SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Department of
Computer Science, Systems and Communication

PhD program in Computer Science Cycle XXXIII

Semantic Table Annotation for Large-Scale Data Enrichment

Surname: Cutrona Name: Vincenzo

Registration number: 745615

Tutor: Prof. Giuseppe Vizzari

Supervisor: Prof. Matteo Palmonari

Coordinator: Prof. Leonardo Mariani

ACADEMIC YEAR 2019/2020

I have never met a man so ignorant that I couldn't learn something from him.

— Galileo Galilei

Dedicated to all the people who have shared this journey with me.

ABSTRACT

Data are the new oil, and they represent one of the main value-creating assets. Data analytics has become a crucial component in scientific studies and business decisions in the last years and has brought researchers to define novel methodologies to represent, manage, and analyze data. Simultaneously, the growth of computing power enabled the analysis of huge amounts of data, allowing people to extract useful information from collected data.

Predictive analytics plays a crucial role in many applications since it provides more knowledge to support business decisions. Among the statistical techniques available to support predictive analytics, machine learning is the technique that features capabilities to solve many different classes of problems, and that has benefited the most from computing power growth. In the last years, more complex and accurate machine learning models have been proposed, requiring an increasing amount of current and historical data to perform the best.

The demand for such a massive amount of data to train machine learning models represents an initial hurdle for data scientists because the information needed is usually scattered in different data sets that have to be manually integrated. As a consequence, data enrichment has become a critical task in the data preparation process, and nowadays, most of all the data science projects involve a time-costly data preparation process aimed at enriching a core data set with additional information from various external sources to improve the sturdiness of resulting trained models. How to ease the design of the enrichment process for data scientists is defying and supporting the enrichment process at a large scale. Despite the growing importance of the enrichment task, it is still supported only to a limited extent by existing solutions, delegating most of the effort to the data scientist, who is in charge of both detecting the data sets that contain the needed information, and integrate them.

In this thesis, we introduce a methodology to support the data enrichment task, which focuses on harnessing the semantics as the key factor by providing users with a semantics-aided tool to design the enrichment process, along with a platform to execute the process at a business scale. We illustrate how the data enrichment can be addressed via tabular data transformations exploiting semantic table interpretation methods, discussing implementation techniques to support the enactment of the resulting process on large data sets. We experimentally demonstrate the scalability and run-time efficiency of the proposed solution by employing it in a real-world scenario. Finally, we introduce a new benchmark dataset to evaluate the performance and the scalability of existing semantic table annotation algorithms, and we propose an efficient novel approach to improve the performance of such algorithms.

PUBLICATIONS

List of publications related to the work described in this thesis:

[1] Federico Bianchi, Mauricio Soto, Matteo Palmonari, and **Vincenzo Cutrona**. “Type Vector Representations from Text: An Empirical Analysis.” In: *Proceedings of the First Workshop on Deep Learning for Knowledge Graphs and Semantic Technologies, DL4KGS@ESWC*. Vol. 2106. CEUR-WS.org, 2018, pp. 72–83.

[2] Michele Ciavotta, **Vincenzo Cutrona**, Flavio De Paoli, Nikolay Nikolov, Matteo Palmonari, and Dumitru Roman. “Supporting Semantic Data Enrichment at Scale.” In: *Technologies and Applications for Big Data Value*. (To appear). 2021.

[3] Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Kavitha Srinivas, and **Vincenzo Cutrona**. “Results of SemTab 2020.” In: *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab@ISWC*. Vol. 2775. CEUR-WS.org, 2020, pp. 1–8.

[4] Shady Abd El Kader, Nikolay Nikolov, Bjørn Marius von Zernichow, **Vincenzo Cutrona**, Matteo Palmonari, Brian Elvesæter, Ahmet Soylu, and Dumitru Roman. “Modeling and Publishing French Business Register (Sirene) Data as Linked Data Using the euBusinessGraph Ontology.” In: *Joint Proceedings of the International Workshops on Sensors and Actuators on the Web, and Semantic Statistics, SemStats@ISWC*. Vol. 2549. CEUR-WS.org, 2019.

[5] **Vincenzo Cutrona**, Federico Bianchi, Ernesto Jiménez-Ruiz, and Matteo Palmonari. “Tough Tables: Carefully Evaluating Entity Linking for Tabular Data.” In: *ISWC*. Vol. 12507. Springer, 2020, pp. 328–343.

[6] **Vincenzo Cutrona**, Michele Ciavotta, Flavio De Paoli, and Matteo Palmonari. “ASIA: a Tool for Assisted Semantic Interpretation and Annotation of Tabular Data.” In: *ISWC Satellite Tracks*. Vol. 2456. CEUR-WS.org, 2019, pp. 209–212.

[7] **Vincenzo Cutrona**, Flavio De Paoli, Aljaz Kosmerlj, Nikolay Nikolov, Matteo Palmonari, Fernando Perales, and Dumitru Roman. “Semantically-Enabled Optimization of Digital Marketing Campaigns.” In: *ISWC*. Vol. 11779. Springer, 2019, pp. 345–362.

[8] **Vincenzo Cutrona**, Gianluca Puleri, Federico Bianchi, and Matteo Palmonari. “NEST: Neural Soft Type Constraints to Improve Entity Linking in Tables.” In: *ESWC*. (Under revision). 2021.

ACKNOWLEDGMENTS

I am extremely grateful to all the people who have played an essential role in this research.

First and foremost, I would like to thank my supervisor Prof. Matteo Palmonari, for his invaluable advice and tutelage during my Ph.D. His continuous support was really influential in shaping my academic research. Thanks for pushing me to give the best and for the constructive comments to correct my mistakes.

I would also thank Prof. Michele Ciavotta, which I consider to be my second supervisor. His immense knowledge and great experience have encouraged me in my academic research and, most importantly, daily life. It has been a real pleasure to collaborate with him all through my Ph.D. course.

I would like to offer my special thanks to Prof. Ernesto Jiménez-Ruiz, who gave me the possibility of joining Prof. Artur d'Avila Garcez's lab at City, University of London. I really enjoyed the time we spent together at City, and I'm deeply grateful to him for all the discussions we had, which helped me better live the academic life and look to the future with confidence.

Many thanks to all the other professors of my research group who gave me continuous advice. A special thank you to Prof. Giuseppe Vizzari, my Ph.D. tutor, for his friendly suggestions, and to Prof. Leonardo Mariani, the school coordinator, who always promptly supported Ph.D. students in these pandemic times.

I would like to thank all my lab mates and colleagues, especially Federico Bianchi, who left our research group last year but continued supporting my research with ideas and suggestions. Special thanks also to Debora Nozza for her invaluable friendly advice. I would also like to thank all the students, especially Gianluca Puleri, who brought valuable contributions to this research.

Finally, I would like to express my gratitude to my family, my girlfriend, and my friends for their encouragement and support throughout my studies. Thanks for having helped me in overcoming difficult moments.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CEA	Cell Entity Annotation
CPA	Column Predicate Annotation
CTA	Column Type Annotation
DNN	Deep Neural Network
FOL	First-Order Logic
KB	Knowledge Base

KG	Knowledge Graph
RDF	Resource Description Framework
STA	Semantic Table Annotation
STI	Semantic Table Interpretation
URI	Universal Resource Identifier

1 INTRODUCTION

We live in the Data era, an era where the volume, velocity, variety, veracity, and value (the *Five V's*) of the data are increasing exponentially. Extracting information from the vast amount of collected data worldwide is a task of paramount importance in all fields, from research to business, and it represents one of the main value-creating assets (estimates indicate yearly earnings in the order of 270 billion dollars by 2022).[1] In the last few years, data science has become a hot spot because of the importance of data analyses in scientific studies and business decisions, leading to the definition of novel methodologies to represent, manage, and analyze data.

Machine Learning and Deep Learning represent some of the most powerful tools available to data scientists, as they have proven to excel in solving a range of problems, from image analysis to text interpretation and understanding. Throughout the years, the increase in computational resource power enables the training of even larger and sophisticated machine learning models (i. e., architecturally more complex, with more parameters to learn), which need more data to be trained. Back in 2013, the *One Billion Words Benchmark for Language Modeling* was released, which contains 0.8 billion tokens.[2] In 2020, just seven years later, OpenAi announced their GPT-3 model, a model with 175 billion parameters, which has been trained on a dataset with 500 billion tokens [10].

Prediction models are useful in different industry-driven and science-driven application domains to support the business decision. Bigger models allow companies to consider more factors in their analysis, thanks to the higher number of model parameters. For example, a marketing company can study the impact of the weather forecasts on their digital campaigns, considering different weather features in their study; or else, an epidemiology study with data collected from patients diagnoses can be extended by considering demographic information, such as the population density of a specific city. However, training datasets in industry-driven and science-driven application domains are mostly represented by first-party data, i. e., data that a company already collects about users or customers, usually about a single domain (e. g., marketing data, diagnoses data). In order to consider external factors (e. g., weather features), different sources of information have to be integrated with the first-party data.

1 Worldwide Semiannual Big Data and Analytics Spending Guide - https://www.idc.com/getdoc.jsp?containerId=IDC_P33195

2 https://opensource.google/projects/lm-benchmark

The extension of the first-party data with the needed second- (i. e., data collected and sold by a different company) or third-party data (i. e., data provided by high volume data aggregators),[3] represents the *data enrichment*, a core step of the data preparation stage. We define the data enrichment as the task of enriching one source dataset (*source*) with information from a second dataset (*target*), which we assume to contain additional features.[4]

Despite the number of powerful tools available nowadays, completing the data preparation stage in a typical Data Science project covers up to the 80% of the whole project duration time, representing a high cost in terms of time and money [70, 120]. The data enrichment step poses specific challenges; indeed, finding the right data to use is not straightforward, and how to push them into the source dataset is still an even more cumbersome task; in fact, it requires to achieve two core steps: (i) to *reconcile* the source dataset against the target datasets, i. e., to identify data fractions that are available both in the source and in the target datasets, which can be exploited as join points, and (ii) to *extend* the source dataset, i. e., actually to add the additional information to the source dataset. The challenges in achieving the data enrichment are mainly due to the following factors:

- *Data exploration*: finding all the datasets of interest is cumbersome because different target datasets are available; for example, extending marketing data with weather features requires to discover which third-party dataset provides the needed information, which are the refresh rate and the geospatial coverage of the selected dataset, and so on. Researchers proposed ad-hoc solutions to consider this aspect in the data analytics pipeline, e. g., the CAVA [16] system integrates the data preparation and the data augmentation with the traditional data exploration and analysis tasks.

- *Knowledge gap*: the user usually knows the source dataset but has no knowledge about the target one; for example, weather datasets contain domain-specific properties, represented with domain-specific formats, which are not known in advance by people working in the marketing domain;

- *Reproducibility*: the data preparation is a recurrent process (e. g., the same process applies to several snapshots of the same dataset, like monthly sales reports), and it is commonly addressed by defining *pipelines* that embed all the steps to reproduce the cleaning/enrichment phases; building such pipelines require

3 Definitions of first-, second-, and third-party data are from [117]. For the sake of simplicity, we will improperly use the term *third-party data* to refer to both second- and third-party data.

4 The target dataset may contain first- or third-party data. We will consider the most challenging case of using third-party data.

users to know in details the source and the target dataset contents and structures;

- *Scalability*: source and target datasets may be very large (millions of rows and more), making scalability a killer factor in the pipeline deployment.

.

The Semantic Web proposed by Tim Berners-Lee [3] in 2001 aims to create a Web where data are connected and semantically annotated, so that to enable artificial agents to access the data more intelligently and perform tasks on behalf of users in order to discover the desired knowledge. Since 2001, the Linked Open Data (LOD) cloud has grown to the dimension of thousands of interconnected datasets.

The connected structures of the LOD cloud and the massive amount of high-quality datasets in it represent two valuable ingredients to exploit in supporting the data enrichment; in fact, once the source dataset has been reconciled against a target dataset in the LOD cloud, we have that, on the one hand, it is possible to help the users in discovering the target dataset and in finding the right information thanks to the intrinsic semantics (addressing the knowledge gap problem); on the other hand, the inter-datasets links enable the guided discovery of new and potentially interesting datasets (helping with the data exploration). The important role that the semantics plays in data enrichment, especially within Big Data contexts, is also acknowledged by the European Big Data Value Strategic Research and Innovation Agenda [129], and dedicated special issues in scientific journals [2, 60, 128].

