VRIJE UNIVERSITEIT

Extending Knowledge Bases from Human-readable Tables

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor of Philosophy
aan de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. C.M. van Praag,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Bètawetenschappen
op woensdag 8 december 2021 om 15.45 uur
in een bijeenkomst van de universiteit,
De Boelelaan 1105

door

Benjamin Bienze Kruit
geboren te Delft

promotor:	prof.dr. P.A. Boncz
copromotor:	dr. J. Urbani
promotiecommissie:	dr. X.L. Dong
	prof.dr. P. Groth
	prof.dr. F.A.H. van Harmelen
	prof.dr. J.R. van Ossenbruggen
	prof.dr. G. Weikum

The research reported in this thesis has been carried out at the CWI, the Dutch National Research Laboratory for Mathematics and Computer Science, within the Database Architectures group, and at the Vrije Universiteit Amsterdam.

This research is financially supported by Amsterdam Data Science (ADS).

SIKS Dissertation Series No. 2021-26

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The cover image of this thesis (photographer unknown, c. 1937) depicts the card index at the Central Register of the Czech Social Security Administration (Prague), built by Vitkovice Ironworks (Ostrava) in 1936. It is still in use.

Acknowledgments

Nobody who starts a Ph.D. knows exactly what they'll be in for. I started mine because I knew I had so much more to learn and to discover, and this certainly held true. Over the years I have been very lucky to get the support from many different people, without whom I would never have been able to complete this work.

First of all, I would like to thank my supervisors Jacopo Urbani and Peter Boncz, for all of their support and guidance during my Ph.D. research. From Jacopo, I've learned what it's like to do research effectively. Our weekly meetings were always engaging, stimulating discussions, which never failed to motivate me. His strategic talent helped me gain a sense of priority in the often jumbled process of science, and his drive for tooling and practical results ("anything that doesn't work yet is called A.I.") gave rise to the most interesting research topics. Above all, I am grateful for his flexibility and broad curiosity for new topics, without which this reseach would never have existed. Peter is not only a profound source of inspiration, but also a patient teacher. His talent for combining the big (ideas) and the small (nanoseconds) consistently led to new insights, and quickly brought out the heart of the matter. I'd like to thank him for opening the doors of computer science to me. I am also very grateful to Henri Bal, for his kindness and guidance in the beginning of my PhD research and his long-term commitment to the amazing diversity of research in his group. His experience and wisdom were always reassuring.

Furthermore, I would like to extend my thanks to my examination committee: Luna Dong, Paul Groth, Frank van Harmelen, Jacco van Ossenbruggen and Gerhard Weikum. Their research has been a great inspiration to me, and it is a humbling experience to have them read and review mine.

During my Ph.D., I was very lucky to be part of two amazing Amsterdam research groups. Having two sets of colleagues not only greatly broadened my view, but also gave me double the companionship over the years. At the CWI, the Database group deeply expanded my respect for foundational technology, and I am very thankful for their support. I would like to thank Mark for all his support over the years, and,

Dean, Thibault, Pedro, Abe, Till, Diego, Tim and Hannes for all their help and fun in and out of the office; I'd also like to thank Arjen, Sjoerd, Stefan and Martin and all other members of the DA group for fostering such a great environment. At the VU, the HPDC group formed a friendly medley that never made me feel out of place. I am very grateful to have worked alongside Unmesh, who is a wonderful friend and teacher of many subjects. His kindness has uplifted my Ph.D. life more than he may realise. I am also thankful for the companionship of Roshan, Vinod, Laurens, Hamid, Vladimir, Leonardos, Corinne and Henk, who made the VU such a nice place to be. My gratitude goes out to Ceriel, Kees, Alexandru, Peter and Marc, for their support and advice. In addition, I have been very fortunate to supervise and collaborate with Hongyu, whose contribution to this thesis is significant, and whose kindness has left a lasting impression.

Finally, this thesis would never have existed if not for the unconditional support from my family. I want to thank my parents for their boundless love and encouragement, and Katja, Mathieu, Toby, Kaðlín, Sander and Roos for all their care. I also want to thank Fenny, Bunna, Tom and Willem for their comfort and their help during the last phase of the work on this thesis. Above all, I am thankful to Hester, who is my everything.

Contents

CHAPTER 1

Introduction

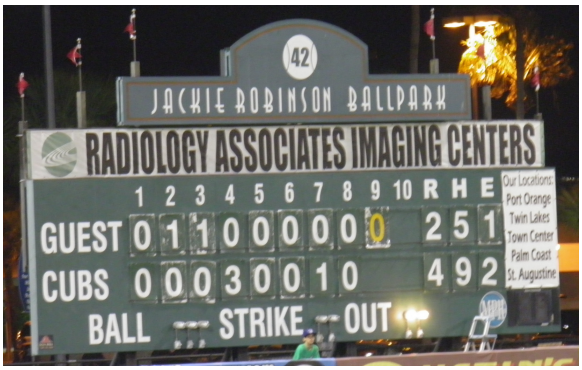
Never before have people had access to as much information as they do now. In its 30-year history, the web has brought about a revolution in the way that data are published and consumed. One might expect that anyone could now effortlessly use any publicly available dataset for new applications and analyses. However, this is not the case. The enormous *volume* of available data has been accompanied by an immense *variety* of data representation. If we want to realize the huge potential of data on the web for innovation, we therefore must understand, and be able to process, the diverse forms in which people structure this data [Berman et al., 2018].

In this thesis, we will focus on *tables*, which people have used for representing structured data since the dawn of writing (Figure 1.1). Organizing information in a table helps people to understand its meaning visually, and also makes it easier to perform mental operations on it [Hurst, 2000]. For instance, we can compare different cells in one column, or look up specific values quickly using labels in the topmost row. In this way, tables are an efficient way to encode human knowledge for effective processing and communication. They occur widely in (non-fiction) text, and have been common in digital documents and web pages for decades [Lopresti and Nagy, 1999].

Figure 1.1: Clay tablet; Pre-Sargonic (*c.* 2500 BC). Sumerian account of silver and other commodities; square tablet, 8 columns.

In the past decade, there has been much research into using tables from the web for various applications [Cafarella et al., 2018]. They have been used to support such diverse tasks as fact lookup for question answering systems [Yin et al., 2011], extending user-provided tables [Yakout and Ganjam, 2012], and providing structured answers to web search queries [Chirigati and Liu, 2016] or complex natural language questions [Pasupat and Liang, 2015].

However, one of the main problems when automatically processing tables is that you always need some *background knowledge* to understand the structure of data on its own. This becomes an important issue when sharing data outside of a single frame of reference, or combining data from multiple sources. Humans have an enormous capacity for interpreting ambiguous structures in context, but computers need explicit definitions. For example, take the bottom table of Figure 1.2: some people might recognize a table like this from baseball scoreboards, inferring that the cells 'Brooklyn' and 'New York' in the leftmost column refer to baseball teams instead of places, and understand that the nine numbered columns refer to the number of runs scored per inning. People without this background knowledge would be just as clueless as the computer, and unable to make such inferences.



Team	1	2	3	4	5	6	7	8	9	R	H	E
Brooklyn	1	0	0	0	0	0	0	3	0	**4**	**8**	**0**
New York	0	0	0	0	0	0	1	0	4	**5**	**8**	**0**

Figure 1.2: A table used on a baseball scoreboard, and a similar web table from an article on Wikipedia. (Wikipedia user web, CC BY-SA 3.0)

The explicit definitions that are used by computers for structuring data are known as data models. The field of *data modeling* aims to discover, define and analyze how people can effectively represent information in software systems [West, 2011]. These days, data modeling is an essential part of the software development process [Kleppmann, 2017]. Organizations that use multiple data models for different software systems, or that want to incorporate external data modeled by third parties, will find themselves needing to reconcile these models in order to combine data in a coherent way. This problem is known as *data integration* [Doan et al., 2012].

Recent developments in large-scale data analysis and machine learning have opened up new, valuable prospects for extracting insights and creating applications that make use of large amounts of data. In 2005, enterprise data integration had already been estimated to yield about half a billion dollars in revenue [Halevy et al., 2005], and the market value had grown to $6.44 billion by 2017 [Marketsandmarkets.com, 2017]. It is one of the building blocks for building succesful *data warehouses* and *data lakes* [Doan et al., 2012], and an essential part of billion-dollar cloud platforms these days such as Amazon Web Services [1], Microsoft Azure [2] and Snowflake [3]. But despite the high

[1] `https://docs.aws.amazon.com/glue/`
[2] `https://azure.microsoft.com/en-us/services/data-factory/`
[3] `https://www.snowflake.com/guides/data-integration/`

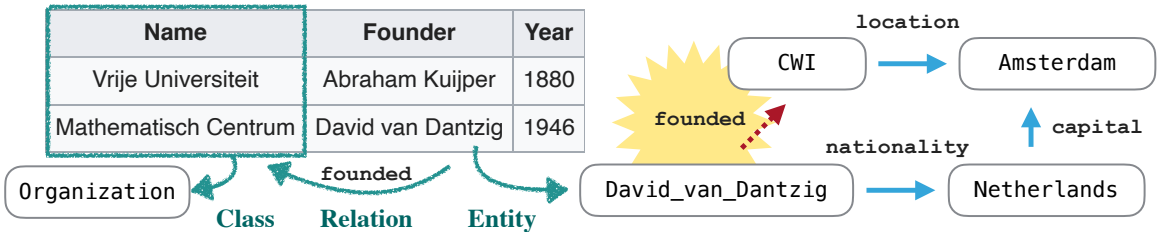


Figure 1.3: A table that expresses information about people who founded organizations (left), and a fragment of a Knowledge Base (right) that is extended with a new fact.

investments, these solutions often do not scale to the enormous variety of data models used in practice, nor are they able to deal with the inconsistencies and conflicts that arise when preparing data for modern applications [Stonebraker and Ilyas, 2018].

Moreover, these systems are designed for integrating data from multiple software applications, but not for processing tables that were created for human eyes. Furthermore, they are built on the assumption of a *closed domain*, where one knows beforehand which topics and types of information should be in the data model. Because of the scale and diversity of tables on the web, these systems are therefore unsuited for our goal of automatically processing arbitrary tables. Instead, to succeed at this task we need a large amount of flexibly modeled, open-domain background knowledge, and effective techniques that leverage this background knowledge for table integration in open-domain settings.

1.1 Integrating Tables with Knowledge Bases

Since the explosive growth of available data on the web, the potential of large-scale data integration has been widely acknowledged [Yoshida et al., 2001]. Whereas the original aim of the web was to make sharing information easier for people, its inventor later started an initiative to make information on the web more accessible to machines: Berners-Lee et al. [2001] coined the term "Semantic Web" for a project that aimed to enrich data with explicit references to well-defined standards, thereby making the web *machine-readable*.

There has been a trend in recent years to leverage this set of standards to create

large databases of general knowledge called *Knowledge Bases* (KBs). These KBs usually define a set of entities (such as `CWI`), entity classes (such as `City`) and relations between entities (such as `location`). The most popular way to express facts in KBs is as *triples* (such as $\langle$`CWI`, `location`, `Amsterdam`$\rangle$). Due to its simplicity, this framework is extremely flexible and thus useful for many scenarios. Such triples can also be seen as describing labeled edges between entity nodes in a graph, so KBs are also commonly known as *Knowledge Graphs* (KGs). Examples of publicly available KBs include DBpedia [Auer et al., 2007] and Wikidata [Vrandečić and Krötzsch, 2014], which are used in many applications such as information retrieval [Seyler et al., 2018], data integration [Kharlamov et al., 2016] and question answering [Diefenbach et al., 2017]. KBs are also used in industrial applications by most major web companies [Noy et al., 2019].

The task of *interpreting* web tables by associating them with concepts from a KB has received much attention from the research community (e.g. [Limaye et al., 2010, Ritze et al., 2015]). This task is typically broken down into three sub-tasks (as shown in Figure 1.3) that associate various components of the table with concepts in the KB, predicting: (1) classes for columns, (2) relations between columns, and (3) entities for cells. After this interpretation step, the information expressed in the table can be transformed using the predicted associations to create new triples for the KB. In this way, the table is not only interpreted using the KB as background knowledge, but it is also used to *extend* the KB with new information [Oulabi and Bizer, 2019]. The potential of extending KBs using web tables has recently emerged as part of a larger trend of semi-automatically constructing KBs from large amounts of data [Suchanek and Weikum, 2013, Zhang et al., 2019]. However, the diversity of real web tables means that neither open-domain table interpretation, nor the design of effective pipelines for extending KBs using web tables is a solved problem.

Therefore, in the next section we will discuss the consequences of table diversity and its effect on extracting and integrating knowledge from tables. This will lead to the four research questions stated in the next section, which we aim to address in this thesis, all based on one fundamental problem:

How do we automatically integrate a diverse set of tables into a coherent, machine-readable format?

1.2 Research Questions and Contributions

The fundamental research problem stated above is very broad, so in this section we will break it down into the four research questions that form the scope of this thesis. We will also give a short description of our contributions with regard to addressing them. Overall, in this thesis we will work to answer the main research problem by investigating which real-world phenomena impede the integration of tables with Knowledge Bases, and develop methods for overcoming these challenges.

1.2.1 Extracting New Facts

There is a trade-off that arises when developing a system that uses a KB for interpreting tables with the goal of extending that same KB. The predicted interpretations of such a system are often largely based on finding *matches* between information in the table and facts in the KB [Limaye et al., 2010, Ritze et al., 2015]. Consequently, the system is able to make more confident predictions if a larger part of the table can be matched. However, this matched information is redundant for the goal of extending the KB because it is already known, while the part of the table that can *not* be matched is most interesting for this goal, as it expresses novel information that is not yet in the KB. Thus, the more the interpretation depends on matches, the more confident its predictions will be, but the less useful it is for KB extension. Existing work has rarely focused on this trade-off. We therefore need to address the following question:

Research Question 1: *How do we measure and account for the trade-off between accuracy and coverage improvement in table interpretation?*

Our contribution towards addressing this problem is twofold. First, we make an empirical contribution by replicating the evaluation of two state-of-the-art techniques

for table interpretation and analyze the amount of novel extractions using two new metrics. We observe that current techniques are biased towards confidence, but this comes at the expense of novelty. Then, we introduce a new algorithm for novelty-oriented table interpretation based on a scalable graphical model using context-dependent entity similarities. Our algorithm further disambiguates cell values using KB embeddings as additional ranking method. Our experiments show that our approach has a higher recall during the interpretation process than the state-of-the-art systems, and that it is more resistant against the bias towards redundancy observed in other systems, because it produces more novel extractions.

1.2.2 Integrating N-ary Tables

Many existing table interpretation methods make simplifying assumptions about the structure of web tables that often do not hold in practice. First, some systems assume that every table has a single *key* column that contains the name of that table's main entities, and all other columns express *attributes* of those entities. However, in many cases web tables express relations between any pair of columns [Braunschweig et al., 2015b]. Second, most systems assume that the table expresses *binary relations*, which involve two entities, while it has been shown that most tables on the web express so-called *n-ary* relations that concern more than two values [Lehmberg and Bizer, 2019]. Third, a common assumption is that tables can be separated into a header, which may contain class or relation labels, and a body, which contains entity names and values. Instead, many web tables actually include entity names and values in their header for editorial layout reasons, especially in the case of tables that express n-ary relations. These phenomena all contribute to the the problem that real-world web tables are more diverse than previously assumed. This motivates us to formulate the following question:

> **Research Question 2:** *How can we extract n-ary facts from diverse real-world web tables and integrate them with a KB?*

We make an empirical contribution towards solving these issues by designing and

evaluating a *pipeline* for large-scale table-to-KB data integration, which consists of several stages. First, our method applies heuristic transformations that reshape tables with the goal of reducing the diversity of their layouts. Next, it clusters similar tables into fewer unified ones to overcome the diversity of their content. Then, the unified tables are linked to the KB so that knowledge about well-described entities propagates to the lesser-known ones. Finally, our method applies a technique to judiciously interpret the table and extract n-ary relations that relies on functional dependencies. Our experiments over 1.5M Wikipedia tables show that our clustering can group together semantically similar tables, which leads to the extraction of many novel n-ary facts.

1.2.3 Creating a New KB from Tables

Although many KBs aim to cover a wide range of topics, it would be unrealistic to expect them to cover all possible domains of interest. In some cases, we would like to be able to use tables to create a new, domain-specific KB from scratch.

The techniques we developed for addressing the previous research questions relied on a large KB for interpreting tables, but in this scenario there is no data at all that we can use for the first step of finding matches. This can be seen as an extreme case of the novelty trade-off from our first research question. In this situation, human effort is needed to create the initial data for training a model that can be used to find and integrate new information. However, labeling individual data points is cumbersome and often prohibitively expensive. Therefore, we are interested in answering the following question:

> **Research Question 3:** *How can we build a coherent KB from tables on a new domain, with minimal human effort?*

Our contribution to answering this question is the design and evaluation of a system called Tab2Know, which we developed for the domain of tables in scientific papers. It addresses the challenge of automatically interpreting the tables in these papers, and of disambiguating the entities that they contain. To solve these problems,

we propose a pipeline that employs both statistical-based classifiers and logic-based reasoning. First, our pipeline applies weakly supervised classifiers to recognize the type of tables and columns, with the help of a data labeling system and an ontology specifically designed for our purpose. Then, logic-based reasoning is used to link equivalent entities in different tables. An empirical evaluation of our approach using a corpus of papers in the Artificial Intelligence domain suggests that ours is a promising step to create a large-scale KB of scientific knowledge.

1.2.4 Designing Pipelines

As we have seen above, new and interesting research challenges arise when applying table integration techniques to real-world data. To account for various phenomena related to table diversity, it is necessary to understand how issues related to the structure, quality and domain of this data influence the integration results. However, current approaches for table integration typically focus on only part of the pipeline, and are hard to debug at the system level.

To facilitate research into real-world table integration, we believe it is therefore essential to move beyond solving table integration subtasks in isolation on simplified benchmarks. By looking at the performance of table integration systems in practice, we can discover new avenues for research and make practical impact. This motivates us to formulate the following research question:

> **Research Question 4:** *How can we support research into table extraction and integration pipelines on realistic data?*

Our contribution towards progress on this challenge is a resource, in the form of a modular library called Takco. While it is tempting to look for a one-fits-all solution, we believe that the most impact can be made by firmly embedding such a library in an existing data science ecosystem. That way researchers and practitioners have the flexibility to adapt and re-configure existing solutions to their needs, and evaluate them on new domains. The design of Takco therefore allows for the loose coupling of various stages in table integration pipelines, and enables users to analyze and

tune these stages for their use-case. We have evaluated Takco on various real-world datasets and found it to return satisfactory performance, while highlighting interesting properties of real-world data that open up new research directions.

1.3 Thesis Outline and Publications

This thesis consists of seven chapters that are mostly based on papers that have been previously published. In this section, we will describe the structure of subsequent chapters and which publications they are based on.

- In **Chapter 2. Background**, we provide an overview of background material necessary to understand this thesis. We discuss some existing work on table interpretation, and its connection to the fields of data management and data integration. We also define some formal aspects of Knowledge Bases and ways to model different types of information in them, as well as techniques that have been developed to construct them (semi-)automatically. Additionally, we discuss several applications of this research and its real-world impact.

- In **Chapter 3. Extracting Novel Facts from Tables**, we address the first research question on measuring and overcoming the novelty trade-off in the table interpretation task.

  The research in this chapter is based on the following published papers:

  - Benno Kruit, Peter Boncz and Jacopo Urbani. Extracting new knowledge from web tables: Novelty or confidence? *Proceedings of KBCOM Workshop at WSDM*, 2018.
  - Benno Kruit, Peter Boncz and Jacopo Urbani. Extracting novel facts from tables for Knowledge Graph completion. *Proceedings of ISWC*, 2019.

- In **Chapter 4. Extracting N-ary Facts from Table Clusters**, we focus on the second research question about the n-ary structure of real-world web tables and their effective integration with KBs.

The research in this chapter is based on the following published paper:

  - Benno Kruit, Peter Boncz and Jacopo Urbani. Extracting N-ary Facts from Wikipedia Table Clusters. *Proceedings of CIKM*, 2020.

- In **Chapter 5. Building a KB from Tables in Scientific Papers**, we deal with the third research question on bootstrapping domain-specific KBs from web tables with minimal human effort.

  The research in this chapter is based on the following published paper:

  - Benno Kruit, Hongyu He and Jacopo Urbani. Tab2Know: Building a Knowledge Base from Tables in Scientific Papers. *Proceedings of ISWC*, 2020.

- In **Chapter 6. A Platform for Web Table Information Extraction**, we address the last research question, on supporting data scientists in creating scalable table integration pipelines on real-world data.

  The research in this chapter is based on the following published work:

  - Benno Kruit, Peter Boncz and Jacopo Urbani. Takco: A Platform for Extracting Novel Facts from Tables. *Proceedings of WWW, Demonstrations*, 2021.

- In **Chapter 7. Conclusion**, we present general conclusions and insights that we developed with regard to work on these problems, as well as future directions for research.

CHAPTER 2

Background and Related Work

Because our goal is to integrate new information from real-world tables into KBs, we must first describe existing techniques for processing these types of data. In this chapter, we will discuss existing work on processing web tables, as well as provide background information to help understand the work presented in this thesis. This will provide the context for subsequent chapters, in particular how the techniques presented therein are related to recent developments in the field.

In Section 2.1 we will introduce some fundamental concepts and terms about data modeling and web tables that we will use in subsequent sections, as well as briefly discuss research on acquiring and pre-processing tables from diverse sources, which will give us an idea of the kind of input data that we will be dealing with. In Section 2.2 we give a short description of Knowledge Bases, how they encode different kinds of information, how to combine them, and also give a brief overview of existing work on creating or expanding them automatically, which is the aim of the work in this thesis. In Section 2.3, we will introduce the problem of *table interpretation*, which will motivate much of the work in this thesis, and discuss existing research that addresses it. In Section 2.4 we will briefly discuss some challenges in the field of Data Integration

and a selection of related work that addresses them, which provides the background for our schema matching and entity linking in Chapters 4 and 5. In Section 2.5 we will discuss the real-world impact and varied applications of web tables, illustrating the relevance of further research in this field.

2.1 Tabular Data

In this section, we will introduce some background concepts related to processing tabular data from different perspectives. It consists of four parts. In Section 2.1.1, we will briefly introduce the fields of data modeling and data integration, and in Section 2.1.2 we will discuss how data is typically presented in tables on the web. Section 2.1.3 concerns the challenges related to detecting and extracting these tables from web pages, and Section 2.1.4 will give an overview of related work on understanding their layout and structure.

In general, it may help to informally visualize many of these approaches as lying on a *spectrum* with varying data heterogeneity. In Figure 2.1, we have adapted the spectrum described by [Ives et al., 2015], and show several aspects of the approaches that we discuss in this chapter.

Due to the large amount of research on the subject of web table processing, we will limit ourselves to topics that are most relevant to later chapters; for a comprehensive survey on web table extraction, retrieval and augmentation, see [Zhang and Balog, 2020].

2.1.1 Data Modeling

Systems for processing large amounts of data are pervasive in the modern world. For these systems to be effective tools, their data must be organized such that it is suitable for human understanding (the *conceptual data model*), and also for efficient processing (the *physical data model*) [West, 2011]. In the following section, we briefly introduce some fundamental concepts in data modeling, which will help us connect our research problems to existing approaches in the field of data integration later on. Readers

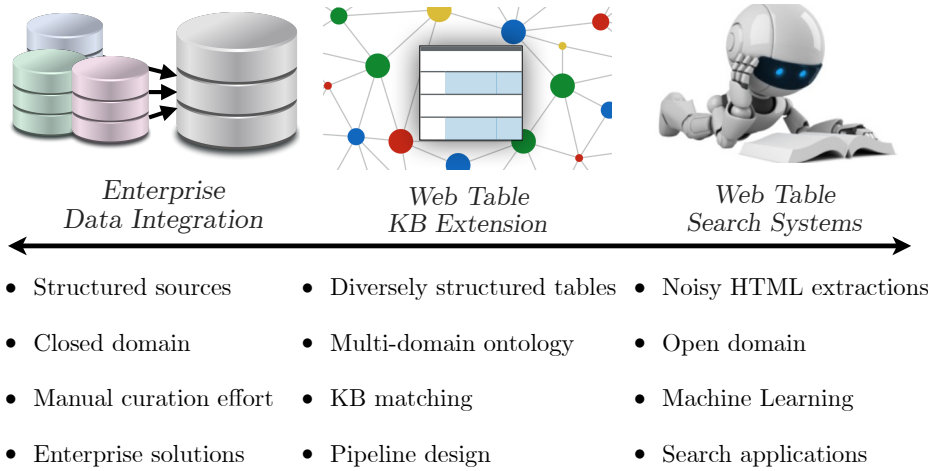


Figure 2.1: A spectrum between Enterprise Data Integration and Web Table Search systems (adapted from [Ives et al., 2015]).

who are familiar with databases may feel free to skip this section, and continue with Section 2.1.2 on web tables.

The Relational Model The most popular approach for managing databases in software applications is known as the *relational model* [Codd, 1970]. Its purpose is to provide a declarative method for specifying data and queries. In this model, users describe the content of a database and what they want from it in a language consistent with first-order predicate logic, and the database management software takes care of the low-level procedures of storing and retrieving the data. Conceptually, the data is organized into *relations*, which are collections of identically structured chunks known as *tuples*. The structure of each tuple in a relation is known as its *schema*, made up of a number of *attributes* constrained by *domains*. For example, these domains may be textual character strings of certain lengths or numbers below a certain size, and are used both for speeding up database operations and enforcing logical structure on the data.

> Adapting the definition of [Abiteboul et al., 2015], we formally state that a relation $r = \{t_1, \ldots, t_n\}$ is a set of n tuples which are all elements of a schema $r \subseteq S$, where S is a Cartesian product of m domains $S = D_1 \times \cdots \times D_m$ and each domain $D \subseteq \mathbf{dom}$ is a subset of all allowed

values **dom**. Each of the m domains corresponds to an attribute of S. In database theory literature, the attributes within each schema are typically given unique names $U \in \mathbf{att}^m$, where **att** is the set of possible attribute names.

The easiest way to visualize a relation is as a table, with tuples as rows and attributes as columns. This has led to people often referring to relations in this model as "database tables". However, for the sake of clarity, in this thesis we will make use of the term *tables* to refer only to tables in (web) documents, and use the term *database relations* for the structures found in relational database systems. In related work on data processing, the term *relational data* is also used to refer to data that is structured like database relations.

Unfortunately, the usage of the term *attribute* in the literature on web tables and information extraction is very ambiguous. Due to the analogy between relations and tables, the term is often used interchangeably to refer to the database concept and to table columns. This can lead to further confusion when discussing attribute *names*: for the relational definition above, attribute names simply refer extensionally to some content of the database, while in literature on tables they are often assumed to intensionally describe some real-world concept. Therefore, in this thesis we will use the term *attribute* to refer to a conceptual characteristic of an entity, while using *database attribute* to refer to the definition above, and using *column* to refer to table columns as they appear in (web) documents. We will refer to a natural-language string that describes some attribute (e.g. "birth place") as an *attribute string*, which is typically accompanied by a *value string* (e.g. "Amsterdam").

In Section 2.1.2 we will use these distinctions to highlight some key characteristics of web tables, and why systems for processing them face challenges that are very different from those of traditional data management systems.

Data Integration Because every computer application has its own unique set of constraints and priorities, the way that data ends up being organized in different systems can vary widely. However, there are many cases where it is necessary to

combine data from different sources that do not have the same structure. The data models of those sources must then be reconciled, which is known as *data integration*. There has been decades of research in the database community to develop techniques for addressing these problems, so we will only briefly and informally sketch the main topics here in order to provide the background for related work that we discuss later on (for an in-depth coverage of the field, see the textbook of Doan et al. [2012]).

Research on data integration has found many applications in enterprise data management software. In such software a distinction is typically made between *physical* and *virtual* data integration [Doan et al., 2012, Ch. 3], concerning how and when the data is stored and accessed. In both approaches, the schema of one data source is defined in terms of the other schema, which specifies how to structurally transform data from one model to the other. In physical data integration, information is copied from the sources into a target database at regular intervals. Typically, this is done in a three-phase process known as *Extract-Transform-Load* (ETL), in which separate modules are designed for each phase to connect the schema and interface of the source to that of the target. In virtual data integration, the target is connected to the source at the moment when information is requested by the application, and the structural transformations of the request and the answer are performed on-the-fly [Van Der Lans, 2012]. Recently, significant advances have been made on using these techniques for creating virtual Knowledge Bases over relational databases [Sequeda and Miranker, 2013, Calvanese et al., 2017].

However, in enterprise scenarios data integration typically takes place under the assumption of a *closed domain*, where the data models of the sources are known upfront, and the transformations are defined by subject matter experts. In situations where less is known about the source data, such as when such knowledge is poorly documented or consists of large volumes of diverse datasets, manually creating these definitions is infeasible, and so these approaches fail to address modern data integration challenges [Stonebraker and Ilyas, 2018]. Thus, the problem of specifying and implementing such transformations is orthogonal to the problem of supporting their (semi-)automatic construction, which is the main research problem of this thesis.

Moreover, we will see in the next section that web tables display an even wider variety of structures than data sources in modern data integration scenarios, and processing this variety at scale requires techniques that lie outside the field of traditional data integration.

2.1.2 Web Tables

In this section, we will discuss tables on the web, and techniques for extracting them and processing their structure. This will give us a general idea of the input data with which we will be working in subsequent chapters.

First, it might be wise to attempt to define what we mean by "tables" on the web and in documents. Although tables are omnipresent in all kinds of communication, they can be notoriously tricky to define in general. [Hurst, 2000] attempted to comprehensively settle the matter with regards to their occurrence in documents, stating that, at the very least, "[a] table is an area of a document which is characterized physically by a grid-like appearance". However, more practically, we would like to consider them in terms of the information that they express, so it might be more useful to follow the statement of Braunschweig [2015] that "[tables] provide a compact representation of similarly structured data."

In most of this thesis, we will assume that this structure is laid out on a row-wise basis, which means we will primarily deal with *horizontal* tables [Yoshida et al., 2001]. For simplicity, we make use of the following informal definition:

> Tables consist of a grid of *cells* made up of horizontal *rows* and vertical *columns*. In some cases, but not always, several rows at the top of this grid form the table *header*, and the rest form its *body*. Within each column, the header cells (if present) express some information about body cells. Each row typically expresses *statements* (or *facts*) that involve values of its cells.

We will expand and formalize these assumptions in Chapter 4, where we discuss the processing of web table structures in more detail.

Compared to database relations discussed in the previous section, it is clear that web tables provide us with fewer guarantees on their structure. Aside from this, there are several other characteristics that we need to keep in mind:

- **Size:** Web tables are much smaller than most database relations; e.g. in the WDC corpus [Lehmberg et al., 2016] they have between 3 and 5 columns, and 12 to 14 rows on average. This provides a weak signal for inferring their meaning, as we will see in Chapter 4.
- **Quality:** Not only must web tables first be extracted from pages in a process that may introduce noise (Section 2.1.3), but we must also account for the fact that any web extraction source may contain spam, incorrect information or inconsistencies.
- **Scale:** Estimates of the number of (clean) web tables range in the hundreds of millions [Cafarella et al., 2008]. Processing these and identifying relevant tables for some use-case requires scalable methods.
- **Variety:** Even when describing similar topics, web tables may use different terms and layouts. Conversely, the same term in different tables may refer to different concepts. This causes ambiguity, which must be resolved for integrating them coherently.

In the following sections, we will give a brief historical overview of existing approaches that have been developed to address these challenges.

2.1.3 Web Table Extraction

The first step when attempting to process web tables is to *extract* them from the web page in which they occur, so we will briefly discuss this step before describing techniques for their further processing (see [Roldán et al., 2019] for a survey of subtasks and approaches). In the work described in subsequent chapters, we assume that this step is already performed, either because we apply existing methods or use a pre-processed corpus of tables. However, the extraction step may introduce artifacts

into the set of tables that we use as input, and so it is important to be aware of the strengths and limitations of these methods.

Perhaps contrary to expectations, the extraction of tables from HTML has received much attention from the research community for over a decade. The reason for this is that although originally HTML tables were given specified semantics in the markup language definition [Raggett, 1996] (based on previous work on military documentation [Bingham, 1995]), their subsequent usage on web pages took on a life of its own when web designers also started (mis-)using them for positioning arbitrary content within page layouts. Therefore, it had become necessary to distinguish these *layout tables* from *content tables* in applications attempting to process tabular information from the web [Hurst, 2001, Wang and Hu, 2002]. Around the same time, Yoshida et al. [2001] introduced a distinction between different types of table structures in terms of the location of value strings and attribute strings in the table. These types include including horizontal tables (i.e. those described by our informal definition) and vertical tables for which the attribute strings form columns instead of rows.

In subsequent ground-breaking research, Cafarella et al. [2008] applied these techniques to create a corpus of 154M "relational" web tables filtered from 14 billion HTML tables in order to support web search for structured data. A more fine-grained table structure taxonomy for table filtering was created by Crestan and Pantel [2011] and used in modified forms by several works to create classifiers for filtering HTML tables at scale [Lautert et al., 2013, Eberius et al., 2015, Lehmberg et al., 2016]. These corpora have been used for various applications (which we describe in Section 2.5), as well as for the creation of annotated benchmark datasets for web table integration [Ritze et al., 2015], which we use in Chapter 3.

2.1.4 Web Table Structures

Even after accurately extracting clean tables, some of their internal structure might remain to be analyzed before it is possible to extract coherent information from them. For example, there might not be a clean separation between cells that express metadata and those that express values that can be integrated, as is shown in several tables of

No. overall		Title	Original airdate
17	1	"A Slight Case of Reincarnation"	31 December 1966
Adam is determined to free an African leader from the spell of the Face. *			
18	2	"Black Echo"	7 January 1967
Adam visits an exiled Russian Grand Duchess, who is requesting him because he is familiar with a pearl necklace that she once owned, and is now trying to reclaim it. However, upon arrival at the Duchess's home, Adam finds two horrifying secrets from his past are about to return...			
19	3	"Conspiracy of Death"	14 January 1967
Adam investigates the murder of an old wartime friend.			

(a) Cells containing long text, wrapped to next row

Catholic, Orthodox, Protestant, and most Oriental Orthodox	Original language (Koine Greek)
Canonical Gospels	
Matthew	Greek (majority view: see note)[N 2][34][35][36]
Mark	Greek
Luke	Greek
John	Greek
Apostolic History	
Acts	Greek

(b) Sub-headers

Sled	Athletes	Event	Run 1		Run 2		Run 3	
			Time	Rank	Time	Rank	Time	Rank
LAT-1	Sandis Prūsis Mārcis Rullis	Two-man	48.10	13	48.06	12	47.92	8
LAT-2	Intars Dīcmanis Gatis Gūts	Two-man	48.07	12	48.22	18	48.14	13

(c) Hierarchical header structure

Year	Population	Year	Population
1749	200	1885	453
1763	273	1907	344
1780	327	1925	330
1800	273	1945	351
1818	329	1955	295
1869	402		

(d) Repeated header

Pos.	Rider	Points	POL	ITA	GER	SWE	GBR	DEN
1	(5) Billy Hamill	113	16	20	9	25	18	25
2	(1) Hans Nielsen	111	18	25	25	9	20	14
3	(4) Greg Hancock	88	12	13	13	16	16	18
4	(2) Tony Rickardsson	86	20	18	16	14	7	11
5	(8) Henrik Gustafsson	80	14	4	18	20	11	13

(e) Hybrid relational-matrix table

Figure 2.2: Examples of real-world web table structures found in Wikipedia articles.

Figure 2.2. In this section, we will discuss existing approaches for dealing with these structures that have inspired some of our work in Chapter 4.

Pivk et al. [2007] propose a "comprehensive functional table model", called TarTar, for transforming tables into logical form. The authors describe a *recapitulation* process which identifies value strings in table headers by decomposing complex headers. In contrast to our approach in Chapter 4, this is only performed when the headers have a tree-like structure, and relies on external resources. TarTar is based on the grounded cognitive table model introduced by Hurst [2000]. More recently, Nagy et al. [2015] found some commonalities in table headers by clustering their cells. Going much further, Halevy et al. [2016] analyzed a large corpus of web tables, discovering structure in table header cells to break them down in terms of multiple values (e.g. "Coffee consumption per person in Sweden, 2005"). Although it extracted many interpretable rules describing structures for expressing n-ary information, the content of the tables themselves was not disambiguated nor integrated with a KB. The method by Lehmberg and Bizer [2016] classifies table columns for detecting layout and list tables, distinguishing tables that express binary facts from those that express facts with a higher arity using inter-table statistics. However, none of these approaches link the n-ary facts to a KB, as we do in Chapter 4.

Spreadsheet Tables Many tables are originally created in spreadsheet software, and then exported to documents or tabular data interchange formats. Some of these formats are extremely simple, using Comma-Separated Values (CSV) in newline-delimited rows of text. However, this simplicity may disregard and throw away much of the original structure of the spreadsheet, which must then be reconstructed before it is suitable for further processing. This is often done manually, a task known as *data wrangling* [Kandel et al., 2011]. Automatic approaches may attempt to extract so-called logical data frames directly from spreadsheets [Chen and Cafarella, 2013, Eberius et al., 2013] or more generally from messy grids in textual files [Christodoulakis et al., 2020].

Recently, much work on table extraction from spreadsheets has concerned predicting

the role of cells within the table, i.e. whether they are part of a hierarchical index or table header, or express values or aggregations. Promising approaches include the use of a wide range of layout and visual markup features in machine learning models [Koci et al., 2016], using pre-trained table cell embeddings from large table corpora [Ghasemi-Gol et al., 2019]. Similar techniques involving deep neural networks have also been applied to predicting the roles of table structures in financial documents for auditing purposes [Li et al., 2020]. However, these techniques are generally limited to a pre-defined domain of spreadsheet tables, and typically require much manual annotation to train. Nonetheless, comparable techniques may be used on web tables with similar structures in the future, and they have inspired some heuristics for table transformations that we describe in Chapter 4.

2.2 Knowledge Bases

In this section, we will look at how knowledge is structured and represented on the Web, and provide an overview of the research on automatically acquiring such knowledge from unstructured web data. It is composed of four parts. In Section 2.2.1, we will introduce Knowledge Bases, and in Section 2.2.2, we will describe approaches for modeling n-ary relations in KBs. In Section 2.2.3, we will give a brief overview of the field of Ontology Matching, which has developed techniques for integrating KBs, and in Section 2.2.4, we will present a short historical overview of techniques for constructing KBs (semi-)automatically. This will provide the background for describing existing work on extending KBs with information from web tables, which we do in Section 2.3.

2.2.1 Overview of Knowledge Bases

Currently, the largest repositories of knowledge on the web are in the form of Knowledge Bases (KBs). We will briefly give a formal definition of KBs, before giving an overview of their history, relevant applications, and challenges concerning their usage and maintenance.

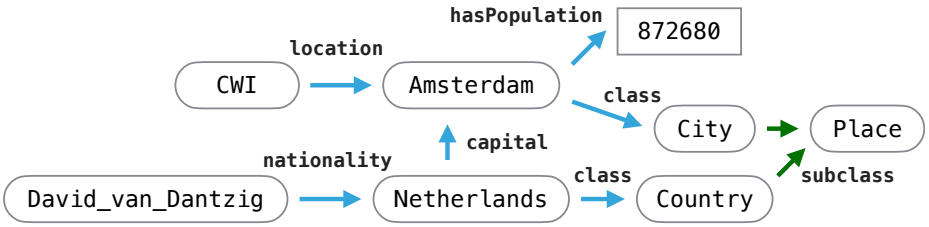


Figure 2.3: An example knowledge base fragment. Entities are represented as nodes, and triples are represented as edges that are labeled with properties. Here, statements about classes and properties (such as the `subclass` relation shown in green) are considered part of the ontology.

> In the following, we will assume that a KB $\mathcal{K}$ is a repository that contains factual statements about a set of entities $\mathcal{E}$, literals $\mathcal{L}$ (which may denote non-entity values of different datatypes) and properties $\mathcal{P}$. It can be seen as a directed labeled graph where the nodes are entities and the edges represent semantic relations, and can therefore encoded with triples of the form $\langle s, p, o \rangle \in \mathcal{K}$ where $s \in \mathcal{E}$, $o \in \mathcal{E} \cup \mathcal{L}$ and $p \in \mathcal{P}$.

Realistic KBs contain facts of various types: For instance, they indicate type memberships (e.g., $\langle$`Netherlands`, `type`, `Country`$\rangle$), encode more generic binary relations (e.g., $\langle$`Amsterdam`, `capitalOf`, `Netherlands`$\rangle$, or numeric attributes, e.g., $\langle$`Amsterdam`, `hasPopulation`, 872680$\rangle$. Additionally, triples may encode complex rules about entities, classes and properties, such as taxonomies and constraints, which together form a so-called *ontology* [McGuinness et al., 2004]. Figure 2.3 shows a fragment of an example KB, represented as a graph.

> In this thesis, given the triple $\langle s, p, o \rangle$, we say that the pair $\langle p, o \rangle$ is an *property-value pair* of s. Similarly, we may refer to a set of values with a function V over entities and properties. For example, an entity e is often associated to a finite set of labels $V(e, \texttt{label})$, which are the values for the `label` property where $V(e, \texttt{label}) = \{l \mid \langle e, \texttt{label}, l \rangle \in \mathcal{K}\}$.