In this asset, the enrichment workflow can be viewed as a pipeline of data reconciliation and extension steps. For example, adding demographic data to the diagnoses data requires a simple two-step pipeline: (i) a reconciliation step that reconciles patients living cities against cities in a target dataset; (ii) an extension step to fetch the information about the population density from the target dataset (e. g., the total population and the surface area of each city). Effectively supporting these steps may lead to better usage of the LOD cloud data, helping users access third-party datasets without acquiring the cross-domain expertise required to integrate the datasets manually.

Most of the currently available solutions that support data enrichment are embedded in data transformation tools (e. g., KNIME and Talend). They are designed for users familiar with programming languages and process definition but usually inexperienced in the particular domain to which the data pertain [108]. Alternatively, some interactive tools support the data enrichment (e. g., OpenRefine) but are plagued with limited scalability and do not offer any functionality to develop/deploy the enrichment process into a production-ready pipeline. The ideal solution is to provide the users with a system that can guide them to interactively explore the target datasets, and retrieve the needed information in a pay-as-you-go fashion.

In this thesis, a different perspective to address the data enrichment task for tabular data is discussed, which frees users from the burden of knowing the target datasets. The proposed methodology uses the Semantic Table Interpretation (STI) (i. e., a semantic-based approach to reconciling tables against a target Knowledge Base (KB) - more details are given in Chapter 2) as the core enrichment enabler: the focus of the user is in annotating the elements of the source dataset, while the system offers support in finding new sources of information (target datasets).

1.1 USE CASE EXAMPLE

We motivate the importance of supporting the data enrichment at scale by describing a real-life data analytics use case consisting of different data manipulation and extension tasks.

A digital marketing company invests in sponsored ads in the main search and display platforms (e. g., Google, Bing, or Facebook), i. e., the company buys a certain amount of ads, and then pays such platforms to display specific ad banners to the users. The higher the number of visualized and clicked ads, the higher the company revenues. Thus, when it comes to starting a marketing campaign, the company has to place a bid, i. e., to choose which ads to display, when, where, and to whom. In fact, nowadays, the advertisement engagement strategy is highly oriented to personalized user targeting to promote relevant content to the right users.

In order to evaluate the current bidding strategy, the company collects daily a considerable amount of data related to different performance indicators, including volume of clicks (number of users that click the ad banner) and impressions (number of users that visualize the ad banner), the user location and the timestamp when the user clicked/visualized the banner, and the device on which the banner has been displayed. The collected data are then analyzed by expert managers that optimize the bidding strategy based on their own experience and the collected data. For example, a manager knows that the weeks immediately preceding rainy seasons are a perfect period to advertise umbrellas.

The company wants to exploit the amount of collected data to investigate the effect of weather conditions on its campaigns' performance at a regional level. Anne, the company manager, assigns this task to a data analyst, Claire, providing her with reports from Google AdWords, the platform currently used to collect campaign performance indicators (an excerpt of such reports is represented by the white columns in Table 1.1). The final goal of Claire is to train a prediction model able to predict the most suitable moment to launch a new campaign.

Table 1.1: A dataset enriched with data from GeoNames (GN) and ECWMF (W) by applying different functions: transformation (gray), reconciliation (red), and extension (orange).

Keyword ID	Clicks	City	Region	Date	Date (ISO 8601)	Region ID (GN)	2t (W)	tp (W)
...	...	...	...	...	...	...	...	...
194906	64	Altenburg	Thuringia	6/9/2017	2017-09-06	2822542	287.70	0.08
517827	50	Ingolstadt	Bavaria	6/9/2017	2017-09-06	2951839	288.18	0.02
459143	42	Berlin	Berlin	6/9/2017	2017-09-06	2950157	290.48	0.00
891139	36	Munich	Bavaria	6/9/2017	2017-09-06	2951839	288.18	0.02
459143	30	Ulm	Baden-Württemberg	6/9/2017	2017-09-06	2953481	288.18	0.02
...	...	...	...	...	...	...	...	...

Claire reserves some time to compare different weather services and finally decides to retrieve weather forecasts from ECMWF,[5] a provider that release dumps of weather observations collected across Europe. Among the several weather features available in ECMWF dumps, Claire is firstly interested in features related to the temperature (2t) and precipitation (tp).

Claire resorts to asking Bob, a data engineer, to add the set of needed features to the original dataset, so that to enable the analytics phase. She also asks him to deliver the data in tabular format because, in this preliminary stage, she prefers working with Jupyter notebooks, which allow her to run Python code interactively; thus, she needs data in a format that can be loaded with standard Python libraries.

Data have been collected for many years by the company, resulting in a large volume dataset. Bob is familiar with data transformation tools and immediately decides to exploit their capability to scale to the dimension of the data. Besides, the created pipeline can be reused in the future on newly collected data until they share the same format and schema of the current dataset. Bob downloads a dump from the ECMWF service and discovers that the dump adheres to the GRIB format, which organizes the weather information on a grid of points represented by WGS84 coordinate pairs. Each point in the grid is assigned a set of weather observations for some dates; dates are represented using the ISO 8601 standard. Thus, to access the needed information, a WGS84 coordinate pair and an ISO 8601 formatted date are required; both the inputs are currently missing in the company dataset.

Bob designs a transformation pipeline starting with a step that computes the ISO 8601 formatted dates; as a result, a new column is appended to the source dataset (the grey column in Table 1.1). As the next step, Bob has to figure out how to get the weather information at the region level: in fact, Claire asked for weather information aggregated at the region level, while the GRIB dataset has much more fine-grained information (at coordinate level). Bob has first to detect

5 `https://www.ecmwf.int`

the *boundary box* of a region, then collect the weather information for those coordinates that fall into the boundary box, and aggregate them (e. g., by computing the average value).

Locations in AdWords reports are from a proprietary geospatial dataset, *Geotargets*,[6] which does not contain the boundary box of locations. Bob finds out that GeoNames,[7] a public geographical database, provides the boundary box of each represented geographical object. Geotargets and GeoNames use different identifiers to describe locations, and the lack of existing direct mappings between the two datasets hinders an easy join between them. Bob has to download also the dumps of GeoNames and Geotargets, so that he can use an existing record linkage tool to find the mappings between locations; however, the information available in Geotargets is poor with respect to the rich representation provided in GeoNames, and the quality of the resulting mappings is low. Since the subsequent analysis will highly rely on the result of this step, Bob has to ensure a high level of quality, and for this reason, he manually revises the mappings to remove false positives and fix wrong results. As a result, Bob obtains a new dataset, where GeoNames and Geotargets identifiers that refer to the same location are in the same row; this new dataset can now be easily integrated with the source dataset by adding a join operation to the pipeline (as a result, the red column in Table 1.1 is created).

At this point, the source dataset contains the information needed to query the GRIB dataset. Bob adds one more step to the pipeline, which contains a short script of code that takes as input a pair of values from the *Region ID (GN)* and *Date (ISO 8601)* columns, retrieves the boundary box of the region from GeoNames, then filters the GRIB dataset by the selected date and the coordinate pairs that fall into the boundary box. The results are then aggregated and appended to the source dataset, resulting in two new columns (orange columns in Table 1.1). Thanks to the data transformation tool's scalability, Bob applies the pipeline to the large volume of data available to the company, thus enabling the analysis mentioned above. Indeed, Claire can now sample the desired fraction of data and load the sample into her most preferred data analytics tool.

Such operations are widespread in data analytics projects, where a source dataset needs to be extended with additional information to enable different kinds of analyses. Moreover, the same process may be repeated many times in the same project because (i) data from different external sources are needed, or (ii) collected data are not sufficient to train the desired model, requiring the data analyst to ask for different features (following a trial-and-error approach). Data processing tools support the repeatability of the task and offer support for data transformations and extensions (for extensions, the user is

6 https://developers.google.com/adwords/api/docs/appendix/geotargeting
7 http://www.geonames.org/

required to write a short code script to query a service, e. g., via API, and post-process the response). However, the reconciliation is mostly in charge of the user, which manually creates the mappings of interest between the involved datasets. Furthermore, the people involved in the process have to find the needed information on their own. For example, none of them know in advance which datasets can be queried by using GeoNames identifiers, if mappings between Geotargets and GeoNames are available, and so on.

The example we provided here motivates the design of a scalable solution, which provides users with a tool to (i) design transformation pipelines on tabular datasets that include data enrichment, helping users to find the right information, and (ii) record and manage these pipelines in a repeatable form over large amounts of data. We believe that all the people involved in data science projects could benefit from this solution because, at the same time, (i) it eases the enrichment task, and (ii) it makes the enrichment process repeatable in a scalable environment, reducing the overall data processing time. Besides, the gap between data engineers and analysts will be bridged: a single figure, potentially a data scientist or a data worker, will be able to both design and deploy the transformation process, avoiding the continuous handover between engineers and analysts. In Chapter 4, we will introduce a methodology and a novel system that employs such methodology that effectively and efficiently supports this scenario.

1.2 CONTRIBUTIONS

In this thesis, we focus on exploiting the semantics to support the data enrichment task at scale, thus providing a methodology to support the *semantic data enrichment*. We leverage and take existing work on STI a step forward; we argue that STI can provide a valuable paradigm to support data enrichment, modularly and at scale, in a broad number of scenarios. Also, we focus on how to evaluate STI approaches when employed in scalable solutions and how to better include the human in the data enrichment process.

More in details, in this thesis, we introduce:

1. The definition of semantic data enrichment, i. e., supporting the data enrichment by exploiting the semantics. Based on our definition, we further introduce:
    - A comprehensive methodology to provide data workers with suitable tools to (i) interactively design transformation pipelines on datasets in tabular format, including semantic enrichment using curated KBs (general-purpose or domain-specific), and (ii) deploy and run such pipelines against massive datasets.

- An interactive framework that modularly supports the definition of semantic data enrichment pipelines and their execution at scale. With *modularly*, we mean that the paradigm can be implemented by an ecosystem of services that provide access to different target datasets to support automatic data reconciliation and extension. Automation is a decisive factor for managing large volumes of data and reaching the *at scale* dimension under certain assumptions.

2. A new benchmark dataset to evaluate STI algorithms at a more significant scale.

3. A methodology to adapt existing STI approaches in such a way to better dealing with inaccurate inputs given by non-semantic-expert users.

We will use (1) to demonstrate the suitability of our proposed methodology by creating both general-purpose and specialized services to support data analytics projects. Since benchmark datasets for STI mainly consist of several small tables, we propose (2) to evaluate STI algorithms in a different context, i. e., when large tables have to be annotated, which better resembles the data enrichment scenario. Lastly, since the proposed framework is interactive and targeted to data workers, we propose (3) to improve the interaction with data workers, as well as to better deal with the information entered by people not expert in semantics technologies, in such a way to adapt existing STI approaches and improve their results.

1.3 THESIS STRUCTURE

The structure of this document is illustrated in the following, with pointers to the relevant publications. The order of the Chapter is to better introduce the reader to the different aspects of the semantic data enrichment, and does not reflect the importance of the contributions; for this reason, we highlight the key publications with the ♦ symbol.

Chapter 2: Preliminaries. This Chapter is devoted to explaining of the basic notions and definitions needed to thoroughly understand the rest of this thesis. We describe fundamentals of KBs, then we focus on the STI task and discuss how STI approaches can be exploited to support the data enrichment.

Chapter 3: Related Work and Contributions. We explore related approaches in the semantic annotation and data enrichment fields, with a specific focus on tabular data. We also discuss alternative approaches to the instance matching problem.

Chapter 4: A Semantic Table Annotation Approach to Large-Scale Data Enrichment. In this Chapter, we outline a methodology for interactively assisting data workers in enriching tabular data at a large scale. We illustrate a system that employs such methodology, and we showcase an experiment conducted in a real-life business application. The research questions answered by this Chapter are:

Q4.1: How can semantics be exploited to support the data enrichment?

Q4.2: Are the existing STI approaches executable in a Big Data environment (to support the enrichment of massive datasets)?

Q4.3: Is it possible to consider the users in the enrichment process in such a way to use their feedback to improve the system performance?