All of this information is typically encoded using the Resource Description Framework (RDF) [Hayes, 2004], which is a W3C standard which ensures that each entity and property of interest has a Uniform Resource Identifier (URI), such as a web

address, and which provides several ways of encoding data structures and types. In most RDF database management systems (also known as *triple stores*), such data can be queried using the SPARQL Protocol and RDF Query Language [W3C, 2013]. Data represented using RDF is also known as *Linked Data* if its identifiers reference other datasets [Bizer et al., 2011], and is said to be part of the *Semantic Web* when incorporated in the metadata of web pages [Berners-Lee et al., 2001].

Examples of popular KBs include Wikidata [Vrandečić and Krötzsch, 2014], which is created collaboratively as a public wiki project similar to Wikipedia with a combination of manual edits and semi-automatic data imports. It is intended to support various projects by the Wikimedia Foundation such as populating pages of low-resource language Wikipedia editions. Another primarily manually built KB, important for historical reasons, is Freebase [Bollacker et al., 2008], which was acquired by Google and now lives on as part of their proprietary KB. Another popular KB is DBpedia [Auer et al., 2007]), which is semi-automatically constructed by transforming Wikipedia structures such as *infoboxes*, which express human-readable attribute-value pairs about the article's subject entity, using manually created templates. Lastly, the various incarnations of the YAGO project (such as [Pellissier Tanon et al., 2020]) constitute a tremendous effort towards automatically creating a rich, open-domain KB from unstructured and semi-structured sources.

Many industry, business, government and non-government organizations use KBs for a wide range of tasks [Hogan et al., 2020, Noy et al., 2019], and their flexibility makes them useful in many different data management paradigms. They have been used for such diverse applications as Information Retrieval [Seyler et al., 2018], Data Integration [Kharlamov et al., 2016], Question Answering [Diefenbach et al., 2017], Semantic Search [Dietz et al., 2018], Data Mining and Knowledge Discovery [Ristoski and Paulheim, 2016].

Graph-structured KBs are also increasingly known as *Knowledge Graphs* (KG) in the literature, especially when they are stored and accessed in software systems that support powerful graph analysis operations. However, not all their representative power is always needed: for instance, Cafarella et al. [2018] state that "[...] in

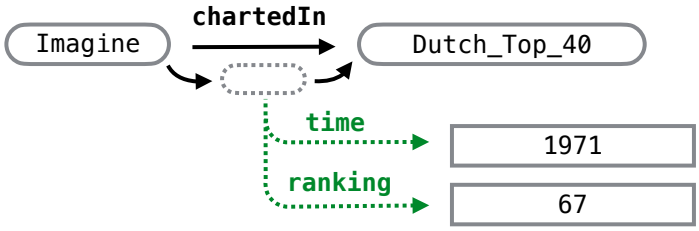


Figure 2.4: Representing n-ary facts in a KB with qualifiers (green dotted arrows), using reification (dotted node).

practice, knowledge 'graphs' act more like 'collections of relational tables with informal schema enforcement': entities tend to exhibit a small number of fixed attributes, and computing graph-style measures is a rare use case." We will use both "KB" and "KG" interchangeably in this thesis, but we will opt more often for the former because of the challenges involved with of formally representing n-ary relations as graph structures, which we will discuss in the next section.

2.2.2 Representing N-ary Information

Almost all major Knowledge Bases are concerned mainly with binary relationships between entities, but it is often necessary to move beyond this focus to express useful knowledge [Suchanek, 2020]. For example, we may want to indicate at which points in time some information was true, or under which circumstances a statement holds. Additionally, for many purposes it is useful to keep track of the provenance of statements: how they became part of a KB, or according to which source some statement is true. In general, we are interested in the ways that KBs represent information about the *scope* of factual statements. First, we will informally define what we mean by n-ary facts:

> A *n-ary fact* is a factual statement with n arguments, where n is also known as statement's *arity*. Whereas a *binary fact* (i.e., 2-ary fact) can be naturally expressed with a triple, facts where $n > 2$ require multiple triples to represent.

Many different approaches exist for modeling n-ary facts in KBs. For example, Wikidata makes use of *qualifiers* to express such complex factual statements. A

qualifier is a subordinate property-value pair assigned to a statement that annotates the fact with additional information [Vrandečić and Krötzsch, 2014]. For instance, the fact $\langle \texttt{Amsterdam}, \texttt{hasPopulation}, 872680 \rangle$ is annotated in Wikidata with the qualifier $\langle \texttt{atTime}, 2020 \rangle$. Other qualifier properties may for instance concern measures of confidence, the mode or subtype of a relation, the role of either entity in the statement, the method of measurement, or the degree to which a statement is disputed. Figure 2.4 shows an example of an n-ary fact represented with qualifiers.

More generally, n-ary statements in KBs are often modeled with reification [Noy et al., 2006], of which we will use the following approach:

> To model n-ary statements, every fact $\langle s, p, o \rangle$ is mapped to a fresh entity $q_{s,p,o} \in \mathcal{E}$ and every qualifier $\langle p', o' \rangle$ of this fact is mapped to a triple $\langle q_{s,p,o}, p', o' \rangle$. In RDF, the fresh entity often takes the form of a *blank node*, which is an entity that does not have a global URI but gets a local, application-specific representation when processed. For convenience, we will consider them entities like any other, denoted as the set $\mathcal{B} \subset \mathcal{E}$.

Other KBs make use of different strategies. In Freebase, n-ary relations are modeled with "Compound Value Types" (CVT). For every n-ary relation, it introduces a separate class for scoping factual statements. For example, the CVT `music.group_membership` is associated with `music.group_membership.from` as a temporal property. The CVTs were created with the explicit goal of adding scopes, resulting in Freebase having a high coverage of n-ary data. Next to being integrated with Google's proprietary KG, much of this data has been imported into Wikidata [Pellissier Tanon et al., 2016]. In YAGO2 [Hoffart et al., 2013] (and subsequent versions) the temporal and geographical scopes of factual statements are modeled explicitly, as well as detailed provenance information. Over the years, the project has pioneered the process of automatically acquiring and reasoning over scoped factual information, recently integrating its ontology with Wikidata [Pellissier Tanon et al., 2020]. In contrast, DBpedia does not deal with n-ary information explicitly, as its goal is to provide up-to-date linked datasets that connect information from Wikipedia to other sources. However, it does feature some classes

in its ontology that may be considered a form of reification, such as `CareerStation`, which captures the temporal scope of employment.

[Hernández et al., 2015] explore the different n-ary data modeling scenarios for Wikidata with respect to storage space, query complexity and processing time in five popular triple stores. A different paradigm for representing n-ary information that does not decompose them into triples works with *property graphs*, in which nodes and edges are annotated with property-value pairs. Although they are similar conceptually to other representations, property graph databases typically have specialized query languages and make other trade-offs in their physical data layout [Francis et al., 2018]. Recently, the RDF* model has been proposed to extend the semantics of RDF for modeling n-ary data [Hartig, 2017]. While this is a promising research direction, it is still in the process of being incorporated into mainstream KBs.

2.2.3 Ontology Matching

Similarly to integrating data from relational databases, KBs have their own data integration challenges. However, the flexibility of data representation in KBs distinguishes those challenges from those in the relational world, and has motivated much research. This forms the field of *ontology matching*, which "aims at finding correspondences between semantically related entities of different ontologies." [Euzenat et al., 2007]. The ontology matching "life cycle", with user interaction, reasoning and maintenance, resembles that of its relational counterpart, but characterized by attention to standards such as RDF, meticulous benchmarking, and powerful, theory-grounded formalisms. In particular, the field has a tradition of yearly benchmarking tracks at the ISWC conference, the Ontology Alignment Evaluation Initiative [Abd Nikooie Pour et al., 2020].

Seminal work in the field includes COMA [Do and Rahm, 2002], which uses a combination of matching approaches, and introduced a comprehensive library of matching functions. Since then, the state-of-the-art has advanced in several directions [Shvaiko and Euzenat, 2013], partly through large benchmark datasets from the biomedical domain [Jiménez-Ruiz et al., 2013], but much work has been done in a

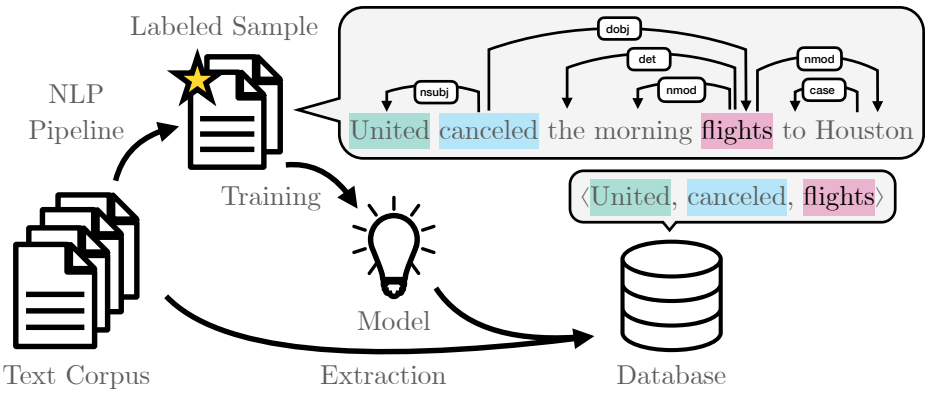


Figure 2.5: Open Information Extraction typically makes use of a complex Natural Language Processing pipeline to weakly supervise the training of a lightweight extraction model, which is then used to create a database of text-based facts.

manner isolated from other fields such as the database community [Euzenat et al., 2007]. However, in its own domain progress continues to be made on challenges such as user validation [Dragisic et al., 2016] and learning matches with the use of embeddings [Zhang et al., 2014, Kolyvakis et al., 2018].

2.2.4 Knowledge Acquisition

Automated Knowledge Base construction from unstructured sources has been a long-standing challenge in AI research (for a comprehensive survey, see [Weikum et al., 2020]; for a survey on Information Extraction in a Semantic Web setting see [Martinez-Rodriguez et al., 2020]). The field has a long history in Natural Language Processing and Information Retrieval, but in the past decade or so has acquired its own identity by exploring new forms of supervision for training machine-learned extraction models. In this section, we present a brief summary of the history of knowledge acquisition for building a KB from text sources and semi-structured data, and discuss open challenges that are relevant to the work presented in this thesis.

The first techniques for extracting structured knowledge from text made use of a fixed vocabulary of properties, aimed at finding specific types of information in large document sets. Later, techniques were developed that scaled to large web datasets without being limited to a pre-defined set of properties, which became known collectively as Open Information Extraction (OpenIE), illustrated in Figure 2.5.

Among them, TextRunner [Banko et al., 2007] was the first to introduce a method that, in contrast to work that had come before, did not require specific annotated data for extracting facts for populating a coherent database. It exploited the structure of language through the use of a pre-trained dependency parser to create training data from a sample of web pages. Wu and Weld [2007] then introduced KYLIN, which uses the structured (but noisy) data in Wikipedia Infoboxes to train Conditional Random Fields (CRFs) for extracting facts from Wikipedia text. They expanded this work to create WOE [Wu and Weld, 2010], but "abstracts these examples to relation-independent training data to learn an unlexicalized extractor, akin to that of TextRunner", At this stage several open challenges remained unresolved, in particular (1) efficient recognition of patterns, (2) disambiguation of concepts, (3) ensuring their consistency, and (4) extracting higher-arity and temporal facts [Weikum and Theobald, 2010].

With regard to challenge (1) and (3), much progress was made in the next decade. The Never-Ending Language Learner NELL [Carlson et al., 2010] bootstrapped category classifiers in a semi-supervised way by combining a hand-crafted taxonomy of entities and properties with large-scale extraction from the web. Prospera [Nakashole et al., 2011] scaled up knowledge harvesting by combining pattern-based gathering of relational fact candidates with weighted consistency reasoning. Google's Knowledge Vault [Dong et al., 2014a] was the first system in this space to combine extractions from different modalities (text, tables, and web page metadata) and score those extractions using embeddings to ensure their quality, while its Biperpedia project [Gupta et al., 2014] resulted in a very large, approximate ontology by using information from its search query stream as a supervision signal. In the LODIE system [Gentile et al., 2015], a novel type of supervision was generated by matching attribute-value string pairs on a page to dictionaries of attribute-value string pairs which were pre-assembled from across many RDF datasets. Further, Lockard et al. [2019] created a clean supervision signal from HTML page structures using label propagation that exploited visual features. Such HTML structures are related to the tabular structures that we reshape in Chapter 4.

Regarding challenge (2), Zhang et al. [2019] integrated OpenIE extractions with KBs by using co-occurrence based embeddings of both KB properties and OpenIE relations. Notably, they do not train entity embeddings, which means they could deal with unseen entities. Because many quality problems are caused by such unseen entities, Peng et al. [2019] performed distant-supervised relation extraction from Wikipedia infoboxes, using type-aware entity linking that takes unlinkable entities into account. The problem of unlinkable entities is closely related to that of extracting novel information, which we address in Chapter 3.

Finally, challenge (4) about higher-arity fact extraction has of yet seen limited attention. One exception is HighLife [Ernst et al., 2018], which infers textual patterns for n-ary fact extraction from grammatical parse trees and combines them while calculating their salience. Then, it matches these to new data while ensuring consistency with rules and constraint reasoning, combining partial matches in a unification step. Another exception is EventKG [Gottschalk and Demidova, 2018], an event-centric KB extracted from several large- scale entity-centric knowledge graphs. However, much work remains to be done on this front, to which we have hoped to contribute with the work described in Chapter 4.

2.3 Table Interpretation

Because of the pervasive presence of tables on the web and their usefulness for representing structured information, the question of how to automatically integrate their content with existing structured data has received much attention from the research community. In particular, the challenge of *table interpretation*, i.e. mapping them to KBs, has been long-standing, but has nonetheless seen increased work in recent years. However, to understand and critically assess this work, it is important to make explicit the various assumptions on which it is founded.

In this section, we will first attempt to describe what these assumptions mean for research in this space, and then discuss this research in terms of the main focus areas. It is composed of three parts. In Section 2.3.1, we consider some common assumptions

Chart (1971)	Rank
Canada Top Singles (*RPM*)[134]	15
Netherlands (Dutch Top 40)[135]	67
Netherlands (Single Top 100)[136]	82

Figure 2.6: A web table about music chart ranks that violates many commonly held assumptions: (1) the rows do not describe attributes of entities, (2) its context is essential for integrating these facts (in this case, the song in question), (3) the header structure contains a value that is part of the n-ary facts, (4) it contains compound cells, and (5) it does not express facts that can be matched to existing knowledge in the KB.

about tables that underpin existing table interpretation techniques, which we briefly discuss in Section 2.3.2. In Section 2.3.3, we then examine a number of systems for table interpretation under specific constraints.

2.3.1 Common Assumptions

Below, we list several distinctions related to the semantic interpretation of tables, that have been highlighted by previous research, which contribute to commonly held assumptions in the field. An example of a table that violates many of these assumptions is shown in Figure 2.6.

1. *Entity-Attribute Tables.* Most work on table interpretation assumes that tables have a *entity-attribute* (EA) structure, in which there is a single column that contains the labels of some *subject entities* and the other columns all express attributes of those entities. Every row of such tables thus neatly describes a single entity, and every non-subject cell potentially allows for the extraction of one KB triple. A minority of research instead considers *multi-concept* tables, attempting to discover multiple subject columns and their complex interdependencies [Braunschweig et al., 2015b, Wang and Ren, 2018].

2. *Self-Contained Tables.* Another commonly held assumption is that web tables are interpretable on their own, that is, they express coherent and accurate

information when analyzed without taking their context into account. However, tables that violate this assumption have been shown to be very common on the web [Lehmberg and Bizer, 2016], and essential temporal metadata has also been shown to often occur on web pages outside of web tables [Oulabi and Bizer, 2017].

3. *Relational Mappings.* Next, the assumption that web tables are structured like database relations influences the way that KB mappings can be expressed. However, in practice the structure of the table may be non-relational, for example containing footnotes, sub-headers, sub-totals, or header cells that contain value strings instead of attribute strings. Research that assumes the tables to have relational structure typically map pairs of columns (of which one is the subject column for EA tables) to KB properties. Alternatively, Vu et al. [2019] describe D-repr, a language for specifying the semantics of tables with complex structures, which allows for the scalable integration of diversely structured data sources with large number of records.

4. *Atomic Cells.* With regard to the content of table cells, most table interpretation approaches assume that each cell contains one value. However, related work that we mentioned in Section 2.4 makes a distinction between *compound* cells, which contain multiple *values*, and *atomic* cells, which contain only one. More generally, the problem of breaking up compound cells into constituents is related to web list parsing [Elmeleegy et al., 2011, Shen et al., 2012, Gupta and Sarawagi, 2009, Chu et al., 2015]. However, as far as we have been able determine, no research has been done on interpreting tables with complex cells.

5. *Isolated Matching.* Finally, most approaches interpret tables on their own, while a minority leverage patterns that can be extracted by analyzing a set of tables collectively. In the first scenario, the interpretation of a table is only based on surface correspondences between its contents and the interpretation model, but in the second case it may be possible to create table unions or propagate predicted labels between mappings. This is closely related to work on table

clustering and similarity search (Section 2.4.1).

Unless otherwise stated, the research below assumes that the input tables are entity-attribute, self-contained, relational, atomic-valued, and isolated.

2.3.2 Approaches and Systems

As described in Chapter 1, the problem of Table Interpretation can be decomposed into three sub-tasks: Entity Linking, Column Typing and Property Matching (see Figure 1.3 in the previous chapter). First, we will describe methods that perform these tasks in isolation, and then describe research into performing them jointly. Afterwards, we will discuss approaches that have been developed for table interpretation with unusual assumptions.

Only Entity Linking First, we will discuss approaches that are limited to linking entities in web tables to KBs. An early approach by Yosef et al. [2011] was primarily oriented towards entity linking in text, but also reported reasonable results on web tables. Bhagavatula et al. [2015] introduced a Probabilistic Graphical Model with prior cell-entity link probabilities based on web hyperlink anchor statistics, disambiguated through inference on a similarity graph based on semantic relatedness with context co-occurrence features. Efthymiou et al. [2017] presented an approach to entity linking that relies on a task-specific index, which consists of labels, descriptions and property-value pairs from multiple KBs; its disambiguation step makes use of word vectors and Personalized PageRank-based inference on a similarity graph.

Only Column Typing Other work limits itself to the task of column typing. One straightforward approach is to exploit an index of entity labels without disambiguating them [Zwicklbauer et al., 2013]. However, recently more advanced approaches have been developed for this task which leverage carefully supervised deep neural networks in ensemble with such candidate indexes in order to generalize across domains, such as ColNet [Chen et al., 2019]. Similarly, Hulsebos et al. [2019] described Sherlock, which leverages a rich feature set that includes statistics, character distributions,

induced regular expression patterns, and word and paragraph embeddings and was trained on a diverse dataset from four sources. This was later extended to create Sato [Zhang et al., 2020], which takes the context of a data column into account. However, recently Khurana and Galhotra [2020] showed that competitive performance can also be achieved without deep learning, instead combining a heterogeneous set of indexes while accounting for mixed-type columns.

Only Property Matching A large separate family of table interpretation systems limit themselves to property matching. The simplest approaches perform string similarities between the column headers and attribute strings or cell values and entity labels [Polfliet and Ichise, 2010, Efthymiou et al., 2016]. The long-running Karma project [Gupta et al., 2012, Taheriyan et al., 2016] considered the entire life-cycle of semi-automatically creating and maintaining complex mappings, also for multi-concept tables. When no overlap between the table and KB can be assumed at all, it uses supervised models based on features of the column header and cell values [Pham et al., 2016]. Other approaches focus on matching tables to relations from Open Information Extraction [Venetis et al., 2011, Wang et al., 2012] or exploit occurrences of cell value pairs in a corpus of text [Sekhavat et al., 2014, Cannaviccio et al., 2018b], and others perform supervised learning using the KB as training data [Ermilov and Ngomo, 2016]. In a different supervised paradigm, Taheriyan et al. [2016] perform full source modeling, making use of a domain ontology and a set of known models, which allows for user-driven source discovery and service composition.

Combined Matching The first system for interpreting web tables was introduced by Limaye et al. [2010], who also formalized the problem into the three sub-tasks described above. This approach consists of a probabilistic graphical model that makes supervised predictions based on a large number of features. Subsequent work approached the sub-tasks as a pipeline, using a task-specific KB [Venetis et al., 2011, Syed et al., 2010, Wang et al., 2012], while others sped up predictions by limiting the feature set [Mulwad et al., 2013] or using distributed processing [Hassanzadeh et al.,

2015].

Two non-proprietary systems that perform all three sub-tasks jointly have been evaluated on realistic datasets, T2KMatch and TableMiner+. Therefore, these are the systems that we examine in Chapter 3. The T2KMatch system [Ritze et al., 2015] implements a series of matching steps that match table rows to entities, using similarities between entity property values and the table columns. Beginning with entity candidate selection from cell values, the value-based similarities between cells and entity properties are then used to filter the candidate set and property predictions, after which they are recomputed on the new selection. This is iteratively repeated until the similarities stop changing and, if it exceeds a confidence threshold, a final prediction is chosen. The TableMiner+ system [Zhang, 2017] consists of two phases that are alternated until a certain confidence level has been reached. The forward-learning phase builds up predictions on a row-by-row basis, after which the backward-update phase uses these to guide the interpretation of the rest of the data. This process is repeated until convergence. Recently, Wang et al. [2021] propose a Deep Learning approach to predict both column types and properties that makes extensive use of intra-table signals and contextual information through an attention mechanism in a multi-task learning objective, which outperforms previous approaches on a large set of tables from the music and Wikipedia domains.

2.3.3 Constrained Table Interpretation

In this section, we briefly discuss systems that associate components of web tables to KBs with additional knowledge or under specific constraints. Such situations may call for radically different approaches by significantly affecting the ambiguity of cell values or enabling the usage of different feature classes.

Closed Domains The systems evaluated in this section are primarily designed for open-domain table interpretation. In closed-domain settings, assumptions can reduce the redundancy of extractions. For example, the work of [Ran et al., 2015] models the incompleteness in the domain subset of the KB by estimating class probabilities

based on relations between entities, which the limited domain makes tractable. The systems of [Wang et al., 2012] and [Venetis et al., 2011] use a probabilistic KB created from a web corpus for supporting table search. This type of KB offers many strategies for improving the recall of new knowledge because it allows for an explicit model of low-confidence facts.

Web Resources Several models use large web text corpora in addition to the information from the KB. TabEL [Bhagavatula et al., 2015] uses the anchor text of hyperlinks on the web to create a prior for instance matching that takes the popularity of entities into account. Additionally, it exploits co-occurrences of anchor links to entity candidates on Wikipedia pages for predicting a coherent set of entities. The work of [Sekhavat et al., 2014] creates a set of syntactic patterns from the ClueWeb09 text corpus featuring entities from properties in the KB. Given query pairs of entities from tables, the syntactic patterns from text featuring the query pair are matched to the patterns in the set. A probabilistic model then allows for the prediction of properties from the KB. A similar approach is taken by [Cannaviccio et al., 2018b], who use a language model instead of extracted syntactic patterns. This approach queries a search engine with the entity pair, and classify the text that occurs between the entity mentions.

Numeric Matching A separate direction is the matching of numeric columns, either with metrics for numeric distribution similarity [Neumaier et al., 2016] or sophisticated ontologies of quantities and statistical models [Ibrahim et al., 2016]. Recent work has tackled this problem using a task-specific quantity KB [Nguyen and Takeda, 2018] and focusing on associations between quantity names and units [Yi et al., 2018].

Collective Matching Using a large collection of tables, some approaches bootstrap using high-confidence or annotated tables to tables that do not match the KB as well. The work of [Ritze and Bizer, 2017] collect statistics from confident predictions on tables with matching entities on which column headers occur for which KB properties.

These statistics are useful for matching properties to columns when the instance matching fails. Another way of using information from a large corpus of tables is by stitching tables together to form higher-quality expanded tables, as in [Lehmberg and Bizer, 2017] which uses both header and row content, along with information in the table context, for construction union tables. Similarly, the work of [Cannaviccio et al., 2018a] exploits high-confidence column-property matches from wikipedia tables with many hyperlinks to match tables with identical headers. They then extract a large number of confidence-weighted facts for KB expansion. These approaches are highly relevant to our work in Chapter 4, where we expand upon this work by focusing on matching n-ary relations.

2.4 Data Integration at Scale

As we discussed in Section 2.1.1, the scale and heterogeneity of modern data integration scenarios have motivated research into new approaches. In this section, we will discuss some techniques that have been developed for these use-cases that have influenced our work in Chapters 4 and 5. Following [Doan et al., 2012, Ch. 5], we make a distinction between *instance matching*, which is concerned with finding matches between individual tuples in the databases and *schema matching*, which finds matches between their schemas. Therefore, this section is composed of two parts. In Section 2.4.1, we will discuss techniques for scalable schema matching, and in Section 2.4.2 we will discuss techniques for scalable instance matching and entity linking.

2.4.1 Schema Matching and Profiling

Because web tables typically have only few rows and columns, they provide a relatively weak signal for further data processing tasks such as integration. Therefore, much research has been done on taking advantage of regularities in a large corpus of web tables, calculating similarities between pairs of tables, or creating web table clusters. In this section we will highlight a selection of techniques from the literature. These techniques are related to traditional schema matching approaches (for an overview of

these, see [Rahm and Bernstein, 2001]), but must additionally account for the diversity of web table data.

For example, one difference between traditional data integration approaches and methods developed for web table integration is the incorporation of the table context. Ling et al. [2013] introduced a method for creating union tables which made use of the table context for constructing "hidden" attribute-value string pairs. These were then aligned and added to the union tables as extra columns, resulting in large, coherent tables. In Chapter 4, we make use of the same idea for integrating the table context and stitching multiple tables, while expanding it by reshaping and aligning table structures in order to boost KB integration performance.

Similarity Estimation When automating schema matching on a large scale, it becomes prohibitively expensive to perform many types of similarity calculations between each pair of tables under analysis. For example, given a collection of ten million web tables which contain 100 cells on average, calculating some similarity score between all tokenized cell pairs would naively result in 10^{18} function calls, which, if a billion of them were performed per second, would take over 300 years.

Instead, there are different kinds of data representations that allow for quick approximate similarity calculation, known as *nearest neighbor search*, which produce an estimate of the most similar instances for any query instance. One powerful example is locality-sensitive hashing (LSH) [Gionis et al., 1999], which uses hashing to create small bit-vectors that represent the sets and efficient indexes for fast retrieval. Even if the estimates are not perfect, with proper configuration it still brings down the number of pairs that need to be compared to a manageable number. This is known as *blocking* (Figure 2.7). An extension of LSH is LSH Ensemble [Zhu et al., 2016], which uses multiple indexes for approximating set containment, which is useful for finding overlapping domains in data sources.

Another technique that is increasingly popular in this space is creating *embeddings*: real-valued dense vectors of objects generated through optimization. These objects may be words, cells, or even columns and tables. Their representations reflect factorized co-

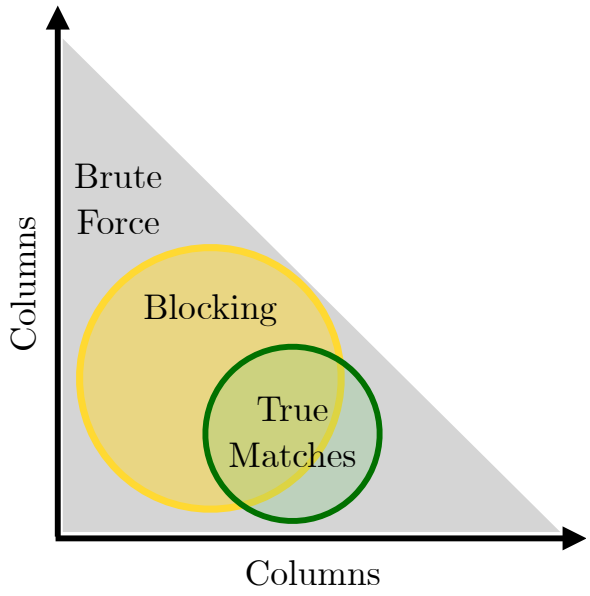


Figure 2.7: In the task of schema matching, we aim to find matches between table columns, but comparing all pairs of columns is expensive. Blocking efficiently finds a subset of column pairs that is designed to overlap the set of true matching pairs as much as possible (adapted from [Papadakis et al., 2020a].

occurrences within the data, and can be used for various data integration tasks [Deng et al., 2019, Gentile et al., 2017]. Scaling up, Dong et al. [2020] explore partitioning strategies on an embedding-based similarity index in the case that the data lake is large and the index cannot fit in main memory. In the recent work of Fetahu et al. [2019], pairs of Wikipedia tables with equivalent or subsumed domains are discovered efficiently in a three-step process. First, they use a blocking approach to find relevant article pairs based on a set of relatedness scores. Then, candidate table pairs are found using a Random Forest classifies trained with a rich set of contextual table features. Finally, these pairs are then analyzed using a neural network classifier which predicts how their domains are related, which identifies many related table pairs. However, none of these techniques have use these schema matches for integration with KBs, for which a semantic interpretation step is needed.

Compound Value Processing In contrast to the well-defined relational attribute domains described above, web tables also often contain compound columns, in which cells contain multiple values arranged in some pattern (for instance, names and birth

dates). Various approaches have been developed for processing such patterns, of which we mention a small selection here. Chaudhuri et al. [2006] describe SSJoin, a system for suggesting similarity joins based on prefix filtering of extracted q-grams. Similarly, Zhu et al. [2017] infer syntactic transformations for complex cells that allow for efficient table join suggestions. Finally, Jin et al. [2020] exploit multilingual Wikipedia tables about the same topics to infer patterns for column datatypes, which can be used for automated data cleaning. Techniques such as these are essential for extracting clean data values, but none of the discussed work has integrated these with KBs, which remains an open problem.

Data Profiling Many techniques for discovering regularities in relational datasets are part of the process of *data profiling* [Naumann, 2014]. This includes *functional dependency (FD) discovery* [Papenbrock et al., 2015], which aims to find constraints between database attributes where the values of one set of attributes (the *key*) uniquely identifies the value of others. However, in noisy datasets it is useful to be less strict: Wang et al. [2009] use *probabilistic* FDs for detecting low-quality data sources and normalizing schemas. Other data profiling tasks include finding *inclusion dependencies*, which express subset constraints, and *unique column combinations* [Roick et al., 2016]. In general, data profiling can be useful for data cleaning [Batini et al., 2009], decrypting the schema of ETL pipelines [Albrecht and Naumann, 2012], and analyzing knowledge graphs [Jentzsch et al., 2015]. We will adapt some of these techniques for finding n-ary KB matches in Chapter 4, where we provide a formal description and algorithm for our use of data profiling in this subtask.

2.4.2 Instance Matching and Entity Linking

Because so many database tuples and column rows describe entities and their attributes, the existing approaches for instance matching and entity linking are closely related. The problem of matching records in databases is very old [Fellegi and Sunter, 1969] and has still received considerable attention in database research in recent years (96+ papers in VLDB, KDD, etc. in 2009-2014). When concerning multiple structured datasets it is

often known as *record linkage* [Christen, 2012] or instance matching [Doan et al., 2012], and it is known as *entity matching* [Böhm et al., 2012], or *entity resolution* [Papadakis et al., 2020b]) when the items of interest are assumed to represent entities. The related task of associating text or semi-structured data to entities in a KB is typically known as *entity linking* (which is the term we will use throughout this thesis) or *entity disambiguation* [Shen et al., 2015]. The literature on this subject is vast, and thus we only give a very brief sketch of recent methods that are relevant to our own work (for recent surveys on the topic, see [Christophides et al., 2020] and [Papadakis et al., 2020c]).

Some works have focused on entity linking using knowledge bases (e.g., Linda [Böhm et al., 2012]). Another line of work has focused on crowdsourcing, e.g., Das et al. [2017] and citations therein. Other, recent research has explored the usage of embeddings for this task: For instance, Cappuzzo et al. [2020] have shown how to construct embeddings from tabular data, training from random walks on a tripartite cell occurrence graph (connecting rows, cells, and columns). Finally, Zhu et al. [2020] designed a scalable Graph Neural Network (GNN) encoder and aggregation method for entity matching between KBs that is makes use of node features to generalize to new nodes by encoding the node neighborhood, handles multiple entity types in one model. One of the most prominent developments in entity linking is Magellan [Konda et al., 2016], which is a tool to help users to perform entity matching, providing different implementations of matching and blocking algorithms. Additionally, Mudgal et al. [2018] have recently studied the application of deep learning for entity matching, but concluded that it does not outperform existing methods on structured data with few property-value pairs per entity, such as our scenario in Chapter 5.

2.5 Impact and Applications

In this section, we discuss the ways that open-domain table processing is used in practice. It is composed of two parts. In Section 2.5.1, we will describe systems that support searching for tables, returning parts of tables from queries, or suggesting

specific ways to integrate them. In Section 2.5.2, we will discuss systems that support explorative, user-driven data integration of heterogeneous datasets.

2.5.1 Search, Query Answering and Integration

Much research has been done on supporting user queries over large collections of web tables. In contrast to the work described in this thesis, the approaches below focus on search and do not integrate the tables with a KB.

Cafarella et al. [2008] described the initial results of the WebTables project at Google, in which they construct a join graph for supporting table-building user queries. They make use of an Attribute Correlation Statistics Database (ACSDb), which disambiguates column headers to some extent. This long-running project led to much work that was integrated into public-facing applications such as data search and spreadsheet tools [Cafarella et al., 2018]. Similarly, Wang et al. [2012] described a system for bootstrapped taxonomy population from web tables, and its application in semantic table search at Microsoft. Related techniques have been integrated into the its Bing search engine as a fact lookup module for question answering [Yin et al., 2011], extracting entities from text context using wrapper induction.

The seminal work of Das Sarma et al. [2012] presented an approach to find related tables based on their schema and contents. This results in table pair candidates for joins or unions, but does not have a way to canonicalize entities or detect duplicates. Similar to the Octopus system described above, Bhagavatula et al. [2013] described a relevant-join prediction approach for Wikipedia tables which allows users to find join candidates for query tables. Pimplikar and Sarawagi [2012] introduced Wwt, a table search engine that returns schema-mapped tables given a query that consists of keywords describing attributes. Similarly, the Infogather system [Yakout and Ganjam, 2012] supports several user query types, and explicitly assumes that tables are entity-attribute tables with a subject column.

Other tasks that have been supported using large collections of web tables are: finding synonyms for attribute strings [He et al., 2016], recommending relevant attribute-value string pairs given entity queries [Kopliku et al., 2011], predicting header cells

for headerless query tables [Braunschweig et al., 2015a, Hancock et al., 2019], and extending tables by using hyperlinks in large web table corpora [Lin et al., 2010].

Apart from tables that appear within the text of web pages, there has been much research into increasing the accessibility of tabular datasets that are published explicitly for re-use. These so-called "Open Data" tables, often distributed in the form of separate files such as spreadsheets, are often difficult to automatically process due to the fact that they deal with specialized topics and narrow domains. Though some progress has been made to (semi-)automatically interpret these datasets (e.g. [van der Waal et al., 2014, Neumaier and Polleres, 2018]), most approaches are still limited to recommending possibly overlapping datasets [Nargesian et al., 2018], recommending datasets based on their meta-data [Wang et al., 2020], and other search techniques [Chapman et al., 2020].

2.5.2 Dataspaces

Although fundamental research into these techniques is vital, it is decidedly non-trivial to put them to use and integrate them into a coherent system. Franklin et al. [2005] introduced the concept of *Dataspaces*, which allow for gradual, on-demand, pay-as-you-go enhancement of schema mappings. The techniques they introduced are useful when the data is heterogeneous yet complete, when the system is not the only application accessing the data, and when the integration approximate and incremental. This was expanded by [Salles et al., 2007] to make use of "trails" that users create when accessing a data search engine, to gradually build schema mappings. The seminal work of Cafarella and Halevy [2009] introduced Octopus, which offers several operations useful for dataspaces with regard to search, context extraction, and table extension.

Braunschweig et al. [2012] described a related Dataspace Support Platform that operates on Open Government Data, which allows non-expert users to explore and integrate data in using relational database operators. Ives et al. [2015] described the requirements and results of creating a similar system in practice for the neuroscience domain. Other examples are Data Civilizer [Deng et al., 2017], which uses effective data discovery techniques for *polystore* systems, where data may reside in heterogeneous

relational databases, and Aurum [Castro Fernandez et al., 2018], which combines efficient techniques to build, maintain and query an *Enterprise Knowledge Graph*, including a profiler that finds approximate dataset matches, and a resource-efficient sampling signature method for keeping track of source data changes. Finally, Galhotra and Khurana [2020] demonstrated S3D, a system for searching tables similar to Octopus, but with added semantic operations and integration of results from KBs.

2.6 Summary

In this chapter, we discussed various existing work on processing web tables, and provided some background information to help understand the work presented in this thesis. First, we described the nature of web tables, and described their characteristics in connection to the relational model, as well as challenges regarding their extraction and structure. The result was a number of issues concerning their size, quality, and variety, which motivate the design of our approaches in subsequent chapters. Then, we introduced Knowledge Bases, together with some formal notation that is most fundamental to the work in this thesis. We also discussed some popular KBs, and provided an overview of techniques for combining them and extending them automatically, which is the main focus of our research. We then provided an overview of existing work on the problem of table interpretation, which we will build upon in our novelty-oriented evaluation of existing methods and the development of our own table interpretation approach in Chapter 3. This was followed by an brief sketch of related work that addresses challenges on data integration at scale, which provides the background for our schema matching and entity linking approaches in Chapters 4 and 5. Finally, we described some examples of systems that show the real-world impact and varied applications of research on web tables, illustrating the relevance of further work in this field.

CHAPTER 3

Extracting Novel Facts from Tables

In this chapter, we first describe our experiments on existing systems and their bias towards extracting known facts. Then, we propose a new, novelty-oriented method for extending a Knowledge Base (KB) from tables.

Existing techniques for table interpretation tend to focus on information that is already in the KB, and therefore extract many redundant facts. Our new method aims to find more novel facts. We introduce a new technique for table interpretation based on a scalable graphical model using entity similarities. Our method further disambiguates cell values using KB embeddings as additional ranking method. Other distinctive features are the lack of assumptions about the underlying KB and the enabling of a fine-grained tuning of the precision/recall trade-off of extracted facts. Our experiments show that our approach has a higher recall during the interpretation process than the state-of-the-art, and is more resistant against the bias observed in extracting mostly redundant facts since it produces more novel extractions.

3.1 Introduction

Much of the world's information exists as tabular data. These are available as HTML tables on web pages, as spreadsheets, or as publicly available datasets in many different formats. There has been more than a decade of research in recognizing, cleaning and capturing these so-called *web tables* [Cafarella et al., 2018]. Because of their relational nature, such large collections of web tables are suitable for supporting table search [Yakout and Ganjam, 2012] or for answering specific factual queries [Sun et al., 2016]. In certain web tables, the rows describe attributes or relationships of entities. This makes them suitable sources for extending the coverage of Knowledge Bases (KBs), which is a task known as *KB completion*.

In order to perform KB completion from web tables, we must first align their structure and content with the KB, a problem broadly referred to as *table interpretation*. Table interpretation has been the subject of several prior works (see Chapter 2, section 2.3). Similar to our research, these works primarily focus on the interpretation of entity tables, i.e., tables where each row describes one entity and columns represent attributes. In this case, the interpretation process consists of two operations. First, each row is linked with an entity in the KB, and optionally the entire table is linked to a class. Then, each column is associated to a KB property.

After the table is correctly interpreted, we can extract novel triples from the table and add them to the KB. This last operation is also known as *slot-filling*, as the empty 'slots' in the KB are filled with new facts [Ritze et al., 2016]. Table interpretation strongly affects the quality of slot-filling, since errors in the former can no longer be corrected. Because of this, state-of-the-art table interpretation techniques (an overview is given in Chapter 2, Section 2.3) aim for high precision by pruning out many potential assignments already at early stages. While high precision is desirable in some contexts (e.g., table search), we have observed that this strategy leads to a high number of redundant extractions during slot-filling, since only the assignments to entities that are well-covered in the KB are retained, which we describe in Section 3.2.

With the goal of maximizing the number of novel extractions without sacrificing

precision, we present a new method for KB completion from web tables. In contrast to existing approaches, our method does not prune out row-entity assignments, but performs the interpretation by performing inference over all possible assignments using a Probabilistic Graphical Model (PGM). The PGM uses label similarities as priors, and then updates its likelihood scoring to maximise the *coherence* of entity assignments across the rows using Loopy Belief Propagation (LBP). Coherence is not computed using a predefined metric (such as class membership) but is automatically selected as a combination of properties that are shared by the entities in the table. This is a novel feature of our method which makes it capable of working with KBs with different topologies and/or data models. Since we use both label similarities and coherence based on salient common attributes, our method is able to maintain a high accuracy for the row-entity assignments. At the same time, it is also able to return many more novel extractions since we did not prune out any assignments.

We also propose an approach to perform slot-filling by disambiguating attribute cells in a novel link-prediction framework. Our approach makes use of embeddings of KB entities and properties to improve the quality of the disambiguation whenever label matching is not sufficient. This furthers our aim to find novel facts for KB completion.

We compared our method to several state-of-the-art systems. Additionally, we evaluated the performance of these systems with regard to the redundancy of the facts that they extract from the tables. Our experiments on popular benchmark datasets show that our approach yields slightly lower precision, but significantly higher recall on entity predictions. This leads to many more novel extractions than what is possible with existing methods. Finally, to test the scalability of our method we perform a large-scale evaluation on 786K tables from Wikipedia.

3.2 Motivation: Measuring Redundancy

We are interested in using tables to expand a knowledge base, which we represent as a set of facts KB over a set of entities E_{KB}. The table extraction technique is expected

to yield a new set of facts F_P over E_{KB}. For a set of tables in an held-out set, it is standard practice to manually annotate a gold standard set of facts F_G and use them for evaluating how many facts in F_P are correct. Notice that F_G might contain facts that are either in KB or not.