The methodology is also described in an upcoming book chapter:

> ♦ Michele Ciavotta, Vincenzo Cutrona, Flavio De Paoli, Nikolay Nikolov, Matteo Palmonari, and Dumitru Roman. "Supporting Semantic Data Enrichment at Scale." In: *Technologies and Applications for Big Data Value*. (To appear). 2021

The experiment we conducted using our system, as well as the application to support the data workers in the enrichment process, have been published in the following articles:

> ♦ Vincenzo Cutrona, Flavio De Paoli, Aljaz Kosmerlj, Nikolay Nikolov, Matteo Palmonari, Fernando Perales, and Dumitru Roman. "Semantically-Enabled Optimization of Digital Marketing Campaigns." In: *ISWC*. Vol. 11779. Springer, 2019, pp. 345–362

> Vincenzo Cutrona, Michele Ciavotta, Flavio De Paoli, and Matteo Palmonari. "ASIA: a Tool for Assisted Semantic Interpretation and Annotation of Tabular Data." In: *ISWC Satellite Tracks*. Vol. 2456. CEUR-WS.org, 2019, pp. 209–212

The same application has been used in a challenge to publish tabular data on the Web as Linked Data:

> Shady Abd El Kader, Nikolay Nikolov, Bjørn Marius von Zernichow, Vincenzo Cutrona, Matteo Palmonari, Brian Elvesæter, Ahmet Soylu, and Dumitru Roman. "Modeling and Publishing French Business Register (Sirene) Data as Linked Data Using the euBusinessGraph Ontology." In: *Joint Proceedings of the International Workshops on Sensors and Actuators on the Web, and Semantic Statistics, SemStats@ISWC*. Vol. 2549. CEUR-WS.org, 2019

Chapter 5: Evaluating Entity Linking for Tables. In this Chapter, we discuss the limitation of current benchmark datasets for the STI related tasks. We thus introduce a novel dataset, which solves some of the current limitations and supports the scalability evaluation of existing STI algorithms. The research questions answered by this Chapter are:

Q5.1: Do existing STI algorithms perform the same on different kinds of tables?

Q5.2: Which are the most difficult challenges to be solved by a STI algorithm?

The dataset has been published in the following work:

♦ Vincenzo Cutrona, Federico Bianchi, Ernesto Jiménez-Ruiz, and Matteo Palmonari. "Tough Tables: Carefully Evaluating Entity Linking for Tabular Data." In: *ISWC*. Vol. 12507. Springer, 2020, pp. 328–343

An adapted version of the same dataset has been used in the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, showing that our new dataset poses new challenges often overlooked by existing algorithms. The results have been published in the challenge proceedings:

Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Kavitha Srinivas, and Vincenzo Cutrona. "Results of SemTab 2020." In: *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab@ISWC*. Vol. 2775. CEUR-WS.org, 2020, pp. 1–8

Chapter 6: Improving Entity Linking for Automatic and Assisted Table Annotation. In this Chapter, we discuss how the incomplete entity typing information used within STI approaches can be managed using soft constraints. Thus, we introduce a neural-based methodology to adapt almost every existing STI algorithm that exploits the entity typing within the filtering or ranking phases. The same approach helps STI approaches to better deal with inaccurate type information provided by a non-expert user. The research question answered by this Chapter is:

Q6.1: Is it possible to mitigate the incompleteness of the type information within STI approaches?

The content of this Chapter has been submitted for publication to an upcoming conference:

♦ Vincenzo Cutrona, Gianluca Puleri, Federico Bianchi, and Matteo Palmonari. "NEST: Neural Soft Type Constraints to Improve Entity Linking in Tables." In: *ESWC*. (Under revision). 2021

The methodology exploits a particular type representation introduced is partially based in a previous article:

> Federico Bianchi, Mauricio Soto, Matteo Palmonari, and Vincenzo Cutrona. "Type Vector Representations from Text: An Empirical Analysis." In: *Proceedings of the First Workshop on Deep Learning for Knowledge Graphs and Semantic Technologies, DL4KGS@ESWC*. Vol. 2106. CEUR-WS.org, 2018, pp. 72–83

Chapter 7: Conclusions. Finally, we conclude this thesis by summarizing the most relevant contributions and insights for future research directions.

1.4 REPRODUCIBILITY

In this thesis, we follow the FAIR (Findable, Accessible, Interoperable, and Reusable) guidelines [121] to release our contributions. We think that reproducibility has become a significant issue, and moreover, publicly releasing all the resources will favor further extensions of our work by the community. In the following, we provide the list of repositories that contain our resources, alongside instructions on how to use them:

- Asia (frontend)

  [code] `https://github.com/datagraft/grafterizer-2.0`

- Asia (backend)

  [code] `https://github.com/UNIMIBInside/asia-backend`

  [code] `https://github.com/UNIMIBInside/conciliator`

  [code] `https://github.com/UNIMIBInside/ASIA-suggester`

  [code] `https://github.com/UNIMIBInside/asia4j`

- 2T

  [code] `https://github.com/vcutrona/tough-tables`

  [dataset] `https://doi.org/10.5281/zenodo.3840646`

- Nest

  [code] `https://github.com/vcutrona/nest`

2 PRELIMINARIES

In the following Chapter, we introduce the reader to the fundamental notions and concepts required to access the rest of this work. We will start by covering fields such as knowledge representation and semantic table annotations, then showing how they are exploited to support the semantic data enrichment.

2.1 KNOWLEDGE BASES

The concept of knowledge demands a systematic function, or system of functions, which explain that relation in which truth appears to consist, between the human intelligence on the one hand and fact or reality on the other [9]. Our world contains several individuals (e. g., people, places), which are related to each other, and we need to give formal representations of these individuals if we want to use them in computational settings. Logic defines means to describe individuals and properties, thus representing a useful tool to describe knowledge formally. Logic embeds the notions of consequences and inferences, thus enabling reasoning over defined properties, and infer new knowledge. Among the family of artificial languages to describe general properties, First-Order Logic (FOL) has served as the paradigm for modern logical systems and their meta-theoretical study [43], but also other logical approaches have been widely used to represent knowledge, like Description Logics [81].

FOL allows us to describe the knowledge about a specific domain; using FOL, we represent domain objects as *individuals*, and we describe properties of the individuals by mean of *predicates*.

KBs are collections of resources described in such a way to enable inferential reasoning over them, where the reasoning is based on the logical consequence. We observe that when we only consider binary predicates, i. e., we have only binary relations between individuals, the KB resembles a graph; we refer to this particular KB as a Knowledge Graph (KG).

2.1.1 *Knowledge Graphs*

A KG represents *entities* of the real world as vertices, while *predicates* between entities are represented as edges. Statements about world facts are represented as *triples* $\langle \mathtt{subject}, \mathtt{predicate}, \mathtt{object} \rangle$ in a KG, e. g., the fact "Albert born in Ulm" is represented with the triple $\langle \mathtt{Albert}, \mathtt{birthPlace}, \mathtt{Ulm} \rangle$. A commonly-agreed formal definition of

KG is missing in the literature, while several conflicting definitions emerged over the past years. In the following, we propose a simplified definition that contains the building blocks needed to understand the remainder of this work. Since the entities are core to the enrichment task we discuss in this thesis, we prefer to distinguish from identifiers of entities and literals. We refer to recent work in the literature for complete definitions and more details about the evolution of the KGs [47]. Before giving our definition of KG, we introduce the reader to the concept of *ontology*. An ontology is a *concrete, formal representation of what terms mean within a given domain* [47]. Thus, we can interpret an ontology as a shared *vocabulary* that proposes terms to capture a particular convention tailored to a specific domain. Even if an ontology is a set of axioms, we provide a simplified definition that focuses on type assertions, which are the only ones we will use in the rest of this thesis.

Definition 1 (Ontology) *An ontology $\mathcal{O}$ is defined as a tuple $\mathcal{O} = \langle C, \subseteq_C \rangle$ where:*

- C *is a set of* concepts, *often referred to as* classes *or* types, *e.g.,* Person, City, Place*;*
- $\subseteq_C \subseteq C \times C$ *is the* subclassOf *relation between concepts in* C, *e.g.,* City $\subseteq$ Place*; it is transitive, reflexive and anti-symmetric, and defines a partial order over* C.

Definition 2 (Knowledge Graph) *Given an ontology $\mathcal{O}$, a Knowledge Graph $\mathcal{G}$ is defined as a tuple $\mathcal{G} = \langle \mathcal{O}, E, S, \in_C, R_E, R_S \rangle$ where:*

- E *is the set of the* entities *that represent domain objects, e.g.,* Albert, Ulm, Germany*;*
- S *is the set of the* literals *that represent values of entities attributes, e.g.,* 14/03/1879*;*[1]
- $\in_C \subseteq E \times C$ *is the* type *relation that holds between an entity and a concept, e.g.,* Ulm $\in_C$ City, Albert $\in_C$ Person*;*
- $R_E = \{r | r \subseteq E \times E\}$ *is a set of* object properties, *i.e., logical relations that hold between entities, e.g.,* Albert birthPlace Ulm*;*
- $R_S = \{r | r \subseteq E \times S\}$ *is a set of* datatype properties, *i.e., logical relations that hold between entities and literals, e.g.,* Albert birthDate 14/03/1879.

Figure 2.1 depicts an example of KG. Since 2012, many companies have announced the development of a KG as the core component to represent data and knowledge; among the multitude of companies,

1 Literals may be assigned *datatypes*. We do not consider this aspect in our simplified definition.

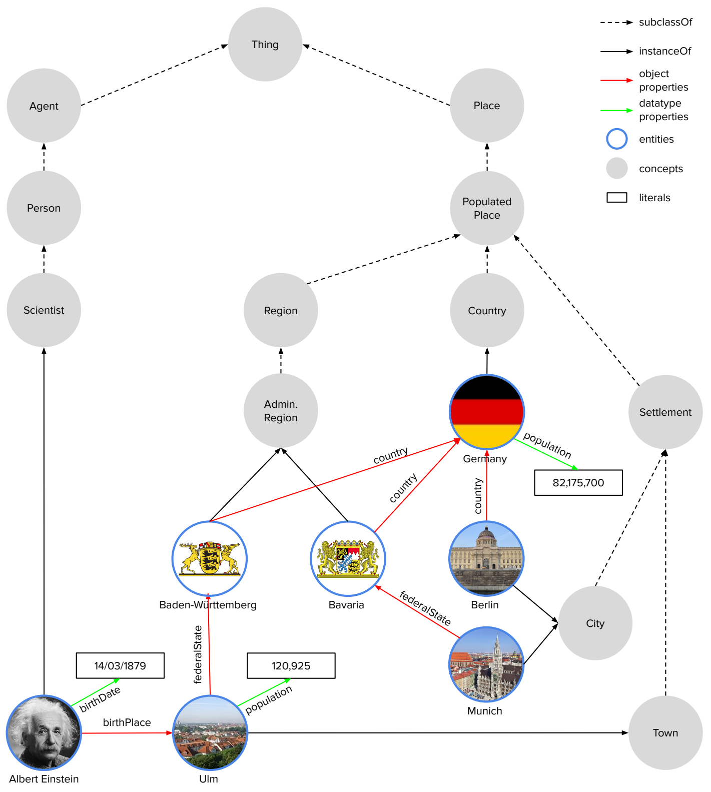


Figure 2.1: An excerpt of the DBpedia KG (images from Wikipedia).

we cite tech giants like Google, Amazon, eBay, and Facebook, which used KGs to integrate and extract value from diverse sources of data at a large scale [47].

2.1.2 *Semantic Web*

The World Wide Web has been conceived and has grown as a medium of documents for people rather than information that can be manipulated automatically [3]. In 2001, Tim Berners Lee illustrated how to transform the Web into the *Semantic Web* by augmenting Web pages with data targeted at computers. Nowadays, thanks to the Semantic Web, we have that artificial agents are able to participate in shared *information space* and help users communicate with each other. The *data targeted at computers* are a key point: in fact, it is not sufficient that the data are well structured and have an exact meaning to humans, like database dumps, to let a machine understand the implications of those data; thus, the Semantic Web introduced new languages to describe data and structured representations, so that enabling machines to reason about the data, and infer new knowledge.

The core components of the Semantic Web are the Resource Description Framework (RDF), a language to describe *resources*, and SPARQL, a query language to query resources represented in RDF. Similarly to

FOL, RDF allows us to represent factual statements about resources through simple binary relations, where individuals are represented with Universal Resource Identifiers (URIs) that uniquely identify the resources across the Web; relations are expressed with the use of RDF *assertions*, i. e., triples structured as $\langle \mathtt{subject}, \mathtt{predicate}, \mathtt{object} \rangle$.

RDF in itself is sufficient to achieve the main goal of the Semantic Web, i. e., the interoperability: resources are uniquely identified, and the *meaning* of the resources is described using a machine-readable language. In different words, we have *annotated* the resources by using their respective URI, and eventually, we linked them using a predicate. Following this approach, we can annotate and share whatever resource in the Web: datasets, people, Web pages, applications, and data that they exchange each other.