So far, current techniques have been evaluated w.r.t. the set of true positives $F_G \cap F_P$ (correctly extracted facts) and false negatives as $F_G \setminus F_P$ (valid facts that were missed). These measures do not capture the *redundant* information that was extracted. We propose two additional metrics to capture it. The first, which we refer to as *positive redundancy* (R^+), is the fraction of correctly extracted facts that are already in the knowledge base, and the second, *negative redundancy* (R^-), is the fraction of annotated but unextracted facts that are in the knowledge base:

$$R^+ = \frac{|(F_G \cap F_P) \cap KB|}{|F_G \cap F_P|} \qquad R^- = \frac{|(F_G \setminus F_P) \cap KB|}{|F_G \setminus F_P|} \tag{3.1}$$

In other words, R^+ is the redundancy of the true positives, and R^- is the redundancy of the false negatives. Notice that these measures work only if $F_G \setminus F_P \neq \emptyset$ and $F_G \cap F_P \neq \emptyset$ but these are conditions largely satisfied in practice. For example, imagine a table of 3 columns and 10 rows yielding $|F_G| = 20$ relational facts, of which 13 are already in the KB. If the technique at hand predicts only 10 correct facts but 8 of these are already in the KB, then $|F_G \cap F_P| = |F_G \setminus F_P| = 10$, $R^+ = 0.8$, and $R^- = 0.5$. Intuitively, R^+ reports the ratio of redundant information that was predicted, while R^- reports the ratio of redundant information that was not predicted. The two measures do not complement each other because they depend on both the predictive power of the technique and on the amount of novel information we can extract from the table. For instance, if the table yields only novel facts, than both R^+ and R^- will be zero regardless how good the extraction technique is.

Therefore, in order to have a more fine-grained view on the actual performance of the technique, we introduce also two *recall* scores that are sensible to the redundancy. The first, *novel recall* ($Q^\star$), is the fraction of new facts that is correctly extracted, and the second, *redundant recall* ($Q^\dagger$), is the fraction of redundant facts that is correctly

extracted:

$$Q^* = \frac{|F_P \cap (F_G \setminus KB)|}{|F_G \setminus KB|} \qquad Q^\dagger = \frac{|F_P \cap (F_G \cap KB)|}{|F_G \cap KB|} \tag{3.2}$$

In other words, Q^* is the recall of novel annotations, and $Q^\dagger$ is the recall of known annotations. For the example above, $|F_G \setminus KB| = 7$, $|F_G \cap KB| = 13$, $Q^* \approx 0.29$, and $Q^\dagger \approx 0.62$. We argue that the measures $R^+, R^-, Q^\star, Q^\dagger$, which we call the *ReNew* measures, offer a better view of the performance than the used precision and recall because they take into account the actual number of novel knowledge that we can extract. Moreover, we can use them to formally state our hypotheses as follows:

$$R^+ > R^- \tag{H1}$$

$$Q^* < Q^\dagger \tag{H2}$$

Note that we are specifically interested in quantifying the extent to which table interpretation systems will extract redundant facts, and not in the general performance of the systems with regard to novel extractions. If we were only interested in the systems performance on the quality of their extracted facts, we could discard all redundant facts, and measure precision and recall of the remaining set of novel extractions. While these measures are useful for tuning systems for performance, in this work we are interested in analysing the behaviour of existing systems with regard to both novel and redundant extractions.

3.2.1 Experiments

In our experiments, we evaluate the systems T2KMatch [Ritze et al., 2015] and Table Miner+ [Zhang, 2017], since they represent the current state-of-the-art for our task. We use two datasets from Ritze et al. [2015], which contain HTML tables from a large, cross-domain web scrape that are known to express relational data (i.e., not used for HTML layout purposes). These datasets contain a realistic selection of tables from the web, with manual annotations from DBPedia Auer et al. [2007], a popular up-to-date

		T2D-instance		
Task	System	Precision	Recall	F_1
Entities Pr.	T2K Match	0.96	0.75	0.84
	TableMiner+	0.97	0.70	0.81
Type Pr.	T2K Match	0.93	0.92	0.92
	TableMiner+	0.94	0.91	0.93
Relations Pr.	T2K Match	0.83	0.60	0.70
	TableMiner+	0.75	0.40	0.51

		T2D-complete		
Task	System	Precision	Recall	F_1
Relations Pr.	T2K Match	0.74	0.33	0.46
	TableMiner+	0.65	0.21	0.32

Table 3.1: Precision, recall and their harmonic mean F_1 for all datasets, tasks and systems.

KB. Thus, they are ideal for our purpose.

We evaluate the performance of the three key operations performed during the table interpretation process: 1) *Entity prediction*, which calculates the entity associated to each cell value. This process yields facts of the type $\langle entity, label, cell\_value \rangle$; 2) *Type Prediction*, which is the process to associate classes to the table's columns. This process yields facts of the type $\langle entity, type, class\_name \rangle$; 3) *Property Prediction*, which is the process that determines the relationships between two different cells. This process yields facts of the type $\langle entity, property, entity \rangle$. In order to evaluate the performance of the system, we need manual annotations for each of these three tasks.

The first dataset, called T2D-instance gold standard, consists of 233 tables with manual annotations of 25703 entities, 233 types and 420 properties from DBpedia. Using these annotations we could extract 75216 facts. The second (much larger) dataset, called T2D-complete gold standard, consists of 1748 tables. In this case, the manual annotations were limited to types (i.e., columns) and properties (between columns) from DBpedia. Entities (e.g. cell values) are not annotated. The lack of entity annotations precluded the usage of this dataset of our purposes. To fix this problem, we created a silver-standard set of entity annotations for each system by leveraging class predictions. If a class was correctly predicted for a column, then we

		T2D-instance			
Task	System	R^+	R^-	$Q^\star$	$Q^\dagger$
Entities Pr.	T2K Match Match	1.00	0.71	0.00	0.77
	TableMiner+	1.00	0.74	0.00	0.73
Types Pr.	T2K Match	1.00	0.71	0.00	0.77
	TableMiner+	1.00	0.74	0.00	0.73
Relations Pr.	T2K Match	0.81	0.22	0.10	0.63
	TableMiner+	0.83	0.32	0.04	0.36

		T2D-complete			
Task	System	R^+	R^-	$Q^\star$	$Q^\dagger$
Relations Pr.	T2K Match	0.82	0.15	0.12	0.78
	TableMiner+	0.83	0.29	0.07	0.47

Table 3.2: ReNew metrics: positive redundancy, negative redundancy, novel recall and redundant recall for all datasets, tasks and systems.

assumed that the matching with the entities was correct. Using this method, we were able to extract 56509 and 48173 facts for T2KMatch and TableMiner+, respectively. This still allows us to extract facts and calculate the redundancy scores. However, by definition in this case the entity and type matchings will be ideal. Therefore, we report the results only for the properties.

Accuracy

Initially, our goal was to reproduce the experiments presented in literature and compare the two systems using the standard precision, recall, and F_1. Running the T2KMatch system was not particularly challenging since the implementation was already configured to use DBPedia. However, the TableMiner+ system Zhang [2017] was originally designed for the Freebase knowledge base, and used services that have been discontinued. To provide a meaningful comparison, we minimally altered the system to use the same KB as the one used by the T2KMatch framework. Moreover, we replaced the Freebase module by a label index and KB query index in Lucene, using the same interface. In this way, we could provide a meaningful comparison of the two systems.

The precision and recall were calculated following the definitions in Ritze et al.

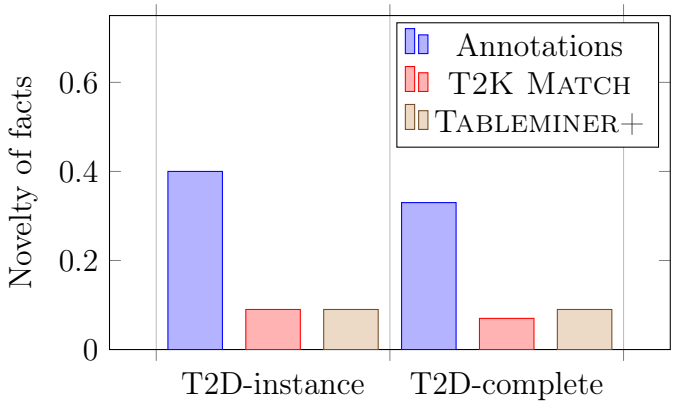


Figure 3.1: Avg. fraction of facts that is new, for facts extracted from annotations ($\frac{|F_G \backslash KB|}{|F_G|}$) and predictions ($\frac{|F_P \backslash KB|}{|F_P|}$).

[2015] and Zhang [2017]. Predictions of equivalent classes and properties were considered correct, and so were single-level superclasses Ritze et al. [2015].

The results we obtained are presented in Table 3.1. We can see that both systems perform very similarly on the T2D-instance dataset, particularly regarding entity and type prediction. The scores for T2KMatch are comparable to the scores published in Ritze et al. [2015] which means we were able to reproduce the experimental analysis presented in literature. With the T2D-complete dataset, T2KMatch significantly outperforms TableMiner+ on the property prediction task. This may be due to the coherence that T2KMatch calculates between all columns of a table and ontology properties of a class, but a further error analysis is outside the scope of this work.

Redundancy

We report in Table 3.2 the four ReNew metrics on the set of facts from entity, types, and property predictions for both systems. First, we observe that R^+ and Q^* is 1 and 0 for the entity and types predictions respectively. These values are expected since by design both systems only accept mappings that are already in the KB. Thus the positive redundancy is maximal while novel recall is minimal. Notice however that R^- and $Q^\dagger$ do not have ideal values, which means that the systems do miss valid entity and type predictions because of this policy.

Furthermore, we can see that both hypotheses hold in every case and with both

systems. This means that a large part of correct extracted facts is redundant (R^+ close to 1) and that a large part of unextracted facts is novel (low R^-). Moreover, the ratio of novel facts that is extracted (Q^*) is lower than the ratio of redundant extractions ($Q^\dagger$). If we compare the two systems, then we observe that negative redundancy (R^-) is higher with TableMiner+, which indicates that a larger fraction of missed facts are known. Also, novel recall (Q^*) is lower, which means that the system retrieved a smaller fraction of all novel facts that could have been extracted.

One could argue that since the tables do contain some redundant information, then it should be expected that the system also returns redundant predictions. To consider this case, Figure 3.1 reports the average fraction of facts that is not in the KB for facts extracted from the gold standard ($\frac{|F_G \backslash KB|}{|F_G|}$) and for predictions returned by the two systems ($\frac{|F_P \backslash KB|}{|F_P|}$). This figure clearly illustrates that the ratio for the two systems is smaller than the amount of redundant information from the tables, which confirms (from an empirical perspective) our conclusion that the state-of-the-art is biased towards the prediction of already-known knowledge rather than novel one.

3.3 Our Approach

3.3.1 Background

In this section, we provide some background on concepts on which our approach is based. For background on Knowledge Bases and Table Interpretation, see Chapter 2.

PGMs In this chapter, we employ Probabilistic Graphical Models (PGMs) to perform the interpretation. PGMs are a well-known formalism for computing joint predictions [Pearl, 1989]. For a given set of random variables, conditional dependences between pairs of variables are expressed as edges in a graph. In these graphs, variables are connected if the value of one influences the value of another. The connection is directed if the influence is one-way, and undirected if both variables influence each other. The behavior of the influence on every edge is expressed by a function known as the *potential function*. When performing inference in a PGM, information from

the nodes is propagated through the network using the potential functions in order to determine the final distribution of the random variables.

Slot filling The survey by Ji and Grishman [2011] discusses approaches and challenges to the slot filling task in the context of textual information extraction. Most systems use distant supervision for textual pattern learning, and some employ cross-slot reasoning to ensure the coherence of multiple extracted values. Recently, work on Universal Schemas by Riedel et al. [2013] has allowed the joint factorization of textual extractions and KB properties and this boosts slot-filling precision.

Data fusion In the field of data fusion, systems explicitly aim for high recall and use a post-processing filter to improve precision. In [Muñoz et al., 2014], the extracted facts are filtered using machine learning models, and in [Dong et al., 2014a] they are filtered using a sophisticated statistical model of the KB. In [Ritze et al., 2016], the system of [Ritze et al., 2015] is used to interpret a large collection of web tables, after which the extracted facts are filtered using several strategies. However, only 2.85% of web tables can be matched, which is attributed to a topical mismatch between the tables and the KB.

KB Embeddings We also make use of latent representations of the KB [Nickel et al., 2016] to filter out incorrect extractions. In particular, we consider TransE [Bordes et al., 2013], one of the most popular methods in this category. The main idea of TransE is to "embed" each entity and property into a real-valued d-dimensional vector (where $d > 0$ is a given hyperparameter). The set of all vectors constitutes a model Θ of $|\mathcal{E}|d + |\mathcal{P}|d$ parameters which is trained so that the distance between the vectors of entities which are connected in $\mathcal{K}$ is smaller than the distance between the ones of entities which are not connected.

To this end, Θ is trained using stochastic gradient descent, minimizing the L_1-distance $d(\mathbf{s} + \mathbf{p}, \mathbf{o})$ (where $\mathbf{s}$, $\mathbf{o}$ and $\mathbf{p}$ are the vectors that are associated with the entities s and o and property p for which $\langle s, p, o\rangle \in \mathcal{K}$) while maximizing an equivalent distance for triples that do not occur in $\mathcal{K}$.

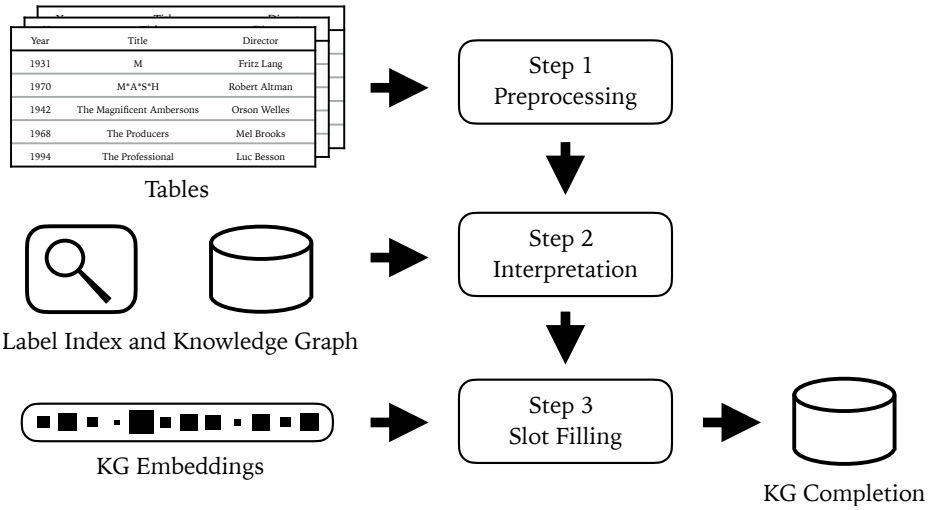


Figure 3.2: Overview of our approach.

Training Θ is done by minimizing the loss function

$$\mathcal{L}_\Theta = \sum_{\langle s,p,o\rangle \in \mathcal{F}} \sum_{\langle s',p,o'\rangle \in S_{\langle s,p,o\rangle}} [\gamma + d(\mathbf{s}+\mathbf{p},\mathbf{o}) - d(\mathbf{s}'+\mathbf{p},\mathbf{o}')]_+ \tag{3.3}$$

where: $\mathbf{s}, \mathbf{s}', \mathbf{o}, \mathbf{o}', \mathbf{p}$ are the vectors associated to the entities s, s', o, o' and type p respectively; $\gamma \geq 0$ is an hyperparameter that defines the minimum acceptable margin; $d(\cdot)$ is a distance function (typically the L_1 norm), $[x]_+$ returns the positive part of x, and $S_{\langle s,p,o\rangle} = \{\langle s,p,o'\rangle \mid \langle s,p,o'\rangle \notin \mathcal{F}\} \cup \{\langle s',p,o\rangle \mid \langle s',p,o\rangle \notin \mathcal{F}\}$, i.e., it is a set of "corrupted" facts which are not in $\mathcal{K}$. Once training is completed, the model can be used to perform link prediction, i.e., to estimate the likelihood of unseen facts: if the distance of these facts is small, then these are more likely to be true.

3.3.2 Intuition

Our approach applies three operations in a sequence (Figure 3.2). The first operations pre-processes the input table to determine whether it is suitable for extracting novel triples. The second one interprets the table, i.e., maps the rows and columns to concepts in $\mathcal{K}$. Finally, the third operation extracts (novel) facts from the table so they can be added to $\mathcal{K}$.

Preprocessing Not all tables are suitable to extract factual information. In this research, we are interested in tables that describe entities and their binary properties, like the one reported in Figure 3.3. First, we identify the key-column (if any) using the heuristics proposed by [Ritze et al., 2015] which consists of selecting the column with most unique non-numeric values breaking ties by choosing the leftmost one. This heuristics works well in practice for this dataset so we apply it without modifications. Only the tables with valid key columns are considered since these are the only ones for which we can (potentially) extract factual knowledge.

Table Interpretation We apply a novel procedure to link the rows and columns in the table to entities and properties in $\mathcal{K}$ respectively. This is the most important task in the pipeline because these links are crucial for performing slot-filling.

In line with other methods, we assume that the entities described in the rows are similar to each other according to some unspecified criterion. It is important that the similarity between entities is computed accurately because it is precisely this value that later allows the system to select certain mappings instead of others. Both T2KMatch and TableMiner+ consider two entities as similar if they are instances of the same class. In principle this is an effective choice, but it becomes suboptimal when the KB does not contain the relevant class or when the similarity is defined over different attributes.

To overcome these limitations, we introduce a more general similarity function which considers all shared attributes between two entities, and then employ a statistical inference procedure to maximize the coherence between the assignments. We use a PGM as statistical framework to infer likelihood scores between rows and candidate mentions, and employ Loopy Belief Propagation (LBP) [Koller et al., 2009] as inferencing method to change these scores depending on how the various candidates are similar to each other.

The scores returned by LBP indicate which are the assignments that maximize the coherence between the entities. We combine these scores with other ones that measure the syntactic overlap between the labels of the candidate entities (and their

attributes) and the content of the row. The score obtained by this combination is used to rank the candidate entities and to select the one with the highest score.

Since the scores for the entity candidates consider all shared neighbors and not only class instantiation, our method can deal with a larger variety of tables. This is beneficial because it leads to higher recall and thus can improve the extraction of novel information.

Slot filling Finally, we proceed with the extraction of novel triples from the content of attribute columns. Suppose that the previous operation has mapped the content of row i and non-key column j to entity e_i and property p_j respectively. If we manage to associate the content of the cell at coordinates (i, j) to an entity $e_{i,j}$, then we can construct the triple $\langle e_i, p_j, e_{i,j} \rangle$ and add it to $\mathcal{K}$. To this end, we first retrieve from $\mathcal{K}$ all entities with a label that is similar to the content of the cell. In case this operation returns multiple candidates, we make use of the context that we have already acquired to rank them. Namely, we know already that $e_{i,j}$ should be linked to e_i through p_j. Therefore, we rank the candidates depending on the likelihood that such link exists. To compute the likelihood scores of potential links, we use the embeddings computed by TransE [Bordes et al., 2013]. After the candidates are re-ranked, we select the one with the highest likelihood score and return the extracted triple.

3.3.3 Table Interpretation

Now, we describe our method for performing table interpretation in more detail. Figure 3.3 shows the computation that takes place during the interpretation, using table (a) as a motivating example. In this case, the key-column is the second one ("title") but its content is ambiguous since the values can refer to movies, TV series, or books. For instance, the second row can refer to the TV serial `M*A*S*H` or to the movie `MASH`, as is shown in Figure 3.3b. The goal of this task is to map as many rows ρ as possible to corresponding entities in $\mathcal{E}$ and each column c to one property in $\mathcal{P}$. To this end, we perform a sequence of five operations, described below.

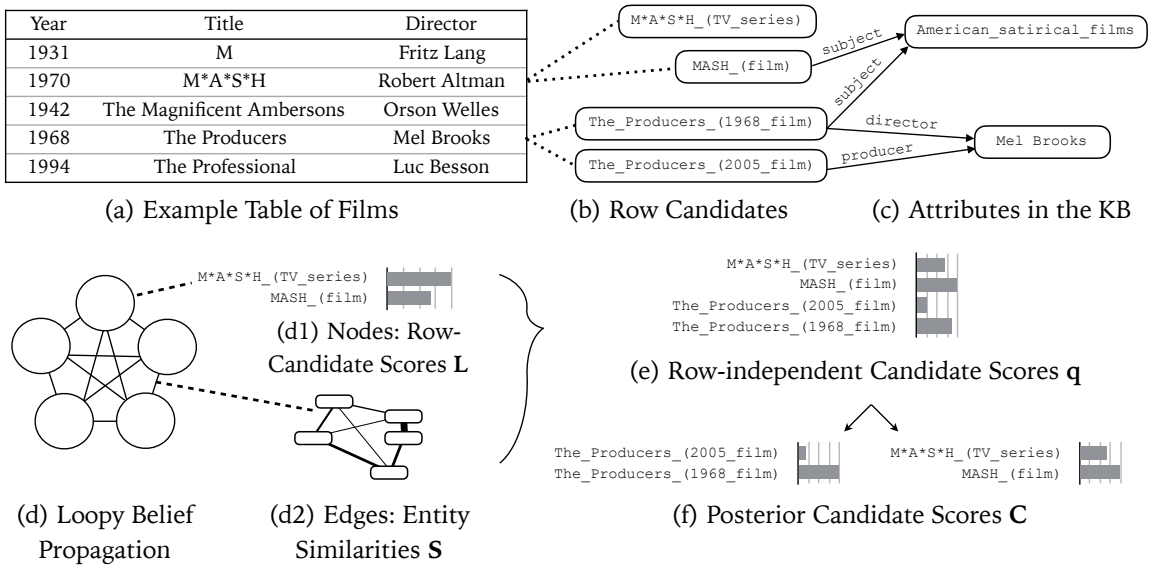


Figure 3.3: Schematic representation of our table interpretation method.

Step 1: Candidate Entity Selection

First, we identify the key-column (if any) using the heuristics proposed by [Ritze et al., 2015], which consists of selecting the column with most unique non-numeric values breaking ties by choosing the leftmost one. This heuristics works well in practice so we apply it without modifications. Only the tables with valid key columns are considered since these are the only ones for which we can (potentially) extract factual knowledge.

For every cell in the key column, we then select a set of entity candidates. We represent this computation with the function $\mathsf{Cand}(\rho)$ which takes in input a generic row ρ and returns all entities in $\mathcal{E}$ which are potential candidates with ρ. This function is implemented by 1) indexing all the labels in $\mathcal{K}$, 2) retrieving the labels which contain the cell value of the key column, 3) returning the entities associated to the labels. Let $e \in \mathsf{Cand}(\rho)$ be a potential entity candidate for row ρ. We call the tuple (ρ, e) a *row-entity assignment*. If $\mathsf{Cand}(\rho)$ is empty, then ρ is ignored. Otherwise, the table interpretation process will determine which row-entity assignment should be selected.

Example 3.3.1. In the table (a) of figure 3.3, we assume that the second column is the key column, because it is the leftmost column with non-numeric unique values. The label index returns a set of scored candidate entities per row (b).

The label matches are ranked using length-normalized smoothed TF-IDF. In our case, the query corresponds to the cell value of the key column, while the documents are all labels in $\mathcal{K}$. Identically to [Ritze et al., 2015], we (1) take only the first result if it is much better than the next and (2) take the top three labels otherwise. The final set of candidates consists of all entities associated with these labels.

Typically entities are explicitly linked to labels with direct properties (e.g., `rdfs:label` [Hayes, 2004]). However, more links can be retrieved if we also consider titles and disambiguation pages. In our approach, we add also these labels to the index because we observed that this leads to a substantial increase of the recall. At this stage, it is important to have a high recall because the subsequent operations cannot recover in case we fail to retrieve the correct mapping. In the definitions below, we denote these sets of labels for each entity as $V(e, \texttt{label})$.

Step 2: Computation of the Priors

In this step, we compute a score of the row-entity assignments by comparing all cell values in the row with all the labels of entities that are connected to the candidate entities. To this end, we first define attribute links, and related labels of an entity e as

$$\mathsf{Links}(e) = \{\langle p, v\rangle \mid \langle e, p, v\rangle \in \mathcal{F}\} \tag{3.4}$$

$$\mathsf{LinkLabels}(e, p) = \{l \mid \langle p, v\rangle \in \mathsf{Links}(e), l \in V(v, \texttt{label})\} \tag{3.5}$$

Intuitively, $\mathsf{Links}(e)$ contains all links of e while $\mathsf{LinkLabels}(e, p)$ represents the labels at the other end of the p-links from e. Then, we introduce the function

$$\mathsf{Match}(c, \rho, e, r) = \max_{s \in \mathsf{Cell}(c,\rho)} \max_{l \in \mathsf{LinkLabels}(e,p)} \mathsf{TokenJaccard}(s, l) \tag{3.6}$$

to compute the highest attainable string similarity between the cell at column c and row ρ and the values of the p-links from e. Here, $\mathsf{Cell}(i, j)$ returns the content of the cell at row i and column j in a table with n rows and m columns, while $\mathsf{TokenJaccard}$ is the Jaccard index $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ of the tokens in each string. For instance, in

the table in Figure 3.3 each cell is matched to each attribute of the corresponding row-entity candidates, e.g., $\mathsf{Match}(3, 4, \texttt{The\_Producers\_(1968\_film)}, \texttt{director})$ is the score that quantifies to what extent the content of the cell at coordinates $(3, 4)$ matches the string "Mel Brooks", which is the label of the director of the film. Note that we treat the content of every cell as a string. There are some approaches that use type-specific cell and column matching methods [Ritze et al., 2015, Pham et al., 2016, Zhang, 2017, Limaye et al., 2010], but a combination of our method with these techniques should be seen as future work. Our motivation for this is two-fold: 1) our method is focused on slot-filling, for which numeric properties are not a part, and 2) the datasets in our study are not sufficient for evaluating numeric property matching, which is a domain in itself with many fine-grained distinctions (e.g. units, precision, interpolation) [Ibrahim et al., 2016, Neumaier et al., 2016, Nguyen and Takeda, 2018].

We can now compute likelihood scores for mapping cells to entities (Eq. 3.7), and for mapping columns to properties (Eq. 3.8) to aggregate and normalize these scores on the row and column levels respectively:

$$\mathsf{CellScore}(c, \rho, p) = \frac{1}{|\mathsf{Cand}(\rho)|} \sum_{e \in \mathsf{Cand}(\rho)} \mathsf{Match}(c, \rho, e, p) \tag{3.7}$$

$$\mathsf{ColScore}(c, p) = \frac{\sum_{i=0}^{n} \mathsf{CellScore}(c, \rho_i, p)}{\sum_{i=0}^{n} \sum_{p' \in c} \mathsf{CellScore}(c, \rho_i, p')} \tag{3.8}$$

For instance, in Figure 3.3a, $\mathsf{CellScore}(4, 3, \texttt{director})$ returns the likelihood that the cell (4,3) matches the property `director`, while $\mathsf{ColScore}(3, \texttt{director})$ returns the aggregated scores for column 3 considering all rows in the table.

Since $\mathsf{ColScore}(c, p)$ is the likelihood score that column c maps to property p, we can use this value to construct the prior distribution of all assignments to c. Furthermore, we can use these scores to refine the likelihood of the possible row-entity matchings. We compute such likelihood as

$$\mathsf{RowScore}(\rho, e) = \frac{1}{m} \sum_{i=0}^{m} \max_{p \in \mathcal{P}} \mathsf{ColScore}(c_i, p) \times \mathsf{Match}(c_i, \rho, e, p) \tag{3.9}$$

In essence, Eq. 3.9 computes the likelihood of an entity-row matching as the average best product that each cell matches to a certain attribute (p, e) ($\mathsf{Match}(\cdot)$) with the general likelihood that the column matches to p ($\mathsf{ColScore}(\cdot)$). We use the values of $\mathsf{RowScore}$ to build a prior distribution for all entity-row matches.

Step 3: Entity Similarity Scores

Both prior distributions computed with Eqs. 3.7 and 3.8 rely on the Jaccard Index. Thus, they are distributions which are ultimately built on the string similarities between the strings in the cells and the entities' labels. We use these scores to compute similarity scores between pairs of candidate entities across the rows. In the next step, we will use these similarities to compute better entity-row likelihood scores than the ones of Eq. 3.9.

First, we weigh all links $\langle p, v\rangle$ depending on their popularities across the entities in the table and the corresponding prior of the assignments that use them. To this end, we define the function $\mathsf{LinkTotal}$ as

$$\mathsf{LinkTotal}(p, v) = \sum_{i=0}^{n} \max_{e \in \mathsf{Cand}(\rho_i)} \mathsf{RowScore}(\rho_i, e)[\langle p, v\rangle \in \mathsf{Links}(e)] \tag{3.10}$$

where $[x]$ returns 1 if x is true or 0 otherwise. Note that since $\mathsf{RowScore}$ returns a value between 0 and 1, $\mathsf{LinkTotal}(\cdot)$ returns n in the best case.

Then, we represent the coverage and saliency of $\langle r, v\rangle$ by normalising the value $\mathsf{LinkTotal}(r, v)$ with respect to the table and the KB:

$$\mathsf{Cover}(p, v) = \frac{\mathsf{LinkTotal}(p, v)}{\sum_{i=1}^{n}[\langle p, v\rangle \in \cup_{e \in \mathsf{Cand}(\rho_i)} \mathsf{Links}(e)]} \tag{3.11}$$

$$\mathsf{Salience}(p, v) = \frac{\mathsf{LinkTotal}(p, v)}{|\{e \in \mathcal{E} \mid \langle p, v\rangle \in \mathsf{Links}(e)\}|} \tag{3.12}$$

Intuitively, $\mathsf{Cover}(\cdot)$ computes the popularity of $\langle p, v\rangle$ among the rows of the table, while $\mathsf{Salience}(\cdot)$ considers all entities in $\mathcal{K}$. We combine them as

$$\mathsf{LinkScore}(p, v) = \mathsf{Cover}(p, v) \times \mathsf{Salience}(p, v) \tag{3.13}$$

so that we can rank the attributes depending both on their coverage within the table and popularity in the KB. This combination allows us to give low ranks to attributes, like ⟨`isA,Resource`⟩, which should not be considered despite their high coverage since they are not informative. In contrast, it can boost up the score of attributes with a medium coverage in case they have a high saliency.

Finally, we use the scores from Eq. 3.13 to compute a similarity score between pairs of entities. We compute the similarity between entities e_1 and e_2 as

$$\mathsf{EntitySimilarity}(e_1, e_2) = \sum_{\langle p,v\rangle \in \mathsf{Links}(e_1) \cap \mathsf{Links}(e_2)} \mathsf{LinkScore}(r, v) \tag{3.14}$$

Step 4: Disambiguation

Now, we compute which are the row-entity assignments which maximise the coherence in the table, i.e., maximise the similarity between the entities. These assignments are determined using Loopy Belief Propagation (LBP) [Pearl, 1989].

We model each row-entity prediction as a categorical random variable, for which the label score $\mathsf{RowScore}(\rho, e)$ is the prior distribution (Figure 3.3d1). For convenience, we can view these scores as a sparse matrix $\boldsymbol{L}$ of size $n \times |\mathcal{E}|$. The variables are connected to each other with the edge potentials being defined by entity-entity similarities $\mathsf{EntitySimilarity}(e_1, e_2)$ (Figure 3.3d2; equivalently represented by a matrix $\boldsymbol{S}$), which forms a complete graph. Since this graph has loops it is not possible to perform exact inference. Therefore we approximate it by executing LBP. Additionally, all our edge potentials are identical. This causes all nodes to receive identical information from each other. Instead of having separate messages for each node, we thus have a single vector-valued message that provides the belief updates for our nodes:

$$q_e = \prod_{\rho=0}^{n} \sum_{e' \in \mathsf{Cand}(\rho)} L_{\rho,e'} \times S_{e,e'} = \prod_{\rho=0}^{n} (\boldsymbol{LS})_{\rho,e} \tag{3.15}$$

$$C_{\rho,e} = L_{\rho,e} \times q_e \tag{3.16}$$

where q_e indicates how similar entity e is to all weighted candidates of all rows, and

$C_{\rho,e}$ is the coherence score of entity e for row ρ (Figs. 3.3e and 3.3f respectively). Because the main operation consists of a single matrix multiplication, computation is fast and can be parallelized by standard matrix processing libraries.

LBP can be run for multiple iterations (in our case, replacing $L_{\rho,e'}$ by $C_{\rho,e'}$), but is not guaranteed to converge [Pearl, 1989]. In fact, we observed that sometimes an excessive number of iterations led to suboptimal assignments. This occurred when the entity similarity scores (Eq. 3.14) were not accurate due to missing attributes in the KB and ended up "overriding" the more accurate priors that were computed considering only label similarities (Eq. 3.9) when they are combined in the following step. From our experimental analysis, we observed that in the overwhelming majority of the cases a single iteration of LBP was enough to converge. Therefore, we apply Eq. 3.16 only once without further iterations.

As we can see from Eq. 3.16, the selection of the entity for row ρ relies on two components, $\boldsymbol{L}$ and $\boldsymbol{q}$: The first takes into account to what extent the entity label matches the label of candidate entities and to what extent the labels of the attributes matches with the remaining cell values. The second considers the coherence, i.e., the mappings that maximise the similarity between the entities.

Finally, we disambiguate rows by choosing the highest-rated candidate $\hat{e}_\rho = \text{argmax}_e\ C_{\rho,e}$. Then, we re-calculate $\mathsf{ColScore}(c, r)$ with the updated set of candidates containing only the predicted entity $\mathsf{Cand}(\rho) = \{\hat{e}_\rho\}$ and disambiguate columns by choosing the highest scoring property $\hat{r}_c = \text{argmax}_p\ \mathsf{ColScore}(c, r)$. After this last step is computed, our procedure has selected one entity per row and one property per attribute column. In the next section, we discuss how we can extract triples from the table.

3.3.4 Slot-Filling

After the table is interpreted, we can extract partial triples of the form $\langle s, p, ? \rangle$ where s are the entities mapped to rows and p are the properties associated to columns. If the cell contains numbers or other datatypes (e.g., dates) that we can add the cell value to the KB as-is, but this is inappropriate if the content of the cell refers to an

entity. In this case, we need to map the content of the cell to an entity in the KB.

The disambiguation of the content of a cell could be done by querying our label index precisely the same way as done in Sec. 3.3.3. However, this extraction is suboptimal since now we have available some context, i.e., $\langle s, p, ? \rangle$ that we can leverage to refine our search space. To this end, we can exploit techniques for predicting the likelihood of triples given the KB's structure, namely KB embeddings provided by the TransE algorithm [Bordes et al., 2013]. Given in input e_i, i.e., the entity associated to row i and p_j, i.e., the property associated to column j, our goal is to extract a fact of the form $\langle e_i, p_j, x \rangle$ where entity x is unknown. We proceed as follows:

1. We query the label index with the content of $\mathsf{Cell}(i, j)$ as done for the computation of $Cand(\cdot)$ in Sec. 3.3.3. This computation returns a list of entity candidates $\langle e_1, \ldots, e_n \rangle$ ranked based on label string similarities.

2. For each candidate $e_k \in \langle e_1, \ldots, e_n \rangle$, we compute $\mathsf{Rank}(k) = d(\boldsymbol{e_i} + \boldsymbol{r_j}, \boldsymbol{e_k})$ where d is the distance measure used to compute the TransE embeddings (we use the L_1 norm), and $\boldsymbol{e_i}, \boldsymbol{r_j}, \boldsymbol{e_k}$ are the TransE vectors of e_k, p_j, e_i respectively.

3. We return $\langle e_i, r_j, e_k \rangle$ where e_k is the entity with the lowest $\mathsf{Rank}(k)$, i.e, has the closest distance hence it is the triple with the highest likelihood score.

3.4 Evaluation

Our implementation uses two additional systems: Trident [Urbani and Jacobs, 2020], an in-house triple store to query the KB; and Elasticsearch[1], a well-known search system based on Lucene [Hatcher and Gospodnetic, 2004] that we used for building and querying the label index. Moreover, we reimplemented the TransE algorithm for creating the KB embeddings. Since our KBs contain millions of nodes and edges, we

[1] `https://elastic.co/`

parallelized the learning using Hogwild! [Recht et al., 2011]. We empirically verified that this form of parallelism does not affect the quality of the embeddings.

Baselines Since our goal is to extract novel facts from tables, we considered existing systems that perform slot-filling as baselines. In particular, we considered the systems T2KMatch [Ritze et al., 2015] and TableMiner+ [Zhang, 2017] because of their state-of-the-art results. There are other systems that implement only parts of the pipeline or under certain circumstances, see Chapter 2, section 2.3 for an overview. Further note that these approaches primarily focus on table interpretation. In contrast, we provide an end-to-end system which considers also the operation of slot-filling.

The two systems that we selected for evaluation were designed to work with different KBs, thus no comparison between them was even made. Moreover, the systems were evaluated against a set of manual annotations, and scored on the individual subtasks in terms of precision and recall. Such evaluation did not consider the facts that the system has extracted, but only the classification accuracy on the entity linking, type prediction, and property prediction tasks.

T2KMatch was designed to work with a specific subselection of DBpedia [Auer et al., 2007] while TableMiner+ was originally built to use the Freebase API. We have performed some slight modifications to their source code so that we could perform a fair comparison. For T2KMatch, we modified the system to be able to use an augmented set of candidates so that in some experiments we could measure precisely the performance of table interpretation. For TableMiner+, we modified the system so that we could use different KBs without online API access.

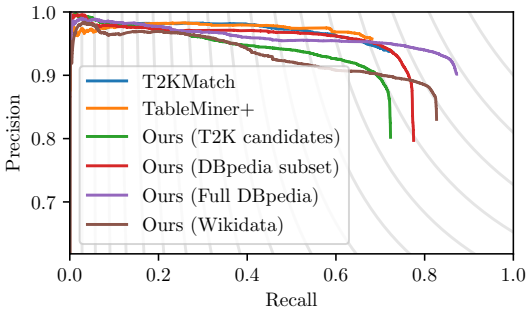
Knowledge Bases Our method can work with any arbitrary KB. We consider DBpedia (so that we could compare against T2KMatch) which is a popular KBs created from Wikipedia and other sources. We use two versions of DBpedia: The first is the triple-based version of the tabular subset used by T2KMatch. This is a subset of DBpedia from 2014 and we consider it so that we can perform an exact comparison. It contains 3.4M entities and 28M facts. Additionally, we also use the latest version

of the full KB (version 2016-10). The full DBpedia contains 15M entities (including entities without labels and redirected entities) and 110M facts. Finally, we compare our performance using Wikidata ("truthy" RDF export, acquired on Oct 2018), which has 106M entities and 1B facts. For evaluation, we map the gold standard to Wikidata using `owl:sameAs` links from DBpedia.

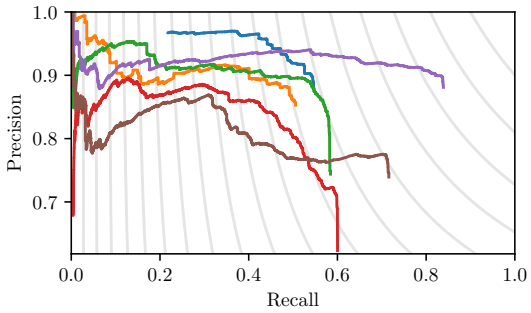
Testsets To the best of our knowledge, there are two openly available datasets of tables that have been annotated for the purpose of table interpretation. The first one is the T2D dataset [Ritze et al., 2015], which contains a subset of the WDC Web Tables Corpus – a set of tables extracted from the CommonCrawl web scrape[2]. We use the latest available version of this dataset (v2, released 2017/02). In our experiments, we disregarded tables without any annotation. The resulting dataset contains 238 entity tables with 659 column annotations and 26106 row-entity annotations. Throughout, we refer to this dataset as T2D-v2.

The second dataset is Webaroo, proposed by [Limaye et al., 2010]. Tables in this dataset were annotated with entities and properties in YAGO. While these tables are a less varied sample of the ones in the T2D, they allow us to study the behaviour of the systems on a dataset with different annotations. This dataset contains 429 entity tables with 389 and 4447 column and row-entity annotations respectively. In order to test the performance of T2KMatch with this dataset, we "ported" the YAGO annotations to DBpedia using the Wikipedia links they refer to. Finally, we tested the scalability of our system by running it on a large set of Wikipedia tables [Bhagavatula et al., 2015].

In the remaining section, we first present experiments that are targeted to evaluate the procedure for table interpretation. Then, we focus on the evaluation of the performance and novelty of extractions. Finally, we report our experiments using our method for processing a large number of tables for performing slot-filling.