The Semantic Web anticipated the popularity of KG representations when logic-based languages had been proposed. Indeed, on top of RDF, we can use in fact more expressive languages that enable reasoning over the resources. RDFS and OWL are two languages that offer different models to describe the data, and enable logical reasoning (e. g., given a set of triples, we infer new knowledge by applying entailment rules[2] in RDFS, or by executing Description Logic reasoners[3] in OWL).

Data described using both RDF and RDFS (or OWL) represent a KG. Throughout the years, structured conceptualizations of knowledge have arisen with the advent of the Semantic Web, with KGs spreading as the most preferred structure. The advantage of using RDF to describe entities in a KG is that entities become resources identified by a URI; in this way we can assert that a resource in a specific KG is related with a resource in a different KG (e. g., we may assert that two resources are equal), creating the so-called Linked Data [8]. The most prominent example of interconnected KGs is the Linked Open Data cloud, a resource that currently counts 1269 datasets, connected with 16201 links (as of May 2020).[4]

2.1.3 *Semantic Gap*

A considerable amount of information, either published on the Web or stored in private repositories, is still not compliant with the Semantic Web principles, generating the *semantic gap*, i. e., a gap between the coverage of structured and unstructured data [75]). This gap calls for semantic annotation techniques, a variety of methodologies able to add a semantic layer to unstructured data in order to make the semantic explicit; in the literature, many techniques have been proposed for annotating almost every resource, including text [72], images [98], Web pages [124], outdoor trajectories [27], and geospatial data [113].

2 `https://www.w3.org/TR/rdf11-mt/#rdfs-entailment`
3 `https://www.w3.org/2001/sw/wiki/Category:OWL_Reasoner`
4 `https://lod-cloud.net/#about`

In the last years, a particular focus has been dedicated to tabular data, since the tabular format is one of the most used formats for storing organized information; in fact, tables are fact easy to understand by humans due to their simple structure, and at the same time allow users to store informative content easily. Also, tabular data, especially in the form of CSV files, is the typical input format in data analytics pipelines. In 2008, at the time when unstructured documents mainly populated the Web, at least 14.1 billion HTML tables were published on the Web, and among them, 154 million were in a "database-like" format [14]. In 2015, approximately 90 million relational tables were extracted from the CommonCrawl Web Table Corpus [66].

The large availability of tabular data led to a growing research interest in exploiting such amount of valuable data for many varied table-centric applications [13], like data search for query answering [12, 86, 97, 116], table extension [4, 12, 67, 103, 104] and completion [1], and KG population [36, 105, 125].

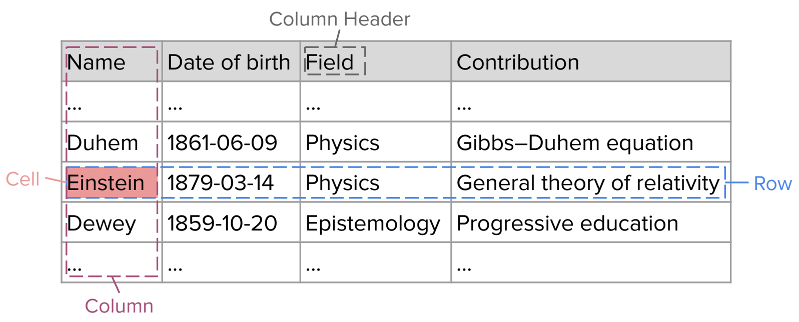
The lack of understanding of the semantic structure and meaning of the data content may represent a hurdle for most of the above applications. Gaining the semantic understanding requires to *interpret* the content of tabular data, i. e., to match the table content against a known KG, or to interpret the table schema into a known ontology. The task of interpreting tables with resources described in a reference KG is referred to as STI in the literature. STI algorithms are in charge of solving different problems, like to detect the subject column of a table (i. e., the column that describes the *main entities* of the table), to understand the semantic type of columns, to disambiguate values in cells, and to find relationships between values in columns.

2.2 SEMANTIC TABLE INTERPRETATION AND ANNOTATION

An STI approach exploits the semantics to make explicit the meaning of values in tables. Even if different types of tables exist, we focus on simple tables like the one depicted in Figure 2.2, where the first row is the *header row*, containing column headers), and the other rows contain the actual data. Examples of simple tables are CSV files and relational database tables.

Given a table and a reference KG as inputs, the output of an STI algorithm is a set of annotations that make explicit the semantics of the table, i. e., annotations that map the source table to the KG. Thus, we consider the Semantic Table Annotation (STA) as the result of STI. The two tasks are strictly related, because different strategies in STI will lead to different annotations. We can distinguish between two different levels of annotation:

- *Instance-level annotation*, which is mainly covered by entity linking algorithms for tables; the instance-level annotation maps the


Column Header
Cell
Row
Column

Name	Date of birth	Field	Contribution
...	...	...	...
Duhem	1861-06-09	Physics	Gibbs–Duhem equation
Einstein	1879-03-14	Physics	General theory of relativity
Dewey	1859-10-20	Epistemology	Progressive education
...	...	...	...

Figure 2.2: A simple table that contains data about philosophers of science.

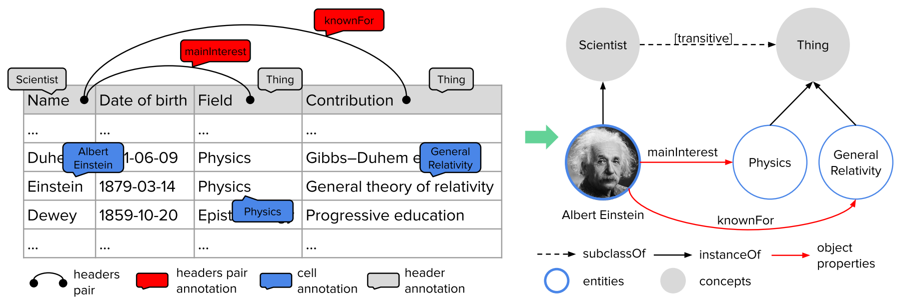
disambiguated content of the table to entities in the reference KG. The most used approaches to the instance-level annotation are:

- The cell-based annotation (see Figure 2.3a), where the content of a cell is mapped to an entity (e. g., given DBpedia as the reference KG, the cells containing the labels *Einstein* and *General theory of relativity* in Figure 2.2 can be mapped to the entities `Albert Einstein` and `General relativity`, respectively).
- The row-based annotation (see Figure 2.3b), where a table row is mapped to an entity (e. g., the third row depicted in Figure 2.2 may be mapped to the entity `Albert Einstein`; this strategy usually follows the assumption that a row refers to only one entity (*one-entity-per-row* assumption).

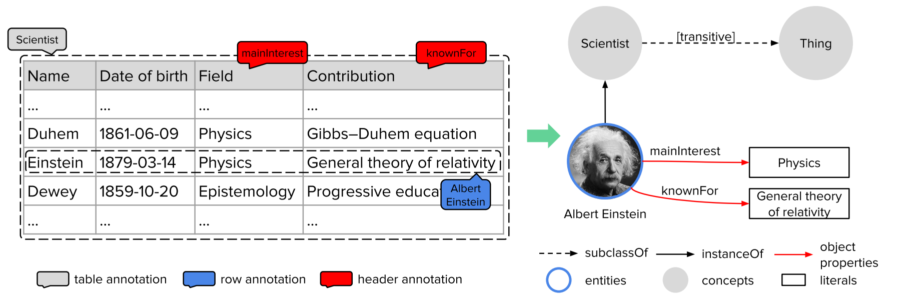
• *Schema-level annotation*, which results from the interpretation of the table schema, thus it maps elements in the table schema to types and properties described in the reference KG. Sometimes the schema-level interpretations must deal with more challenging tables where the header row is missing or contains meaningless column headers (e. g., codes from legacy sources).[5] An annotation strategy may map column headers to classes (e. g., the column header *Name* in Figure 2.2 could be mapped to the class `Philosopher` in DBpedia), and between pairs of column headers and properties (e. g., the column headers pair ⟨*Name*, *Field*⟩ in Figure 2.2 could be mapped to the property `mainInterest`). A possible annotation strategy is to map the whole table to a type from the ontology and map every column header to a property from the ontology when working under the *one-entity-per-row* assumption.

As depicted in Figure 2.3, different annotations may result from the same STI process, which in turn lead to two different KGs in output, depending on the employed strategy. In the following, we refer to

5 We assume a table always to have a header row; when missing, the header row is created with dummy column headers.



(a) Cell-based annotation.



(b) Row-based annotation.

Figure 2.3: Different annotations resulting from the same interpretation (images from Wikipedia).

STI approaches as to all the approaches that automatically solve the STA by interpreting a table, while we refer to STA approaches as to the approaches that do not interpret the table (e. g., approaches that support a user in defining the schema-level annotation).

In 2019, the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab) was organized to standardize the definition and evaluation of STI [54]. According to the terminology introduced in SemTab, an STI approach delivers three possible annotations:

- Cell Entity Annotation (CEA), that is the instance-level annotation (cell-based);
- Column Type Annotation (CTA), that is the schema-level annotation focused only on linking columns to ontology classes;
- Column Predicate Annotation (CPA), that is the schema-level annotation focused only on finding binary relations between columns, i. e., linking pairs of columns to ontology properties. Currently, this task is not intended to solve the general case of n-ary relations, which is instead useful in many applications [65].

An STI approach may produce one or more of the above annotations. Depending on the specific application (e. g., fully-automated matching, or interactive matching), the annotation tasks may be addressed together, so that they can mutually inform each other. For example, CEA can provide evidence for CTA and CPA, i. e., linked entities are exploited to infer the right class for annotating the columns and checking the existing properties between them, while CTA may help the disambiguation in CEA, e. g., by narrowing the set of acceptable types for entities. In some cases, it is not required to deliver all the annotations, e. g., when the application's goal is not to build a new KG, CPA may be useless.

Approaches that address all the STI tasks usually implement complex pipelines to calibrate how the evidence is collected and propagated across tasks. Usually, algorithms that solve different matching tasks face different problems; for this reason, in Chapter 3, we will discuss instance- and schema-level annotation algorithms separately.

2.2.1 *Instance-level Annotation*

Given a table and a reference KG, an entity linking algorithm aims at disambiguating textual values in table cells, referred to as *labels*, after retrieving a set of candidate entities from the reference KG. Labels are expected to mention (or, refer to)[6] some entities, and the goal of a matching algorithm is to *disambiguate* the mentions, i. e., to recognize if a label is a *mention* of an entity described in the reference KG; if this is the case, then the cell is annotated with the mentioned entity. For each label to disambiguate, the algorithm decides whether to annotate the cell with an entity from the KG or leave it not linked.

We refer to the cell to annotate as i, j, which is the cell in the ith row and jth column, which we indicate as R_i and C_j, respectively. Since the label is not sufficient to disambiguate the mentioned entity in almost the cases, STI algorithms exploit the structure of the table by extracting contextual information from C_j and R_i, thus looking at values occurring on the same row or column of the cell to annotate to support the disambiguation. We refer to the column C_j as to the *entity column* and all the other columns as *context columns*; similarly, we refer to the row R_i as to the *entity row* and all the other rows as *context rows*. We refer to the label in the cell to annotate as to $label_{i,j}$, while for the sake of clarity we will use $value_j$ and $value_i$ to refer to the content of cells in the jth context column or the ith context row, respectively.

If e is an entity identifier assigned to the cell i, j, then $e\ P\ value_j$ or $value_j\ P\ e$ are facts in the reference KG that relate e to $value_j$ with a property P, according to the formalization provided in Chapter 2. For $e\ P\ value_j$, $value_j$ can be an entity identifier, a type identifier or any literal, while for $value_j\ P\ e$, $value_j$ must be an entity identifier.

Algorithms for entity linking in tables usually combine three primary operations into complex matching pipelines:

6 We use the term "mention" as it is used for entity linking in textual documents.

1. *Candidates retrieval*, where some value in the table, usually the label of the cell to annotate, is matched against KG facts; this operation returns a - possibly empty - set of candidate entities, i. e., entities that are potential candidates for the annotation.

2. *Ranking*, where candidate entities are ranked according to some criterion, which may combine matching scores, e. g., the score given by the retrieval function, other similarity scores, filters, and more sophisticated mechanisms.

3. *Decision making*, where the collected evidence supports the decision of whether to link or not the label and, in the first case, which entity to consider as the final annotation.

The combination of ranking and decision making is the core of a disambiguation algorithm. Filters over candidates and scoring used in ranking can contribute to decision making: if after filtering the ranked list of candidates is not empty, the top candidate can be selected for the annotation, otherwise, the cell is not annotated. Other decision-making strategies can use thresholds; however, it is difficult to apply thresholds over scores that are not bounded, typical for scores returned by matching functions powered by search engines and available *lookup* services. On the other hand, search engines offer very efficient search over the vast amount of information stored in KGs. As a result, *lookup services* usually combine string similarity, document frequencies (e. g.Lucene-based scores), and even other aspects like popularity (the DBpedia Lookup Service[7] exploits entity popularity measures, i. e.in-links pointing at the candidate). These considerations make filtering and scoring particularly relevant.

2.2.2 *Schema-level Annotation*

Given a table and a reference KG, the schema-level annotation is the task devoted to mapping the table's schema to the graph-based schema of the KG, i. e., its ontology. Often this task is addressed downstream the instance-level annotation, thus getting also annotated entities as input. Each column is classified into two different classes:

- *Named entity column*, a column containing some values that are mentions of entities; columns *Name*, *Field* and *Contribution* in Figure 2.2 are named entity columns.

- *Literal column*, which features plain values that do not mention entities; the column *Date of birth* in Figure 2.2 is a literal column containing dates.

7 https://wiki.dbpedia.org/lookup

For each column, the algorithm decides whether to annotate named entity column with one or more classes from the KG ontology,[8] or leave it not annotated. Eventually, the algorithm may assign a datatype to literal columns.[9] We refer to the column to annotate as C_j, which is the jth column of the table. Moreover, the algorithm may detect the *subject column* of the table among the named entity columns. The subject column is the column that contains the main entities of the table; as an example, the table in Figure 2.2 contains data about philosophers, thus its subject column is the column *Name*, which contains mentions of philosophers. When the one-entity-per-row assumption holds, the only named entity column of the table is also its subject column.

The schema-level annotation also embraces the task of finding relations between columns. In this case, we refer to the columns to annotate as the ordered pair $\langle C_j, C_k \rangle (j \neq k)$. For each pair of columns, the algorithm decided whether to annotate the pair with one or many properties from the KG or leave it not annotated. Eventually, a direct property between two columns may be missing in the reference KG, thus requiring to consider *property paths*[10] as possible annotations: for example, the property `nationality` is almost equivalent to the *property path* `birthPlace/country`. Moreover, a column may represent complex ontology mappings, e. g., a measurement with a unit of measure, or many columns should be considered at once to be mapped to a single property, e. g., the *Name* and *Surname* columns are both targets of the `fullname` property. How to deal with these particular cases is an open issue not covered yet by the SemTab standardization process.

2.3 SEMANTIC ENRICHMENT OF TABULAR DATA

Advancements in data science technology have made it possible to analyze massive amounts of data and develop highly accurate data analytics models. Among them, predictive analyses are usually based on machine learning models; decision tree, random forest, neural network, and deep neural network are just a few examples of the vast number of machine learning models available nowadays.

Machine learning models require a massive amount of collected data to achieve acceptable confidence in making predictions. The data are usually collected as (or transformed to) tabular data, a data format that allows users to feed machine learning models easily. However, many analyses are focused on studying variables that may be missing in the collected data; for example, it is not possible to study the weather

8 Multi-typing is encouraged in some KGs, e. g., schema.org (`https://schema.org/additionalType`).

9 In this thesis, we focus on named entity columns, which are the most relevant for the semantic data enrichment and the only ones considered in the CTA task defined by SemTab.

10 `https://www.w3.org/TR/sparql11-property-paths/`

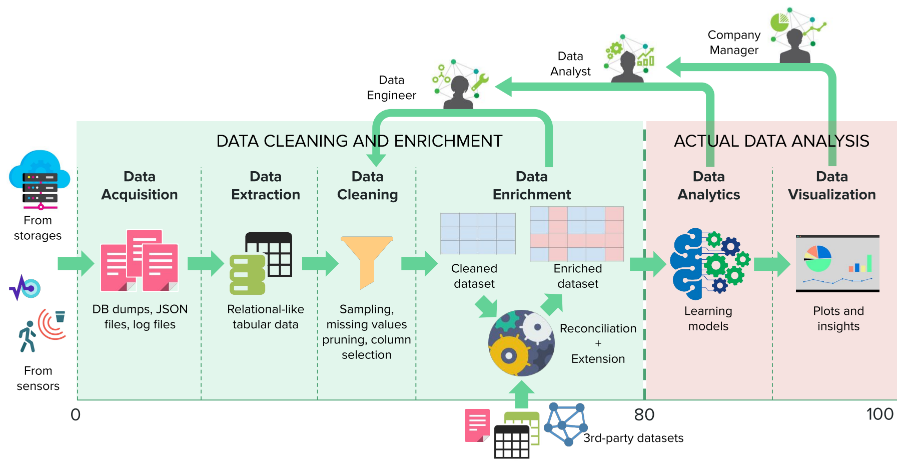


Figure 2.4: Infographic representing the main stages of a data project and the related stakeholders.

impact over digital marketing campaigns' performance if we only collect data about campaign performance indicators.

Thus, we are in a scenario where we have a dataset that we need to *enrich* with data stored in external sources of information. We refer to this task as *data enrichment*, a specific data integration problem, in which a source dataset is extended with additional data from external sources. While the semantic of the source dataset is known to the user, the semantic of the external sources is usually unknown. The problem is usually part of the data preparation step, where a *data transformation pipeline* is built to combine different sources of information manually. Nonetheless, figuring out how to combine different datasets is a cumbersome task, because in almost the cases, the external data sources are unknown to the user.

Numbers support the evidence about the effort needed to achieve this task: a typical data science project is mostly focused on the data preparation stage (Figure 2.4), which takes up to 80% of the time required by the project for cleaning enriching the data. Only the remaining 20% is spent on data analysis [70].

This issue is now widely recognized and requires appropriate tools and methodologies because it will worsen as data volume and variability increase [45]. In this panorama, semantic techniques can support data enrichment. We discussed in Section 2.1 how the Linked Open Data (LOD) cloud is a source of valuable information (high-quality and well-maintained data), which is represented in such a way to increase the interoperability. A lot of this information is open and accessible, but few attempts to use it to enrich tabular data have been made, mainly due to the specific requirements of the task. The *semantic data enrichment* harnesses the semantics to support the user in the enrichment task by adding two additional building blocks to the data transformation pipeline:

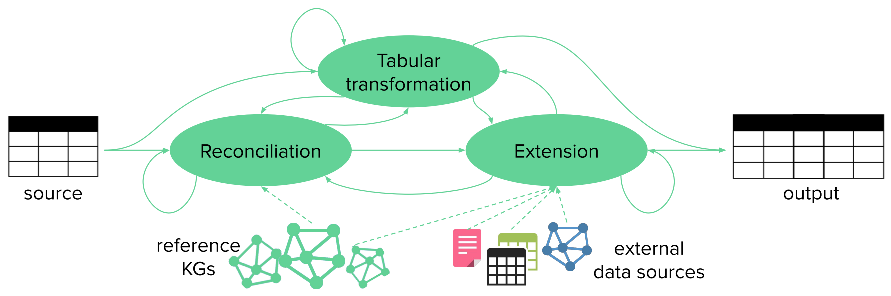


Figure 2.5: The data transformation graph.

- The *reconciliation*, which maps tabular data to a pool of reference KGs; the core component of the reconciliation is the entity linking, which enables the next task;
- The *extension*, where the identifiers of linked entities are used to get data from different external data sources, by exploiting the relations between entities available in Linked Data.

Given these two new steps, we define a data transformation graph G^T as the one depicted in Figure 2.5, and we say that a semantic data enrichment pipeline is a path on G^T where at least one node is a reconciliation step. As an example, consider the dataset mentioned above, containing marketing performance indicators. We suppose this dataset also contains geographical information, such as the city to which the performance indicator is related. A simple semantic data enrichment pipeline first reconciles our dataset against a geospatial KG, e. g., GeoNames, then relies on locations identifiers to retrieve the information needed to query external weather providers, e. g., ECMWF. The key idea in the semantic data enrichment is to find a *shared space of identifiers* between the source and the target dataset; in our example, the shared space of identifiers is represented by WGS84 coordinate pairs. When needed, tabular transformations are available to change the format of the data.

Current approaches do not provide comprehensive solutions to the problem of semantic data enrichment, with many challenges still to be faced, including: (i) the democratization of structured knowledge, which is difficult to access for non-expert people; (ii) the creation of enrichment pipelines that involve various data sources, potentially from third parties; (iii) the development of tools to ease the design and execution of semantic enrichment pipelines, also considering the user in the loop.

On the one hand, there are plenty of tools that support the data pipeline life-cycle, which are suitable for users familiar with programming languages and process definition [108]. These tools usually offer a variety of configurable components to create data pipelines, but are unsuitable for non-technical data workers with only spreadsheet-level of knowledge; besides, these solutions do not allow the users to incorporate their specific knowledge, which is crucial to perform the data reconciliation task effectively.

On the other hand, some tools provide users with easy-to-use functionalities for data preparation, offering support to the semantic enrichment only to a limited extent. These solutions are mainly suitable for the exploratory phases of a project, without supporting the life-cycle management and guaranteeing automation and scalability.

Consequently, we observe a working environment where different actors, which have different knowledge backgrounds, are forced to use different tools that are not compatible with each other: domain experts design the data transformations using dedicated tools. In contrast, engineers have to rewrite the data transformations in such a way as to deploy them into a production environment. The knowledge gap between these two groups causes delays in the development and issues in maintaining the solutions, calling for a unified framework to support users in designing semantic data enrichment pipelines and deploying them in an effective and scalable way.

3 RELATED WORK AND CONTRIBUTIONS

Supporting both the design and execution stages of a data enrichment task demands to address different topics that have been investigated in the literature. Two main fields are relevant to semantic data enrichment:

- *Semantic Annotation*, which encompasses all the algorithms that add a semantic layer to the input data so that artificial agents can understand the content of the input data;
- *Data Enrichment*, which focuses on applying transformations to different datasets to integrate them, paying particular attention to efficiency.

In this Chapter, we provide a snapshot of the most recent advancements relevant to these two fields, with a specific focus on tabular data, and then we discuss how we take the best from each field to support the semantic data enrichment both effectively and efficiently.

3.1 SEMANTIC ANNOTATION OVERWIEW

In the last decade, the important role of the table annotation has been discussed in several works. We hereby discuss different semantic annotation approaches for tabular data proposed in the literature (Table 3.1); we refer to a recent survey [73] for the discussion of semantic annotation approaches in different contexts and applications.

3.1.1 *Instance- and Schema-level Annotation*

In the context of tabular data, CEA is the task of annotating the entities mentioned in a table against a reference KG. In the last years, different algorithms and systems have been proposed to achieve this task. A typical methodology in the literature is to couple the CEA task with the CTA and CPA tasks, since solving the three tasks collectively gives accuracy benefits, compared to making local decisions [69]. In this Section, we discuss the algorithms that exploit the cell-entity annotations to find the type of each column and the relations between columns.

In [69], entities and classes to link are modeled as random variables, then evaluated jointly in a probabilistic graphical model. The approach assumes that a column may be annotated with more classes, resembling many concrete situations. Candidate entities are obtained and

Table 3.1: Comparison between semantic annotation approaches.

Approach	**CEA**	**CTA**	**CPA**	**One-entity-per-row**	**Supervised (S) / Unsupervised (U)**	**Interactive**
Limaye et al. [69]	✓	✓	✓		S	
Mulwad et al. [79]	✓	✓	✓		S	
Ritze et al. [102]	✓	✓	✓	✓	U	
Zhang et al. [126]	✓	✓	✓	✓	U	
Zhang [127]	✓	✓	✓	✓	U	
Efthymiou et al. [38] (FactBase)	✓	✓	✓	✓	U	
Eslahi et al. [41] (Lookup-based)	✓	✓	✓	✓	U	
Cremaschi et al. [22]	✓	✓	✓		U	
Ermilov et al. [40] + Usbeck et al. [114]	✓		✓	✓	S	
Kruit et al. [61]	✓		✓	✓	U	
Bhagavatula et al. [5]	✓				S	
Efthymiou et al. [38] (Embeddings)	✓			✓	U	
Efthymiou et al. [38] (Ontology Matching)	✓			✓	U	
Eslahi et al. [41] (Looping)	✓			✓	U	
Luo et al. [71]	✓				S	
Deng et al. [33]	✓				S	
Knoblock et al. [59]		✓	✓		U	✓
Taheriyan et al. [111]		✓	✓		S	✓
Pham et al. [96]		✓	✓		S	✓
Deng et al. [32]		✓			U	
Cruz et al. [25]		✓	✓		U	
Quercini et al. [99]		✓			U	
Ramnandan et al. [100]		✓			S	
Chen et al. [19]		✓			S	
Hulsebos et al. [49]		✓			S	
Zhang et al. [123]		✓			S	

ranked according to the TF-IDF similarity between the cell label, and the entity label; candidate classes are obtained and ranked in the same way, comparing column headers with class labels. Variables related to the classes have a lower impact in the probabilistic model than the variables for entities, because column headers may be omitted, or not match any type. Relations are also modeled as variables of the probabilistic model, but in this case, the feature represents the compatibility between pairs of types assigned to a pair of columns and binary relations in the reference KG. All the variables are jointly considered in a generalized Support Vector Machine classifier, which learns their weights.