(a) Performance tradeoff, T2D-v2



(b) Performance tradeoff, Webaroo

System	Pr.	Re.	F_1
T2KMatch	.94	.73	.82
TableMiner+	**.96**	.68	.80
Ours (T2K candidates)	.88	.72	.79
Ours (DBpedia subset)	.90	.76	.83
Ours (Full DBpedia)	.92	**.86**	**.89**
Ours (Wikidata)	.87	.82	.84

(c) Row-entity evaluation, T2D-v2

System	Pr.	Re.	F_1
T2KMatch	.88	.55	.67
TableMiner+	.85	.51	.63
Ours (T2K candidates)	.74	.58	.65
Ours (DBpedia subset)	.72	.59	.65
Ours (Full DBpedia)	**.88**	**.84**	**.86**
Ours (Wikidata)	.77	.71	.74

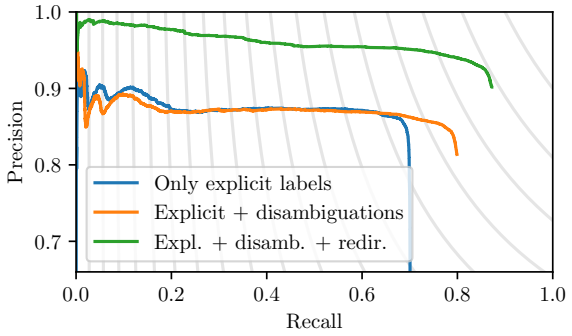
(d) Row-entity evaluation, Webaroo

Figure 3.4: Row-entity evaluation scores and precision-recall tradeoff for the T2D-v2 and Webaroo datasets (the isolines of constant F_1 score are shown in grey). Precision, recall, and F_1 are calculated at the threshold of maximum F_1.

3.4.1 Table Interpretation

We evaluate the performance of determining the correct row-entity assignments, which are the key output for table interpretation. Figure 3.4b,d and Figure 3.4a,c report a comparison of the performance of our method against the baselines. We measure the precision/recall tradeoff (obtained by altering the threshold value for accepting mappings), and precision, recall, and F_1 (shown at the threshold of maximum F_1) on all predictions. The precision decreases whenever a system makes a wrong prediction while the recall is affected when no entity is selected. Not predicting a match for a row can have several causes: the candidate set for that row might have been empty, the annotated entity might not have been in the KB (this occurs when we use a subset), or when all candidates have been pruned away during the interpretation (this occurs with the baselines).

[2] `http://webdatacommons.org/webtables/`



System	Pr.	Re.	F_1
Only explicit labels	.85	.69	.76
Explicit + disambiguations	.84	.79	.81
Expl. + disamb. + redir.	**.92**	**.86**	**.89**

Figure 3.5: Row-entity evaluation scores and precision-recall tradeoff of our approach given different label sources, on T2D-v2.

In these experiments, we configured our system with three different settings: First, we use the same KB and the candidates (i.e., the output of $\mathsf{Cand}(\cdot)$) used by the other two systems. We refer to this setting as "T2K candidates". Then, we use the KB subset used by T2KMatch in our own label index and disambiguation ("DBpedia subset"). Finally, we use our own candidate set generation and full KB ("Full DBpedia"). By evaluating the performance of our method with these settings, we can compare the performance of our approach given the limitations of the inputs that the other systems face.

From the results reported in the figures, we can make a few considerations. First, our method returns a comparable recall but an inferior precision than the baselines if we use the set of candidates from T2KMatch, but is able to match its performance in terms of F_1 when using the same KB. However, the baselines are limited with respect to KBs. In fact, T2KMatch requires that DBpedia is translated into a tabular format while our method does not have this restriction. If our method is configured to use the full DBpedia KB, then it returns the highest recall with only a small loss in terms of precision. This translates in a significantly higher F_1 score than the best of the baselines. These are positive results since a high recall is important for extracting novel facts.

While the precision of our system is low in the limited-input setting, many of the errors that it makes are due to problems with the candidate set and the KB. Therefore, we evaluated a scenario in which we artificially expanded the candidate set to always

include the gold standard. This means we are artificially making use of a "perfect" candidate index. Even with this addition, T2KMatch is unable to use these candidates for prediction and returns the same results. In contrast, manually adding them to our system leads to both a notably higher recall and precision.

This indicates that our method is sensitive to the candidate generation, i.e., to the very first selection of candidates using the index label. To evaluate how our system behaves with richer label indices, we evaluated our method on T2D-v2 with three different label indices. The first index only uses the explicit labels of the entities. The second one includes also the labels that we obtain from redirect pages in Wikipedia. The third one adds also the labels we obtain from the disambiguation pages. The results of this experiment are reported in Figure 3.5. As we can see from these results, including more labels per entity significantly improves both the precision and recall of our system.

Additionally, in the settings where we use the entire DBpedia KB, we are able to predict more entities that were not in the subset that T2KMatch was restricted to. In the case where we use Wikidata, our performance is lower due to missing entity correspondences between the KB and the gold standard.

Webaroo dataset

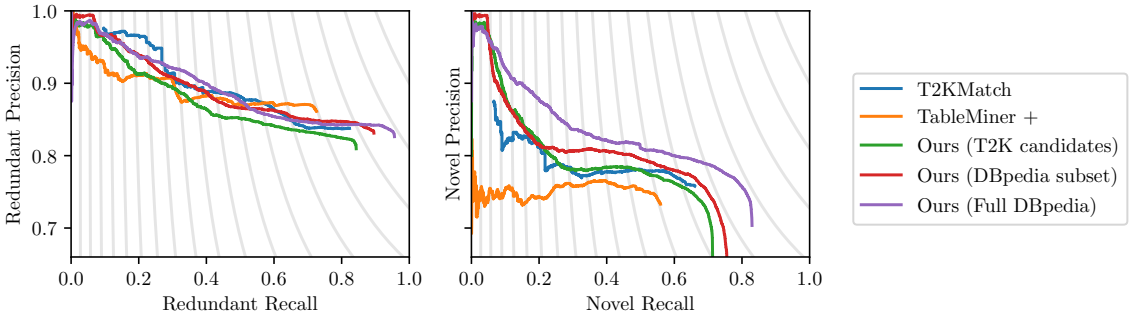
To evaluate the systems in another context, we also performed an experiment on the Webaroo dataset. From Table 3.4d and Figure 3.4b, we can see that our approach again achieves higher recall at the expense of some precision. Interestingly, it is when using the KB on which the dataset was annotated that our system achieves highest performance.

3.4.2 Measuring Redundancy

Current systems (e.g., [Ritze et al., 2015, Zhang, 2017]) were evaluated against a set of manual annotations, and scored on the individual subtasks of table interpretation. Such evaluation did not consider the novelty of facts that the system has extracted.

System	Redundant			Novel		
	Pr.	Re.	F_1	Pr.	Re.	F_1
T2KMatch	.84	.82	.83	**.76**	.66	.71
TableMiner+	**.86**	.73	.79	.73	.56	.63
Ours (T2K candidates)	.81	.84	.83	.61	.71	.66
Ours (DBpedia subset)	.83	.90	.86	.59	.76	.66
Ours (Full DBpedia)	.83	**.96**	**.89**	.70	**.83**	**.76**

(a) The scores for extracting novel and redundant triples from T2D-v2, measured at the acceptance threshold of maximum F_1.



(b) The precision-recall tradeoff curve on T2D-v2.

Figure 3.6: The novel and redundant precision-recall tradeoff for the T2D-v2 dataset (in gray, the isolines of constant F_1 score). Unlike the experiments in the previous figures, here the bias towards extracting known (redundant) facts is made explicit and we focus on finding novel KB facts in web tables.

In other words, no difference was made between predictions of already known facts or new knowledge, but this difference is important in our context. In order to fill this gap, we need to distinguish between these cases when measuring performance.

Given in input a KB $\mathcal{K} = (\mathcal{E}, \mathcal{P}, \mathcal{F})$, an extraction technique like ours is expected to yield a new set of predicted facts $\mathcal{F}_P$ over $\mathcal{E}$ and $\mathcal{P}$ from an input source like web tables. If we have gold standard table annotations, we can generate another set of facts $\mathcal{F}_G$ and use them for evaluating how many facts in $\mathcal{F}_P$ are correct. Note that both $\mathcal{F}_P$ and $\mathcal{F}_G$ might contain facts that are either in $\mathcal{F}$ or not. So far, current techniques have been evaluated w.r.t. the set of true positives $\mathcal{F}_G \cap \mathcal{F}_P$ (correctly extracted facts) and false negatives as $\mathcal{F}_G \setminus \mathcal{F}_P$ (valid facts that were missed). These measures do not take the redundancy of the extracted facts into account, while the redundant information exceeds the novel information for benchmark datasets (see Section 3.2).

Ranking	Dataset	Prec@1	Prec@3
Only Label Index (TF-IDF score)	Wikitable	0.37	0.42
	T2D-v2	0.24	0.31
Labels + Embeddings (TransE)	Wikitable	**0.61**	**0.72**
	T2D-v2	**0.62**	**0.74**

Table 3.3: Precision of slot-filling with/out KB embeddings, calculated on redundant extractions.

In Figure 3.6, we show the evaluation of the correctness of *novel* and *redundant* facts separately. Crucially, our system significantly outperforms the baselines with respect to the recall of novel facts, which is paramount to KB completion. In the tradeoff curve for novel triples (Figure 3.6b), we also outperform the state-of-the-art regarding precision for most threshold values.

3.4.3 Slot-filling

To test the scalability of our system, we have run it on all 1.6M tables in the Wikitable dataset. The first step concerns detecting entity tables with key columns that contain entity labels. This process returned 786K tables. Then, we proceeded with the retrieval of entity candidates. About 288K tables did not contain any entity in DBpedia, thus were discarded. The table interpretation process was launched on the remaining 498K tables. Our approach is trivially parallelizable, and runs in 391 ms per table on average.

From these tables, we extracted 2.818.205 unique facts for 1.880.808 unique slots of the form $\langle s, p, ? \rangle$. Of those slots, 823.806 already contain at least one entity o in the KB. However, we do not know whether our extractions are redundant, or t represents a new extraction that should be added to the existing ones in the KB. To determine the novelty, we queried the label index for every extracted fact and discovered that in 307.729 cases the labels were matching. We can assume these extracted facts to be redundant. From these numbers, we conclude that our extraction process has produced about 1.6M extractions for which we have no evidence of redundancy and thus can be considered as novel. A manual analysis over a sample confirmed this

conclusion.

Finally, we evaluated the effectiveness of our procedure for re-ranking the extractions using the KB embeddings on the Wikitable and T2D-v2 datasets. To this end, we compare the naïve index-based ranking obtained by simply picking the top result returned from the label index against our strategy or re-ranking considering the distance of the corresponding embeddings (Sec. 3.3.4). We chose to measure the precision for the first or top-3 ranked candidates since this is a standard metric used to evaluate the performance of link prediction [Nickel et al., 2016].

Since we need to know the correct entity, we restricted this analysis to the redundant extractions (i.e., the ones already in the KB) and disregarded the novel ones. Table 3.3 reports the results both when we consider only the best result and the top three. We see that that our embedding-based ranking outperforms the index-based ranking in both cases, and predicts the correct entity at the top of the ranking in 61% of the time, compared to 37% for the Wikitable dataset. Moreover, the relatively low results obtained with the index-based ranking strategy indicate that labels are in general not reliable for disambiguating attributes.

3.5 Conclusion and Discussion

In this chapter, we investigated the problem of extending KBs using the data found in Web tables. Existing approaches have focused on overall precision and recall of facts extracted from web tables, but it is important for the purpose of KB completion that the extraction process returns as many (correct) *novel* facts as possible.

We developed and evaluated a new table interpretation method to counter this problem. Our method uses a flexible similarity criterion for the disambiguation of entity-row matches, and employs a PGM to compute new likelihood scores depending on how the various candidates are similar to each other to maximize the coherence of assignments. Because it combines the syntactic match between the tables and the KB with the coherence of the entity predictions, it can confidently predict more candidates for which the attributes in the table are not yet in the KB. Consequently, it extracts

more novel facts for KB completion. For the task of slot-filling, we introduced a novel approach for attribute disambiguation based on KB embeddings, which outperforms a naive label-based approach.

We compared our method to two state-of-the art systems, and performed an extensive comparative evaluation on multiple knowledge bases. Our evaluation shows that our system achieves a higher recall during the interpretation process, which is necessary to extract novel information. Furthermore, it is able to extract more (correct) facts that are not yet in the KB.

Interesting directions for future work include the development of extensions for tables where the entity is identified by multiple columns or where rows do not necessarily describe entities. In particular, the heuristics for determining the key column of the table (and whether such a column is present) would need to be replaced by a model that reliably detects the type of table. Moreover, the inclusion of external sources can be useful to extract more novel information from the table. Finally, despite the remarkable work by different research teams to produce good benchmark datasets, there is still the need for larger and more diverse benchmarks to further challenge the state-of-the-art.

We conclude with a note about the possibility of using the embeddings during the table interpretation instead of using them as as a "post-processing" tool during slot-filling. In fact, their likelihood scores can be used to boost some candidates rather than others. The problem is that we discovered that the average scores per properties varies considerably due to differences and coverage of the KB (e.g., the scores for triples with the property `director` are significantly different than the scores with the property `producer`). These differences hint to the fact that a normalization step would be needed to avoid that the system is biased towards a subset of properties, which is prohibitively expensive for KBs of this size. Our usage of the embeddings during slot-filling does not suffer of this problem since both the head entity (e_i) and property (r_j) are already fixed.

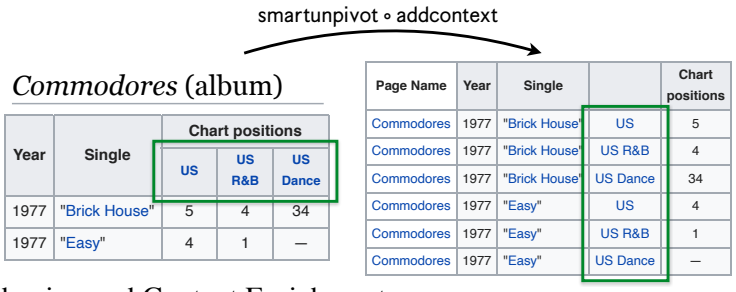
CHAPTER 4

Extracting N-ary Facts from Table Clusters

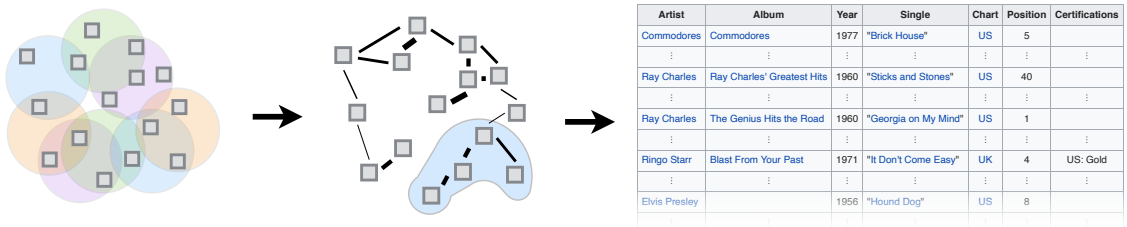
In this chapter, we describe our research on real-world tables that express n-ary facts and how to integrate them with KBs, by focusing on tables from Wikipedia.

Tables in Wikipedia articles contain a wealth of knowledge that would be useful for many applications if it were structured in a more coherent, queryable form. An important problem is that many of such tables contain the same type of knowledge, but have different layouts and/or schemata. Moreover, some tables refer to entities that we can link to Knowledge Bases (KBs), while others do not. Finally, some tables express entity-attribute relations, while others contain more complex n-ary relations.

We propose a novel knowledge extraction technique that tackles these problems. Our method first transforms and clusters similar tables into fewer unified ones to overcome the problem of table diversity. Then, the unified tables are linked to the KB so that knowledge about popular entities propagates to the unpopular ones. Finally, our method applies a technique that relies on functional dependencies to judiciously interpret the table and extract n-ary facts. Our experiments over 1.5M Wikipedia tables show that our clustering can group many semantically similar tables. This leads to the extraction of many novel n-ary facts.



Phase 1: Table Reshaping and Context Enrichment

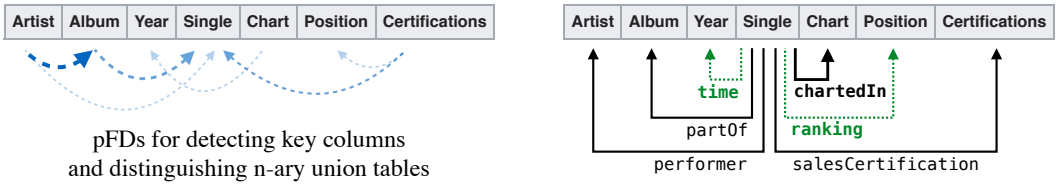


Blocking: table sets of approximate neighbourhoods

Matching: weighted graph of aggregated similarity scores

Union Table

Phase 2: Clustering



pFDs for detecting key columns and distinguishing n-ary union tables

Disambiguating column pairs to KB relations to extract binary and n-ary facts

Phase 3: Table Union Integration

Figure 4.1: Schematic overview of our pipeline. In Phase 1, we process the set of all Wikipedia tables to clean up editorial structures (Section 4.3.1). In Phase 2, we cluster them to form larger union tables (Section 4.3.2). In Phase 3, we integrate them with Wikidata and extract binary and n-ary facts (Section 4.3.3)

4.1 Introduction

Tables in Wikipedia articles express many interesting relations that can improve tasks like web search [Yakout and Ganjam, 2012], or entity linking [Wang et al., 2012]. Some of the most popular Knowledge Bases were constructed from Wikipedia, in particular considering the content of infoboxes. Non-infobox tables on Wikipedia, however, are often used to state knowledge that is complementary to the knowledge contained in infoboxes. This makes tables an excellent source of additional knowledge to extend the coverage of current Wikipedia-based KBs, like Wikidata [Vrandečić and Krötzsch, 2014] or DBPedia [Auer et al., 2007].

Wikipedia tables are created to present structured knowledge for human readers, but not necessarily for machines. To process this knowledge automatically, the table must first be transformed into a coherent structure that is machine-readable. One way to reach this goal is to integrate the content of the tables into a Knowledge Base (KB), but this is challenging due to the diversity of layouts, schemas, and symbols that human contributors use to display knowledge in tabular format. Conceptual relations can be expressed by many different schemas, their meaning might depend on background knowledge that is expressed in the table context, columns may contain multiple attributes in list form, and many cell values are homonyms or synonyms. Secondly, many tables express knowledge about long-tail entities that are not present in KBs. Moreover, many tables on Wikipedia express n-ary relations which are challenging to interpret. In these cases, there is not a single key column that contains the entities of which the other columns express attributes, which is something that is assumed to exist by state-of-the-art table extraction systems [Ritze et al., 2015, Zhang, 2017].

While there are existing works that addresses these problems in isolation (we cover these in Chapter 2), in this chapter we propose a novel method that solves them conjointly. Our method consists of a sequence of *three* main operations. First, it efficiently combines a diverse set of table corpus statistics to perform holistic schema normalization on tables that have different layouts. Then, the tables are clustered together in fewer larger tables. Finally, our method extracts both entity-attribute and n-ary facts from the clustered tables so that new facts can be added to the KB.

The novelty of our approach hinges on three components:

First, we present several techniques to normalize tables which would otherwise be ignored by current knowledge extraction approaches. These techniques rely on some heuristics which transform the tables by removing rows that span all columns, unpivoting column headers, and adding extra contextual columns.

Second, we introduce features for clustering together many similar tables into a unified collection so that knowledge about entities in one table can propagate to other tables. These features measure the potential alignment of columns using Jaccard

similarities, cell embeddings, and semantic types. To scale our clustering to 1.5M tables in Wikipedia, we use set- and embedding-based approximate neighbor search to reduce the similarity search space.

Third, we describe a new method, based on probabilistic functional dependencies [Wang et al., 2009], to distinguish entity-attribute tables from the ones that describe n-ary relations and extract knowledge from both types. We show that using functional dependencies instead of simple heuristics (like picking the leftmost column with unique values, as in Chapter 3), as current methods do, returns a better accuracy that is beneficial to improve the downstream fact extraction. Moreover, to the best of our knowledge, ours is the first method that can extract n-ary facts from tables to a KB.

We evaluate our approach on a large sample of Wikipedia tables using Wikidata [Vrandečić and Krötzsch, 2014], one of the most popular KBs, as reference KB. We report on the performance of different components of the pipeline separately, and compare to a set of strong baselines. Additionally, we extended our evaluation to a set of 1.5M tables extracted from Wikipedia, to evaluate the scalability. In this case, our system managed to extract 29.5M facts which comprise 15.8M binary facts and 6.9M more complex n-ary facts. A large percentage (approx. 77%) was novel, i.e., facts not yet in Wikidata.

4.2 Background

We start with a short recap of some well-known notions on KBs and tables, and introduce some notation used throughout the chapter. For a comprehensive background on Knowledge Bases and Table Interpretation, see Chapter 2.

Tables We model our collection of tables as a corpus $\mathcal{T}$ of tables, which we have extracted from the HTML of Wikipedia articles. For a given table T, we denote with $T[i][j]$ the *cell* of table T at the i^{th} row and j^{th} column. Every cell c is associated to a *cell value* $\mathsf{val}(c)$ which represents the content of a cell and is either a string or

`NULL`. In the last case, we say that the cell is empty and denote it with the symbol $\boxed{\emptyset}$. Additionally, a cell may contain some links to entity-related Wikipedia pages. In Wikidata, every Wikipedia page is mapped to an entity. We denote with $\mathcal{E}_W \subseteq \mathcal{E}$ the set of such entities, and write $\mathsf{links}(c) \subseteq \mathcal{E}_W$ to refer to the entities pointed by the links in c. Some cells are marked with a *span* that extends multiple columns. We write $\mathsf{span}(c) = [i, j]$ when cell c spans columns i to j (included) in its row. If c at row k has span $[i, j]$ then $\mathsf{val}(T[k][i]) = \ldots = \mathsf{val}(T[k][j])$. However, the opposite does not hold, namely two adjacent cells can have the same value but without an extended span. Finally, we write $\mathsf{span}(c) \subset \mathsf{span}(d)$ if $\mathsf{span}(c) = [i, j], \mathsf{span}(d) = [k, l], i \geq l, j \leq l$, and $j - i < l - k$.

We denote with $\mathsf{cols}(T)$ and $\mathsf{rows}(T)$ the list of all columns and rows in T respectively. We represent each column as a tuple of $|\mathsf{rows}(T)|$ cells and each row as a tuple of $|\mathsf{cols}(T)|$ cells. We distinguish header rows from body rows based on the table's HTML. We write $\mathsf{head}(T)$ to refer to a list of rows in T marked as headers, while $\mathsf{body}(T)$ refers to the remaining rows. Abusing notation, we write $\mathsf{cols}(\mathsf{body}(T))$ to refer to the list of columns of the table's body. We view $T, \mathsf{cols}(T), \mathsf{rows}(T), \mathsf{body}(T), \mathsf{cols}(\mathsf{body}(T))$, and $\mathsf{head}(T)$ as sets if the order of the tuples does not matter, otherwise we use the suffix $[i]$ to refer to the i^{th} element in the collection (e.g., $\mathsf{cols}(T)[1]$ is the first column of T).

Tuples are denoted with delimiters $\langle\rangle$. We introduce two auxiliary functions to operate on tuples. Function $\mathsf{append}(a, B)$ returns a tuple where element a is appended to tuple B. Function $\cdot$, written $A \cdot B$, returns a tuple where tuple A is concatenated to tuple B. A *union table* is a table created by concatenating the body of multiple tables. In a union table, columns may be aligned into single ones or not. In the second case, empty cells are used to fill the gaps [Ling et al., 2013].

In this chapter, we use the relational model to specify some operations on tables. This model views a table schema as a *database relation* R with *attributes* $A_1, \ldots, A_m$, denoted as $R(A_1, \ldots, A_m)$, and calls a table with such a schema an *instance* of R. Attributes in the relational model are mapped to header cells in the table. Thus, they are different than attributes used in KBs. In the former, attributes (informally) map to the header names of the table while in the latter they are property-value pairs of

entities.

Page Name	No.	Title	Writer(s)	Recording date	Length
Elvis Presley	1.	"Blue Suede Shoes"	Carl Perkins	January 30, 1956	2:00
Elvis Presley	2.	"I'm Counting on You"	Don Robertson	January 11, 1956	2:25
Elvis Presley	3.	"I Got a Woman"	• Ray Charles • Renald Richard	January 10, 1956	2:25
Elvis Presley	4.	"One Sided Love Affair"	Bill Campbell	January 30, 1956	2:11
Elvis Presley	5.	"I Love You Because"	Leon Payne	July 5, 1954	2:43
Elvis Presley	6.	"Just Because"	• Bob Shelton • Joe Shelton • Sydney Robin	September 10, 1954	2:34

(a) Entity-Attribute (EA) table

Page Name	Year	Chart	Positions
"Hound Dog"	1956	US *Billboard*	8
"Hound Dog"	1956	US Singles	15
"Hound Dog"	1956	Australia (Kent Music Report)	17
"Hound Dog"	1956	Belgium (Ultratop 50 Flanders)	13
"Hound Dog"	1956	UK Singles	2
"Hound Dog"	1956	US Cash Box Magazine	1

(b) N-ary (NA) table

Figure 4.2: Examples of tables that express different information structures.

We make a distinction between *Entity-Attribute (EA)* and *N-Ary (NA)* tables. EA tables contain one column with the names of entities, which we call *key column*, and every row expresses attributes of that entity in the other columns [Yakout and Ganjam, 2012]. Therefore, one row can be translated into a set of attributes of an entity, and represented in $\mathcal{K}$ with triples of the form $\langle \texttt{entity}, \texttt{attribute\_relation}, \texttt{attribute\_value} \rangle$.

NA tables lack a key column and each row expresses one n-ary fact and typically $n > 2$. In this case, we say that the table expresses a *n-ary relation*. It has been shown that NA tables make up a significant portion of tables on the Web [Lehmberg and Bizer, 2019]. Table (a) in Figure 1 is an example of a EA table while Table (b) reports a NA table.

To improve the extraction coverage, we consider the content that we can extract from the page that contains the table. For instance, we consider the title of the article or the table caption. We represent contextual information as strings. To distinguish the various types of contextual information, we use pairs of the form $\langle X, Y \rangle$ where X is the type of information and Y is the content. For instance, the pair $\langle$"Page Name","Elvis Presley"$\rangle$ is an example of contextual information for the table in Figure 4.2a. We refer to the set of all contextual tuples associated to table T as $\mathsf{context}(T)$.

Table unpivoting Tables can be categorized either as *wide* or *narrow*, depending on how they express information [Wickham, 2014]. If they are wide (i.e., have a

wide layout), then it is more likely that single attribute values are represented with dedicated columns. For instance, the header cell "US" in the third column of the top-left table in Figure 4.1 expresses the qualifier $\langle \texttt{chartedIn}, \texttt{US\_Billboard\_200} \rangle$ for all the triples extracted from the cells below it. For our purposes, it is more convenient if tables are in a narrow shape, i.e., header cells express attribute properties rather than attribute values (the top-right table in Figure 4.1). Converting a table from a wide shape to a narrow shape is known as *unpivoting* (Figure 4.3).

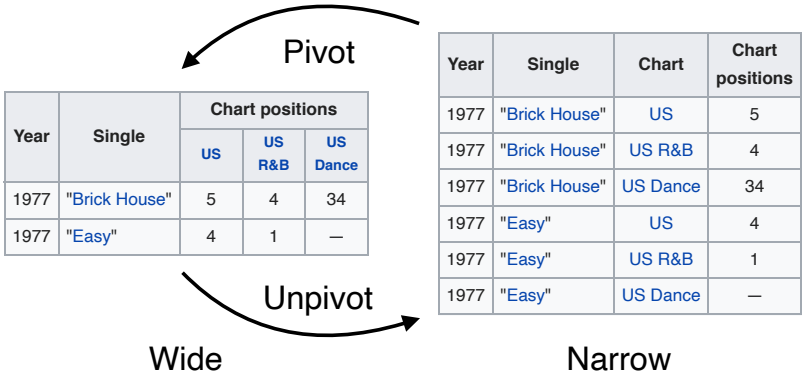


Figure 4.3: Pivoting and unpivoting.

Wyss and Robertson [2005] provide a formal definition of unpivoting, which we outline below for self-containment. This definition uses two additional relational algebra operators: δ (metadata demotion) and Δ (column deference). Given a relation schema $R(A_1, \ldots, A_m)$, let r be an instance of R with n rows and let $\texttt{y}$ be an attribute that is not in R. Then $\delta_{\texttt{y}}(r)$ appends every $A_1, \ldots, A_m$ to each row in r, returning a new instance with $|r| \times n$ rows and schema $R(A_1, \ldots, A_m, \texttt{y})$. The operator Δ is used to further process the relation. Given a relation $R(A_1, \ldots, A_m, B)$, let r be an instance of R and $\texttt{z}$ a column name that is not in R. Then, $\Delta^{\texttt{z}}_B(r)$ searches if an element of B at row i equals to column name at position j, and if this occurs then it copies the cell value at row i and column j in a new column with name $\texttt{z}$.

These two operators, in combination with the standard relational operators projection (Π) and selection (σ), can be used to formally define the operation of unpivoting. Let $R(A_1, \ldots, A_i, B_{i+1}, \ldots, B_m)$ be a relation, r be an instance of R, and $B_{i+1}, \ldots, B_m$

be the attributes to unpivot. Then, unpivoting r can be expressed as:

$$\mathsf{UNPIVOT}^{\mathtt{y}\to\mathtt{z}}_{A_1,\ldots,A_i}(r) \coloneqq \Pi_{A_1,\ldots,A_i,\mathtt{y},\mathtt{z}}(\sigma_{\mathtt{y}=A_1\wedge\ldots\wedge\mathtt{y}\neq A_i}(\Delta^{\mathtt{z}}_{\mathtt{y}}(\delta_{\mathtt{y}}(r))))$$

where y is the name of the column with the unpivoted schema and z is the column name with the values of the unpivoted columns.

Functional Dependencies To distinguish EA tables from NA tables, we make use of probabilistic functional dependencies (pFDs), first introduced by Wang et al. [2009]. Let X, Y be two attributes of a relation R, and r be an instance of R. Then, the pFD $X \to^p Y$ indicates that two tuples in r that share the same value for X also share the same value for Y with probability p. To compute pFDs, we use their perTuple algorithm , which returns pFDs using probabilities computed on r.

4.3 Method

Our goal is to extract clean, unified, and linked n-ary facts from a large set of tables to enrich a KB with new knowledge. For example, we would like to extract from the top-left table in Figure 4.1 the n-ary fact that the song "Brick House" charted in the "US Billboard 200" chart at position 5 in 1977. To represent this fact, we use three triples: The triple $t = \langle \texttt{Brick\_House}, \texttt{chartedIn}, \texttt{US\_Billboard\_200}\rangle$ and the triples $\langle \mathtt{q_t}, \texttt{pointInTime}, \texttt{1977}\rangle$ and $\langle \mathtt{q_t}, \texttt{ranking}, \texttt{5}\rangle$, where $\mathtt{q_t}$ is a fresh entity used to represent the qualifiers mapped to triple t.

We use Wikidata [Vrandečić and Krötzsch, 2014] as target KB because of its popularity and large coverage, and focus on Wikipedia tables since they contain a large amount of interesting factual information related to Wikidata entities. Our method, which is graphically depicted in Figure 4.1, can be viewed as a pipeline of three main operations: *Table Reshaping and Enrichment* (Section 4.3.1), *Clustering* (Section 4.3.2), and *KB Integration* (Section 4.3.3), each discussed below.

4.3.1 Reshaping

In Wikipedia, some tables are generated using well-defined and popular templates while others are built using modified copies of templates taken from related pages. Consequently, tables that express similar content can be very diverse from each other and this hinders a successful factual extraction.

To counter this problem, we apply a procedure to "normalize" the tables. This procedure, which we refer to as $\mathsf{reshape}(\mathcal{T})$, performs three operations on every table $T \in \mathcal{T}$. The first operation merges or removes cells that span all the columns ($\mathsf{mergechunks}(T)$, Section 4.3.1). The second operation unpivots some columns to transform wide tables into narrow ones ($\mathsf{smartunpivot}(T, \mathcal{U})$, Section 4.3.1). Finally, tables are further enriched with extra contextual information ($\mathsf{addcontext}(T)$, Section 4.3.1).

Merging table chunks

Sometimes, Wikipedia contributors decide to add cells that span all the columns for various purposes. For example, such cells are added below the row they belong to keep the table from becoming too wide, or at the bottom as a footnote.

These cells can confuse the interpretation procedure since they can, for instance, be recognized as separate rows with new entities. To avoid these cases, the function mergechunks (which is formally defined in Algorithm 1), identifies these cells and copies their content to other parts of the table. The algorithm applies three heuristics H_1, H_2, H_3 that we observed work well in practice:

- H_1 If cells that span all the columns appear at every even row of the body, i.e., at row index $i = 2, 4, \ldots$, then we assume that the cells contain extra information about the preceding row. Thus, we add an extra column with empty cells, remove the i^{th} row and copy its content in the extra column at row $i - 1$ (Figure 4.4a);
- H_2 If H_1 does not apply, but there are cells that span all columns as last rows in the table, then we assume that they contain a footnote. In this case, we remove the rows and add their content as contextual information of type "footnote" to the table;

Algorithm 1 mergechunks(T)

if $|\mathsf{cols}(T)| = 1$ **then return** T
$B \coloneqq \mathsf{body}(T)$ $H \coloneqq \mathsf{head}(T)$ $n \coloneqq |B|$ $m \coloneqq |\mathsf{cols}(T)|$
$H' \coloneqq \langle\rangle$ $B' \coloneqq \langle\rangle$ $C \coloneqq \langle\rangle$
if $\mathsf{span}(B[2i][1]) = [1, m]$ for each $1 \leq i \leq \frac{n}{2}$ **then**
    for $i \coloneqq 1$ **to** $|H|$ **do** $\mathsf{append}(H[i] \cdot \langle\boxed{\emptyset}\rangle, H')$
    for $i \coloneqq 1$ **to** $\frac{n}{2}$ **do** $\mathsf{append}(B[2i-1] \cdot \langle B[2i][1]\rangle, B')$
else
    $i \coloneqq n$
    while $i \geq 1$ **do**
        if $\mathsf{span}(B[i][1]) = [1, m]$ **then**
            $\mathsf{append}(\langle \text{“}footnote\text{”}, \mathsf{val}(B[i][1])\rangle, C)$
        else break
        $i \coloneqq i - 1$
    if $\exists j$ such that $1 \leq j \leq i$ and $\mathsf{span}(B[j][1]) = [1, m]$ **then**
        $D \coloneqq \boxed{\emptyset}$
        for $i \coloneqq 1$ to $|H|$ **do** $\mathsf{append}(\langle D\rangle \cdot H[i], H')$
        for $j \coloneqq 1$ to i **do**
            if $\mathsf{span}(B[j][1]) = [1, m]$ **then** $D \coloneqq B[j][1]$
            else $\mathsf{append}(\langle D\rangle \cdot B[j], B')$
    else
        $H' \coloneqq H$ $B' \coloneqq \langle B[1], \ldots, B[i]\rangle$
Let T' be a table s.t. $\mathsf{head}(T') = H', \mathsf{body}(T') = B'$, and $\mathsf{context}(T') = \mathsf{context}(T) \cup C$
return T'

- H_3 If H_1 does not apply, but there are multiple cells that span all columns that appear in the body, then we treat them as extra information about the rows below them. To this end, we add an extra column with empty cells, remove every row with index i with a cell that spans all columns and copy its content in the extra column at row $i + 1, \ldots, j$ where j is either the end of the table or the row index of the following cell that spans columns (Figure 4.4b).

After removing all-column spanning cells, we analyze the table headers. Wikipedia tables typically have one or more header rows at the top of the table, but can also have sub-headers at other positions. Whenever a header row is encountered after a body row, we split the table horizontally into distinct new tables. In other cases, table schemas are repeated side-by-side. When we detect such a repetition, we split the table vertically and concatenate the left rows to the right rows of the table.

No. overall		Title	Original airdate
17	1	"A Slight Case of Reincarnation"	31 December 1966
Adam is determined to free an African leader from the spell of the Face. *			
18	2	"Black Echo"	7 January 1967
Adam visits an exiled Russian Grand Duchess, who is requesting him because he is familiar with a pearl necklace that she once owned, and is now trying to reclaim it. However, upon arrival at the Duchess's home, Adam finds two horrifying secrets from his past are about to return...			
19	3	"Conspiracy of Death"	14 January 1967
Adam investigates the murder of an old wartime friend.			

(a) Table processed by H_1

Catholic, Orthodox, Protestant, and most Oriental Orthodox	Original language (Koine Greek)
Canonical Gospels	
Matthew	Greek (majority view: see note)[N 2][34][35][36]
Mark	Greek
Luke	Greek
John	Greek
Apostolic History	
Acts	Greek

(b) Table processed by H_2

Figure 4.4: Examples of tables processed by mergechunks heuristics.

Table Unpivoting

Algorithm 2 smartunpivot$(T, \mathcal{U})$

$H \coloneqq \mathsf{head}(T) \quad m \coloneqq |\mathsf{cols}(T)| \quad b \coloneqq c \coloneqq d \coloneqq 0$
for $i \coloneqq 1$ **to** $|H|$ **do**
  for all $U \in \mathcal{U}$ **do**
    Let u_l be the returned value of $U(T, H[i][l])$
    Let $[j, k]$ be the largest interval s.t. $u_j = \ldots = u_k =$ **true**
    if $(k - j) > (d - c)$ **then**
      $b \coloneqq i \quad c \coloneqq j \quad d \coloneqq k$
if $b = 0$ **then return** T
$y \coloneqq z \coloneqq \boxed{\emptyset}$
if $b > 1 \wedge \mathsf{span}(H[b-1][c]) = [c, d]$ **then** $z \coloneqq H[b-1][c]$
Let $R(A_1, \ldots, A_{c-1}, B_c, \ldots, B_d, A_{d+1}, \ldots, A_m)$ be a relation with $A_i \coloneqq H[b][i]$ and $B_i \coloneqq H[b][i]$. Also, let r be an instance of R with $\mathsf{body}(T)$
$s = \mathsf{UNPIVOT}^{y \to z}_{A_1,\ldots,A_{c-1},A_{d+1},\ldots,A_m}(r)$
Let T' be a table where:
  - $\mathsf{head}(T') = \langle\langle A_1, \ldots, A_{c-1}, A_{d+1}, \ldots, A_m, y, z\rangle\rangle$
  - $\mathsf{body}(T') = s$
return T'

Wide tables tend to contain column headers that express value strings rather than attribute strings. We would like to transform such tables so that the content of these cells appears in the body instead of the header. For instance, the top-left table in Figure 4.1 contains the columns `US`, `US R&B`, `US Dance` which are the values of attributes with property `chartedIn`. This table should be transformed into the

top-right table in Figure 4.1.

To this end, we must tackle two challenges. First, we need to define a procedure that, given an input table, detects a sequence of horizontally adjacent header cells that encode attribute values. The second challenge consists of extracting the new column header associated with these values so that we can unpivot the table.

We tackle the first challenge with a set of six boolean functions that encode some heuristics, while we rely on the content of previous headers to extract the new column header. Our procedure for unpivoting the table can be viewed as a function smartunpivot which, given in input table T and set of boolean functions $\mathcal{U}$, returns an unpivoted version of T (or T if no unpivoting was possible).

We outline the functioning of smartunpivot on table T below (the pseudocode is in Algorithm 2). Each boolean function $U_1, \ldots, U_6 \in \mathcal{U}$ receives in input a table cell and returns true if the encoded heuristics matches with the cell. First, the procedure scans the headers of T row-by-row and invokes all boolean functions in $\mathcal{U}$ with every cell in the input. An interval of adjacent cells for which a function has returned true maps to a potential set of columns with attribute values. We select the largest interval of such cells for unpivoting the table. Let us assume that this interval occurs at row i and spans columns $[j, k]$. To retrieve the new column header, we consider the cell at header row $i - 1$ (if any) and column j. If this cell has a span $[j, k]$, then we pick its value as column header, otherwise, we set the new column header with an empty cell.

Let $R(A_1, \ldots, A_{i-1}, A_i, \ldots, A_k, A_{k+1}, \ldots, A_m)$ be a relation that represents the schema of T where each attribute $A_1, \ldots, A_m$ maps to the header cell at $\mathsf{head}(T)[i]$. Moreover, let y, z be two fresh attributes that will contain the unpivoted attributes and the content of the unpivoted columns respectively. We map y to $\boxed{\emptyset}$, while z maps either to a cell with the new column header or to $\boxed{\emptyset}$ if no relation was found. In the top-right table of Figure 4.1, y would map to the 4^{th} column while z is the 5^{th} column.

Finally, let r be an instance of R with the body of T. We unpivot T by first executing $r' \coloneqq \mathsf{UNPIVOT}^{y \to z}_{A_1,\ldots,A_{i-1},A_{j+1},\ldots,A_m}(r)$, and then creating a new table T' with $\mathsf{body}(T') \coloneqq r'$ and $\mathsf{head}(T') \coloneqq \langle\langle A_1, \ldots, A_{i-1}, A_{j+1}, \ldots, A_m, y, z\rangle\rangle$.

In the remaining, we describe the heuristics encoded by the boolean functions.

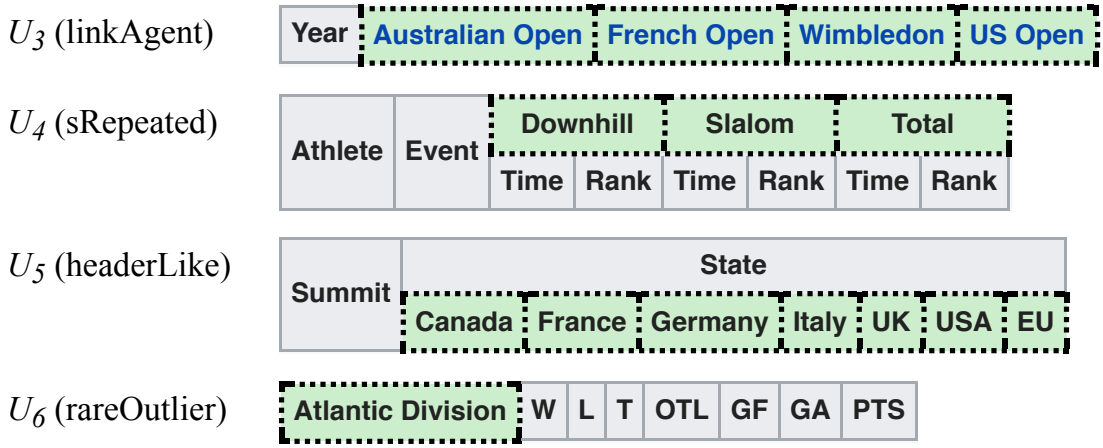


Figure 4.5: Examples of candidate tables headers for unpivoting. Cells in green are returned by the named heuristic.