A different approach is proposed in [79], where a list of ranked entities is collected for each column by querying Wikitology, a hybrid KB that contains data from Wikipedia, DBpedia, YAGO, Wordnet and Freebase. The rank is based on the entities popularity (PageRank), which is a good indicator in Web tables, where the most popular entity is often the correct answer in ambiguous cases. A score is then assigned to each class as the weighted sum of the ranks of entities of that class. Finally, the column is annotated with the class with the highest score. The predicted classes are used to refine the results, then Wikitology is queried again by restricting the type of returned entities (type filtering). A classifier is trained to decide whether the evidence is strong enough to link to the top ranked-entity or not. Candidate relations between columns are found by querying the DBpedia SPARQL endpoint with pairs of concepts assigned to columns. Then, pairs of strings in different columns are used to score candidate relations: a new query is made against the endpoint, and each pair of strings vote for the candidate relation if the candidate relation appears in the set of relations between the pair of strings. The most voted relation is used for annotating the column pair.

The T2K algorithm [102] works under the one-entity-per-row assumption, thus assigning each row with an entity in the KB, each column to a predicate, the whole table to a concept. A set of candidates is retrieved for each row by querying the KB; then, the types distribution among the sets of candidates for each row is observed to decide the best type to assign to the table. To annotate a column with a property, the value in each row is used for querying the KB and finding a property that relates the top-ranked candidate in the same row and the value itself. The property with the highest frequency is then used to annotate the column. Once the table is assigned a type, the type is used to filter out properties with a different domain type and entities of a different type.

The approach proposed in [126] works similarly to the approach described in [102], querying the reference KB to retrieve candidates, using the majority voting strategy to decide the right type, and applying the one-entity-per-row assumption. Candidates are retrieved based on lexical similarity measures (i. e., Levenshtein and Jaccard), and a deep semantic matching method to assess the semantic similarity between cells values, and entity labels and descriptions. Based on the one-entity-per-row assumption, a property is assigned to each column. Properties are found by filtering the set of properties of the candidates in a column, thus using a feature-based binary classifier that predicts the best property to assign to the considered column; the classifier considers both the values in the columns and the text in the column header.

A more complex strategy is proposed by [127], where the column classification and the entity disambiguation are solved in an incremental, mutually recursive, approach. Entities in cells and concepts of columns are revised iteratively, by enforcing interdependence between columns, and between the classification and disambiguation results; one column is assumed to be the subject column of the table. At the end of the iterative phase, the algorithm finds relationships between the entities in the subject column, and all the other columns, based on the one-entity-per-row assumption. The algorithm firstly finds relations between the subject column and any other columns on each row independently. Given two cells in the same row, one in the subject column and the other in a different column, a set of properties is retrieved by the reference KG by querying triples where the annotated entity in the subject column occurs as the subject. Then, the set of triple object values is matched against the value of the cells in the other columns, using the frequency weighted dice function; the highest score is assigned to be the confidence score for the candidate relation. When candidate relations have been found for each row, the algorithm aggregates the results by scoring the candidate relations (also considering metadata outside the table, like surrounding paragraphs and semantic markups), then selecting the top-ranked relation.

An improved version of the above algorithm [127] has been proposed to increase efficiency [22]. The improved algorithm applies an iterative approach to solve CTA first, based on preliminary results in CEA (i. e., by finding entities for a subset of rows), then solves the CPA task using the already discovered knowledge. The improved approach removes some external dependencies (e. g., queries to external Web search engines), and simplifies some heuristics, making the algorithm more effective, according to the results obtained on different benchmark datasets.

FactBase is an approach proposed to annotate rows in a table, based on the one-entity-per-row assumption [38]. The algorithm implements a pipeline that starts with retrieving candidates from a private KB that contains data from Wikidata, DBpedia, and Wikipedia. Then, candidates are filtered by observing their types and descriptions: representative types and description tokens are computed using a majority voting strategy on the types and descriptions of the top-ranked candidate in each set of candidates. The algorithm assumes that when a cell has only one candidate, that candidate is the right annotation. Building on this hypothesis, the algorithm uses unique candidates to understand which columns in the table describe *facts* about the entities in the subject column. Pairs or candidate-value from the subject column and a different context column are used to query again the index, collecting triples where the candidate and the given value appear as subject and object, respectively. Properties in the returned triples are collected, and the context column is finally annotated with the most frequent property occurring among rows. If a row does not have candidates, the algorithm uses the properties found to query the KB, looking for entities related to the values in the context columns through the property of each context row. The new candidates are then sorted by the edit distance between the candidate label, and the label in the subject column and the best candidate is used to annotate the row.

An improved version of the FactBase algorithm [38] does not rely only on the top-ranked candidate to infer the right column type [41]. This choice allows the algorithm to deal with more candidates, considering the entire context of the table and filtering the correct annotations using a majority function.

In the following, we discuss two algorithms that solve CEA and CPA only, without considering CTA. The main reason is that both algorithms focus on extracting new triples from the table, aiming to populate or create a KG.

TAIPAN extracts RDF triples from a table, assuming the existence of a subject column, thus looking for relations between the subject column and all the other columns [40]. The algorithm starts by selecting a subset of the table rows, then, disambiguates cell values against the reference KG. The disambiguation phase is in charge of AGDISTIS, an approach based on combining the Hypertext-Induced Topic Search (HITS) algorithm with label expansion strategies and string similarity measures [114]. A set of candidate entities is retrieved to create a disambiguation graph; to collect entities, several heuristics are used, as well as known surface forms (i. e., strings that are usually used to refer to given resources) for resources in the reference KB. The HITS algorithm is run over the disambiguation graph to find the most authoritative candidates; the resources with the highest authority values represent the correct candidate. After the disambiguation phase, for each column, two features are computed: (i) the *support*, i. e., the ratio between the disambiguated cells and the total number of cell, and (ii) the *connectivity*, i. e., the number of connections of the column to other columns and the total number of columns. Finally, a binary classifier is used to classify columns as being either subject columns or not. A set of seed properties is then retrieved in addition to properties extracted via triple patterns. Properties between pairs of columns are evaluated using a probabilistic model, finally solving CPA using the top-ranked property for each column.

A different approach relies on the heuristics described in [102] to detect the subject column, working under the one-entity-per-row assumption [61]; for each row, a set of candidates is retrieved and ranked based on a normalized TF-IDF measure. Candidates are evaluated using a probabilistic graphical model, which is initialized with priors and then updates its likelihood scoring to maximize the coherence of candidate entities across the rows. The coherence is computed as a combination of properties shared by the entities in the table, not by checking types of candidates. Finally, the best candidate of each row is taken for finding relations between the subject columns and the other columns: all the properties of the selected candidate are considered, and sorted according to the Jaccard similarity between their value, and the value listed in the considered column; in this way, each property is given a score for each row, then the scores are aggregated at the column level, and the top-ranked property is used to annotate the column.

The algorithms we discussed in this Section are suitable for annotating Web tables and showed high performance in annotating several benchmark datasets. However, we will discuss in Chapter 5 that these approaches are mostly targeted to annotate Web tables, thus they scale in terms of the number of tables to annotate, but do not scale when the dimension of a single table increases. In fact, Web tables are many (benchmark datasets like T2D [102] counts thousands of tables), but they are very small (a few hundred rows per table). Consequently, these approaches can not be directly integrated into data enrichment tools employed in big data environments, where the goal is likely to annotate large tables. Among the discussed approaches, FactBase [38] better fits the requirements of a scalable enrichment process, like the one we will discuss in Chapter 4, even if it is plagued by some limitations discussed in Chapter 6. Moreover, in Chapter 6, we will propose a novel methodology to increase the tolerance of the type-based filtering phase in algorithms like FactBase, so that to better support the user interaction in semantic data enrichment.

SEMTAB CHALLENGE SYSTEMS We briefly discuss systems that, in the last years, participated in the SemTab challenge [54], a specific challenge that aims at standardizing the evaluation of table annotation algorithms. The following algorithms performed the best in SemTab, but they are tailored to the challenge specifications, lacking generality. In some cases, the participants have employed hard-coded workarounds to increase the performance of their system [115]. Among the participant systems, we mention MTab [85], which applied a majority voting strategy to select the best candidate in a pool of candidates retrieved from different lookup services, Tabularisi [112], which creates a feature space from the lookup service results using TF-IDF, CSV2KG, which exploits different lookup strategies to collect candidates (based on external services, e. g., DBpedia Spotlight)[1], and DAGOBAH [17], which is based on entity embeddings and K-means clustering. All the algorithms jointly solve the three annotation tasks, using preliminary results in CEA to further refine CPA and CTA results.

3.1.2 *Instance-level Annotation*

In this Section, we discuss the algorithms that focus specifically on the CEA task. In previous Sections, we discussed instance-level annotation approaches mainly based on lexical features, which try to match the text in the table against a target KG. In this Section, we focus on recent approaches that tried to go beyond the lexical matching.

TabEL exploits a graphical model and uses a collective classification technique to optimize a global coherence score for a set of entities in a table [5]. Differently from other approaches that employ graphical models, TabEL does not consider the assumption that the semantics of a table can be mapped to predefined types and relations. Candidates of each cell are set as variables in the graphical model. A classifier is then run iteratively to update the maximum-likelihood value for each variable; the classifier is trained to prefer sets of related entities. Different features are used to train the classifier, such as a prior probability based on Wikipedia hyperlinks, semantic relatedness feature (e. g., the similarity between Wikipedia pages, which is based on their in-links and out-links overlap), mention-entity similarity features (i. e., the similarity between the cell context and the entity context; the cell context is the content of the cells in its row and column, while the entity context is the aggregation of the contexts in which the entity occurs in the training data), existing links features (i. e., the system checks if exists a mention in the cell context with the same surface that links to the candidate entity), and surface features (e. g., a feature represents the exact match between the cell content and the name of the entity in the KG).

An approach purely based on similarities between embeddings is proposed in [38]. The algorithm computes offline the entity embeddings for all the entities in the reference KG. Candidates for each cell are then retrieved in the online phase by querying an index containing surface forms of entities; candidates are put as nodes in a disambiguation graph, and a prior probability is assigned to each node, based on the degree of the entity in the reference KG. Nodes in the graph are linked with weighted edges, where the weight of the edge between two nodes is the cosine similarity between the embeddings of the two nodes. The assumption is that since entities appearing in sentences or paragraph tend to form coherent sets, the cosine similarity between

1 https://www.dbpedia-spotlight.org/

vectors aids the global disambiguation. The PageRank algorithm is run over the disambiguation graph to update the priors, and candidates with the highest score are used to annotate their respective rows.

The above algorithm has been extended with the Looping algorithm [41], which does not create the disambiguation graph by considering all the candidates at once, but it starts by constructing a small graph with only unambiguous entities (i. e., rows with a unique candidate). In this way, the graph does not grow exponentially with the number of candidates, and the results are not affected adversely. Thus, at each iteration of the Looping method, only one ambiguous entity is considered, and its candidates are added to the initial graph; using PageRank, the priors of the new graph are updated, and the candidate with the highest score is considered as the right candidate. Building a correct initial graph is crucial to the disambiguation of entities with multiple candidates in this asset.

A different approach tried to reuse ontology matching methods to solve CEA [38] (we will further discuss this topic at the end of this Section). The method builds on the assumption that a table can be converted into a *table ontology*, where each row is mapped to an instance in the new ontology. Once the conversion is done, a generic ontology matching tool can find relationships between the table ontology and the reference KG, thus returning mappings between entities and rows.