Figure 4.5 shows examples of Wikipedia table headers for which these functions will apply.

- U_1 *(nPrefix)* Returns true if the cell starts with numeric characters;
- U_2 *(nSuffix)* Returns true if the cell ends with numeric characters;
- U_3 *(linkAgent)* Returns true if the cell contains a hyperlink to the Wikipedia page of an entity with type `Agent` in Wikidata, i.e.,

$$U_3(c) \coloneqq \exists e \in \mathsf{links}(c) \text{ s.t. } \langle e, \texttt{isA}, \texttt{Agent} \rangle \in \mathcal{K} \tag{4.1}$$

  The underlying intuition is that entities of the type `Agent`, which in Wikidata includes people and organizations, are unlikely to refer to properties but refer instead to attribute values.

- U_4 *(sRepeated)* Returns true if the cell spans an interval of columns and there is another row where the cells have equal value in the same interval. More formally, let T and ρ be the table and row respectively where c appears, and let $[i, j] = \mathsf{span}(c)$.

$$U_4(c) \coloneqq \exists s \in \mathsf{head}(T) \text{ s.t. } \rho \neq s \wedge \mathsf{val}(s[i]) = \ldots = \mathsf{val}(s[j]) \tag{4.2}$$

- U_5 *(headerLike)* Returns true if the cell appears in $\mathcal{T}$ more frequently either in the

body or in a header cell that is spanned by another cell. Before we define U_5, let N_t, N_b, N_s be as follows:

$$N_t(c) = |\{T \in \mathcal{T} : c \in T\}| \tag{4.3}$$

$$N_b(c) = |\{T \in \mathcal{T} : c \in \mathsf{body}(T)\}| \tag{4.4}$$

$$N_s(c) = |\{T \in \mathcal{T} : c, d \in \mathsf{head}(T) \wedge \mathsf{span}(c) \subset \mathsf{span}(d)\}| \tag{4.5}$$

Then, $U_5(c) \coloneqq \frac{N_b(c)+N_s(c)}{N_t(c)} > 0.5$.

- U_6 *(rareOutlier)* Returns true if the frequency of the cell in the headers of the tables in $\mathcal{T}$ is more than one standard deviation smaller than the average frequency of the cells of its header. Let T be the table where cell c appears, and $\mu(X)$ and $\sigma(X)$ be the mean and standard deviation of X. Then,

$$N_h(c) = |\{T \in \mathcal{T} : c \in \mathsf{head}(T)\}| \tag{4.6}$$

$$\mu_h(T) = \mu(\{N_h(c) : c \in \mathsf{head}(T)\}) \tag{4.7}$$

$$\sigma_h(T) = \sigma(\{N_h(c) : c \in \mathsf{head}(T)\}) \tag{4.8}$$

and $U_6(c) \coloneqq N_h(c) < \mu_h(T) - \sigma_h(T)$.

Adding Contextual Information

Algorithm 3 addcontext(T)

38: $A \coloneqq \langle\rangle$ $B \coloneqq \langle\rangle$ $H \coloneqq \langle\rangle$ $J \coloneqq \langle\rangle$
39: **for all** $C \in \mathsf{context}(T)$ **do**
40: $\mathsf{append}(\langle C[1]\rangle, A)$ $\mathsf{append}(\langle C[2]\rangle, B)$
41: **for** $i \coloneqq 1$ **to** $|\mathsf{head}(T)|$ **do** $\mathsf{append}(A \cdot \mathsf{head}(T)[i], H)$
42: **for** $i \coloneqq 1$ **to** $|\mathsf{body}(T)|$ **do** $\mathsf{append}(B \cdot \mathsf{body}(T)[i], J)$
43: Let T' be a table where $\mathsf{head}(T') = H$ and $\mathsf{body}(T') = J$
44: **return** T'

Often, the context of a table can help the interpretation procedure to link entities to the KB more precisely. For example, a table may be located in a section whose

header contains a keyword (e.g., "song") or a date that is important for linking some entities. To this end, we would like to add such important contextual information to the table.

For each table $T \in \mathcal{T}$, we add to $\mathsf{context}(T)$ three pairs: one with the page title, one the section title, one with the table caption, provided these are available in the associated Wikipedia page. Moreover, procedure mergechunks can optionally add another set of pairs with footnotes. Procedure $\mathsf{addcontext}(T)$, (pseudocode in Algorithm 3) has the task of adding the pairs in $\mathsf{context}(T)$ to T. For each pair $\langle X, Y \rangle \in \mathsf{context}(T)$, it adds an extra column with header X and cells values equal to Y. The table in Figure 4.2b shows an example of a table modified by this procedure. Here, addcontext has added an extra column with the title of the page (*"Hound Dog"*).

4.3.2 Clustering

Some tables can be easily matched to the KB while others are more problematic, especially if they cover a domain that is not yet covered by the KB. For example, consider the table in Figure 4.2b. This table contains a column titled "Chart" but Wikidata does not contain any fact that involves the property `chartedIn` in combination with the entities mentioned in this table. Because of this incompleteness, interpreting this table on its own is hard. However, if there is another table with a similar schema but that can be matched to Wikidata, we can cluster them together so that we can propagate the knowledge obtained by matching one table to the other. Following this intuition, the next step in our pipeline consists of finding clusters of tables that express the same latent semantic relation. To this end, we construct union tables using a set of approximate indexes followed by similarity functions designed for our specific use-case.

Due to the large size of our corpus (1.5M tables), computing our similarity scores for each pair of tables is computationally too expensive. To counter this problem, we perform a two-level clustering. First, we compute a set of table *blocks*, i.e., groups of tables that appear to be similar according to an approximate k-Nearest Neighbors procedure (k-NN) (Section 4.3.2). Then, we calculate our similarity scores on the

reduced set of table pairs (Section 4.3.2). Finally, we construct a graph $\mathcal{G} = \langle \mathcal{T}, \mathcal{W} \rangle$, which has the tables in $\mathcal{T}$ as vertices, and the similarity scores defined in the sparse matrix $\mathcal{W} \in \mathbb{R}^{\mathcal{T} \times \mathcal{T}}$ as weighted edges. We partition this graph into clusters of tables, from which we construct the final set of union tables (Section 4.3.2).

Blocking

Since computing the similarity score between the 1.5Mx1.5M pairs of tables in $\mathcal{T}$ is computationally expensive, we construct four approximate indexes to retrieve, for a given table, the top-k similar tables using a more coarse-grained definition of similarity. We refer to the collection of similar $k + 1$ tables as a *table block*. In our experiments, we use a fairly high value of k (100) to avoid that good table pairs are ignored.

The four approximate indexes $I_1, \ldots, I_4$ are constructed as follows. Two indexes I_1 and I_2 employ Locality Sensitive Hashing (LSH) with MinHash [Zhu et al., 2016], which provides approximate Jaccard similarity between the cell values in two tables. We employ LSH because we have observed that two tables are more likely to express the same semantic relation if many cell values overlap. Therefore, we construct one approximate LSH index considering the header of the table (I_1), and another considering the body of the table (I_2).

The other two indexes I_3 and I_4 perform approximate k-NN with the embeddings of header rows and body columns. We use embeddings because we noticed that sometimes table pairs might not contain exactly the same values, but words are nevertheless semantically similar. We exploit this observation considering pre-trained GloVe word embeddings [Pennington et al., 2014] for every column and header row in our tables. To create cell embeddings, we simply sum the word embeddings of the words in the cell, and create header row and column embeddings by averaging the cell embeddings. Then, we index the vectors of the headers (I_3) and of the columns (I_4) and query them using approximate k-NN search offered by Facebook's library FAISS [Johnson et al., 2019], one of the most scalable implementations of k-NN with embeddings.

After the indexes are computed, we retrieve the top k similar tables for each table

in $\mathcal{T}$ using each index. This yields a set of $4(k+1) * |\mathcal{T}|$ blocks. We consider the union of all blocks returned by all indexes because at this stage we do not want to further sacrifice recall. For every pair of tables that appear in the same block, we compute a more fine-grained similarity score as described below.

Matching

The goal of our proposed similarity score between two tables is to measure to what extent pairs of columns in the two tables can be aligned. This follows the intuition that two tables express the same semantic relation if many of their columns can be aligned.

The similarity score between two tables is an aggregation of the similarity scores between column pairs that are computed considering either the body or headers of two tables. The similarity scores between column pairs rely on a set $\mathcal{M}$ of functions, which we call *matching functions*. These functions, described below, receive in inputs two sets of cells, and compare the overlap of cell values, and the similarity with word embeddings and semantic types.

When comparing the columns of two tables A and B with m_A and m_B columns respectively, using matching function $f \in \mathcal{M}$, we compare every possible pair of columns between A and B. We create an alignment between columns in A and columns in B using the greedy procedure $\mathsf{greedycolsim}(A, B, f)$ shown in Algorithm 4. This procedure creates $\min(m_A, m_B)$ alignments by selecting the best column matches according to f, and aggregates those similarity scores by averaging over both m_A and m_B.

The procedure greedycolsim returns a table alignment score using one matching function f. We invoke this procedure with every $f \in \mathcal{M}$. The scores are then aggregated in a manner described by procedure aggsim, Algorithm 5. The application of $\mathsf{aggsim}(A, B, \mathcal{M}, \theta)$ on tables A and B is as follows. Generally, the aggregation of semantic matcher scores depends on whether they compute "optimistic" or "pessimistic" similarities and whether there is supervision or heuristics available [Do and Rahm, 2002]. In our case, we take an "optimistic" approach assuming that any of our matching

functions may be the most relevant for a given pair of tables. Therefore, we take the best scored obtained by any matching function (line 53). Note that the application in line 53 considers only the body of the tables. We have observed that often correctly matching table pairs have also aligned headers. Therefore, we invoke the matching functions considering the tables' headers and optimistically max-aggregate them (line 55). Then, the final score is obtained by combining them using their weighted mean (line 56). The aggregation weight θ is found using cross-validated grid-search.

Algorithm 4 $\mathsf{greedycolsim}(A, B, f)$

45: $C \coloneqq \mathsf{body}(A)$ $\quad D \coloneqq \mathsf{body}(B)$
46: $m_A \coloneqq |\mathsf{cols}(A)|$ $\quad m_B \coloneqq |\mathsf{cols}(B)|$
47: $E_A \coloneqq \{1, \ldots, m_A\}$ $\quad E_B \coloneqq \{1, \ldots, m_B\}$ $\quad M \coloneqq \emptyset$
48: **while** $|E_A| > 0$ **and** $|E_B| > 0$ **do**
49: $\quad \langle i, j\rangle \coloneqq \underset{\langle i,j\rangle \in E_A \times E_B}{\arg\max} f(\mathsf{cols}(C)[i], \mathsf{cols}(D)[j])$
50: $\quad E_A \coloneqq E_A \setminus \{i\}$ $\quad E_B \coloneqq E_B \setminus \{j\}$ $\quad M \coloneqq M \cup \{\langle i, j\rangle\}$
51: $S \coloneqq \sum_{\langle i,j\rangle \in M} f(\mathsf{cols}(C)[i], \mathsf{cols}(D)[j])$
52: **return** $\frac{1}{2}(\frac{S}{m_A} + \frac{S}{m_B})$

Algorithm 5 $\mathsf{aggsim}(A, B, \mathcal{M}, \theta)$

53: $s_b = \max_{f \in \mathcal{M}} \mathsf{greedycolsim}(A, B, f)$
54: $h_A = \mathsf{head}(A)$ $\quad h_B = \mathsf{head}(B)$
55: $s_h = \max_{f \in \mathcal{M}} f(h_A, h_B)$
56: **return** $\theta s_h + (1 - \theta) s_b$

Next, we discuss our matching functions $f_j, f_e, f_d \in \mathcal{M}$.

f_j: Set Similarity. The simplest way to view of headers and columns is as a set of discrete cell values. To model whether two sets of discrete values are similar, we use their Jaccard index:

$$f_j(a, b) = \frac{|a \cap b|}{|a \cup b|} \tag{4.9}$$

f_e: Word Embedding Similarity. Following [Nargesian et al., 2018], we create word embeddings for cells by summing the word embeddings of the tokens in their

values. For computing the similarity score, we use the positive cosine distance between the cell embedding, i.e.,

$$f_e(a,b) = \max(0, \frac{\bar{\boldsymbol{w}}(a) \cdot \bar{\boldsymbol{w}}(b)}{\|\bar{\boldsymbol{w}}(a)\|\|\bar{\boldsymbol{w}}(b)\|}) \tag{4.10}$$

where $\bar{\boldsymbol{w}}(X)$ is the mean of the embeddings of the cell values in X.

f_d: Datatype Similarity. The functions above consider only the cell values for computing the alignment score. The hyperlinks to Wikipedia pages that are present in cells can be used to create a semantic representation based on the types of entities that they link to. Additionally, we can exploit the repeated patterns in cell sets when they contain composite values involving multiple datatypes.

We proceed as follows: for every cell, we extract a number of patterns corresponding to possible semantic types. The patterns are created by detecting the named entities in the cell (we use the library Spacy (spacy.io)), and combining them with their hyperlinks. We replace each named entity in the cell with all the types of the entity in the KB. This results in patterns such as `[Football Cup] final [YEAR]`. Let a and b be two sets of cells, $N_p(a)$ be the number of unique cells in a from which pattern p is extracted, and P be the set of all patterns extracted from a and b. For every pattern extracted from a, we calculate its *overlap score* as $O_p(a) = N_p(a)$ / $|a|$ and keep only those patterns for which $O_p(a) > \tau$ (default value $\tau = 0.5$). Our datatype similarity function is the cosine similarity between the pattern overlap vectors $\boldsymbol{O}(a), \boldsymbol{O}(b) \in [0,1]^P$ of two cell sets:

$$f_c(a,b) = \frac{\boldsymbol{O}(a) \cdot \boldsymbol{O}(b)}{\|\boldsymbol{O}(a)\|\|\boldsymbol{O}(b)\|} \tag{4.11}$$

Clustering

Given the weighted graph $\mathcal{G}$ of table union candidate pairs, we perform clustering to find sets of unionable tables. This is equivalent to partitioning a similarity graph [Lehmberg and Hassanzadeh, 2018]. To this end, we employ Louvain Community Detection [Blondel et al., 2008] – a state-of-the-art algorithm that scales to large

graphs such as ours.

The Louvain algorithm optimizes a value known as *modularity*, which measures the density of links between different communities compared to those inside communities themselves. Recall that $\mathcal{W}$ is the matrix of weights in the edges, and let $z = \sum_{ij} \mathcal{W}_{ij}$ be the sum of its values. Given an assignment of a community c_i for each node i, the modularity is defined

$$Q = \frac{1}{2z} \sum_{ij} \left[\mathcal{W}_{ij} - \frac{k_i k_j}{2z} \right] \delta(c_i, c_j) \tag{4.12}$$

where k_i and k_j are the sum of the weights of the edges attached to nodes i and j respectively, and δ is the Kronecker delta function. Initially each node is in its own community, after which two steps are alternated until convergence. In the first step, each node is moved to the community that maximizes modularity. In the second step, the procedure constructs a new weighted graph $\mathcal{G}'$ and replaces $\mathcal{G}$ with it. In $\mathcal{G}'$, the nodes map to communities, weighted edges are an aggregated score of the edges between nodes in different communities in $\mathcal{G}$, while edges between nodes in the same community in $\mathcal{G}$ are represented by self-loops. The runtime of this procedure appears to scale with $O(n \cdot log^2 n)$ in the number of nodes [Lancichinetti and Fortunato, 2009].

After finding clusters of similar tables, we align all columns of the tables within each cluster. First, we create a matrix of max-aggregated column similarities using the matching functions in $\mathcal{M}$ for each pair of columns in the tables in the cluster. Then, we run agglomerative clustering [Manning et al., 2008] with complete linkage on this matrix to identify groups of similar columns. This iteratively combines the two clusters (i.e., two groups of columns) which are separated by the shortest distance.

Once the columns are clustered together, we create a union table with as many columns as clusters. Then, the tables are concatenated filling the gaps with empty cells. To create a header for this table, we take the most frequent header cell of each column cluster. The set of union tables will be the input of the next stage of our pipeline.

4.3.3 Integration

The last step in our pipeline consists of extracting facts from the union tables. This phase, shown at the bottom of Figure 4.1, determines the type of the union table and extracts the facts from it.

Detecting n-ary union tables

A key challenge in extracting facts from the union tables is distinguishing between EA and NA union tables. To this end, we make use of pFDs. This gives us a robust signal for union tables because their large number of rows prevents the pFDs from expressing noise, which occurs very frequently in small tables. Let $R(A_1, \ldots, A_m)$ be the relation associated to union table T and r be the instance of R with the body of T. We run perTuple on r to compute the set F_T of pFDs. Let B be the attribute of R with the highest harmonic mean of the multiset $\{p : A \rightarrow^p B \in F_T\}$ (ties are broken by taking the leftmost column). If the harmonic mean is greater than a given threshold υ (default value is 0.95) then we assume that T is a EA table and the column associated to B is the key column. Otherwise, T is an NA table.

Entity Linking

The extraction of factual knowledge from the union table is split into two phases. First, we link the cells in T to entities in $\mathcal{K}$, regardless the type of T. We make use of the hyperlinks whenever they are available and maximize the coherence of entities if multiple matches are possible, as described in Chapter 3. Note that here the large number of rows in the union tables is particularly helpful as it provides a clearer signal for linking the entities. In the following, we denote with $\mathsf{entity}(c) \in \mathcal{E}$ the entity associated with cell c if we found a match, otherwise $\mathsf{entity}(c) = \texttt{NULL}$.

Fact Extraction

We proceed differently depending on whether T is an EA or a NA table.

- *If T is a EA table*, we first retain all pFDs with a sufficiently high probability,

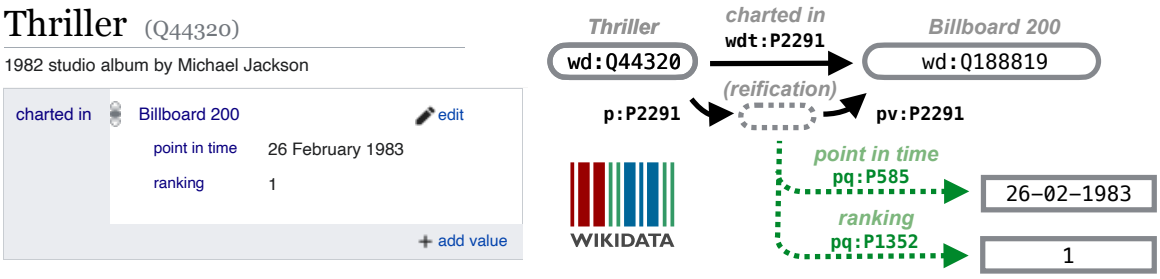


Figure 4.6: A statement on Wikidata with qualifiers (left), and its reification in RDF (right). Note that Wikidata represents the main fact with a single triple, but also uses an auxiliary entity to which the qualifiers are associated (URI prefixes are shown here with standard abbreviations).

i.e., greater than υ, which we call $F_T^{>\upsilon}$. For each pFD $A \to^p B \in F_T^{>\upsilon}$, we search for a property in $\mathcal{K}$ suitable to represent the dependency between A and B. Let $Col_X \in \mathsf{cols}(T)$ be the column of T associated with the attribute X in R. First, we compute the set of all pairs of entities mentioned in the columns, i.e., $E_{A,B} \coloneqq \{\langle a, b\rangle : \forall i.\ a = \mathsf{entity}(Col_A[i]) \wedge b = \mathsf{entity}(Col_B[i] \wedge a \neq \texttt{NULL} \wedge b \neq \texttt{NULL})\}$. Then, the set of matched facts for $A \to^p B$ and property $p \in \mathcal{P}$ in $\mathcal{K}$ is $M_{A,B}(p) \coloneqq \{\langle b, p, a\rangle : \langle a, b\rangle \in E_{A,B}\} \cap \mathcal{K}$. We pick the property $p \in R$ such that $p = \arg\max_{p \in \mathcal{P}} |M_{A,B}(p)|$, that is, the property with the maximum overlap, like [Muñoz et al., 2014]. Then, we output the fact $\langle b, p, a\rangle$ for each $\langle a, b\rangle \in E_{A,B}$ so that it can be added to $\mathcal{K}$.

- *If T is a NA table*, let $E_{A,B,C}$ be the set of tuples for attributes A, B, C defined analogously to $E_{A,B}$. For every possible pair of attributes A, B and property $p \in \mathcal{P}$, we first identify the columns that contain entities that appear in qualifiers of facts in $M_{A,B}(p)$. To this end, we denote with $N(A, B, C, p) \coloneqq \{\langle y, c\rangle : \langle a, b, c\rangle \in E_{A,B,C} \wedge \langle q_{b,p,a}, y, c\rangle \in \mathcal{K}\}$ the set of qualifiers that could be retrieved considering the entities in Col_C, and with $Q(A, B, p) \coloneqq \{C : N(A, B, C, p) \neq \emptyset\}$ the set of attributes where some qualifiers were found. Then, we consider all A, B, p with the highest number of qualifier-matching columns $|Q(A, B, p)|$ because these are the potential n-ary relations with the largest coverage of columns. If there are multiple A, B, p with the same highest $|Q(A, B, p)|$, then we choose the one with the highest number of matched facts $|M_{A,B}(p)|$. Finally,

for each $X \in Q(A, B, p)$, we identify the property r_X that has the highest frequency in the multiset $\{y : \langle y, c\rangle \in N(A, B, X, p)\}$. This property will be the one used to create the qualifiers with the entities in Col_X. At this point we are ready to extract the facts from T: We output the fact $\langle b, p, a\rangle$ for each $\langle a, b\rangle \in E_{A,B}$, and, for each $X \in Q(A, B, p)$, we output the triple $\langle q_{d,p,c}, r_X, e\rangle$ for each $\langle c, d, e\rangle \in E_{A,B,X}$.

Example 4.3.1. In Phase 2 of Figure 4.1, we show the union table constructed from the clustering step. Due to its detected pFDs, we classify it as an NA union table, and attempt to find matching qualifiers in $\mathcal{K}$. Let C, S, and P be the columns in this table that have the headers "Chart", "Single" and "Position", respectively. Then, we have that $\langle \texttt{US\_Billboard\_200}, \texttt{Brick\_House}, 5\rangle \in E_{C,S,P}$ (`US_Billboard_200` is the entity that matches the cell "US" in the table). Let's assume that the triple $\langle q_x, \texttt{ranking}, 5\rangle \in \mathcal{K}$ where $x = \langle \texttt{Brick\_House}, \texttt{chartedIn}, \texttt{US\_Billboard\_200}\rangle$. This means that $\langle \texttt{ranking}, 5\rangle \in N(C, S, P, \texttt{chartedIn})$ and $P \in Q(C, S, \texttt{chartedIn})$. If C, S, and `chartedIn` have the highest number of these qualifier-matching columns $Q(C, S, \texttt{chartedIn})$ and the highest number of matches $M_{C,S}(\texttt{chartedIn})$ of all column pairs and properties, we use the property `chartedIn` to extract facts from columns C and S. Finally, if `ranking` is the most frequent property in $N(C, S, P, \texttt{chartedIn})$, we use it for extracting qualifiers from column P. Let us assume that $E_{C,S,P}$ contains another tuple $\langle \texttt{US\_Billboard\_200}, \texttt{Thriller}, 1\rangle$. In this case, the system will output the facts $f = \langle \texttt{Thriller}, \texttt{chartedIn}, \texttt{US\_Billboard\_200}\rangle$ and $\langle \texttt{q}_\texttt{f}, \texttt{ranking}, 1\rangle$, which is graphically depicted in Figure 4.6.

4.4 Evaluation

For our empirical evaluation, we considered the corpus of 1,535,332 Wikipedia tables from [Bhagavatula et al., 2013], with 1,426,303 unique tables of which 26,260 occur on more than one page. There are 330,221 unique headers, of which 247,403 (75%) occur only once. This means there are 1,287,929 tables that have a header that is shared by some other table. On average, these tables have 11 rows. The experiments here

presented were performed with a Wikidata dump from December 2019.

4.4.1 Annotations

Since there were no available gold standard to test our method, we created it using three human annotators. To this end, we developed a GUI that showed the page title, description, section title and caption, and table contents. We sampled 1000 random tables, all from different Wikipedia pages, which have 3449 columns in total. We aggregated these annotations by majority vote, with moderate agreement between annotators (Fleiss' $\kappa = 0.57$).

First, the annotators annotated the columns that should be unpivoted by selecting a horizontal sequence of cells in the header of a table. The guidelines specified that the sequence should "contain names of a *related* set of concepts that *do not* describe the content of the column below them." After annotation, the table was shown to the annotator in unpivoted form for verification. This resulted in 151 tables from the sample to unpivot.

Then, the annotators were asked to create table unions from the unpivoted tables resulting from the previous phase by iteratively merging clusters. They were presented with one query cluster and several candidate clusters, ranked according to the matchers described above. All clusters were presented as a union table (i.e. "vertical stack") of all tables in that cluster. From these candidate clusters, the annotators were asked to identify the clusters that *expressed the same relation* as the query table. As a proxy, the guidelines suggested that they could ask themselves: "Would it make sense to add every row in the candidate table to the query table?" Then, they identified the aligning column pairs by either selecting a query column or adding a new column to the clustered union table for every candidate column. This resulted in 577 clusters with a total of 2479 columns, in which annotators identified a key column (marking it as an EA table), or identified it as an NA table.

Finally, we let annotators evaluate the property predictions for union table columns that were returned by our system. They assessed the correctness of every KB property that was assigned to a column based on whether its semantics corresponded to that of

Heuristic	Pr.	Re.	F_1
headerLike	0.30	0.33	0.31
numPrefix	0.73	0.27	0.39
sRepeated	**0.89**	0.12	0.21
numSuffix	0.52	0.09	0.16
linkAgent	0.70	0.05	0.09
rareOutlier	0.14	0.02	0.04
All Heuristics	0.44	**0.88**	**0.58**

Table 4.1: Precision, Recall and F_1 of unpivot detection heuristics

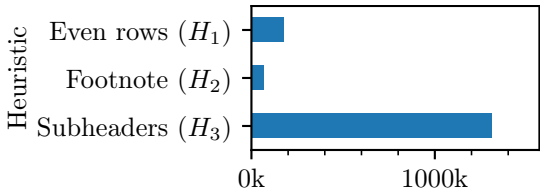


Figure 4.7: Number of cells processed by each heuristic in the mergechunks algorithm

all rows.

Although the collected annotations do not allow us to test every single component of the system, they are sufficient to evaluate the critical ones and can give us an overview of the overall performance. Below we report the results of some key experiments.

4.4.2 Table Reshaping

In Figure 4.7, we show the number of cells that span all columns which were identified by each heuristic from Section 4.3.1. Without the application of the mergechunks procedure, each of these 1,554,692 cells would be located in the wrong columns, where they would interfere with the subsequent operations.

On the entire dataset with 1.5M tables, smartunpivot predicted that 260,528 unique headers should be unpivoted, corresponding to 933,949 different tables. A breakdown of prediction scores per heuristic is shown in Table 4.1. The heuristics are designed to be complementary because they cover a different type of header. Therefore, they are not expected to individually have high recall, but ideally the recall should be high when they are combined together. From Table 4.1, we can see that this is indeed the case with a F_1 of 0.58. Examples of false positives of these heuristics include attribute labels that occur frequently in table bodies (such as "director"). The false negatives

Reshape	Clusters	ARI	AMI	H.	C.	V.
-	One per table	0.00	0.00	1.00	0.85	0.92
-	Same Header	0.55	0.55	1.00	0.90	0.95
Model	Same Header	0.55	0.56	1.00	0.90	0.95
Oracle	Same Header	0.56	0.57	1.00	0.90	0.95
Model	Only Header Sim.	0.83	0.77	0.97	0.95	0.96
Model	Only Body Sim.	0.69	0.67	0.99	0.93	0.96
Model	Both Sim.	**0.88**	**0.85**	0.99	0.97	0.98
Model	Oracle	0.93	0.94	1.00	0.98	0.99

Table 4.2: Clustering ablation for different configurations

include values for which we do not have enough statistics to result in high heuristic scores.

4.4.3 Table Clustering

Table 4.2 reports a feature ablation study with various combination of reshaping and clustering methods. The "Oracle" reshaping strategy unpivots the tables on annotations while "Model" uses our reshape(). The "Same Header" clusters are made by grouping tables with the same header and the "Oracle" clusters are based on the gold-standard annotations. We show the performance of our clustering phase when using only the header similarities ($\theta = 1$ in Alg. 5), the body similarities ($\theta = 0$ in Alg. 5), and all similarity functions together (θ set in Alg. 5 with cross validation).

We use several scoring functions to evaluate the generated clusters. Because clusters can have very different sizes, and there are many clusters of a single element, some of these metrics are adjusted for chance to reduce the scores of random or degenerate clusterings. The *Adjusted Rand Index (ARI)* expresses the cluster and class agreement of item pairs, adjusted for chance [Hubert and Arabie, 1985]. The *Adjusted Mutual Information (AMI)* expresses the mutual information between the clusters and classes, adjusted for chance [Vinh et al., 2010]. Furthermore, a clustering result satisfies *Homogeneity (H.)* if all of its clusters contain only data points which are members of a single class. A clustering result satisfies *Completeness (C.)* if all the data points that are members of a given class are elements of the same cluster. The *V-measure (V.)* is their harmonic mean (similar to the F_1 score) [Rosenberg and

	Accuracy			
Method	EA	NA	Both	EA pct.
Non-num. baseline	0.29	0.10	0.39	90%
Entity baseline	0.24	0.20	0.43	74%
Ours	**0.45**	**0.77**	**0.66**	**36%**

Table 4.3: Key column prediction accuracy for EA tables, NA table detection accuracy, both tasks combined, and the EA table percentage predicted (EA pct.). The true percentage of EA tables in our annotated sample is 35%

		Existing Facts in $\mathcal{K}$		New Facts	
	Precision	#	Context	#	Context
Binary	0.65	4,708,865	33%	15,800,524	29%
N-ary	**0.92**	2,122,518	22%	6,901,790	19%
All	0.74	**6,831,383**	30%	**22,702,314**	26%

Table 4.4: Relation matching precision, number of matched and new facts, and percentage of matched and new facts that involve the table context

Hirschberg, 2007].

From Table 4.2, we see that clustering using only similarities of headers ("Only Head Sim.") outperforms the one that considers the body ("Only Body Sim."), but their aggregation gets us nearest to the performance of the oracle, which is built with human annotations.

4.4.4 Table Interpretation and KB Integration

In Table 4.3, we report the performance of our key-column and n-ary table detection approach, and compare it to two baselines. The "Non-numeric" baseline selects the rightmost non-numeric column that contains at least 95% unique values if it exists, and the "Entity" baseline does the same thing for columns which contain entities [Ritze et al., 2016]. If such columns do not exist, they predict the table is n-ary. Note that these baselines are the most commonly used ones by state-of-the-art systems, and our own system described in Chapter 3. Our approach is more accurate for both EA tables and NA tables, and closest to predict the correct rate of EA tables. In fact, the baselines are severely biased and select EA tables in 90% and 74% of the times, incorrectly labeling many NA tables and consequently precluding the extraction of

	Tables	Columns
Individual	173 (17%)	267 (8%)
Union	193 (**33%**)	419 (**17%**)

Table 4.5: Number of matched tables and columns in annotated sample

n-ary facts. In contrast, our performance is much closer to the true rate of 35%.

In Table 4.5, we compare the property matching counts for individual tables from the annotated sample to the counts for our union tables. The percentage of (union) tables and columns for which we could match KB properties is higher for the union tables, which indicates that creating union tables from clusters leads to more matched properties that we can use for fact extraction.

Finally, Table 4.4 reports the precision for binary and n-ary facts extracted by our system on the gold standard. Interestingly, we observe that our system is more precise at extracting n-ary facts instead of binary facts. We applied our pipeline to the corpus of 1.5M tables and our system extracted 29.5M facts. Of these, 77% are novel facts, i.e., not available in Wikidata. The columns titled "Context" report the percentage of facts that involved the extra columns added by addcontext(). The large ratio indicates that including external context is beneficial as it leads to more extractions.

4.5 Conclusion

In this chapter, we tackled the problem of automatically integrating the knowledge in Wikipedia tables into a KB. In particular, we focused on the extraction of both EA and NA tables. Our pipeline introduces new heuristics for table transformation based on, among others, unpivoting, which allows us to consider more tables. Then, our method uses clustering to alleviate the problems of KB incompleteness and table diversity. Finally, the integration step judiciously interprets the table, distinguishing EA tables from NA ones and extracts facts in a format suitable for a KB integration.

We carried out an empirical evaluation, relying on manual annotators to verify the high quality of our extractions. Our implementation is scalable as we were able to process all 1.5M tables from Wikipedia. More generally, all steps except the Louvain

algorithm for clustering scale linearly with the number of tables.

Future work includes creating a model for filtering the extracted facts to boost their precision, developing a semi-supervised model for table transformation, and designing more sophisticated alignment functions for improving the quality of the table clusters. Additionally, the performance of our pipeline on non-Wikipedia tables remains an open question. In particular, our pipeline assumes that there is some overlap between the entities mentioned in the tables and in the KB and that such entity mentions can be successfully linked to the KB. It is interesting to study whether we can relax such assumptions so that our pipeline can be applied on different table corpora.

CHAPTER 5

Building a KB from Tables in Scientific Papers

In this chapter, we describe our research on how to build a coherent KB from tables on a new domain, with minimal human effort. By focusing on tables from scientific papers, we put our work on information extraction from tables in practice, within a highly dynamic domain that warrants flexible human supervision and control.

Tables in scientific papers contain a wealth of valuable knowledge for the scientific enterprise. To help the many of us who frequently consult this type of knowledge, we present Tab2Know, a new end-to-end system to build a Knowledge Base (KB) from tables in scientific papers. Tab2Know addresses the challenge of automatically interpreting the tables in papers and of disambiguating the entities that they contain. To solve these problems, we propose a pipeline that employs both statistical-based classifiers and logic-based reasoning. First, our pipeline applies weakly supervised classifiers to recognize the type of tables and columns, with the help of a data labeling system and an ontology specifically designed for our purpose. Then, logic-based reasoning is used to link equivalent entities (via *sameAs* links) in different tables. An empirical evaluation of our approach using a corpus of papers in the Computer Science domain has returned satisfactory performance. This suggests that ours is a promising

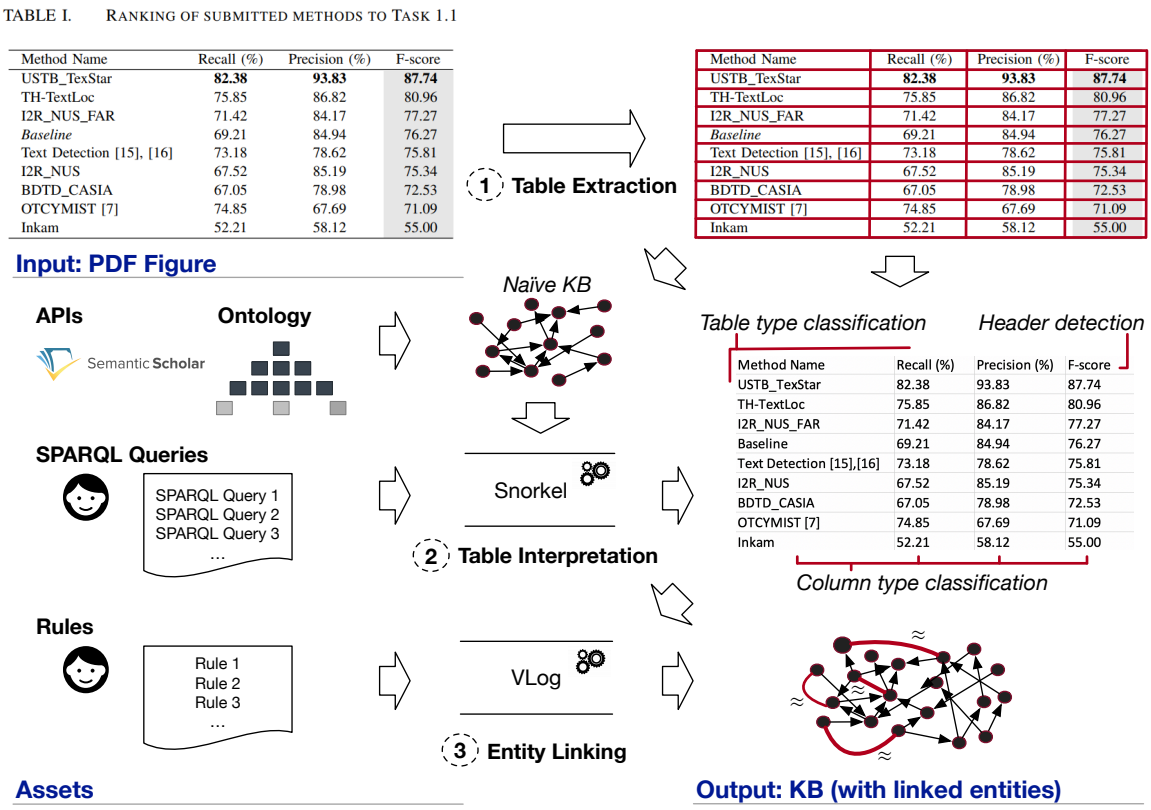


Figure 5.1: Tab2Know: System Overview

step to create a large-scale KB of scientific knowledge.

5.1 Introduction

Often, scientific advancement requires an extensive analysis of pre-existing techniques or a careful comparison with previous experimental results. For instance, it is common for researchers in Artificial Intelligence (AI) to ask questions like “Which are the most popular datasets used for graph embeddings?” or “What is the F1 of BERT on TACRED?". Finding the answers obliges the researchers to spend much time in perusing existing literature, looking for experimental results, techniques, or other valuable resources.

The answers to such questions can be frequently found in tabular form, especially the ones that describe the output of experiments. Unfortunately, tables in papers are made for human consumption; thus, their layout can be irregular or contain

specific abbreviations that are hard to disambiguate automatically. It would be very useful if their content were copied into a clean Knowledge Base (KB) where tables are disambiguated and connected using a single standardized vocabulary. This KB could assist the users in finding those answers without accessing the papers or could be used for many other purposes, like categorizing papers, finding inconsistencies or plagiarized content.

To build such a KB, we present Tab2Know, an end-to-end system designed to interpret the tables in scientific papers. The main challenge tackled by Tab2Know lies in the interpretation of the table, which is a necessary step to build a KB. In this context, the peculiarities of tables in scientific literature make our domain quite different from previous work (e.g., [Bhagavatula et al., 2015, Ritze et al., 2015]), which mainly focused on Web tables. First, the interpretation of Web tables benefits from the existence of large, curated KBs (e.g., DBPedia [Auer et al., 2007]), which allows the linking of many entities. In our case, there is no such KB. Second, a large number of Web tables can be categorized as *entity-attribute* tables, i.e., tables where each row describes one entity, and the columns represent attributes [Ritze et al., 2015, Zhang, 2017]. In our context, we observed that many tables are of different types, namely they express n-ary relations, such as the results of experiments. For such tables, existing techniques designed for entity-attribute tables cannot be reused.

With Tab2Know, we propose a pipeline for knowledge extraction that includes both weakly supervised learning methods and logical reasoning. Tab2Know is designed to 1) detect the type of the table; 2) disambiguate the types of columns, and 3) link the entities between tables. The first operation is applied to distinguish, for instance, tables that report experiments from tables that report examples. The second operation recognizes the rows that contain the headers of the table and disambiguates the columns, linking them to classes of an ontology. The third operation links entities in different tables.

We implement the first two operations using statistical classifiers trained with bag-of-words and context-based features. These classifiers have an accuracy that largely depends on the quality and amount of training data. Unfortunately, labeling

training data is increasingly the largest bottleneck as it often requires an expensive manual effort and/or expertise that might not be readily available. To counter this problem, we propose a weakly supervised method that relies on SPARQL queries and Snorkel [Ratner et al., 2020]. The SPARQL queries are used to automatically retrieve samples of a given class, type, etc., while Snorkel resolves potential conflicts in the prediction with a sophisticated voting mechanism.

After the first two operations are completed, we transform the tables into an RDF KB and apply reasoning with existentially quantified rules to identify and link entities in different tables. Reasoning with existentially quantified rules is a well-known technology for data integration and wrangling [Konstantinou et al., 2019]. For our problem, we designed a set of rules that considers the types of columns and string similarities to establish links using the *sameAs* property. Then, we used VLog [Carral et al., 2019] to materialize the derivations and link the entities across the tables.

We evaluated our approach considering open access CS papers. In particular, we evaluated the performance of our pipeline using gold standards and compared it to another state-of-the-art method. We also applied our method to a larger corpus with 73k scientific tables. In these tables, we found 312k entities, which are linked to the table structure and metadata in our large-scale KB.

We release the datasets, gold standards, and resulting KB as an open resource for the research community at `https://doi.org/10.5281/zenodo.3983012`. The code, ruleset, and instructions to replicate our experiments in this chapter are also publicly available at `https://github.com/karmaresearch/tab2know`.

5.2 Background

Extracting knowledge from tables from PDF documents is a process that can be divided into *three* main tasks: *table extraction*, *table interpretation*, and *entity linking*. Below, we refer to the related work in Chapter 2 and how it relates to the research described in this chapter.

Table extraction The task of table extraction consists of detecting the parts of a PDF/image which contain a table, and then recognizing the table's structure so that it can be correctly extracted.

When attempting to extract tables from PDFs or images, the first step is to detect them inside the document. Existing methods can be categorized either as heuristic (e.g., Oro and Ruffolo [2009], Clark and Divvala [2016]) or supervised (e.g., Pinto et al. [2003]). Methods in the first category are typically unsupervised and apply heuristics like the presence of captions, etc. Prominent systems in this category are Trex [Oro and Ruffolo, 2009] and Tableseer [Liu et al., 2007]. In contrast, methods in the second category train statistical models. Prominent methods are the ones by [Pinto et al., 2003, Choudhury et al., 2015]. The system *PDFFigures* [Clark and Divvala, 2016] is a recent approach based on heuristics with very high precision and recall ($\geq 90\%$) on the domain of scientific papers, and is used in Semantic Scholar [Ammar et al., 2018]. Therefore, this is the system we employed for table detection in this work.

Given as input the detected image-like representation of a table from the first step, there are various systems that focus on subsequently recognizing the table's structure so that it can be correctly extracted. A popular system is Tabula[1]), which recognizes the table's structure using rules. More recently, some deep learning methods based on Convolutional Neural Networks (CNN) [Schreiber et al., 2017], Conditional Generative Adversarial Networks (CGAN) [Vine et al., 2019], and a combination of a CNN, saliency and graphical models [Kavasidis et al., 2018] have been evaluated. The performance of these methods is good ($F_1 \geq 0.95$), but not much different from Tabula, which returns a F_1 between 0.86 and 0.96 and has the advantage that is unsupervised.

Because the main difficulty in this space is the diversity of table layouts across domains, subsequent work has emphasized the creation of, and evaluation on, large corpora gold-standard datasets from diverse sources [Li et al., 2019, Zhong et al., 2020]. However, this challenge is not yet solved robustly for down-stream applications such as ours. In our current research, we have therefore chosen to adapt the commonly

[1] `https://tabula.technology/`

used, domain-invariant Tabula system for performing this step.

Table interpretation The goal of table interpretation consists of linking the content of the table to a KB so that new knowledge can be extracted from the table. We provide an extensive overview of techniques in Chapter 2, section 2.3. However, these techniques are mostly designed for Web tables and rely on a rich KB like DBPedia [Auer et al., 2007], which we do not have.

The only work that has focused on the interpretation of tables from scientific literature is [Yu et al., 2020]. The authors describe an approach to automatically extract experimental data from tables based on ensemble learning. Although we view this work as the most relevant to our problem, there are several important differences between our work and theirs. First, our approach employs a different set of technologies and performs entity linking, which is not considered in [Yu et al., 2020]. Then, our approach is more general. In fact, [Yu et al., 2020] focuses only on the extraction of tuples $(method, dataset, metric, score, source)$ while ours extracts a larger variety of knowledge. Finally, our approach yields a better accuracy (see Section 5.6).

Entity Linking See Chapter 2, Section 2.4.2 for an overview of related work on entity linking. Our work differs from these because they either focus on highly structured table sets or require the existence of KBs (which we do not have). Moreover, another important difference is that we take a declarative approach with rules. Rules are useful because they can be easily debugged/extended directly by domain experts, and they can be integrated with ontological reasoning.

Other related works We mention, as further related work, the systems by [Das Sarma et al., 2012] and *TableNet* [Fetahu et al., 2019] which focus on *searching* for tables related to a given query. Other, less relevant works focus on extracting and searching for figures on papers [Siegel et al., 2016]. These works complement our approach and can further assist the user to find relevant knowledge in papers.

5.3 Overview

Our goal is to construct a clean and large KB from the content of tables in scientific papers stored as PDFs. To do so, we need to address two main challenges: first, we must resolve the ambiguities that might arise during the noisy extraction process and reduce the error rate as much as possible. Second, we must counter the problem that we lack both: 1) a pre-existing KB that can guide the extraction process and 2) a large amount of training data. We must, in other words, find a way to build a KB from scratch.

Our proposal is a pipeline with three steps, as shown in Figure 5.1:

- **Preprocessing: Table Extraction.** The system receives as input an image-like representation of a table, recognizes its structure, and returns its content as a CSV file. For this step, we use external tools. We provide more details below;
- **Task 1: Table Interpretation.** The system processes the CSV to recognize the headers and the type of the table. Then, it disambiguates the columns by mapping them to ontological classes. We describe this task in Section 5.4;
- **Task 2: Entity Linking.** Finally, the system performs logical-based reasoning to link the entities across tables. We describe this task in Section 5.5.

While in principle our method can be applied to scientific papers in any domain, we restrict our analysis to papers in Computer Science, which is our area of expertise. In particular, we consider Open Access papers and have been published in top-tier venues in subfields like AI, semantic web, databases, etc.

Before we describe the components, we describe *two* additional assets that we use for different purposes. The first one is an ontology constructed annotating a sample of random tables. A first version of this ontology contained 44 classes organized in a hierarchy with a maximum depth of 6. After further annotations, we decided to simplify it to a set of of 27 classes (depth 3) for which we had substantial evidence in our corpus. The final ontology has 4 root classes: **Example**, **Input**, **Observation**, and **Other**. These classes define general table types. Then, the subclasses describe column

```
owl:Thing
    :Example
    :Input
        :Category
        :Dataset
        :ExperimentalVariable
            :Feature
            :InputSize
        :Method
            :Task
    :Observation
        :Maximum
        :Metric
            :Accuracy
            :AreaUnderCurve
            :BLEU
            :Correlation
            :Count
            :Error
            :Mean
            :Median
            :Precision
            :Recall
            :F-score
            :StandardDeviation
        :Runtime
    :Other
        :SymbolDescription
```

Figure 5.2: Taxonomy of our column types.

types, e.g., **Dataset**, **Runtime**, or **Mean**. This taxonomy is shown in Figure 5.2. The ontology is serialized in OWL using WebProtégé [Horridge et al., 2019] and is publicly available as resource.

The second asset is an external KB that contains metadata of the papers, namely Semantic Scholar [Ammar et al., 2018]. We access it using the provided APIs to retrieve the list of authors, the venue, and other contextual data.

Preprocessing: Table Extraction Our input consists of a collection of papers in PDF format. The first operation consists of launching PDFFigures [Clark and Divvala, 2016] to extract from the PDFs the coordinates of tables and related captions. We use

the coordinates to extract an image-like representation of the tables, see for instance the table reported in Figure 5.1. Then, we invoke Tabula, which is a tool also used in similar prior works [Yu et al., 2020], to recognize the structure of the tables using their coordinates and to translate them into CSV files.

After the images are converted, we perform a naïve conversion of the tables into RDF triples. We assign a URI to every table, column, row, and cell and link every cell, row, and column to the respective table with positional coordinates.

Example 5.3.1. Consider the table in Figure 5.1. We report below some triples that are generated while dumping its content into RDF.

```
PREFIX : http://xzy/tab2know
:Table1 :hasRow :Table1-r1              :Table1 :hasCol :Table1-c1
:Table1-r1 rdf:type :Row                :Table1-c1 rdf:type :Column
:Table1-r1 :rowIndex 1^^⟨xsd:int⟩       :Table1-c1 :colIndex 1^^⟨xsd:int⟩
:Table1-r1c1 :cellOf :Table1            :Table1-r1c1 rdf:type :Cell
:Table1-r1c1 :rowIdx 1^^⟨xsd:int⟩       :Table1-r1c1 :colIdx 1^^⟨xsd:int⟩
:Table1-r1c1 rdf:value "Method name" :Table1-r2c1 rdf:value "USTB_TexStar"
                                        ...
```

As we can see from the triples in Example 5.3.1, the KB generated at this stage is a direct conversion of the tabular structure into triples. Despite its simplicity, however, such a KB is already useful because it can be used to query the n-ary relations expressed in the tables in combination with the papers' metadata. For instance, we can write a SPARQL query to retrieve all the tables created by one author with a caption containing the word "results", or to retrieve the tables containing "F1" and which appear as proceedings of a certain venue.

The main problem at this stage is that we can only query using string similarities, which severely reduces the recall. For instance, a query could miss a column titled `Prec.` if it searches for `Precision`. The next operation, described below, attempts to disambiguate the tables to create a KB that is more robust against the syntactic diversity of the surface form of their content.

5.4 Task 1: Table Interpretation

Tab2Know performs three main operations to interpret the tables. First, it identifies the rows with the table's header (Section 5.4.2). Then, it detects the type of the table (Section 5.4.3). Finally, it maps each column to an ontological class (Section 5.4.4). First, we describe the procedure to obtain training data.

5.4.1 Training Data Generation

Statistical models are ideal for implementing a table interpretation that is robust against noise. However, their accuracy depends on high-quality training data, which we do not have (and it is expensive to obtain such data with human annotators). We counter this problem following the paradigm of *weak supervision*. The idea is to employ many annotators, which are much cheaper than a human expert but also much noisier. These annotators can deliver a large volume of labeled data, but the labels might be incorrect or conflicting. To resolve these problems, we can either rely on procedures like majority voting or train a dedicated model to computed the most likely correct label. In the second case, we can use Snorkel, one of the most popular models for this purpose [Ratner et al., 2020].

Snorkel's goal is to facilitate the learning of a model θ that, given a data point $x \in \mathcal{X}$, predicts its label $y \in \mathcal{Y}$. Instead of training θ by fitting it to a set of pre-labeled data points, as it would happen in a traditional supervised approach, Snorkel trains an additional generative model with unlabeled data and uses pre-labeled data only for validation and testing. For these two tasks, the amount of pre-labeled data can be much smaller, and thus cheaper to obtain. Then, the generative model can be used to train θ.

Snorkel introduces the term *labeling function* to indicate a data annotator with possibly low accuracy. A labeling function $\lambda : \mathcal{X} \to \mathcal{Y} \cup \{\emptyset\}$ can encode a heuristic or be a simple predictor. It receives a data point x in input and either returns a label in $\mathcal{Y}$ or *abstains*, i.e., returns $\emptyset$. Given m unlabeled data points and n labeling functions, Snorkel applies the labeling functions to the data points and computes a

matrix $M \in (\mathcal{Y} \cup \{\emptyset\})^{m \times n}$.

Then, Snorkel processes M to compute, for each x_i where $i \in \{1, \ldots, m\}$, a *probabilistic training label* $\tilde{y}_i$. The processing consists of creating a generative model using a matrix completion-style algorithm over the covariance matrix of the labels [Ratner et al., 2019]. Then, this model can be used to generate labeled data for training θ. In this work, we considered three families of models, Naïve Bayes (NB), Support Vector Machine (SVM), and Logistic Regression (LR) [Bishop, 2006], to implement θ. We have also experimented with deep learning models, but we did not obtain improvements because such models are more prone to overfitting if training data is scarce or contains many regularities, which is a well-known feature of weak supervision labels [Wyatte, 2019].

The effectiveness of Snorkel largely depends on the number and quality of the labeling functions. In our context, we implemented them using SPARQL queries, which are supposed to be entered by a (human) user. SPARQL queries are ideal because they can assign labels to many data points at once. For each query Q, we create a labeling function that receives in input a column/table x and returns an assigned class label (e.g., a table type, or the class of a column) if x is among the answers of Q. Otherwise, the function abstains.

Example 5.4.1. We show below an example of a SPARQL query that labels columns with the class `:F-score` if they have a header cell with value "f1" and contain any cell with a numeric type.

```
select distinct ?column ?coltype where {
      ?table :column ?column ; :cell ?cell .
      ?column :hasTitle "f1" .
      ?cell rdf:type xsd:decimal .
      bind( :F-score as ?coltype )
   }
```

Clearly, this query is not a good predictor if taken alone, but if we combine its output with the ones of many other functions, then the resulting predictive power is likely to be superior. This is the key observation used by Snorkel.

In our pipeline, we execute all the user-provided SPARQL queries and then use their outputs to build the matrix M for a large number of data points. Next, we train the final discriminative model θ. We compute two different θ: One to generate training data for predicting the tables' types (Section 5.4.3) while the other is for predicting the columns' types (Section 5.4.4).

5.4.2 Table Header Detection

First, we identify the rows that define the headers. To this end, we can either always select the first row as header or employ more sophisticated methods to recognize multi-row headers, like [Fang et al., 2012]. We observed that a simplified unsupervised version of [Fang et al., 2012] yields a good accuracy on our dataset. We describe it below.

Our procedure exploits the observation that header rows differ significantly from the rest of the table with respect to character-based statistics. Hence, we categorize characters either as *numeric*, *uppercase*, *lowercase*, *space*, *non-alphanumeric*, or *other*. Then, for each column, we count how many characters of each class (e.g., numeric) appear in its cell. We compute the average count per class across the column and use these values to determine the standard deviation for each cell. The *outlier score* of a row r is determined as the average of the standard deviations of all classes of its cells. If the outlier score or r is greater than τ (default value is 1, set after cross-validation), then r is marked as header.

5.4.3 Table Type Detection

In scientific papers, tables are used for various reasons. We classified them in the classes **Observation**, **Input**, **Example**, and **Other** (See Figure 5.3 for examples).

Knowing the class of a table is useful for reducing the search space when the user is interested in some specific content (e.g., The F_1 score is typically not mentioned in tables of type **Example**). Moreover, we can also use this information as a feature for the column disambiguation.

l_1-l_2	#S	#l_1-W	#l_2-W	#l_1-V	#l_2-V
en-de	1.9M	55M	52M	40k	50k
en-fr	2.0M	50M	51M	40k	50k
en-es	1.9M	49M	51M	40k	50k

(a) Input

Models	Rerank size	Beam size	GMV	Latency
miDNN	50	-	2.91%	9%
miRNN	50	5	5.03%	58%
miRNN+att.	50	5	5.82%	401%

(b) Observation

Type	**Example Words**
Offensive	disgusting, filthy, nasty, rude, horrible, terrible, awful, worst, idiotic, stupid, dumb, ugly, etc.
Non-offensive	help, love, respect, believe, congrats, hi, like, great, fun, nice, neat, happy, good, best, etc.

(c) Example

α_c	DP concentration parameter for each $c \in V$
$P_0(e\|c)$	CFG base distribution
$\boldsymbol{x}$	Set of non-terminal nodes in the treebank
$\mathcal{S}$	Set of sampling sites (one for each $x \in \boldsymbol{x}$)
S	A block of sampling sites, where $S \subseteq \mathcal{S}$
$\boldsymbol{b} = \{b_s\}_{s \in \mathcal{S}}$	Binary variables to be sampled ($b_s = 1 \rightarrow$ frontier node)
$\boldsymbol{z}$	Latent state of the segmented treebank
m	Number of sites $s \in S$ s.t. $b_S = 1$
$\boldsymbol{n} = \{n_{c,e}\}$	Sufficient statistics of $\boldsymbol{z}$
$\Delta n^{S:m}$	Change in counts by setting m sites in S

(d) Other

Figure 5.3: Examples of tables of each category

We predict the table type with a statistical classifier. As features for the classifier, we selected bags-of-ngrams of lengths 1 to 3 that occurred more than once, weighted by their TF-IDF score. Tables often contain abbreviations and domain-specific symbols that address an audience of experts. These provide strong hints for determining the type of the table; thus we consider the ngram in the content of the cells and the table caption. We also included other numerical features. In particular, we use the fraction of numeric cells in the table and the minimum, maximum, median, mean and standard deviation of numerical columns. This resulted in a total of 5804 features.

To train the models, we first ask the users to specify some SPARQL queries which will be used by Snorkel to create a large volume of training data. Then, we experimented with three well-known types of classifiers: NB, SVM, and LR. Eventually, we selected LR because it returned the best performance on the noisiest dataset.

5.4.4 Column Type Detection

Finally, the interpretation procedure attempts at linking the columns to one of the available classes in our ontology. The ontology includes popular classes that we identified while annotating a sample (e.g., **Dataset**, **Runtime**,. . .), while infrequent classes with very few columns are mapped to the class **Other**. In general, we assume that a column is *untyped* if it is mapped to **Other**.

For this task, we also used bag-of-ngram features of lengths 1 to 3, extracted from the table caption, the column header cells, the header cells of the other columns, and the column body. We restricted the set of ngrams to only the top 1000 most frequent per extraction source. Additionally, we added features about the numerical columns, identical to those in Section 5.4.3. This resulted in a total of 3076 features.

Similarly as before, we first rely on user-provided SPARQL queries to generate training data. Then, we considered NB, SVM, and LR as classifiers. Once the models for the table and column types are trained, we use them to predict the types of every table and column in our corpus. Finally, we use the predicted class to annotate the table/column in the KB with a semantic type.

5.5 Task 2: Entity Linking

Rationale Predicting the types of tables and columns is useful to map the table schema into a meaningful n-ary relation. The last operation in our pipeline consists of associating cells to entities so that we can populate the n-ary relations with new instances.

We start by assuming that every non-numerical cell contains an entity mention, which implies the existence of one entity. This assumption is not unrealistic. Indeed, if we look at the table in Figure 5.1, then we see that every non-numerical cell that is not in the table's header refers to an entity (e.g., the cell "USTB_TextStar" refers to an algorithm to detect text inside images).

In practice, it is likely that some entities are mentioned multiple times. This consideration motivates us to discover whether two entity mentions (possibly on different tables) refer to the same entity. When we do so, then we gain more knowledge about the entity and reduce the number of entities in the target KB. We call this task *entity linking* because we are linking, with the *sameAs* property, equivalent entities across tables.

With this goal in mind, we start by assuming that every entity has the content of the corresponding cell as label. For instance, the entity mentioned in the cell with

"USTB_TextStar" has "USTB_TextStar" as label. Using the labels to determine equality can be surprisingly effective in practice, but it is not an operation without risks. In fact, there are cases where different entities have the same label, or the same entity has multiple labels. These cases call for a more sophisticated procedure to discover equalities.

Reasoning Reasoning with existentially quantified rules is an ideal tool to establish non-trivial equalities between entities since it was already previously used for data integration problems [Fagin et al., 2003, Geerts et al., 2014]. For our purposes, we are interested in applying two types of rules: *Tuple Generating Dependencies (TGDs)* and *Equality Generating Dependencies (EGDs)*. We describe those below.

Consider a vocabulary consisting of infinite and mutually disjoint sets of properties $\mathcal{P}$, constants $\mathcal{C}$, null values $\mathcal{N}$, and variables $\mathcal{V}$. A *term* is either a constant, a variable, or a null value. An *atom* is an expression of the form $p(\vec{x})$ where $p \in \mathcal{P}$, $\vec{x}$ is a tuple of terms of length equal to the arity of p, which is fixed. A *fact* is an atom without variables. A TGD is a rule of the form:

$$\forall \vec{x}, \vec{y}.(B \rightarrow \exists \vec{z}.H) \tag{5.1}$$

where B is a conjunction of atoms over $\vec{x}$ and $\vec{y}$ while H is a conjunctions of atoms over $\vec{y}$ and $\vec{z}$. Let $x, y \in \vec{x}$. A EGD is a rule of the form:

$$\forall \vec{x}.(B \rightarrow x \approx y) \tag{5.2}$$

Intuitively, TGDs are used to infer new facts from an existing set of facts (i.e., the database). Their execution consists of finding in the database suitable replacements for the variables in $\vec{x}$ and $\vec{y}$ that render B a set of facts in the database. Then, these replacements and mappings from $\vec{z}$ to fresh values in $\mathcal{N}$ are used to map H into a set of facts, which is the set of *inferred* facts.

EGDs are used to establish the equivalence between terms. Their execution is similar to the one of TGDs, with the difference that whenever they infer that $a \approx b$,

where a and b are terms and $a < b$ according to a predefined ordering, then every occurrence of b in the database is replaced with a.

The *chase* [Fagin et al., 2003] is a class of forward-chaining procedures that exhaustively apply TGDs and EGDs to infer new knowledge with the rules. A formal definition of various chase procedures is available at [Benedikt et al., 2017]. In this work, we apply the restricted chase, one of the most popular variants. It is known that sometimes the chase may not terminate, but this is not our case since we use an *acyclic* ruleset [Fagin et al., 2003].

We first map the content of the KB extracted from the tables into a set of facts. For example, the first two RDF triples in Example 5.3.1 map to the facts `hasRow(Table1,Table1-r1)` and `hasCol(Table1,Table1-c1)` respectively.

Then, we use the two TGDs

$$type(X, \texttt{Column}) \rightarrow \exists Y.colEntity(X, Y) \qquad (r_1)$$

$$type(X, \texttt{Cell}) \rightarrow \exists Y.cellEntity(X, Y) \qquad (r_2)$$

to introduce fresh entities for every column and cell in the tables. The predicates *colEntity* and *cellEntity* link entities (Y) to the columns and cells respectively. Note that we use null values to represent entities, thus we are simply stating their existence with some placeholders. To reason and discover whether two different entities are equivalent, we employ EGDs. In particular, we use five EGDs, reported below:

$$ceNoTypLabel(X, L), ceNoTypLabel(Y, L) \rightarrow X \approx Y \qquad (r_3)$$

$$eNoTypLabel(X, C, L), eNoTypLabel(Y, C, L) \rightarrow X \approx Y \qquad (r_4)$$

$$eTableLabel(X, T, L), eTableLabel(Y, T, L) \rightarrow X \approx Y \qquad (r_5)$$

$$eTypLabel(X, S, L), eTypLabel(Y, S, M), STR\_EQ(L, M) \rightarrow X \approx Y \qquad (r_6)$$

$$eAuthLabel(X, A, L), eAuthLabel(Y, A, M), STR\_EQ(L, M) \rightarrow X \approx Y \qquad (r_7)$$

where *ceNoTypLabel*, *eNoTypLabel*, *eTableLabel*, *eTypLabel*, and *eAuthLabel* are auxiliary predicates that we introduce for improving the readability. We describe their

intended meaning as follows. The fact $ceNoTypeLabel(X, L)$ is true if $colEntity(Y, X)$ is true and Y is an untyped column with header value L; $eNoTypeLabel(X, C, L)$ is true if X is an entity with a label L that appears in a cell inside an untyped column associated to entity C; $eTableLabel(X, T, L)$ is true if entity X with label L appears in table T; $eTypeLabel(X, S, L)$ is true if entity X with label L appears in a column with type S; $eAuthLabel(X, A, L)$ is true if entity X with label L appears in a table authored by author A.

The rationale behind each EGD is the following:

- **Rule $\mathbf{r_3}$** : This rule is introduced to disambiguate untyped columns. Since we were unable to discover the columns' types and assigned them to the class `Other`, we use the value of the header to determine whether they contain the same type of entities. Thus, the rule will infer that their associated entities are equal if they share the same header.

- **Rule $\mathbf{r_4}$** : This rule infers that two entities are equal if they appear in the same group of columns (created by r_3), and they share the same label.

- **Rule $\mathbf{r_5}$** : This rule encodes a simple heuristics, namely that if two entities with the same label appear in the same table, then they should be equal, irrespective of the type of columns where they appear.

- **Rule $\mathbf{r_6}$** : This rule disambiguates entities in columns of the same type. Here, we no longer consider the header of the column (as done by r_3 and r_4) but compare the entities' labels. After experimenting with approximate string similarity measures, like the Levenshtein distance, we decided to use a case insensitive string equality (STR_EQ) to reduce the number of false positives. Case-insensitive similarity is more expensive than an exact string match because it requires dictionary lookups. We use it here and not in r_3, r_4, and r_5 because the comparisons are done only between entities of the same type.

- **Rule $\mathbf{r_7}$** : This rule implements another heuristic which takes into account the authors of the paper. It assumes that two entities are equal if they appear in

two tables authored by the same author (we used the IDs provided by Semantic Scholar to disambiguate authors) and have the same label.

Once the reasoning has terminated, we introduce a new entity for each different null value and add RDF triples that link them to the corresponding cells and columns. Notice that the list of presented rules is not meant to be exhaustive. The ones that we describe show how we can exploit the predictions computed in the previous step (r_6) and external knowledge (r_7) relying on string similarity when no extra knowledge is available. We believe that additional EGDs, possibly designed to capture some specific cases, can further improve the performance.

5.6 Evaluation

Inputs We considered two datasets: A corpus of tables that we manually constructed, and the dataset by [Yu et al., 2020], which is called *Tablepedia*.

Our corpus of tables contains 142,966 open-access PDFs distributed by Semantic Scholar. These papers appear in the proceedings of top venues in CS: AAAI, ACL, Artif. Intell., ArXiv, CIKM, COLING, CoNLL, EACL, ECAI, EMNLP, HLT-NAACL, IJCAI, NeurIPS, NIPS, (P)VLDB, and WWW. Notably, we excluded papers from non-open-access publishers Elsevier, IEEE, and ACM. From these papers, we extracted 73,236 tables with PDFFigures and Tabula. These tables have 6.23 rows on average (SD = 6.58), and they have 7.11 columns (SD = 6.27). We converted the tables into RDF, resulting in a KB with 23M triples. We used Blazegraph to execute the SPARQL queries. After adding the table types and column types, we loaded the KB into VLog [Carral et al., 2019] to perform rule-based reasoning.

Tablepedia contains 451 tables, which have the columns annotated only with three classes: **Method**, **Dataset**, and **Metric**. To use this dataset in our pipeline, we created a graph representation of the tables without the annotations. Then, we translate the 15 *seed concepts* that are used in [Yu et al., 2020] to create the tables into labelling queries, so that we could apply Snorkel using both datasets. In contrast to Tablepedia, our annotated dataset maps to a much larger number of classes. Notice

that the most frequent column types in our dataset (**Observation**, **Accuracy**, and **Count**), do not occur in Tablepedia.

Training data To create the training data for weak supervision, two human annotators (one PhD and one bachelor CS student) wrote SPARQL queries for labeling with the aid of a web interface designed for this purpose. The annotators examined the results of these queries on a sample of 400 tables, ensuring that the queries represented heuristics that covered a reasonable amount of the data. The quality of the SPARQL queries is fundamental to produce a good training dataset, and hence return good predictions. It is crucial that the queries have *large coverage* to avoid introducing a bias and to increase the training data size. For instance, if the queries label only a few tables, then the model will not receive enough evidence. To this end, we encouraged them to write queries which also matched a large number of items on the entire set of tables, and that did not excessively overlap. This resulted in 39 queries for labeling 98,570 tables with the corresponding type and 55 queries for labeling 165,302 columns.

Gold standards To test the performance, the same human annotators as before manually annotated 400 random tables. The tables in this sample have, on average, 9.92 rows (SD 7.28) and 5.07 columns (SD 3.20). These tables were annotated with the number of header rows, and table and column types. This process resulted in 321 table type and 873 column type annotations (excluding **Other**). Most tables were annotated with the **Observation** class (258), followed by **Input** (50); the smallest class was **Example** (13). The human annotators have annotated the table and column types looking at the images of the tables, the table captions, and possibly the full paper in case it was still not clear. The annotators have annotated the tables independently and resolved the conflicts together whenever they disagreed. After the first round of annotation using the first version of the ontology (44 classes), we marked as infrequent all classes with fewer than 10 annotations. These classes were removed from the ontology and the annotations were redirected to **Other**. For the Tablepedia dataset, we used the annotations provided by the original authors.

Method	Accuracy
1^{st} Row	0.71
Ours	**0.76**

Table 5.1: Accuracy of header detection methods.

Table Type Model	Precision	Recall	F1	AUC
SVM	0.71	0.79	0.74	0.86
LR	0.72	0.79	0.74	0.84
NB	**0.80**	**0.82**	**0.79**	**0.91**

Table 5.2: Performance of semi-supervised trained models on table type prediction of our own gold standard with respect to different metrics.

We highlight two aspects of our gold standard that have a direct impact on the evaluation. First, in contrast to [Yu et al., 2020], we decided not to filter out tables that were incorrectly extracted by Tabula. This makes our corpus more challenging because it might contain errors due to incorrect parsing. Second, our choice of merging infrequent column types into the type **Other** ensures that for each type there is always some evidence, but it has the downside that some classes in the long tail are ignored. Interpreting such types is an additional challenge that deserves a thorough study in future work.

5.6.1 Table Interpretation

Table 5.1 reports the accuracy of our header detection heuristic compared to the baseline that consists of always selecting the 1^{st} row. We observe that our technique has superior performance, although it still makes some mistakes.

In Table 5.2, we report the performance of our table type detection models on our gold standard. In general, we observe that all three models return reasonably high performance. Naïve Bayes (NB) outperformed the others, especially in terms of F_1 and AUC. Thus, we decided to select this as the default one for this task.

In Table 5.3, we report the classifiers' performance for the column types on our gold standard, while Table 5.4 reports the same for Tablepedia. In both cases, we see that LR performs best, likely due to the combined importance of textual and

Model	Precision	Recall	F1	AUC
NB	0.52	0.48	0.47	**0.87**
SVM	**0.58**	**0.56**	**0.53**	0.83
LR	**0.58**	**0.56**	**0.53**	**0.85**

Table 5.3: Performance of semi-supervised trained models on column property prediction of our own gold standard with respect to different metrics.

Column Property Model	Precision	Recall	F1	AUC
Yu et al. [Yu et al., 2020]	0.82	0.81	0.81	0.90
NB	0.84	0.82	0.81	0.96
SVM	0.90	0.89	0.89	0.97
LR	**0.92**	**0.91**	**0.91**	**0.98**

Table 5.4: Performance of semi-supervised trained models on column property prediction of the Tablepedia data with respect to different metrics.

numeric features for this task. Additionally, we observe that our model significantly outperforms the model of [Yu et al., 2020] on their dataset. If we compare the scores between the two datasets, then we see that they are significantly lower with our dataset. The reason is two-fold: First, the authors of Tablepedia have manually removed much noise from the extracted tables while no pre-processing took place on our dataset. Second, our dataset contains many more classes than Tablepedia, which makes it more challenging to predict.

Finally, we studied the added value of using Snorkel and compared it with a simpler majority voting (MV), i.e., labeling a data point using the most frequently predicted class. In Table 5.5, we report both the accuracy obtained with MV and with Snorkel on different tasks. While Snorkel outperforms MV for the table type detection on our corpus and column type detection in Tablepedia, MV is better when detecting the column types of our corpus. This is disappointing, but can be explained. In

Task	MV	Snorkel
Table Types (Our corpus)	0.50	**0.71**
Column Types (Our corpus)	**0.56**	0.49
Column Types (Tablepedia)	0.39	**0.65**

Table 5.5: Accuracy of label query aggregation methods (Snorkel and majority voting (MV)) on different tasks.

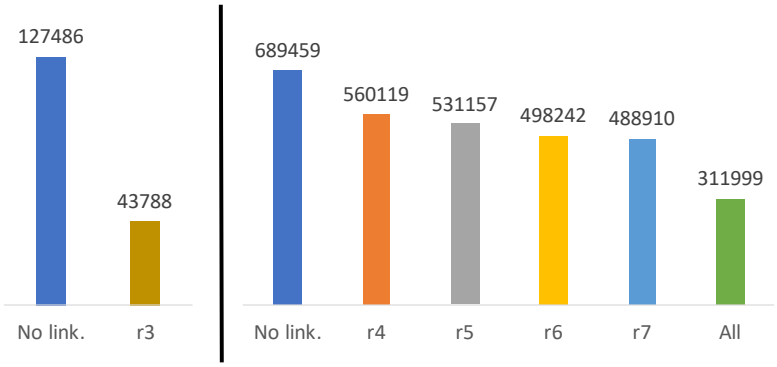


Figure 5.4: Ablation study. The bar marked with r_i reports the number of entities when only EGD r_i is included in the rule set

	Label	# Links
Good ✓	mnist	288
	knn	211
	wiki	146
	cifar-10	108
	en-es	65
Bad ✗	after	183
	analysis	66
	subset	49
	0/0/0	9
	f4(x)	6

Figure 5.5: Examples of good and bad entities labels, with their number of linked occurrences

the column type detection case, our labeling functions (i.e., SPARQL queries) have frequently abstained. Consequently, the labling matrix M has a low label density, and whenever this occurs, Snorkel is unable to compute optimal weights that diverge from MV [Ratner et al., 2020]. The solution to this shortcoming of our approach is to solicit additional labeling functions, in order to increase the coverage of annotations. However, to support users in effectively writing such functions, it is neccesary to provide them with adequate information about which items are insufficiently labeled. This requires a modification of the user interface and a procedure to identify such items with sufficient diversity, which is an interesting issue for future work to explore.

5.6.2 Entity Linking

Figure 5.4 reports the number of entities before and after the execution of the EGD rules. The left side compares the number of entities that refer to columns before and after r_3 was executed. As we can see, r_3 merged many entities, and this reduced the number of distinct entities of 65%. The right side shows the decrease of entities that refer to cells after the execution of rules $r_4, \ldots, r_7$. Here, the bar titled r_i reports the number of entities if only r_i is executed while the right-most column indicates the number of entities when all rules are included. We observe that every EGD contributes to merge some entities, but the best results are obtained when all EGDs are activated: here, the EGDs merged about 55% of the entities.

To evaluate the quality of entity links, we manually evaluated a sample of 100 merged entities. For each sampled entity, we first determined whether the entity was a meaningful one. From this analysis, we discovered that 65% of the entities are correct while the remaining have either nonsensical labels or some text resulted from errors of Tabula. In Figure 5.5, we report examples of good and bad entities with their number of links.

Then, we looked at the cells which referred to the entity, which were 541 in total. Since the rules could make a mistake and link two cells to the same entity although they meant different ones, we evaluated, for each entity, the precision of its links. Given the set of n cells that link to the same entity, the precision is computed by taking the cardinality of the largest subset of cells that refer to the same concept and divide it by n. For instance, consider an entity X with label Y which is linked to $n = 4$ cells. Three of these cells contain the text Y but refer to a dataset while one cell contains Y but refers to something else. In this case, the precision for X is $\frac{3}{4}$. In our sample, the average precision over the meaningful entities was about 97%, which is a relatively high value. This indicates that reasoning produced an accurate entity linking.

5.7 Conclusion

We presented Tab2Know, an end-to-end system for building a KB from the knowledge in scientific tables. One distinctive feature of Tab2Know is the usage of SPARQL queries for weak supervision to counter the lack of training data. Another distinctive feature is the usage of existentially quantified rules to link the entities without the help of a pre-existing KB.

Our pipeline effectively combines statistical-based classification and logical reasoning, exploiting SPARQL and remote KBs like Semantic Scholar. Therefore, we believe that ours is an excellent example of how semantic web technologies, statistical- and logic-based AI can be used side-by-side.

Future work Although our results are encouraging, and the current KB is already able to answer some non-trivial queries, future work is required to improve the performance. First, a more accurate table extraction procedure is needed to improve the accuracy of table interpretation and entity linking. Moreover, our current ontology links classes only via $\sqsubseteq$. It is interesting to study whether new relations can lead to better extractions. For instance, specifying the range of some classes could be used to exclude mappings to columns with incompatible values. Finally, a natural continuation of our work is to further research whether additional rules can return a better entity linking. In particular, we believe that rules that take into account the context of the table or co-authorship networks will be particularly useful.

To conclude, we believe that Tab2Know represents one more step that brings us closer to solve the problem of constructing an extensive and accurate KB of scientific knowledge. Such a KB is a useful asset for assisting the researchers, and it can play a crucial role in turning the vision of open science into a reality.

CHAPTER 6

A Platform for Web Table Information Extraction

In this chapter, we describe our work on a software library for supporting the design and analysis of table integration pipelines.

To extract knowledge from web tables, it is often useful to create data cleaning and semantic matching pipelines which leverage patterns from large table corpora. However, current approaches typically focus on only one part of the pipeline, and are hard to debug at the system level. To support research into the entire process of knowledge extraction from tables, we argue that it is necessary to move beyond solving subtasks, and evaluate pipelines on real-world data. We present Takco [1] (TAbles for Knowledge base COmpletion), an open-source platform designed for extracting novel facts from tables that can be added to Knowledge Bases such as Wikidata. It has a modular design that allows for analysis of every step of the pipeline, and can be flexibly used for both explorative work on a single laptop as well as large-scale work on a compute cluster. Additionally, it allows for the fine-grained analysis of the novelty of facts extracted from benchmark datasets and large-scale corpora. We describe the challenges raised by Takco, present preliminary analyses, and discuss its practical

[1] `https://takco.readthedocs.io/`

usage.

6.1 Introduction

Semantically integrating tables on the web into Knowledge Bases is a long-standing challenge in data management [Cafarella et al., 2018, Zhang and Balog, 2020]. However, the variety of real-world web tables is so wide that current approaches cannot help but make simplifying assumptions. Although such assumptions are necessary for constraining the scope of research, it creates the risk of narrowing the attention of the field to problems that are easy to define, while overlooking real-world phenomena that may give rise to interesting research challenges. We believe that letting go of several common assumptions about web tables will lead to productive new avenues of research, and the potential of making larger real-world impact. At the same time, focusing on a wider range of phenomena in practice creates the need for more emphasis on building end-to-end approaches that combine multiple subtasks in a modular fashion. However, creating and evaluating such systems takes much engineering effort, which currently impedes research into the practical applicability of semantic table integration approaches.

In this chapter, we therefore introduce Takco, a modular Python library for creating and analyzing pipelines for web table data integration. The main goal of our effort is to facilitate research and development into the extraction of novel facts from real-world web tables. Therefore, we have developed several modules that can be combined in a flexible way, according to the requirements of the project or the characteristics of the data in the domain of interest. These modules are designed to address a number of challenges we encountered when approaching this problem from the perspective of modern data science.

First, data science workflows typically feature a *development* and a *production* environment, where the development environment allows for the rapid prototyping of candidate solutions, and the production environment scales to large datasets. Current systems for knowledge extraction from web tables either require users to manually

adapt large KBs and datasets to their own environment, or have difficulty scaling up to realistic workloads. To support agile development, it is essential to instead allow users to explore solutions incrementally, making use of ad-hoc data labeling, pre-trained models and existing resources, while supporting the scalability of components in production.

Second, most web tables require *data cleaning* and *reshaping* in order to be suitable for semantic integration. Issues with data formatting, extraction artifacts, context, and table structure impede the ability of current approaches to extract meaningful information. However, such operations are heavily dependent on the data domain and data modeling assumptions that underpin the KB of interest. Therefore, users must have the ability to freely "patch" any pipeline with custom tailored modules or annotated data for their specific use-case, and explore the influence of different configurations on their extraction results.

Finally, there is an inherent trade-off in the table interpretation subtask between *novelty* and *confidence*. As we have shown in Chapter 3, the more similar some dataset is to information that is already known in the KB, the more confident most models are about the semantics of that dataset. Datasets with the potential to add mostly new facts are therefore harder to correctly interpret. This means that it is vital to tune web table integration systems for *generalization*. However, at present there exist no practical tools for assessing and analyzing how well current approaches generalize to novel information.

6.2 Design of Takco

To address the challenges mentioned before, we decided to spend some engineering effort to build a large-scale platform for table integration, called Takco. The development has been driven by the following desiderata:

1. *Generality.* We did not want to develop yet another tool that caters only to the needs of a single research lab, but a tool that can solve this problem in a wide variety of scenarios;

2. *Scalability.* Our system should be able to process a number of tables that is well beyond the size of current benchmarks. Ideally, we would like to support very large datasets, like all the tables contained in Wikipedia;

3. *Integration with existing tools.* The current data science landscape offers many high quality frameworks, libraries, and tools to ease the processing of tabular data. It makes no sense to reinvent the wheel. Thus, our goal was to develop a platform that can be integrated as seamless as possible in a typical data science pipeline;

4. *Usability.* Our system should be well-documented and should provide an implementation for basic operations. The goal is to avoid that new researchers spend considerable time on developing so that more time is available for research.

We (briefly) describe below how such desiderata have been addressed in our system. More information is available in the online documentation [2].

Generality In order to support table interpretation in a wide range of scenarios, we decided to take a KB-agnostic approach. By leveraging the RDFlib[3] API, our table interpretation module can flexibly support a wide range of back-ends, including SPARQL, Header-Dictionary Triples (HDT) [Fernández et al., 2013] and Trident[4]. This interface is integrated with the support of different label index back-ends, such as Elasticsearch[5] and SQlite[6] Full-Text Search.

We have also implemented an interface for flexibly supporting different schema blocking and matching techniques, with which we provide wrappers for Locality-Sensitive Hashing (LSH) [Gionis et al., 1999], Approximate Nearest Neighbors (ANN) using word embeddings [Pennington et al., 2014, Johnson et al., 2019], and a number of heuristics based on the table context, which optionally restrict the clustering based on table metadata such as web-domain or wiki-category. These modules are easy to

[2] `https://takco.readthedocs.io/`
[3] `https://rdflib.dev/`
[4] `https://github.com/karmaresearch/trident`
[5] `https://elastic.co/`
[6] `http://sqlite.org/`

configure, and provide a standard set of matcher aggregation functions and similarity functions.

Scalability For processing large datasets of millions of web tables, we designed Takco to be runnable on multi-core and multi-machine hardware, while keeping the modules themselves hardware-agnostic, so that the same functions can be used for exploration as well as large extraction applications. This was achieved by using the Dask[7] distributed computing library, which facilitates running Python code in parallel on multiple CPUs in one or multiple machines. Additionally, the Takco modules were designed to flexibly serialize and deserialize indexes, caches and intermediate results to and from the Hadoop Distributed File System (HDFS) [Borthakur and Others, 2008], which ensures that the data is physically close to where the computation happens.