Focusing on the multilingual aspect, the approach proposed in [71] focuses on solving the CEA task when the given table and the reference KG are in different languages. The framework is based on neural networks and vector representations, aiming to bridge the language gap by vector space transformation. The neural network represents a joint model that takes two tables as input: the original input table, and a *candidate table* where each cell of the original table is replaced with a candidate entity. Candidates are found by translating the cell content into the language of the reference KG (using translation tools), thus querying the KG with the translated text. Text in cell and candidate entities are both converted in embeddings, which are trained on two corpora of different languages separately. Since different vector spaces of embeddings are not naturally comparable, the algorithm also employs a bilingual translation layer in the network, that learns a linear transformation between the different vector spaces (translation parameters are pre-trained with a small set of already translated pairs). The network finally considers three different features to capture the correlations between entities in the table: the mention and context features, which represents the relevance or compatibility between the original table and the entity table, and the coherence feature, which evaluates the inner relationship of entities in the correct linked table. The neural network features are learned in a supervised way, thus requiring a set of training data.

Differently from the other approaches, the one proposed in [33] introduces the pre-training/fine-tuning paradigm to relational Web tables. A model is pre-trained and learns deep contextualized representations on relational tables in an unsupervised manner. The trained model is then fine-tuned on the specific CEA task. Given a gold standard, the model is fine-tuned by considering each cell in a table as a potential entity, thus using the cell text as well as the table metadata into the pre-trained model; in this way, a contextualized representation is obtained for each cell. Candidates for each cell are collected using a lookup function. A dense vector representation of each candidate is obtained by considering the information about its name and description (using word embeddings, which are shared with the aforementioned pre-trained model), as well as its type (the type embeddings are learned during the fine-tuning phase). A matching score is then used to learn the similarity between the candidate and cell representations.

ENTITY RESOLUTION We mostly focused on STI algorithms that solve the CEA task, i. e., the table-to-KG matching, which is more related to the semantic data enrichment topic discussed in this thesis. However, this task is part of the vast *entity resolution* research area, which includes two kinds of matching that could approximate the table-to-KG matching: the table-to-table matching (*record linkage*), and the KG-to-KG matching (*link discovery*). In the following, we report a brief literature overview of these matching tasks, discussing the challenges that, as of today, prevent the adoption of existing algorithms to solve the CEA task. We refer to a recent survey [20] for an in-depth explanation of the record linkage algorithms, with a specific focus on the new challenges brought by the big data, e. g., entity descriptions published as linked open data.

Record linkage is the task of matching structured descriptions (i. e., records) that refer to the same real-world entity, appearing across different data sources [20]. Many approaches have been proposed in the literature to solve this task, and they can be exploited to approximate a solution also the table-to-KG problem if we manage to convert the KG in a set of records (i. e., a table). However, KGs exhibit properties that challenge existing record linkage approaches: (i) KGs potentially contain billion of triples and thousands of KGs are available in the LOD cloud (*volume*); (ii) KGs are extremely heterogeneous, even in the same domain (*variety*); (iii) KGs in the LOD cloud are very dynamic (*velocity*); KGs are of widely differing quality in terms of coverage, accuracy and timeliness (*veracity*). For example, KGs feature highly heterogeneous entity descriptions, which hinder the usage of schema-aware comparisons [92]; approximate string matching similarities [37] between values of a pair of entities is weak due to veracity, requiring the algorithm to gather more sources of evidence (e. g., by checking the similarity of neighboring entities, which are connected via relations in KGs).

In the last years, different approaches to tackle the above problems individually have been proposed. However, they are still challenged when many characteristics have to be considered simultaneously [20]; for example, approaches based on hierarchical matching strategies [44] or distributed representations [15] have been proposed to better deal with the data variety. Neural methods are the core component of novel entity resolution approaches such as Ditto [68] or DeepMatcher [78], which adopt language models and different neural architectures for serializing records and considering domain knowledge within the matching process. When dealing with entities that do not share a common schema, a soft-alignment calculation can consider many attribute pairs at once [87].

Nowadays we have different methods focused on different aspects, thus identifying the best workflow that fit the real user needs demands for end-to-end frameworks such as MinoanER [39] and JedAI [93], which allow the users to combine different sub-components of the most recent entity resolution approaches, and extensively evaluate their combination.

Concerning the link discovery, it is the task of finding links across KGs, which is critical in ensuring the interoperability between different KGs. In the specific context of the entity resolution, the task is usually restricted to find `sameAs` relationships between entities [82]. The most powerful link discovery framework[2] are AgreementMakerLight [42] and LogMap [53]. Both the frameworks focus on computational efficiency, handle very large ontologies, and allow the users to customize the link discovery process. Alongside these frameworks, other tools like Silk [118] and LIMES [84] allow the users to manually define generic matching pipelines (i. e., pipelines that do not necessarily find `sameAs` links), and expose learning-based approaches for determining linking specifications. For example, LIMES employs decision tree models to suggest the best set of similarity metrics and thresholds to use to compare a pair of attributes [90]).

The link discovery matching approximates the table-to-KG problem if we manage to convert the table into a KG (this task mainly requires to define the schema-level annotation of the table). However, it has been evaluated that link discovery approaches do not perform well in CEA [38]. The main reason that hinders a full mapping between the two matching problems rely on the variety of the data: most link discovery tools are not designed to provide mappings between heterogeneous KGs, and the number of attributes used to describe entities in a record differs from the information amount available in the KG. For example, DBpedia entities are described with 11.44

2 According to the results observed in several editions of the Ontology Alignment Evaluation Initiative.

attributes on average, whereas the number of attributes used in a Web table corpus is typically between 4 and 6. [38].

3.1.3 *Schema-level Annotation*

The schema-level annotation is devoted to the annotation of the table structure. The task is broad and covers many other sub-tasks besides CTA and CPA, such as table structure understanding [88], datatype prediction [56, 83], and holistic matching across tables [64]. In this Section, we are more interested in discussing algorithms tightly related to the specific CTA and CPA task.

We hereby focus on works that tried to decouple these tasks from CEA, by exploiting the power of text classification models, showing promising results.

An STA approach to learning how to assign a type to a column by observing its values is described in [59]. The proposed modeling framework assumes a semantic type to be either a class, or a pair consisting of a data property and a class. As an example, ⟨`mainInterest`, `Scientist`⟩ is considered as a semantic type. The algorithm computes a set of similarity metrics, which are used to retrieve a set of candidate types for each column to put as nodes in a disambiguation graph. Concepts in the graph are then linked with all the possible relationships existing in the reference ontology. Finally, concepts and relationships are extracted from the graph by computing the Steiner tree, i. e., the most succinct model that connects all the semantic types and relates all the source columns. The approach was first extended to reuse learned models to annotate tables in the same context [111]; subsequently, a further extension enabled the reuse of a learned model in different domains [96]. Both the approaches have been integrated into Karma, a tool to interactively assist the users in mapping tabular data schema to ontologies [46].

A different approach for finding the top-k concepts for a given column is provided in [32]. The algorithm ranks the concepts for a column by computing different exact and fuzzy similarities between a concept and the column; in particular, this similarity is quantified based on the matching between the entities in the concept and the cell values of the column. This work proposes an efficient way to compute this kind of similarity by extending MapReduce-based similarity-joins.

Another table annotation approach has been proposed as part of the GIVA framework [24] to integrate geospatial information from heterogeneous sources [25]. In this approach, tables are first transformed into corresponding *feature-rich* tables, by correctly identifying the column headers when missing; then, the approach exploits the particular hierarchical characteristics of geospatial classification schemes, which can be modeled using a *part-of* or *is-a* relationship, to extract ontologies from relational tables (but also from XML and RDF documents). A further improvement of the instance matching module in GIVA has been proposed to improve the dataset integration when dealing with business names and addresses [106].

The strategy of using snippets returned by a search engine (e. g., Bing) has been investigated in [99], where the snippets are used to determine the type of entities in a table. Each table row is submitted to the search engine as a query; then the returned snippet is used as input for a multi-class text classifier, which determines whether a snippet is the description of an entity of a given type. The classifier is trained on a set of entities; for each entity, a query label+type is submitted to the search engine, and up to 10 snippets are used as training data for the classifier.

The approach used in SemanticTyper learns a semantic labeling function from a set of values (strings, numbers, or a mix) to a set of semantic types, in a supervised way [100]. A semantic type can be either a class or a pair property-class. Since the full column is taken as input, the algorithm is able to learn the distribution and hence characteristic properties of the data corresponding to a semantic type as a whole, rather than extracting features from individual data values.

Recently, approaches based on machine learning models have been proposed for solving the CTA task.

ColNet [18, 19] is a supervised system based on convolutional neural networks to model the contextual semantics of a column. ColNet can also learn inter-column semantics features from a KG and query answering algorithm, without assuming that table cells have KB entity correspondences. Furthermore, the algorithm assumes no metadata is available for the given table (e. g., column headers or table description), typical in many real-world contexts. In addition to the evidence available in the table, ColNet builds the *property vectors*, i. e., vectors representing the degree of existence of a property for each considered class, which are exploited in the prediction model. In its first version [18], ColNet trained a single binary classifier for each class to consider. Subsequently, the approach has been improved by proposing a single Hybrid Neural Network to solve the multi-class classification prediction problem [19].

Sherlock [49] relies on a multi-input deep neural network for detecting the semantic type of a column, by looking at the table header row. In Sherlock, columns are modeled as mappings from column values to a column header, and the column headers are treated as ground truth labels of the semantic type. Thus, the CTA problem is modeled as a multi-class classification problem. Sherlock has been trained on about 680 thousands data columns, learning ~1,500 features of columns from VizNet [48], a repository of real-world datasets collected from the Web. These features include global statistics (e. g., column entropy that describes how uniformly values are distributed), character-level distributions (e. g., the frequencies of 96 ASCII-printable characters within each value of a column), word embeddings (i. e., high-dimensional vectors that represent words in a continuous space; words in columns are mapped to vectors using the GloVe pre-trained embeddings [94]), and paragraph vectors (i. e., high-dimensional vectors that represent paragraphs in a continuous space; the column text is considered as a paragraph, then it is used to fed a Distributed Bag of Words version of Paragraph Vector [63]).

Sato is a hybrid machine learning model that automatically detects the semantic types of columns, by exploiting column values, as well as contextual information. Sato combines Sherlock's single-column type prediction with topic modeling and structured learning to solve the multi-class classification problem. Differently from Sherlock, Sato uses signals from the global context (values from the entire table) and the local context (predicted types of neighboring columns). In addition, Sato is designed in a modular fashion, so that it can be easily plugged in other single-column models with nearly-zero effort.

Despite the small subset of DBpedia classes used in the experimental phase (34 for Colnet, 78 for both Sherlock and Sato, out of 685 classes currently covered by the DBpedia ontology,[3]) all the approaches showed promising results for the CTA task.

The schema-level algorithms we described in this Section are mainly based on the text analysis of table columns. This strategy is not suitable for supporting the data enrichment of large tables, because it prevents the scalability of the enrichment algorithm. However, some models can be pre-trained on a dedicated corpus (e. g., ColNet), but the small number of predictable types does not support many different scenarios. Besides these reasons, in this thesis, we do not include a specific schema-level annotation algorithm in the semantic enrichment workflow presented in Chapter 4, mainly because we prefer to support the user in the STA task interactively. Indeed, the number of columns in a table is usually limited, allowing the user to manually annotate all the columns, while the instance-level annotation is the crucial task that requires automation. Moreover, in Chapter 6, we will propose a methodology to soften STI algorithms filtering and ranking strategies, in such a way to better deal with inaccurate schema-level annotations provided by possibly non-expert users.

3.1.4 *Evaluating Table Annotation Algorithms*

In the last decade, different benchmark datasets have been proposed in the literature to evaluate table annotation algorithms. For many years, Limaye [69] and T2D [102] have been the reference benchmark datasets in the literature. Subsequently, the novel W2D dataset [38] has been proposed, which contains a larger number of tables, if compared with previous existing datasets. The three datasets are suitable for

3 https://wiki.dbpedia.org/services-resources/ontology

testing different annotation tasks; as an example, T2D can be used to test algorithms in solving CPA, but W2D does not cover this task very well, because only a few tables are annotated with properties. Also, only W2D is suitable for testing the capability of an algorithm to efficiently annotate many tables, since it counts more than 485k tables, while T2D counts ~200 tables.

Even if these datasets have been the reference in the literature, and many works discussed in the previous Sections have been tested on them, they come with some limitations:

- These datasets are focused only on Web tables, which are usually very small, thus do not allow to test the scalability of annotation algorithms, when the size of a single table increases; in fact, the average number of rows per table is 123 for T2D, 29 for Limaye and only 15 for W2D, according to [38].