Integration with existing tools Takco represents tables as Pandas[8] Dataframes, which is the most used Python library for data science. Consequently, it is easy for users to write components for "patching" existing pipelines to adjust them to changing requirements or new domains. By using the Dataframes API, industry-standard packages for Machine Learning and Data Cleaning can also be interwoven with Takco components. Additionally, we provide modules that load standard benchmark datasets into a common format, accounting for differences in serialization, annotation structure and table formats.

Usability By integrating with the Python data science ecosystem, Takco automatically gains many advantages from a usability perspective. Existing tools for interactive development and debugging, such as Jupyter notebooks[9], facilitate the exploration of data processing solutions in a tight loop of user feedback. We have also used Jupyter for writing tutorials and example pipelines which are part of the documentation of Takco. This makes it easier for researchers to get started with the library, and frees up their time for doing experiments.

[7] `dask.org/`
[8] `https://pandas.pydata.org/`
[9] `https://jupyter.org/`

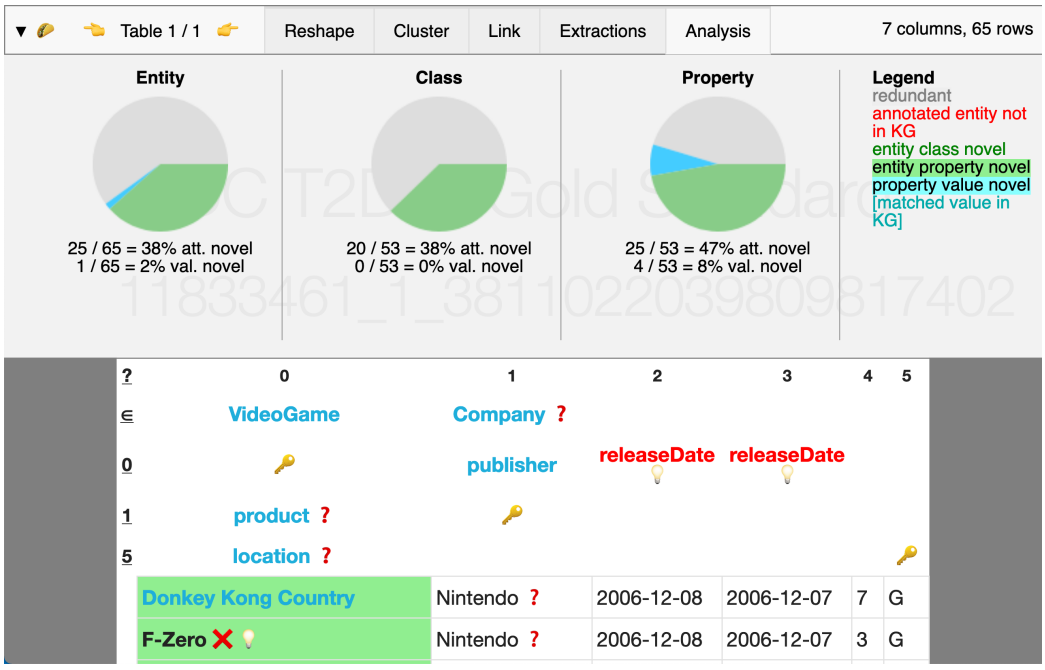


Figure 6.1: Measuring the novelty of extracted facts

Additionally, Takco provides a user interface for interactively exploring the parameter space of table integration pipelines. The user interface can be run in two ways: (1) as a browser bookmarklet that allows for the processing of web tables as they appear on web pages, (2) as a stand-alone web service for the processing of benchmark datasets. In both cases, users may step though the consecutive stages of a pipeline, select parameter values and configurations, and analyze their effect on the tables in question. For example, users may select a KB and inspect which information expressed in the table is novel to that KB (Figure 6.2). We will expand upon how the interface of Takco may be used by different users in the next section.

6.2.1 Target Users

The set of target users of Takco can be divided into three categories: *System developers*, *Knowledge base contributors*, and *Benchmark creators* (note that the categories are often overlapping).

System Developers Takco enables a rapid exploration of configurations, and this is useful for system developers to evaluate the impact of design decisions on extraction quality and novelty. For instance, it is possible to observe how the performance varies with different components in the extraction pipeline. Moreover, Takco shows which tables are easy or hard (Figure 6.1) and the performance in different contexts, such as tables of different sizes or with a different KB.

Knowledge Base Contributors One of the most obvious usage of Takco is to extract novel facts to be added to a KB. Moreover, Takco is useful for KB contributors because it shows how well a pipeline tuned for one dataset would perform on another dataset (Figure 6.2), which gives a hint about the completeness of the extraction.

Benchmark Creators With the growing popularity of table interpretation, it is fundamental to design better benchmark tools that can stress the systems in key areas. To this end, we argue that benchmark creators need fine-grained control over the knowledge gap in order to tune for desirable properties of competing systems, such as extraction novelty. To support our position, Takco can be used to show characteristics of existing or new benchmarks, which offer some insights into what makes novel facts from tables easy or difficult to extract (Figure 6.3).

6.3 Usage of Takco

In this section, we describe the main features of Takco. In particular, we focus on its ability to load multiple benchmark datasets and interface with various KBs. Then, we describe how it can be used either as an (novelty) analysis tool or as a platform to aid the development of new methods.

Datasets In recent years, several datasets have been created by multiple research groups for benchmarking table interpretation systems. These datasets consist of annotations of disambiguated entity links, ontology classes, and KB relations for different columns. Often, the datasets were created with different assumptions. For

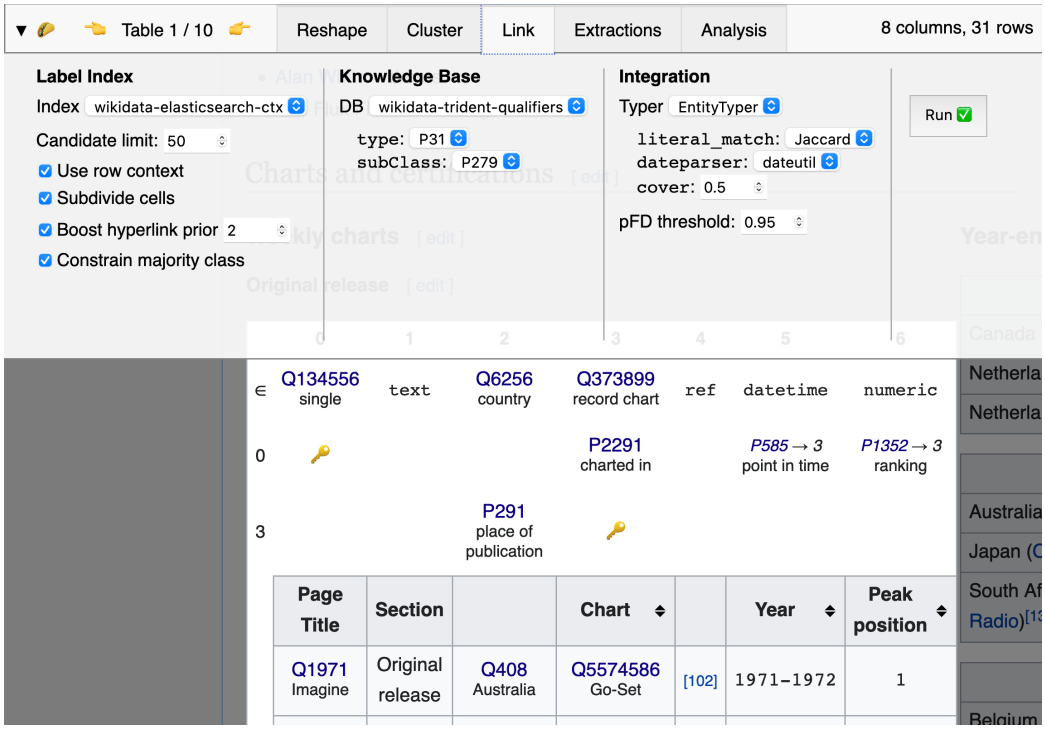


Figure 6.2: Fine tuning interpretation on Wikipedia tables

example, sometimes the tables are assumed to express relations only involving a certain key entity column, while in other datasets any pair of columns can express a relation. The same holds for the annotations of classes, which in the first case is available only for the key column. Additionally, these datasets were annotated using different KBs. Such a diverse landscape makes a fair comparison between systems a difficult task. In order to ease it, we have loaded into our tool the following datasets: T2D-v2 [Ritze et al., 2015], WWT [Limaye et al., 2010], WikipediaGS [Efthymiou et al., 2017], and the recent Tough Tables benchmark [Cutrona et al., 2020], which are the most commonly used in the field. By providing a single interface to these datasets, we were able to perform a more comprehensive study of the performance of multiple state-of-the-art approaches.

KBs Some engines were designed to work only for a certain KB. For example, T2KMatch makes use of a specifically tuned subset of DBpedia [Auer et al., 2007]. In contrast, Takco is fully KB-agnostic. To assess the ability of table interpretation systems to extract novel knowledge, it is crucial to study the knowledge gap between

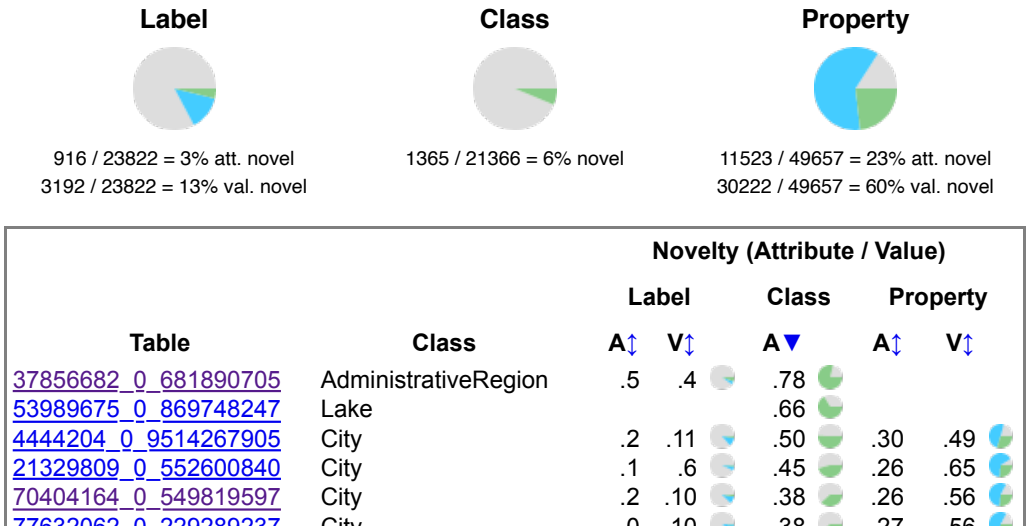


Figure 6.3: Analyzing the novelty of benchmark datasets

the KB and the table. To this end, we have loaded both the full and subsets of DBpedia, YAGO, Freebase and Wikidata in order to determine the impact of varying the KB and hence the knowledge gaps.

For DBpedia, we loaded the subset as used in T2KMatch and several past versions, including the current “live" distribution. Because most KBs include entity references to Wikipedia pages, we created a module that maps entity annotations from one KB to another. This allows us to evaluate entity disambiguation with respect to Wikidata, which is the largest KB but for which little annotated data exists.

6.3.1 Usage as an Analysis Tool

Our platform can be used to analyze the results of a generic extraction system. In particular, it offers a fine-grained analysis of the novelty, since extracting novel facts is one of the primary goals of our research. To perform this analysis, it is sufficient to return the annotations produced by the extraction system in a specific format that can be parsed by our tool.

As described above, gold-standard annotations and predictions for this task consist of three types: rows to entities, columns to classes, and column pairs to relations. Each of these annotation types result in fact extractions of entity *labels*,

class memberships, and *relations* respectively. For instance, imagine a table in a dataset with two columns, "Cities" and "Capital of", and one row with values ("A'dam","NL"). Moreover, suppose that the dataset contains a gold row to entity annotation which links the row to the KB entity `Amsterdam`, a column to class annotation which links "Cities" to `City`, and an annotation of a column pair to a relation which links "Capital of" to `capitalOf`. The first annotation can be used to extract the fact $\langle$`Amsterdam`, `hasLabel`, "*A'dam*"$\rangle$, the second extracts the fact $\langle$`Amsterdam`, `isA`, `City`$\rangle$, and the third $\langle$`Amsterdam`, `capitalOf`, `Netherlands`$\rangle$ (assuming that "NL" is also disambiguated to `Netherlands`).

Each of these facts can be novel or not. We make a distinction between two types of novelty. For *value novelty*, we consider any value (label, class, or relation value) to be novel that does not *match* an existing value in the KB, where the matching function can involve approximate comparisons of strings, numbers and other datatypes. For *attribute novelty*, we consider any value to be novel if the KB has no value at all for that entity and attribute. For instance, let us consider the fact $s = \langle$`Amsterdam`, `capitalOf`, `Netherlands`$\rangle$ and let $E = \{x \mid \langle$`Amsterdam`, `capitalOf`, $x\rangle \in KB\}$, i.e., E contains all the entities in the KB linked to `Amsterdam` with `capitalOf`. In this case, with the value novelty, we say that s is novel if no entity in E has a label that matches with "NL". With the attribute novelty, s is novel if E is empty.

Our tool aggregates the two types of novelty scores per table and dataset. For entity links, the aggregated score is the percentage of cells that do not match the label of the annotated entity in the KB. For classes, it is the percentage of entities that are not assigned in the KB to the class annotated for that column. For other relation annotations, it is the percentage of cells in the annotated column that contain a value that does not match (or exist) for that (subject entity, relation) pair in the KB.

The aggregated scores are shown in an overview page that can be used to analyze the various types of novelties. For instance, Figure 6.3 contains a screenshot of such a page considering the annotations in the gold standard of the dataset T2D-v2. Note that these scores can be computed either considering the annotations in the gold

standard or the annotations produced by an extraction system. Visualizing the novelty of the gold standard annotations is useful to determine how appropriate the benchmark is for completing the KB. In contrast, analyzing the annotations of an extraction system is useful to assess its ability to return new facts.

6.3.2 Usage as a Modular Platform

Takco can also be used as a platform that provides basic building blocks for developing a new table interpretation technique. The building blocks consist of some common operations that are typically performed during the extraction pipeline. The design of Takco allows for the fine-grained analysis of the different components that make up a KB completion pipeline. In particular, we have used it to develop our KB completion methods of Chapters 3 and 4.

For example, one of such building blocks can be used to retrieve a set of entity candidates for a given table cell based on string similarities. In this case, the choice of the pool of labels in the KB has a large impact on the recall. Previous methods have considered different sources of labels, such as anchor text of hyperlinks on Wikipedia. To test the performance with different label sets, we loaded the labels from the KB itself, from Wikipedia article titles, disambiguations and redirects, from anchor text of Wikipedia hyperlinks, and anchor text from the Web. Furthermore, the way the label index is queried is also influential since some good (or bad) candidates might be missed (or included). We have implemented several querying methods (e.g., approximate or exact string matching), and allow importing candidate sets from other systems, e.g., T2KMatch. This allows us to cleanly separate these dependencies in the pipeline, and analyze the impact of single components. For instance, Figure 6.1 shows a screenshot of the analysis tool that highlights which cell values in a table match values in the KB, based on one particular label set, along with the novelty scores.

Our platform reports matching scores for every candidate prediction, along with its novelty and correctness, along the entire integration pipeline. This shows when and where the pipeline discards predictions that are correct but have low support due to insufficient matches. Tracking this information is crucial to maximizing the amount

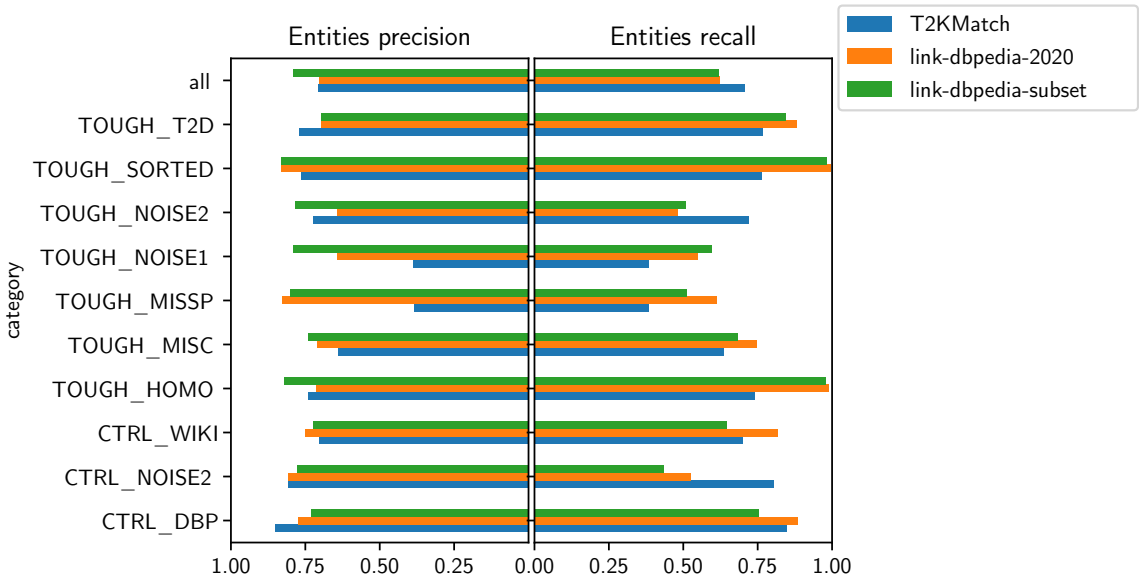


Figure 6.4: Entity Linking performance on Tough Tables benchmark subsets, for T2KMatch and two Takco baseline label search indexes.

of extracted novel knowledge.

6.4 Evaluation Results

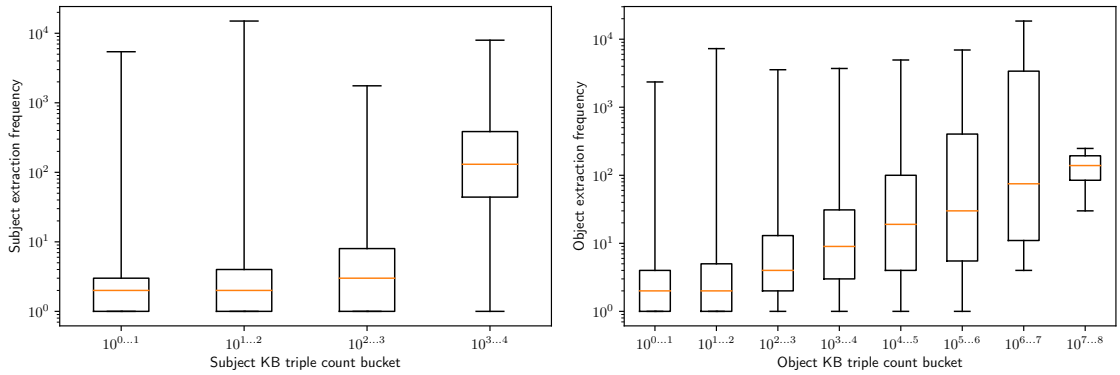
In this section, we provide a brief evaluation of Takco, illustrating several of its functions and their potential for supporting research into the integration of large volumes of diverse web table data with KBs.

Entity Linking In Figure 6.4, we show an entity linking performance comparison of two baseline label index configurations from Takco with the performance of T2KMatch [Ritze et al., 2015] on the recently introduced Tough Tables benchmark [Cutrona et al., 2020]. This benchmark contains tables from various sources, including artificially generated from DBpedia. The annotations are based on DBpedia version 2016-10. The tables are selected and created for the explicit purpose of identifying strengths and weaknesses in table interpretation methods. This is achieved by comparing performance over different subsets of the benchmark dataset, which represent different challenges for such systems, such as ambiguous or misspelled entity names.

The Takco label search indexes that we used are Elasticsearch instances with custom query functions that combine several matching scores. The index contains one document for each entity, which contains the labels of all its property values, as well as a set of weighted surface forms that are derived from hyperlink anchor text frequencies on Wikipedia. For each row in the table, the entity relevance is calculated by combining two matching scores: (1) the weighted score of surface form that best matches the cell in the key column, and (2) the similarity between the non-key cells and the entity's property values. The two indexes we evaluated both function in the same way, but one is based on the entities in the DBpedia subset of T2KMatch, while the other is based on the 2020-10 version of DBpedia. Note that in contrast to our work in Chapter 3, we only select the top-1 query result if present, and do not perform any further ranking of candidates.

From these results, we can identify several strengths and weaknesses of our baselines compared to T2KMatch. Comparing the baselines, we see that the DBpedia-2020 index often achieves higher recall than the subset, though in some cases at the price of precision. For the NOISE2 subsets of the benchmark, which are based on random character changes in cells, T2KMatch outperforms our baselines, which do not support character-based fuzzy string matching. However, the baselines do perform well on the NOISE1 and MISSP subsets, which is based on realistic typos and misspellings. Further, the baselines achieve high recall on the HOMO subset, which contains ambiguous entity labels, also when it is adversarially sorted (SORTED). Finally, the strong precision of T2KMatch on the T2D and DBP subsets illustrates that its has strong in-domain performance.

Long Tail Extractions Figure 6.5 is based on an analysis of triples that were extracted from 100k Wikipedia tables by an integration pipeline similar to the one described in Chapter 4. For each entity, we count how often it occurs in the subject and object positions of extracted triples and triples in the KB. To display the relation between these numbers, we group the entities in exponentially distributed buckets by frequency (i.e. sets of entities with counts in the KB between exponentially growing



(a) Extractions by subject count.

(b) Extractions by object count.

Figure 6.5: Fact extraction frequency per entity on a sample of 100k Wikipedia tables, by Wikidata triple count in exponentially sized buckets. For each bucket (range of $10^i \dots 10^{i+1}$) of triple counts, we count how often entities in the subject (or object) role occur in the same role in extracted facts. The boxes represent the interquartile range, the orange line denotes the median, and the whiskers show the minimum and maximum values. Entities that are well-described by the KB are also generally well-represented in extractions.

intervals), and show box-plots of extraction counts per bucket. We can see that overall, there is a connection between the number of triples that are extracted for some subject entity, and the number of triples in the KB with that entity as subject (i.e. its number of property-value pairs). The same holds largely for the object position. In general, this is an indication of how a matching-based interpretation approach may reinforce the bias of a KB towards certain topics, by extracting facts about popular topics that are already extensively covered in a "rich-get-richer" dynamic. However, the spread of frequencies is very large, especially at the bottom: many entities that are well-described by the KB do not occur in extractions frequently at all.

6.5 Conclusion

In this chapter, we introduced Takco, a modular library for designing and analyzing web table integration pipelines. It allows for the flexible specification of consecutive processing stages, which can be easily configured and inspected.

Takco was designed to be general, scalable, integrated with existing tools, and easy to use. We have addressed these goals through incorporating best practices from the Python data science ecosystem. We have leveraged various open-source libraries to make Takco KB-agnostic, efficient to run on distributed hardware or multiple CPUs, and easy to extend with patches and custom functions.

We have discussed how its user interface can be used both as an an analysis tool, providing insight into the novelty of extractions from large corpora or benchmark datasets, and also as an exploration tool for comparing the effect of different configurations and designs of table integration pipelines. This may be useful for system developers, who may compare the impact of different label indexes, KBs, or pre-processing configurations, for KB contributors, who may explore the optimal settings for table integration in their domain, and for benchmark creators, who may evaluate the difficulty of data subsets in different tasks.

Finally, we presented some preliminary results of running Takco components on benchmark data and a large corpus. We have found that our baseline label indexes may achieve competitive performance on a recent benchmark with little fine-tuning, with reasonable results across varied and adversarially constructed tables. Additionally, we have shown that there is a correlation between how often information about an entity is extracted, and how much is already known about that entity. This may cause a bias in extraction pipelines to expand the coverage of the KB within topics that are already extensively covered. By identifying situations in which this bias occurs, we hope Takco may be of use in developing solutions for counteracting it.

CHAPTER 7

Conclusion

In this thesis, we have attempted to address one fundamental research problem:

> *How do we automatically integrate a diverse set of tables into a coherent, machine-readable format?*

We broke down this problem into several challenges, and introduced methods to overcome them. In Section 7.1, we will discuss each of these challenges in turn, and highlight how our contributions have addressed the fundamental problem and advanced the state-of-the-art. In Section 7.2, we will reflect upon the work presented in this thesis and share some lessons learned.

7.1 Main Contributions and Future Work

7.1.1 Extracting Novel Facts

In Chapter 3, we described a trade-off that arises when developing a system which uses a KB for interpreting tables with the goal of extending that same KB. We showed that information that is already in the KB can be extracted from tables with high

precision and recall, while the novel information expressed in tables, which is most interesting for KB extension, is much harder to correctly interpret. This led us to formulate the research question:

> **Research Question 1:** *How do we measure and account for the trade-off between accuracy and coverage improvement in table interpretation?*

Contributions

We attempted to answer this research question on two fronts. First, we evaluated two state-of-the-art techniques for table interpretation [Ritze et al., 2015, Zhang, 2017] and analyzed the amount of novel extractions using two new metrics (Section 3.2). We observed that these techniques are very accurate at interpreting tables that mostly express information that is already in the KB, and from which little novel knowledge can thus be extracted. We additionally showed that fewer novel facts than redundant facts could be extracted with these techniques from benchmark datasets.

These results motivated us to develop a new method for table interpretation that was less reliant on matches between the table and the KB. We introduced a new technique for novelty-oriented table interpretation based on a scalable graphical model (Section 3.3). The model enforces coherence within a column between the set of predicted entity links, by relying on context-dependent entity similarities instead of a single pre-defined property such as their class. In our experiments, this technique attained higher recall in the row-entity annotation task than existing approaches on two benchmark datasets, and it was additionally able to correctly extract more novel facts (Section 3.4). Our approach also allowed us to compare the usage of different KBs for interpretation, through which we found that using the full DBpedia KB outperformed the subset of [Ritze et al., 2015], but that using the larger Wikidata KB did not improve performance. These results indicate that although our method is able to achieve a good trade-off between confidence and novelty for fact extraction, it is still important to consider which KB to use when interpreting tables for any benchmark under consideration.

Finally, we developed and evaluated a method to disambiguate cell values in the slot-filling task (Section 3.3.4). Using the predicted properties and subject entities from the interpretation step, this method ranked object entity candidates by scoring triples with a KB embedding model. We found that this method improved over a text-based baseline, allowing us to identify the correct slot-filling candidates with reasonable accuracy without relying on explicit KB matches that might be biased in favor of redundant extractions.

Limitations and Future Work

Interpretation Features In Section 3.3 we introduced a method that relies less on matches between information expressed in the table and the KB, but instead on a novel signal, namely the coherence between entity predictions within one column. Many other features may additionally be used for table interpretation that also represent less direct, more fuzzy associations between the table and KB, and have been researched previously [Limaye et al., 2010, Ritze and Bizer, 2017, Muñoz et al., 2014]. However, it has not been evaluated whether these features increase or mitigate the bias towards confident predictions on redundant data, nor what effect they have on the novelty of extractions. Extending our coherence-based prediction signal using these features is a direction for future work which has the potential to further increase the extraction accuracy of novel facts.

Knowledge Fusion In the experiment described in Section 3.3.4, we rank the object candidates of our extracted facts using a simple KB completion model based on embeddings, but there is much potential for improving this step. In recent years there has been an explosion of research on embedding-based KB link prediction models (see [Cai et al., 2018] for a survey), but few of them have been evaluated as part of information extraction systems. Future research should determine their applicability in this context. Such KB embedding models likewise play an increasingly important role in the the challenging related problem of *knowledge fusion* [Dong et al., 2014b], in which a single correct value must be found for slot-filling among multiple conflicting

values. However, in this case temporal meta-information is essential for assessing the correctness of such values [Dong et al., 2015, Oulabi et al., 2016], and thus would require an embedding model that accounts for temporal n-ary facts (e.g. [Guan et al., 2019, Goel et al., 2020]). Significant future research is needed to evaluate these complex KB embedding models in a knowledge fusion setting.

7.1.2 Integrating N-ary Tables

In Chapter 4, we identified a number of popular simplifying assumptions about the structure of web tables that often do not hold in practice. We argued that these assumptions impede the extraction of *n-ary* facts from diverse real-world tables, which motivated us to address the following question:

> **Research Question 2:** *How can we extract n-ary facts from diverse real-world web tables and integrate them with a KB?*

Contributions

To address this research question, we focused on tables from Wikipedia. This focus allowed us to develop and evaluate a pipeline for large-scale table-to-KB data integration, which consisted of several stages (Section 4.3). The first stage reshapes tables using heuristic transformations, with the goal of reducing the diversity of table layouts. The second stage creates so-called union tables by clustering similar tables together, to overcome the diversity of table contents. The third stage links the entities of these union tables to the KB. The last stage relies on functional dependencies to identify matches between table rows and n-ary facts in the KB, extracting such facts with high precision.

We evaluated the performance of this pipeline by examining the operation of each stage separately, as well as by analyzing the resulting extractions from the whole pipeline (Section 4.4). We found that our heuristics for detecting header cells on which to unpivot tables returned satisfactory performance on a sample of manually annotated tables. Individually, some heuristics attained high precision, but they needed to be

combined in order to achieve acceptable recall. We evaluated the clustering stage on data that was annotated with gold-standard column correspondences. Our results show that the similarity metrics which we formulated for matching the headers and bodies of tables allow the clustering algorithm to accurately create union tables that express coherent relations. On the gold-standard union tables, we also evaluated our key column prediction method, which accurately distinguishes between entity-attribute tables and n-ary tables and outperforms baselines used in related work. We evaluated the final n-ary KB integration stage by manually checking the Wikidata properties and qualifiers that our method predicted for table columns. At 65% precision for binary relations and 92% precision for n-ary relations, we estimate that the full pipeline could extract over 22 million novel facts for expanding the coverage of Wikidata from 1.5 million Wikipedia tables. We also showed that these facts very frequently involved the table context, and could more often be extracted from union tables than individual tables. These results indicate that performing context enrichment and schema matching on web tables can improve KB integration performance, compared to interpreting tables in isolation.

Limitations and Future Work

Structure Understanding We argued in Section 4.3.1 that many real-world tables are structured in such a way that reshaping them is beneficial to extracting n-ary facts, in particular with respect to values in their headers. However, we have only brushed the surface in terms of identifying these cases automatically. Due to the observed variety and the template-like occurrence of these cases within different domains, we used a set of general heuristics for detecting pivoted headers, but there are still many cases that the heuristics do not cover. Alternative approaches to understanding such table structures, e.g. the machine learning model of [Ghasemi-Gol et al., 2019], rely on annotated data which is expensive to acquire across many domain of interest. Therefore, a promising future research direction would be to leverage heuristics such as ours for weak supervision, in order to effectively train a model to detect and transform complex table structures in real-world tables.

Heterogeneous Table Context In our research, we make use of the table context to extract a limited set of values that we add as extra columns, such as the page title and table caption. However, there are potentially many more sources of contextual information that may be vital for correctly extracting information from the tables, such as temporal meta-information [Oulabi and Bizer, 2017]. Previous work has extracted "hidden attributes" from template-like repeating structures in the table context [Ling et al., 2013], but for merging tables from very heterogeneous sources such attributes would have to be aligned. Interesting future work would thus consist of matching attribute and value strings across table extraction sources in order to correctly integrate such contextual information.

7.1.3 Creating a New KB from Tables

Whereas our first two research questions concerned the extension of an existing KB, we focused on creating a new KB from scratch in Chapter 5. In this situation, human effort is needed to create the initial data for training a model that can be used to interpret tables. However, labeling individual training data points is inflexible and often prohibitively expensive. To overcome this obstacle, we formulated the following question:

> **Research Question 3:** *How can we build a coherent KB from tables on a new domain, with minimal human effort?*

Contributions

Our work on answering this question focused on tables from scientific papers. We developed a system called Tab2Know, which addressed the challenge of automatically interpreting these tables using a purpose-built ontology, and of disambiguating the entities that they contain.

Our pipeline employed both statistical classifiers and logical reasoning. First, we created an RDF representation of the table structures extracted from PDFs and their scholarly metadata (such as authors and venues), and combined it with an ontology

about concepts in the domain of interest. Using a set of labeling functions implemented as SPARQL queries over this data, we trained a set of weakly supervised classifiers to predict the type of tables and columns (Section 5.4). Then, we employed a scalable reasoning engine to link equivalent entities in different tables (Section 5.5). Using a set of rules that described the existence and equivalence of entities which made use of the statistical predictions, we were able to match a large number of entities in the KB.

Our empirical evaluation of this approach, using a corpus of recent papers from several AI venues, gave encouraging results (Section 5.6). On a sample of these tables which were manually annotated to create a gold standard, we achieved reasonable performance with light-weight statistical models on the table interpretation task, while outperforming related work on an existing, simpler benchmark dataset. An examination of our entity links suggested that their quality is promising, and that all rules contribute to the high number of matches found. The combination of weakly supervised interpretation models and rule-based reasoning was conducive to the low manual effort needed to create this KB, while the ontology contributed to its coherence.

Limitations and Future Work

Complex Ontologies The ontology that we used for table interpretation in Section 5.4.1 consisted of a shallow taxonomy of classes related to the concepts we encountered in a sample of tables from our corpus. One promising avenue of future work is the creation of a richer ontology, expressing more relations between classes such as disjointness. Similarly, information about the range and domain of properties in the ontology could boost the precision of our table interpretation models. It is an interesting open question for future research how to incorporate such ontological knowledge into the training of weakly supervised statistical models.

Expressive Labeling Functions In Section 5.4.1, we demonstrated the potential of using the SPARQL query language for writing labeling functions for weak supervision, which opens up many possibilities for future research in this domain. In particular, it would be interesting to incorporate more information from the table context into these

queries, such as interactions between venues, keywords in paper abstracts, citation links and author information. A further avenue for future research would be to incorporate more background information into the KB of extracted structures that is used for data labeling, such as existing links to external KBs from paper metadata or predicted topics.

Robust PDF Table Extraction One limitation of our pipeline is its reliance on a baseline table extraction pre-processing step, which introduces noise into the dataset, in particular for tables with complex structure. There has been much recent research into table extraction methods using deep neural networks, which may significantly improve the data extraction quality if they are able to generalize sufficiently across domains. Additionally, future research may use table interpretation techniques to provide a supervision signal for training such extraction models, e.g. by preventing them from splitting cells that contain entity names.

7.1.4 Designing Pipelines

In Chapter 6, we turned our attention to the application of table integration techniques in real-world settings. Here, the development and evaluation of systems is vital. To understand the challenges of integrating data from diverse sources and domains, researchers and developers must have the tools to inspect and debug each system component in practice. Therefore, we posed the following question:

> **Research Question 4:** *How can we support research into table extraction and integration pipelines on realistic data?*

Contributions

To address this challenge, we developed a modular Python library called Takco. Its design allows for the loose coupling of various stages in table integration pipelines, and enables users to analyze and tune these stages for their use-case. That way, researchers

and practitioners have the flexibility to adapt and re-configure existing solutions to their needs, and evaluate them on new domains.

In Section 6.3, we have detailed how it may support research and development into KB extension from web tables on several fronts. As an analysis tool, it can be used to provide insight into the novelty of individual extractions, and to explore the set of features that were used to link entities, classes or properties. Additionally, it can be used to evaluate the extraction novelty aggregated per table or benchmark dataset, which allows for the identification of challenging data subsets, or the comparison of benchmarks on their potential for evaluating systems on the task of KB extension.

In Section 6.3, we also described several scenarios in which Takco could be used by researchers and practitioners. We highlighted its usefulness for system developers, who may compare the impact of different label indexes, KBs, or pre-processing configurations, for KB contributors, who may explore the optimal settings for table integration in their domain, and for benchmark creators, who may evaluate the difficulty of data subsets in different tasks. Finally, we illustrated the usage of Takco on examples of various domains in Section 6.4, which highlighted interesting properties of real-world data that open up new research directions.

Limitations and Future Work

N-ary Interpretation Models Like the existing methods we discuss in Chapter 3, the n-ary table interpretation module of Takco relies on factual overlap between the KB and table. Though we are nonetheless able to extract novel facts due to the schema matching and stitching step, there is still much potential for exploring n-ary interpretation methods that may be less sensitive to the bias that we described in Chapter 3. One potential approach is to exploit co-occurrence statistics between datatypes and qualifiers. Intuitively, we expect that given a set of predicted column classes such as $\{\texttt{Politician}, \texttt{Electoral District}, \texttt{Election}, \texttt{Percentage}\}$, our common sense would tell us that the table expresses a relation that has something to do with votes. Future work may improve upon our interpretation approach by leveraging such statistics for interpreting n-ary tables without relying on KB matches, thereby extracting

more novel n-ary facts.

Large-scale Evaluation Though Takco has returned satisfactory results on samples of data from different domains, it is an open question which configurations are most suited for large-scale web table integration. However, it is not straightforward to find such optimal configurations in the large combinatorial space of parameter values. One possible starting point is the running of ablation studies, in which the effect of removing entire modules is measured on the integration quality for different sources of web tables. However, this requires a large amount of manual labeling of extracted facts, and does not provide insight into the way the different modules may interact in various settings. Alternatively, future research may focus on finding configurations that are good enough for practical impact, by working with KB contributors to judge and filter extractions for extending a real-world KB.

7.2 Final Reflections

In this final section, we will reflect upon several aspects of doing research in this field, and share some lessons learned from the work we have done.

Benchmarks and Runnable Systems The research described in this thesis could not have been done without the tremendous effort of other researchers to create benchmark datasets, and open-source code with which to reproduce their efforts. It is hard to overstate the importance of shared resources to fine-grained comparisons of methods such as those in Chapters 3 and 5. For the task of table interpretation, small variations in implementation can make all the difference. While research papers can only fit a limited amount of information, the data and code provide the bedrock on which further research can be built. It is therefore vital that the community values the clear documentation of seemingly trivial details, such as table pre-processing operations, the exact version of the KB used both for annotation and interpretation, the exact set of entity labels used and the configuration of the search index in which they are stored, and handling of edge-cases. Far from being implementation details,

these issues are central to the benchmarking effort. In this way, the systems that we compared against have set an example of open research worthy of following.

Heuristics and Learning In recent years, there has been a trend of applying progressively more complex machine learning models in virtually every aspect of software engineering. To train these models, this trend has been accompanied by the need for ever larger labeled datasets, which are expensive to obtain. This dynamic leads to inflexible models which need to be re-trained on new data when requirements change, and whose predictions are often hard to explain or audit. In contrast, rules and ontologies written by humans encode models that can be flexibly adapted to changing requirements, and can be transparently inspected and understood. The techniques introduced in this thesis do not require manually labeled training data, nor are they based only on the straightforward application of rules. Instead, our approach has been to leverage precisely specified knowledge to make approximate predictions. In Chapter 3, this was done through a graphical model and KB embeddings, in Chapter 4 through heuristics and unsupervised clustering, and in Chapter 5 through weak supervision and reasoning. Perhaps the last case most clearly shows how statistical and logical models may work together: by manually specifying labeling functions and rules, we retain a level of control over the method that we would not have had otherwise, while still being able to accurately process a wide variety of input data. We expect such systems to thrive in knowledge-intensive applications in the future, and that research on this topic will have much practical impact.

Real Data Throughout all research presented in this thesis, we have focused on working with data from real-world sources. We have attempted to develop techniques that deal with phenomena that appear in actual tables on the web and in research papers, motivated by a close look at the data itself. Though this can be overwhelming when confronted with a flood of niche issues that are hard to capture in a single scientific model, it never fails to open up new research avenues. Indeed, it is the amazing variety of the information in these corpora that may inspire the development

of novel data integration methods. By taking on the challenge of processing a wider range of realistic phenomena, we believe that research on web tables can not only expand KBs with new facts, but also answer fundamental questions about how people represent information.

Bibliography

Mina Abd Nikooie Pour, Alsayed Algergawy, Reihaneh Amini, Daniel Faria, Irini Fundulaki, Ian Harrow, Sven Hertling, Ernesto Jimenez-Ruiz, Clement Jonquet, Naouel Karam, and Others. Results of the Ontology Alignment Evaluation Initiative 2020. In *15th International Workshop on Ontology Matching*, pages 92–138. CEUR Workshop proceedings, 2020.

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundation of Databases*, volume 1. 2015.

Alexander Albrecht and Felix Naumann. Schema decryption for large extract-transform-load systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7532 LNCS:116–125, 2012.

Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu A Ha, Rodney Michael Kinney, Sebastian Kohlmeier, Kyle Lo, Tyler C Murray, Hsu-Han Ooi, Matthew E Peters, Joanna L Power, Sam Skjonsberg, Lucy Lu Wang, Christopher Wilhelm, Zheng Yuan, Madeleine van Zuylen, and Oren Etzioni. Construction of the Literature Graph in Semantic Scholar. In *Proceedings of NAACL-HLT*, 2018.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a Web of open data. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4825 LNCS, pages 722–735, 2007. doi: 10.1007/978-3-540-76298-0_52.

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open Information Extraction from the Web. In *Proceedings of IJCAI*, pages 2670–2676, 2007. doi: 10.1145/1409360.1409378.

Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM Computing Surveys (CSUR)*, 41(3):16:1–16:52, 2009. doi: 10.1145/1541880.1541883.

Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. *Proceedings of ACM PODS*, 2017.

Francine Berman, Rob Rutenbar, Brent Hailpern, Henrik Christensen, Susan Davidson, Deborah Estrin, Michael Franklin, Margaret Martonosi, Padma Raghavan, Victoria Stodden, and Others. Realizing the potential of data science. *Communications of the ACM*, 61(4):67–72, 2018.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Methods for exploring and mining tables on Wikipedia. In *Proceedings of IDEA @ SIGKDD*, pages 18–26, 2013. doi: 10.1145/2501511.2501516.

Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. TabEL: entity linking in web tables. In *Proceedings of ISWC*, pages 425–441. Springer, 2015. doi: 10.1007/978-3-319-25007-6_25.

Harvey Bingham. CALS table model document type definition. Technical Memorandum, 1995.

Christopher M Bishop. *Pattern recognition and machine learning*, volume 4. 2006. doi: 10.1117/1.2819119.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

Christoph Böhm, Gerard de Melo, Felix Naumann, and Gerhard Weikum. LINDA: distributed web-of-data-scale entity matching. In *Proceedings of CIKM*, pages 2104–2108, 2012.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase : A Collaboratively Created Graph Database For Structuring Human Knowledge. In *Proceedings of ACM SIGMOD*, pages 1247–1249, 2008.

Antoine Bordes, Nicolas Usunier, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-Relational Data. In *Proceedings of NIPS*, volume 26, pages 2787–2795, 2013.

Dhruba Borthakur and Others. HDFS architecture guide. *Hadoop Apache Project*, 53 (1-13):2, 2008.

Katrin Braunschweig. *Recovering the Semantics of Tabular Web Data*. Phd thesis, 2015.

Katrin Braunschweig, Julian Eberius, Maik Thiele, and Wolfgang Lehner. OPEN—Enabling Non-expert Users to Extract, Integrate, and Analyze Open Data. *Datenbank-Spektrum*, 12(2):121–130, 2012. doi: 10.1007/s13222-012-0091-9.

Katrin Braunschweig, Maik Thiele, Julian Eberius, and Wolfgang Lehner. Column-specific context extraction for web tables. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1072–1077. ACM, 2015a.

Katrin Braunschweig, Maik Thiele, and Wolfgang Lehner. From web tables to concepts: A semantic normalization approach. In *Proceedings of ER*, pages 247–260. Springer, 2015b. doi: 10.1007/978-3-319-25264-3.

Michael J Cafarella and Alon Halevy. Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101, 2009. doi: 10.14778/1687627.1687750.

Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. WebTables: Exploring the Power of Tables on the Web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008. doi: 10.14778/1453856.1453916.

Michael J Cafarella, Alon Halevy, Hongrae Lee, Jayant Madhavan, Cong Yu, Daisy Zhe Wang, and Eugene Wu. Ten years of webtables. *Proceedings of the VLDB Endowment*, 11(12):2140–2149, 2018. doi: 10.14778/3229863.3240492.

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

Diego Calvanese, B Cogrel, Sarah Komla-Ebri, R Kontchakov, Davide Lanti, M Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8:471–487, 2017.

Matteo Cannaviccio, Lorenzo Ariemma, Denilson Barbosa, and Paolo Merialdo. Leveraging Wikipedia Table Schemas for Knowledge Graph Augmentation. In *Proceedings of the 21st International Workshop on the Web and Databases*, number July, page 5. ACM, 2018a. doi: 10.1145/2932194.2932203.

Matteo Cannaviccio, Denilson Barbosa, and Paolo Merialdo. Towards Annotating Relational Data on the Web with Language Models. In *Proceedings of WWW*, pages 1307–1316, 2018b. doi: 10.1145/3178876.3186029.

Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. *Proceedings of ACM SIGMOD*, 2020.

Andrew Carlson, Justin Betteridge, and Bryan Kisiel. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of AAAI*, pages 1306–1313, 2010. doi: 10.1002/ajp.20927.

David Carral, Irina Dragoste, Larry González, Ceriel Jacobs, Markus Krötzsch, and Jacopo Urbani. VLog: A Rule Engine for Knowledge Graphs. In *Proceedings of ISWC*, pages 19–35, 2019.

Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. Aurum: a data discovery system. In *Proceedings of ICDE*, pages 1001–1012, 2018. doi: 10.1109/ICDE.2018.00094.

Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. Dataset search: a survey. *The VLDB Journal*, 29(1):251–272, 2020. doi: 10.1007/s00778-019-00564-x.

Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of ICDE*, volume 2006, page 5, 2006. doi: 10.1109/ICDE.2006.9.

Jiaoyan Chen, Ernesto Jimenez-Ruiz, Ian Horrocks, Charles A Sutton, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles A Sutton. ColNet: Embedding the Semantics of Web Tables for Column Type Prediction. In *AAAI*, 2019.

Zhe Chen and Michael J Cafarella. Automatic Web Spreadsheet Data Extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, page 1. ACM, 2013.

Fernando Chirigati and Jialu Liu. Knowledge Exploration Using Tables on the Web. *Proceedings of the VLDB Endowment*, 10(3):193–204, 2016. doi: 10.14778/3021924.3021935.

Sagnik Ray Choudhury, P Mitra, and C Lee Giles. Automatic Extraction of Figures from Scholarly Documents. *Proceedings of the 2015 ACM Symposium on Document Engineering*, 2015.

Peter Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *Proceedings of TKDE*, 24(9):1537–1555, 2012.

Christina Christodoulakis, Eric B Munson, Moshe Gabel, Angela Demke Brown, and Renée J Miller. Pytheas: pattern-based table discovery in CSV files. *Proceedings of the VLDB Endowment*, 13(12):2075–2089, 2020.

Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.*, 53(6), dec 2020. doi: 10.1145/3418896.

Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. TEGRA: Table Extraction by Global Record Alignment. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015.

Christopher Clark and S Divvala. PDFFigures 2.0: Mining figures from research papers. *Proceedings of JCDL*, pages 143–152, 2016.

Edgar F Codd. A relational model of data for large shared data banks. *M.D. computing : computers in medical practice*, 15 3:162–166, 1970.

Eric Crestan and Patrick Pantel. Web-scale table census and classification. In *Proceedings of WSDM*, pages 545–554. ACM, 2011. doi: 10.1145/1935826.1935904.

Vincenzo Cutrona, Federico Bianchi, Ernesto Jiménez-Ruiz, and Matteo Palmonari. Tough Tables: Carefully Evaluating Entity Linking for Tabular Data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12507 LNCS:328–343, 2020. doi: 10.1007/978-3-030-62466-8_21.

Sanjib Das, Paul Suganthan G C, AnHai Doan, J Naughton, Ganesh Krishnan, Rohit Deep, E Arcaute, V Raghavendra, and Y Park. Falcon: Scaling Up Hands-Off Crowdsourced Entity Matching to Build Cloud Services. *Proceedings of ACM SIGMOD*, 2017.

Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of ACM SIGMOD*, page 817, 2012. doi: 10.1145/2213836.2213962.

Dong Deng, Raul Castro, Fernandez Ziawasch, Abedjan Sibo, Ahmed Elmagarmid, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. The Data Civilizer System. In *Proceedings of CIDR*, 2017.

Li Deng, Shuo Zhang, and Krisztian Balog. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *Proceedings of ACM SIGIR*, SIGIR '19, 2019.

Dennis Diefenbach, V López, K Singh, and P Maret. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information Systems*, 55:529–569, 2017.

Laura Dietz, Alexander Kotov, and E Meij. Utilizing Knowledge Graphs for Text-Centric Information Retrieval. *Proceedings of ACM SIGIR*, 2018.

Hong-Hai Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the VLDB Endowment*, pages 610–621, 2002. doi: 10.1016/B978-155860869-6/50060-3.

AnHai Doan, Alon Halevy, and Zachary G Ives. *Principles of Data Integration*. 2012.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *Proceedings of ACM SIGKDD*, pages 601–610, 2014a. doi: 10.1145/2623330.2623623.

Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *Proceedings of the VLDB Endowment*, 7(10):881–892, 2014b. doi: 10.14778/2732951.2732962.

Xin Luna Dong, Wang-Chiew Tan, and Winston Churchill. A time machine for information: Looking back to look forward. *Proceedings of the VLDB Endowment*, 8(12):2044–2045, 2015.

Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. *arXiv*, 2020.

Zlatan Dragisic, V Ivanova, P Lambrix, Daniel Faria, Ernesto Jiménez-Ruiz, and Catia Pesquita. User Validation in Ontology Alignment. In *International Semantic Web Conference*, 2016.

Julian Eberius, Christoper Werner, Maik Thiele, Katrin Braunschweig, Lars Dannecker, and Wolfgang Lehner. DeExcelerator: a framework for extracting relational data from partially structured documents. In *Proceedings of CIKM*, pages 2477–2480, 2013. doi: 10.1145/2505515.2508210.

Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. Building the Dresden Web Table Corpus: A Classification Approach. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 41–50. IEEE, 2015.

Vasilis Efthymiou, Oktie Hassanzadeh, Mohammad Sadoghi, and Mariano Rodriguez-Muro. Annotating web tables through ontology matching. In *CEUR Workshop Proceedings*, volume 1766, pages 229–230, 2016. doi: 10.13140/RG.2.2.11208.52485.

Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In *Proceedings of ISWC*, volume 10587 LNCS, pages 260–277. Springer, 2017. doi: 10.1007/978-3-319-68288-4_16.

Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *VLDB Journal*, 20(2):209–226, 2011. doi: 10.1007/s00778-011-0223-0.

Ivan Ermilov and Axel Cyrille Ngonga Ngomo. TAIPAN: Automatic property mapping for tabular data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10024 LNAI: 163–179, 2016. doi: 10.1007/978-3-319-49004-5_11.

Patrick Ernst, Amy Siu, and Gerhard Weikum. HighLife: Higher-arity Fact Harvesting. In *Proceedings of WWW*, volume 2, pages 1013–1022, 2018. doi: 10.1145/3178876.3186000.

Jérôme Euzenat, Pavel Shvaiko, and Others. *Ontology matching*, volume 333. Springer, 2007.

Ronald Fagin, P Kolaitis, R Miller, and L Popa. Data Exchange: Semantics and Query Answering. In *Proceedings of ICDT*, 2003.

Jing Fang, Prasenjit Mitra, Zhi Tang, and C Lee Giles. Table Header Detection and Classification. In *Proceedings of AAAI*, pages 599–605, 2012.

Ivan P. Fellegi and Alan B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969. doi: 10.1080/01621459.1969.10501049.

Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary RDF Representation for Publication and Exchange (HDT). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.

Besnik Fetahu, Avishek Anand, and Maria Koutraki. TableNet: An Approach for Determining Fine-grained Relations for Wikipedia Tables. In *Proceedings of WWW*, 2019.

Nadime Francis, Alastair Green, P Guagliardo, L Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, M Rydberg, P Selmer, and A Taylor. Cypher: An Evolving Query Language for Property Graphs. *Proceedings of SIGMODC*, 2018.

Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005. doi: 10.1145/1107499.1107502.

Sainyam Galhotra and Udayan Khurana. Semantic Search over Structured Data. In *Proceedings of CIKM*, pages 3381–3384, New York, NY, USA, 2020. Association for Computing Machinery.

Floris Geerts, G Mecca, Paolo Papotti, and Donatello Santoro. That's All Folks! LLUNATIC Goes Open Source. *Proceedings of the VLDB Endowment*, 7(13): 1565–1568, 2014.

Anna Lisa Gentile, Ziqi Zhang, and Fabio Ciravegna. Early Steps Toward Web-Scale Information Extraction with LODIE. *AI magazine*, 36(1):55–64, 2015.

Anna Lisa Gentile, Petar Ristoski, Steffen Eckel, Dominique Ritze, and Heiko Paulheim. Entity Matching on Web Tables : a Table Embeddings Approach for Blocking. In *Proceedings of EDBT*, volume 20, pages 510–513, 2017.

Majid Ghasemi-Gol, Jay Pujara, and Pedro A Szekely. Tabular Cell Classification Using Pre-Trained Cell Embeddings. *Proceedings of ICDM*, pages 230–239, 2019.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, and Others. Similarity search in high dimensions via hashing. In *Proceedings of the VLDB Conference*, volume 99, pages 518–529, 1999.

Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. Diachronic embedding for temporal knowledge graph completion. In *Proceedings of AAAI*, volume 34, pages 3988–3995, 2020. doi: 10.1609/aaai.v34i04.5815.

Simon Gottschalk and Elena Demidova. EventKG: A Multilingual Event-Centric Temporal Knowledge Graph. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10843 LNCS:272–287, 2018. doi: 10.1007/978-3-319-93417-4_18.

Saiping Guan, Xiaolong Jin, Yuanzhuo Wang, and Xueqi Cheng. Link Prediction on N-ary Relational Data. In *Proceedings of WWW*, pages 583–593. ACM, 2019. doi: 10.1145/3308558.3313414.

Rahul Gupta and Sunita Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment*, 2(1):289–300, 2009. doi: 10.14778/1687627.1687661.

Rahul Gupta, Alon Halevy, Xuezhi Wang, Steven Euijong Whang, and Fei Wu. Biperpedia: An Ontology for Search Applications. *Proceedings of the VLDB Endowment*, 7(7):505–516, 2014. doi: 10.14778/2732286.2732288.

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

Alon Halevy, Natalya Noy, Sunita Sarawagi, Steven Euijong Whang, and Xiao Yu. Discovering Structure in the Universe of Attribute Names. In *Proceedings of WWW*, pages 939–949. International World Wide Web Conferences Steering Committee, 2016. doi: 10.1145/2872427.2882975.

Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05*, page 778, New York, New York, USA, 2005. ACM Press. doi: 10.1145/1066157.1066246.

Braden Hancock, Hongrae Lee, and Cong Yu. Generating Titles for Web Tables. *Proceedings of WWW*, 2019.

Olaf Hartig. Foundations of RDF* and SPARQL*:(An alternative approach to statement-level metadata in RDF). In *AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017.*, volume 1912. Juan Reutter, Divesh Srivastava, 2017.

Oktie Hassanzadeh, Michael J Ward, Mariano Rodriguez-Muro, and Kavitha Srinivas. Understanding a large corpus of web tables through matching with knowledge bases: an empirical study. In *SEMWEB*, 2015.

Erik Hatcher and Otis Gospodnetic. *Lucene in action*. Manning Publications, 2004.

Patrick Hayes. RDF Semantics. W3C Recommendation. Available at http://www.w3.org/TR/rdf-mt/, 2004.

Yeye He, Kaushik Chakrabarti, Tao Cheng, and Tomasz Tylenda. Automatic Discovery of Attribute Synonyms Using Query Logs and Table Corpora. In *Proceedings of WWW*, pages 1429–1439, 2016. doi: 10.1145/2872427.2874816.

Daniel Hernández, Aidan Hogan, and Markus Krötzsch. Reifying RDF: What works well with wikidata? In *CEUR Workshop Proceedings*, volume 1457, pages 32–47, 2015.

Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge Graphs. *arXiv*, mar 2020. doi: 10.1145/3340531.3412176.

Matthew Horridge, Rafael S Gonçalves, Csongor Nyulas, and M Musen. WebProtégé: A Cloud-Based Ontology Editor. *Proceedings of WWW*, 2019.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

Madelon Hulsebos, Arvind Satyanarayan, Kevin Hu, Tim Kraska, Michiel Bakker, Çagatay Demiralp, Emanuel Zgraggen, and César Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of ACM SIGKDD*, pages 1500–1508, 2019. doi: 10.1145/3292500.3330993.

Matthew Hurst. Layout and language: Challenges for table understanding on the web. In *Proceedings of the International Workshop on Web Document Analysis*, pages 27–30, 2001.

Matthew Francis Hurst. *The Interpretation of Tables in Texts*. Phd thesis, 2000.

Yusra Ibrahim, Mirek Riedewald, and Gerhard Weikum. Making Sense of Entities and Quantities in Web Tables. In *Proceedings of CIKM*, pages 1703–1712, 2016. doi: 10.1145/2983323.2983772.

Zachary G Ives, Zhepeng Yan, Nan Zheng, Brian Litt, and Joost B Wagenaar. Looking at Everything in Context. In *Proceedings of CIDR*, 2015.

Anja Jentzsch, Hannes Mühleisen, and Felix Naumann. Uniqueness, density, and keyness: Exploring class hierarchies. In *CEUR Workshop Proceedings*, volume 1426, 2015.

Heng Ji and Ralph Grishman. Knowledge base population: Successful approaches and challenges. In *Proceedings of NAACL-HLT*, pages 1148–1158. Association for Computational Linguistics, 2011.

Ernesto Jiménez-Ruiz, Christian Meilicke, B C Grau, and I Horrocks. Evaluating Mapping Repair Systems with Large Biomedical Ontologies. In *Description Logics*, 2013.

Zhongjun Jin, Yeye He, and Surajit Chauduri. Auto-transform: Learning-to-transform by patterns. *Proceedings of the VLDB Endowment*, 13(11):2368–2381, 2020. doi: 10.14778/3407790.3407831.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 2019.

Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of ACM SIGCHI*, pages 3363–3372, 2011.

I Kavasidis, S Palazzo, C Spampinato, C Pino, D Giordano, D Giuffrida, and P Messina. A Saliency-based Convolutional Neural Network for Table and Chart Detection in Digitized Documents. *ArXiv*, abs/1804.0, apr 2018.

E Kharlamov, S Brandt, Ernesto Jiménez-Ruiz, Y Kotidis, S Lamparter, T Mailis, C Neuenstadt, Ö Özçep, C Pinkel, C Svingos, D Zheleznyakov, I Horrocks, Y Ioannidis, and R Möller. Ontology-Based Integration of Streaming and Static Relational Data with Optique. *Proceedings of ACM SIGMOD*, 2016.

Udayan Khurana and Sainyam Galhotra. Semantic Annotation for Tabular Data. *arXiv*, 2020.

Martin Kleppmann. *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media, Inc., 2017.

Elvis Koci, Maik Thiele, Óscar Romero Moral, Wolfgang Lehner, Oscar Romero, and Wolfgang Lehner. A Machine Learning Approach for Layout Inference in Spreadsheets. In *Proceedings of IC3K*, pages 77–88. SciTePress, 2016. doi: 10.5220/0006052200770088.

Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Prodromos Kolyvakis, Alexandros Kalousis, and Dimitris Kiritsis. Deep alignment: Unsupervised ontology matching with refined word vectors. *Proceedings of NAACL-HLT*, 1:787–798, 2018. doi: 10.18653/v1/n18-1072.

Pradap Konda, Sanjib Das, Paul Suganthan G C, AnHai Hai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, G. C. Paul Suganthan,

Philip Martinkus, AnHai Hai Doan, Adel Ardalan, Jeffrey R. Ballard, Yash Govind, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. Magellan: Toward Building Entity Matching Management Systems. In *Proceedings of the VLDB Endowment*, volume 9, pages 1197–1208, 2016. doi: 10.1145/3277006.3277015.

N. Konstantinou, E. Abel, Luigi Bellomarini, Alex Bogatu, Cristina Civili, Endri Irfanie, M. Koehler, Lacramioara Mazilu, Emanuel Sallinger, A. Fernandes, G. Gottlob, J. Keane, and N. Paton. VADA: an architecture for end user informed data preparation. *Journal of Big Data*, 6(1):74, 2019.

Arlind Kopliku, Karen Pinel-Sauvagnat, and Mohand Boughanem. Attribute Retrieval from Relational Web Tables. In *SPIRE*, 2011.

Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: A comparative analysis. *Phys. Rev. E*, 80(5):56117, 2009.

Larissa R. Lautert, Marcelo M. Scheidt, and Carina F. Dorneles. Web table taxonomy and formalization. *ACM SIGMOD Record*, 42(3):28–33, 2013. doi: 10.1145/2536669.2536674.

Oliver Lehmberg and Christian Bizer. Web table column categorisation and profiling. In *Proceedings of WebDB*, 2016.

Oliver Lehmberg and Christian Bizer. Stitching Web Tables for Improving Matching Quality. *Proceedings of the VLDB Endowment*, 10(11):1502–1513, 2017. doi: 10.14778/3137628.3137657.

Oliver Lehmberg and Christian Bizer. Profiling the semantics of n-ary web table data. In *Proceedings of SBD @ ACM SIGMOD*, number 2, pages 1–6, 2019. doi: 10.1145/3323878.3325806.

Oliver Lehmberg and Oktie Hassanzadeh. Ontology augmentation through matching with web tables. In *CEUR Workshop Proceedings*, volume 2288, pages 37–48, 2018.

Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. A large public corpus of web tables containing time and context metadata. In *Proceedings of WWW*, pages 75–76. International World Wide Web Conferences Steering Committee, 2016.

Hongwei Li, Qingping Yang, Yixuan Cao, Jiaquan Yao, and Ping Luo. Cracking Tabular Presentation Diversity for Automatic Cross-Checking over Numerical Facts. In *Proceedings of ACM SIGKDD*, pages 2599–2607, 2020. doi: 10.1145/3394486.3403310.

Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, Ming Zhou, and Zhoujun Li. TableBank: A Benchmark Dataset for Table Detection and Recognition. In *Proceedings of LREC*, number May, pages 1918–1925, mar 2019.

Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proceedings of the VLDB Endowment*, 3(1):1338–1347, 2010. doi: 10.14778/1920841.1921005.

Cindy Xide Lin, Bo Zhao, Tim Weninger, Jiawei Han, and Bing Liu. Entity relation discovery from web tables and links. In *Proceedings of WWW*, pages 1145–1146, 2010. doi: 10.1145/1772690.1772846.

Xiao Ling, Alon Halevy, Fei Wu, and Cong Yu. Synthesizing union tables from the Web. In *Proceedings of IJCAI*, pages 2677–2683, 2013.

Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. TableSeer : Automatic Table Metadata Extraction and Searching in Digital Libraries Categories and Subject Descriptors. *Information Sciences*, pages 91–100, 2007. doi: 10.1145/1255175.1255193.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. OpenCeres: When Open Information Extraction Meets the Semi-Structured Web. In *Proceedings of NAACL-HLT*, 2019.

Daniel Lopresti and George Nagy. Automated Table Processing: An (Opinionated) Survey. 1999.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008. doi: 10.1109/LPT.2009.2020494.

Marketsandmarkets.com. Data Integration Market by Component (Tools and Services), Business Application (Marketing, Sales, Operations, Finance, and HR), Deployment Model, Organization Size, Vertical, and Region - Global Forecast to 2022, TC 5249. Technical report, 2017.

Jose L. Martinez-Rodriguez, Aidan Hogan, and Ivan Lopez-Arevalo. Information Extraction meets the Semantic Web: A Survey. *Semantic Web*, 11(2), 2020. doi: 10.3233/SW-180333.

Deborah L McGuinness, Frank Van Harmelen, and Others. OWL web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

Sidharth Mudgal, H Li, Theodoros Rekatsinas, A Doan, Y Park, G Krishnan, Rohit Deep, E Arcaute, and V Raghavendra. Deep Learning for Entity Matching: A Design Space Exploration. *Proceedings of ACM SIGMOD*, 2018.

Varish Mulwad, Tim Finin, and Anupam Joshi. Semantic message passing for generating linked data from tables. In *Proceedings of ISWC*. Springer, 2013. doi: 10.1007/978-3-642-41335-3_23.

Emir Muñoz, Aidan Hogan, and Alessandra Mileo. Using linked data to mine RDF from wikipedia's tables. In *Proceedings of WSDM*, 2014. doi: 10.1145/2556195.2556266.

George Nagy, David W Embley, Mukkai S Krishnamoorthy, and Sharad C Seth. Clustering header categories extracted from web tables. In *Proceedings of DRR*, 2015.

Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of WSDM*, number 1955, page 227, 2011. doi: 10.1145/1935826.1935869.

Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. Table Union Search on Open Data. *Proceedings of the VLDB Endowment*, 11(7):813–825, 2018. doi: 10.14778/3192965.3192973.

Felix Naumann. Data profiling revisited. *SIGMOD Rec.*, 42(4):40–49, 2014. doi: 10.1145/2590989.2590995.

Sebastian Neumaier and Axel Polleres. Geo-semantic labelling of Open Data. In *ESWC*, 2018.

Sebastian Neumaier, Jürgen Umbrich, Josiane Xavier Parreira, and Axel Polleres. Multi-level semantic labelling of numerical values. In *Proceedings of ISWC*, pages 428–445. Springer, 2016.

Phuc Nguyen and Hideaki Takeda. Semantic labeling for quantitative data using Wikidata. In *ESWC*, 2018.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. In *Proceedings of the IEEE*, volume 104, pages 11–33. IEEE, 2016.

Natasha Noy, Alan Rector, Pat Hayes, and Chris Welty. Defining n-ary relations on the semantic web. *W3C working group note*, 12(4), 2006.

Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, Jamie Taylor, Anant Narayanan, and Jamie Taylor. Industry-scale knowledge graphs: lessons and challenges. *ACM Queue*, 17(2):48–75, 2019. doi: 10.1145/3329781.3332266.

Ermelinda Oro and M Ruffolo. PDF-TREX: An Approach for Recognizing and Extracting Tables from PDF Documents. *Proceedings of ICDAR*, pages 906–910, 2009.

Yaser Oulabi and Christian Bizer. Estimating Missing Temporal Meta-Information using Knowledge-Based-Trust. In *KDWeb*, 2017.

Yaser Oulabi and Christian Bizer. Extending Cross-Domain Knowledge Bases with Long Tail Entities using Web Table Data. In *Proceedings of EDBT*, 2019.

Yaser Oulabi, Robert Meusel, and Christian Bizer. Fusing time-dependent web table data. In *Proceedings of WebDB*, number 1, 2016. doi: 10.1145/2932194.2932197.

George Papadakis, Ekaterini Ioannou, and Themis Palpanas. Entity Resolution: Past, Present and Yet-to-come. In *Proceedings of CIDR*, 2020a. doi: 10.1007/978-3-030-12375-8_3.

George Papadakis, Ekaterini Ioannou, and Themis Palpanas. Entity Resolution: Past, Present and Yet-to-Come. In *Proceedings of EDBT*, pages 647–650, 2020b.

George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2):1–42, 2020c.

T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015. doi: 10.14778/2794367.2794377.

Panupong Pasupat and Percy Liang. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of ACL*, pages 1470–1480, 2015.

Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1989.

Thomas Pellissier Tanon, Denny Vrandečić, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher. From freebase to wikidata: The great migration. In *Proceedings of WWW*, pages 1419–1428, 2016.

Thomas Pellissier Tanon, Gerhard Weikum, Fabian Suchanek, Thomas Pellissier-Tanon, Gerhard Weikum, and Fabian Suchanek. YAGO 4: A Reason-able Knowledge Base. In Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, Peter Haase, and Michael Cochez, editors, *Proceedings of ESWC*, volume 1, pages 583–596, Cham, 2020. Springer International Publishing. doi: 10.1007/978-3-030-49461-2.

Boya Peng, Yejin Huh, Xiao Ling, and Michele Banko. Improving Knowledge Base Construction from Robust Infobox Extraction. In *Proceedings of NAACL-HLT*, number June 2018, pages 138–148, Stroudsburg, PA, USA, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-2018.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of EMNLP*, 2014.

Minh Pham, Suresh Alse, Craig A Knoblock, and Pedro Szekely. Semantic labeling : A domain-independent approach. In *Proceedings of ISWC*, 2016.

Rakesh Pimplikar and Sunita Sarawagi. Answering table queries on the web using column keywords. *Proceedings of the VLDB Endowment*, 5(10):908–919, 2012. doi: 10.14778/2336664.2336665.

David Pinto, Andrew McCallum, Xing Wei, and W Bruce Croft. Table extraction using conditional random fields. In *Proceedings of ACM SIGIR*, pages 235–242, 2003.

Aleksander Pivk, Philipp Cimiano, York Sure, Matjaz Gams, Vladislav Rajkovič, and Rudi Studer. Transforming arbitrary tables into logical form with TARTAR. *Data and Knowledge Engineering*, 60(3):567–595, 2007. doi: 10.1016/j.datak.2006.04.002.

Simeon Polfliet and Ryutaro Ichise. Automated mapping generation for converting databases into linked data. In *CEUR Workshop Proceedings*, volume 658, pages 173–176, 2010.

Dave Raggett. HTML Tables. Internet Requests for Comments, 1996.

Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001. doi: 10.1007/s007780100057.

Chenwei Ran, Wei Shen, Jianyong Wang, and Xuan Zhu. Domain-Specific Knowledge Base Enrichment Using Wikipedia Tables. In *2015 IEEE International Conference on Data Mining*, pages 349–358. IEEE, 2015. doi: 10.1109/ICDM.2015.124.

Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. Training complex models with multi-task weak supervision. In *Proceedings of AAAI*, pages 4763–4771, 2019.

Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: rapid training data creation with weak supervision. *The VLDB Journal*, 29(2):709–730, 2020. doi: 10.1007/s00778-019-00552-1.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of NIPS*, pages 693–701, 2011.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation Extraction with Matrix Factorization and Universal Schemas. In *Proceedings of NAACL-HLT*, pages 74–84, 2013.

Petar Ristoski and Heiko Paulheim. Semantic Web in data mining and knowledge discovery: A comprehensive survey. In *Proceedings of WWW*, volume 36, pages 1–22. Elsevier, 2016. doi: 10.1016/j.websem.2016.01.001.

Dominique Ritze and Christian Bizer. Matching Web Tables To DBpedia - A Feature Utility Study. In *Proceedings of EDBT*, pages 210–221, 2017. doi: 10.5441/002/edbt.2017.20.

Dominique Ritze, Oliver Lehmberg, and Christian Bizer. Matching HTML Tables to DBpedia. In *WIMS*, 2015.

Dominique Ritze, Oliver Lehmberg, Yaser Oulabi, and Christian Bizer. Profiling the Potential of Web Tables for Augmenting Cross-domain Knowledge Bases. In *Proceedings of WWW*, pages 251–261, 2016.

Mandy Roick, Thorsten Papenbrock, Lukas Schulze, and Felix Naumann. Holistic Data Profiling : Simultaneous Discovery of Various Metadata Categories and Subject Descriptors. In *Proceedings of EDBT*, pages 305–316, 2016.

Juan C. Roldán, Patricia Jiménez, and Rafael Corchuelo. On extracting data from tables that are encoded using HTML. *Knowledge-Based Systems*, page 105157, 2019. doi: 10.1016/j.knosys.2019.105157.

Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of EMNLP*, pages 410–420, 2007.

Marcos Antonio Vaz Salles, Jens Peter Dittrich, Shant Kirakos Karakashian, Olivier René Girard, and Lukas Blunschi. Itrails: Pay-as-you-go information integration in dataspaces. *Proceedings of the VLDB Endowment*, pages 663–674, 2007.

Sebastian Schreiber, S Agne, I Wolf, A Dengel, and S Ahmed. DeepDeSRT: Deep Learning for Detection and Structure Recognition of Tables in Document Images. *Proceedings of ICDAR*, 01:1162–1167, 2017.

Yoones A Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. Knowledge Base Augmentation using Tabular Data. In *Proceedings of WWW*, 2014.

Juan F Sequeda and Daniel P Miranker. Ultrawrap: SPARQL execution on relational data. *Journal of Web Semantics*, 22:19–39, 2013.

Dominic Seyler, Tatiana Dembelova, Luciano Del Corro, Johannes Hoffart, and G Weikum. A Study of the Importance of External Knowledge in the Named Entity Recognition Task. In *Proceedings of ACL*, 2018.

Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. LIEGE: link entities in web lists with knowledge base. In *Proceedings of ACM SIGKDD*, pages 1424–1432, 2012. doi: 10.1145/2339530.2339753.

Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015. doi: 10.1109/TKDE.2014.2327028.

Pavel Shvaiko and J. Euzenat. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, 2013. doi: 10.1109/TKDE.2011.253.

N Siegel, Zachary Horvitz, Roie Levin, S Divvala, and Ali Farhadi. FigureSeer: Parsing Result-Figures in Research Papers. In *Proceedings of ECCV*, 2016.

M Stonebraker and Ihab F Ilyas. Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.*, 41:3–9, 2018.

Fabian Suchanek. The need to move beyond triples. *CEUR Workshop Proceedings*, 2593:95–104, 2020.

Fabian M Suchanek and G Weikum. Knowledge harvesting in the big-data era. In *Proceedings of ACM SIGMOD*, 2013.

Huan Sun, Hao Ma, Xiaodong He, Wen-Tau Yih, Yu Su, and Xifeng Yan. Table Cell Search for Question Answering. In *Proceedings of WWW*, pages 771–782, 2016. doi: 10.1145/2872427.2883080.

Zareen Syed, Tim Finin, Varish Mulwad, and Anupam Joshi. Exploiting a Web of Semantic Data for Interpreting Tables. In *Proceedings of the WebSci10: Extending the Frontiers of Society On-Line, April 26-27th, 2010, Raleigh, NC: US*, number April, pages 26– 27, 2010.

Mohsen Taheriyan, Craig A. Knoblock, Pedro A Szekely, and José Luis Ambite. Learning the semantics of structured data sources. *Journal of Web Semantics*, 37-38: 152–169, 2016. doi: 10.1016/j.websem.2015.12.003.

Jacopo Urbani and Ceriel Jacobs. Adaptive low-level storage of very large knowledge graphs. In *Proceedings of WWW*, pages 1761–1772, 2020.

Rick Van Der Lans. *Data Virtualization for business intelligence systems: revolutionizing data integration for data warehouses*. Elsevier, 2012.

Sander van der Waal, Krzysztof Węcel, Ivan Ermilov, Valentina Janev, Uroš Milošević, Mark Wainwright, Sander Van Der Waal B, W B Krzysztof, and Ivan Ermilov. Lifting open data portals to the data web. In *Linked Open Data - Creating Knowledge Out of Interlinked Data*, volume 8661, pages 175–195. Springer, 2014. doi: 10.1007/978-3-319-09846-3.

Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering Semantics of Tables on the Web. *Proceedings of the VLDB Endowment*, 4(9):528–538, 2011. doi: 10.14778/2002938.2002939.

N L Vine, Matthew D Zeigenfuse, and M Rowan. Extracting Tables from Documents using Conditional Generative Adversarial Networks and Genetic Algorithms. *Proceedings of IJCNN*, pages 1–8, 2019.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledge base. *Communications of the ACM*, 57(10):78–85, 2014.

Binh Vu, Jay Pujara, and Craig A Knoblock. D-REPR : A Language for Describing and Mapping Diversely-Structured Data Sources to RDF. In *Proceedings of the 10th International Conference on Knowledge Capture*, pages 189–196, 2019.

W3C. SPARQL 1.1 overview. 2013.

Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Luna Dong, and Meng Jiang. TCN: Table Convolutional Network for Web Table Interpretation. In *Proceedings of the Web Conference 2021*, pages 4020–4032, 2021.

D.Z. Wang, Luna Dong, a.D. Sarma, M.J. Franklin, and Alon Halevy. Functional Dependency Generation and Applications in pay-as-you-go data integration systems. In *Proceedings of WebDB*, pages 1–6, 2009.

J Wang, Bin Shao, and Haixun Wang. Understanding tables on the web. In *Proceedings of ER*. Springer, 2012.

Ning Wang and Xiangran Ren. Identifying Multiple Entity Columns in Web Tables. *International Journal of Software Engineering and Knowledge Engineering*, 28(03): 287–309, 2018.

Xu Wang, Zhisheng Huang, and Frank van Harmelen. Evaluating Similarity Measures for Dataset Search. In *International Conference on Web Information Systems Engineering*, pages 38–51. Springer, 2020.

Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *Proceedings of WWW*, volume 9, page 242, 2002. doi: 10.1145/511475.511478.

Gerhard Weikum and Martin Theobald. From information to knowledge: harvesting entities and relationships from web sources. In *Proceedings of ACM PODS*, pages 65–76, 2010. doi: 10.1145/1807085.1807097.

Gerhard Weikum, Simon Razniewski, Luna Dong, and Fabian Suchanek. Machine knowledge: Creation and curation of comprehensive knowledge bases. Technical report, 2020.

Matthew West. *Developing high quality data models*. Elsevier, 2011.

Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014.

Fei Wu and Daniel S. Weld. Autonomously semantifying wikipedia. *Proceedings of CIKM*, 2007. doi: 10.1145/1321440.1321449.

Fei Wu and Daniel S Weld. Open Information Extraction using Wikipedia. In *Proceedings of ACL*, number July, pages 118–127, 2010.

Dean Wyatte. De-biasing Weakly Supervised Learning by Regularizing Prediction Entropy. *Proceedings of ICLR 2019 Workshop LLD*, 2019.

Catharine M Wyss and Edward L Robertson. A formal characterization of {PIVOT}/{UNPIVOT}. In *Proceedings of CIKM*, pages 602–608. ACM, 2005.

M Yakout and Kris Ganjam. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of ACM SIGMOD*, pages 97–108, 2012. doi: 10.1145/2213836.2213848.

Yang Yi, Zhiyu Chen, Jeff Heflin, and Brian D Davison. Recognizing Quantity Names for Tabular Data. In *Proceedings of ProfS/KG4IR/Data:Search@SIGIR*, 2018.

Xiaoxin Yin, Wenzhao Tan, and Chao Liu. FACTO : A Fact Lookup Engine Based on Web Tables. In *Proceedings of WWW*, pages 507–516, 2011. doi: 10.1145/1963405.1963477.

Mohamed Amir Yosef, Johannes Hoffart, Ilaria Bordino, Marc Spaniol, and Gerhard Weikum. AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. *Proceedings of the VLDB Endowment*, 4(12):1450–1453, 2011.

Minoru Yoshida, Kentaro Torisawa, and Jun'ichi Tsujii. A method to integrate tables of the World Wide Web. 2001.

Wenhao Yu, Wei Peng, Yu Shu, Qing-kai Zeng, and Meng Jiang. Experimental Evidence Extraction System in Data Science with Hybrid Table Features and Ensemble Learning. *Proceedings of WWW*, 2020.

Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, Wang-Chiew Tan, cCaugatay Demiralp, and Wang-Chiew Tan. Sato: Contextual Semantic Type Detection in Tables. *Proceedings of the VLDB Endowment*, 13(12): 1835–1848, 2020.

Dongxu Zhang, Subhabrata Mukherjee, Colin Lockard, Xin Luna Dong, and Andrew Mccallum. OpenKI: Integrating Open Information Extraction and Knowledge Bases with Relation Inference. In *Proceedings of NAACL-HLT*, 2019.

Shuo Zhang and Krisztian Balog. Web table extraction, retrieval, and augmentation: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(2): 1–35, 2020. doi: 10.1145/3331184.3331385.

Yuanzhe Zhang, Xuepeng Wang, Siwei Lai, Shizhu He, Kang Liu, Jun Zhao, and X Lv. Ontology Matching with Word Embeddings. In *Proceedings of CCL*, 2014.

Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web Journal*, 8(6):921–957, 2017.

Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno-Yepes. Image-based table recognition: data, model, and evaluation. In *Proceedings of ECCV*, 2020.

Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J. Miller. LSH Ensemble: Internet-Scale Domain Search. *Proceedings of the VLDB Endowment*, 9(12):1185–1196, 2016. doi: 10.14778/2994509.2994534.

Erkang Zhu, Yeye He, and Surajit Chaudhuri. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment*, 10(10):1034–1045, 2017. doi: 10.14778/3115404.3115409.

Qi Zhu, Hao Wei, Bunyamin Sisman, Da Zheng, Christos Faloutsos, Xin Luna Dong, and Jiawei Han. Collective Knowledge Graph Multi-type Entity Alignment. In *Proceedings of WWW*, 2020.

Stefan Zwicklbauer, Christoph Einsiedler, Michael Granitzer, and Christin Seifert. Towards disambiguating web tables. In *Proceedings of ISWC (Posters & Demos)*, pages 205–208. CEUR-WS. org, 2013.

Summary

Tables are used for representing similarly structured data in an enormous variety of documents and formats. Over the past 30 years, the huge increase in the volume of published documents on the web has opened up the potential for automatically extracting information from human-readable tables in those documents. Such tables may express information that is useful for many applications (such as web search and question answering), but they often have diverse contents and structures. To deal with this diversity, these applications can benefit from automatically integrating the information from such tables into coherent, machine-readable Knowledge Bases (KBs).

The first step of this integration process is table interpretation, which consists of finding associations between table contents and some background knowledge. However, effectively leveraging this background knowledge introduces several challenges. One such challenge is that on existing domains tables are not only interpreted using a KB as background knowledge, but are also used to extend the same KB with new information, which may cause a bias towards redundant extractions. A further issue is that there may also be tables for which the KB does not provide enough background knowledge to permit interpretation on their own. Additionally, on new domains this background knowledge must be specified manually, which may involve much human effort. Existing approaches are often unsuited to deal with such real-world phenomena.

In this thesis, we present several approaches towards overcoming these challenges in order to extend KBs with new information extracted from human-readable tables. We investigate several real-world phenomena that impede the integration of tables with KBs, and develop methods for overcoming these challenges. First, we describe the challenge of extracting novel information from tables that is not yet present in a KB, when the same KB is used for interpreting them. For addressing this problem, we develop a new evaluation approach, as well as a novel table interpretation method. Second, we describe the challenge of extending KBs from tables with extracted n-ary facts, which involve more than two entities or values. To address this, we present and evaluate a data extraction pipeline for Wikipedia tables that overcomes the diversity of table layouts and the sparsity of n-ary information in the Wikidata KB. Third, we investigate the problem of creating a KB on a new domain from tables with minimal human supervision. We propose a solution that combines weakly supervised machine learning and logical reasoning, and we evaluate it on tables from scientific papers. Fourth, we describe a new system for supporting research into data extraction and KB extension pipelines from tables.

In conclusion, the techniques we have developed effectively leverage background knowledge to overcome the variety of table contents and structures, thereby contributing towards the extension of KBs with new information from human-readable tables. Nevertheless, there is still much potential for research into KB extension from tables. We close this thesis with suggestions for future research, and reflections on research in this field.

Curriculum Vitae

Benjamin Bienze Kruit geboren op 15 december 1990 te Delft

2016 - 2021	PhD candidate Database Architectures group, Centrum Wiskunde & Informatica (CWI Amsterdam) and KARMA research, HPDC group, VU Amsterdam Supervised by Jacopo Urbani and Peter Boncz
2013 - 2016	Master of Science Artificial Intelligence University of Amsterdam
2009 - 2013	Bachelor of Science (Cognitive) Artificial Intelligence Utrecht University