- They include only *entity tables*, i. e., tables where each row represents only one entity; algorithms that work under the one-entity-per-row assumption benefit the most from this limitation, but many real-world tables do not have this property.

- Tables in these datasets are extracted from the Web, and the entities mentioned in tables are usually referred with their canonical name (e. g., Albert Einstein is almost always mentioned as "Albert Einstein" - it is very unusual to see "A. Einstein", "Einstein, A.", "Einstein", and similar mentions in a Web table). Consequently, it is not possible to test the capability of annotation algorithms in dealing with acronyms, abbreviations, and misspelled words by using these benchmark dataset.

A recent work studied the behaviour of STI algorithms on such datasets, finding out that many of the existing approaches are focused on *obviously linkable* cells [126]. According to the study, a tool like T2K [102] matches only 2.85% of a large corpus of Web tables to DBpedia, while the performance in CEA of the same tool evaluated on T2D is very high (F1: 0.82, Precision: 0.90, Recall: 0.76); as a consequence, we conclude that many tables in T2D are easy to annotate. Indeed, experiments conducted in [102] report that the entity linking in T2K fails to annotate ambiguous mentions, spotlighting that there are a few ambiguous mentions in T2D.

In 2019, the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab) was introduced to unify the community efforts towards the systematic evaluation of table annotation algorithms [54]. Different datasets were constructed to evaluate systems in different rounds (ST19-R1, ST19-R2, ST19-R3, and ST19-R4). Some of the datasets feature tables generated through a novel dataset generator, which automatically build tables from a generic KG by processing the results of SPARQL queries. ST19-R1 and ST19-R2 include revised tables from T2D and W2D datasets (e. g., more columns have been annotated). The analysis of the results achieved by different competing approaches revealed some challenges to address in future work:

- Wikipedia and DBpedia lookup mechanisms are adequate for candidate selection tasks, but they might fail when the entity mention is misspelled (e. g., Winsconsin vs Wisconsin) or different from the canonical name (e. g., football player nicknames: *La Pulga* to refer `Lionel Messi`).

- Real-world tables are noisy and not well-formed, in general. Algorithms should also be evaluated with respect to their ability to deal with such tables.

- Missing data can affect the results of the algorithms, but this aspect has not been correctly evaluated.

- Although the overall quality of the SemTab 2019 datasets is higher compared with the previous datasets, a manual inspection of the tables in SemTab datasets brought to the surface some malformed and wrongly annotated tables, e. g., empty rows mapped to an entity, or long descriptions (with mentions of different entities) mapped to a single entity (usually, the first-mentioned entity).[4]

Finally, existing datasets, as well as SemTab datasets, have contributed to benchmarking STI algorithms, but none of them provides enough fine-grained information about the achievement of a specific score (e. g., if the algorithm fails to annotate a specific type of tables). Some tools have been developed to at least highlight the main error patterns, but those patterns must be manually inspected [23].

In conclusion, existing datasets are not suitable for testing scalability and performance of STI algorithms in more realistic scenarios (e. g., enriching a noisy dataset). In Chapter 5, we will propose a novel dataset that faces some of the challenges discussed above. The dataset is mainly focused on testing algorithms in CEA, which is crucial in semantic data enrichment. However, the dataset has been subsequently extended also to cover the CTA task evaluation.

3.2 DATA ENRICHMENT

The *data enrichment* is a specific data integration task where a source dataset is manipulated in order to append additional data, typically from external sources. The data enrichment differs from a pure data integration task because the involved data sources (i) have to be

4 See Tables *53822652_0_5767892317858575530* and *12th_Goya_Awards#1* from ST19-R1 and ST19-R2, respectively. These errors come from the T2D and W2D datasets used in SemTab 2019.

partially integrated, and (ii) are heterogeneous, i. e., they are about different topics. These differences prevent the smooth adoption of well-studied record linkage techniques. Similarly, the problem can be mapped to a link discovery problem, where correspondences between entities in different KGs have to be found, but the same limitations are there, as discusses in Section 3.1.2.

The data enrichment task is often viewed as a data transformation task, where a sequence of data transformation functions is applied to the involved datasets to integrate them. In this Section, we focus on methods and tools to facilitate the data enrichment task, when it is approached as a data transformation problem. We point the reader to recent surveys in the field for in-depth details about the data integration accomplished with record linkage [20] and link discovery [82] techniques.

In general, the data transformation task deals with preliminary profiling and transformation of the data, and is usually aimed at identifying and addressing possible data anomalies. Eventually, the task objective is also to change the shape of the data, in such a way to ease to work with for further tasks. For example, streaming data from sensors are collected as plain-text (logs), then are usually transformed into tabular data to ease the downstream analytics; finally missing values are filled or removed. Even if existing tools are able to cope with different input data formats, our focus is on tools able to manipulate tabular data.

A variety of tools have been proposed to help expert users in defining data transformation pipelines, which differ in the capabilities they offer. Such tools include powerful command line interface tools (e. g., *csvkit*)[5] and libraries for statistical data analysis (e. g., *Agate*,[6] a Python library for data analysis), spreadsheet software (e. g., *Microsoft Excel*),[7] and complex systems for the interactive definition of extract-transform-load processes (e. g., *Pentaho Data Integration*).[8] More details about advancements in this area, with a focus on the automation of detection and fixing of data anomalies, also in the context of Big Data, have been discussed in recent work [109, 119].

While the above tools offer a variety of configurable components to create data transformation pipelines, no hints are provided to the user to ease the integration task, i. e., to find the joins between several data sources. The state-of-the-art lacks solutions like the one described in Section 2.3, where the semantics is exploited to ease the integration task.

A popular data transformation tool that supports instance-level reconciliation and extension in tabular data is OpenRefine.[9] It provides

5 https://pypi.org/project/csvkit/
6 https://pypi.org/project/agate/
7 https://products.office.com/en/excel
8 http://community.pentaho.com/projects/data-integration
9 http://openrefine.org

interactive user-interfaces with spreadsheet-style interaction embedded in a desktop application designed for experts in semantics. The tool also allows users to extend tables only with the information contained in the same KB used for the reconciliation task (`sameAs` links cannot be used for accessing a different KB), or by manually invoking Web services with the string-content of a cell as a parameter (i. e., it requires a short script to invoke an external service via HTTP requests). Although the design phase is well supported in OpenRefine, the tool comes without any support for batch execution of pipelines; thus, it can only process data that can be entirely stored in memory, which is not suitable for the Big Data context [95].

The community around OpenRefine proposed some tools for extending such tool with support to extensive data processing. Among them, the most remarkable is OpenRefine-HD,[10] which extends OpenRefine to use Hadoop's MapReduce jobs on HDFS clusters. Unfortunately, documentation is missing for such a solution, and it is not stated how it can support scalability when more distributed datasets, exposed by external services, are involved. The tool we will present in Chapter 4 implements all the functionalities available in OpenRefine, but moreover, it also supports the batch execution of enrichment pipelines and provides support to define the schema-level annotation of tables.

Different commercial tools have been developed to solve the data integration problem from the data transformation perspective, without offering any support to the semantic enrichment.

Trifacta Wrangler[11] is a commercial suite of Web applications for the transformation of raw datasets. Many smart tools, also based on machine learning techniques, are provided to the user to prepare datasets for different analysis types (e. g., statistics about values in columns are brought to the user to ease the data cleaning). The tool comes with parallel processing, thus supports large volumes of data.

KNIME[12] is a tool mainly dedicated to data analytics, but it comes as a modular platform for building and executing workflows. A set of standard operations is provided to the user, but some other operations can be manually added by writing short scripts. The execution on large datasets is supported by ad-hoc extensions, which allow the user to deploy the designed workflows on Apache Spark and Hadoop clusters.

Talend[13] is a programming editor that offers a predefined set of components to setup pipelines (similarly to KNIME), which can be compiled into executable code. Also, Talend provides an open-source data integration platform with big data extensions.

The three aforementioned commercial tools focus on functionality to design data transformation pipelines and support their scalable

10 https://github.com/rmalla1/OpenRefine-HD
11 https://www.trifacta.com
12 https://www.knime.com
13 https://www.talend.com

execution. However, none of them comes with specific functionalities for the semantic data enrichment, which can be manually included by expert users with hard-coded workarounds (e. g., a user can write a script for KNIME, which reconciles data against a SPARQL endpoint).

Finally, we report that the topic of data extension has not been addressed adequately in the scientific literature. Most of the approaches to fuse two tables have focused on conflict resolution strategies [11], i. e., how to deal with overlapping information. In data extension, though, data to be added can be assumed to be new, and the conflict resolution problem may occur but is not the objective of the fusion operation. A few solutions tailored to the data augmentation have been proposed in the last years, but they do not consider the information available in existing KBs.

InfoGather+ is a tool for the data augmentation, focused on filling missing attributes in a table. The algorithm is mainly indicated for numeric and time-varying attributes. Infogather+ builds a *semantic graph* over different Web tables. Table columns are annotated with unit, scale, and timestamp in the semantic graph, then links are discovered between columns representing the same semantic attribute (the algorithm also deals with values expressed in different units and scales). Finally, the linked columns in the graph are used to extract the information missing in one table, from another table in the graph. Experiments show that the system scales when the number of considered tables increases, but no evidence is given about the system scalability when dealing with large tables.

Deeper is a data enrichment system powered by deep Web, a hidden database [120]. Deeper is useful to help data practitioners in linking a local database to a hidden database. It offers enrichment functions, so that the user can choose additional attributes from the hidden database. The approach is focused on studying how to crawl a hidden database, but issues related to system scalability are not covered. In addition, the system does not exploit the semantics but exposes an API to query the hidden database using only textual information.

3.3 CONTRIBUTIONS AND RELATED WORK

In this thesis, we introduce a system to assist data workers in enriching source datasets with information from third-party data sources. Differently from the usual approach to the data extension, which is mainly in charge of the data worker, we exploit the semantics to ease the task, without requiring the data worker to design the underlying data transformation pipeline, as well as to explore the information stored in KGs for guiding the user in the data enrichment process. We will illustrate how the designed solution can support a data worker in enriching large-scale datasets.

However, smoothly introducing STI algorithms from the literature in our systems is not possible, because existing algorithms do not support the human interaction, and scale only to a limited extent (i. e., they are able to annotate many Web tables, that are usually small). Moreover, benchmark datasets featuring large tables to test the scalability of STI algorithms still lack in the literature, with the existing ones mostly focused on small Web tables. Thus, we first introduce a novel challenging dataset to test different capabilities of STI algorithms, including their scalability; then, we propose a neural-based methodology to allow STI algorithms to deal with the information entered by a user, also when the user is not expert, and then the information may be inaccurate.

SUMMARY OF THE CONTRIBUTIONS The contributions of this research work improve three different aspects of the semantic data enrichment:

- We propose a methodology to support the interactive data enrichment harnessing the semantics; we also design a system that employs this methodology and can execute data enrichment pipelines at a large scale (Chapter 4).
- We introduce a novel benchmark dataset to test also the scalability of existing STI algorithms (Chapter 5).
- We propose a methodology to deal with the incomplete entity typing information in STI approaches, either entered by non-expert users or stored in the reference KGs, which is based on neural prediction models and distributed representations (Chapter 6).

4 A SEMANTIC TABLE ANNOTATION APPROACH TO LARGE-SCALE DATA ENRICHMENT

In Chapter 2, we discussed how KGs represent a well-known semantic paradigm relevant to data enrichment nowadays. Semantic Web technologies support the publication of KGs with standards and shared vocabularies that facilitate access and manipulation of knowledge via web protocols. Several approaches have been proposed to integrate data from different KGs, e. g., using entity reconciliation techniques [2]. However, we also discussed that in most data analytics projects, the user is provided with some source legacy dataset that is structured as a table, not as a KG.

STI approaches to transform legacy datasets (e. g., CSV tables) by giving them a graph structure enriched with shared vocabularies, and, possibly, with background knowledge already available in a graph structure, have been proposed in the literature [46, 80]. The transformation of legacy sources into a final KG, possibly enriched with background knowledge, is a complex process. It can be addressed as an STA task, where users specify mappings manually (e. g., by using specific mapping languages like R2RML[1] or RML[2] to transform CSV data into RDF graphs [62], or by employing the Ontology Based Data Access (OBDA) approach for exposing a database as a graph [58]). In contrast, the same process can be partly sustained by STI approaches, which aim to automatically map the table to the schema of a reference KG and link values in the source table to entities in the KG [40, 61].

Linking values to entity was found to be particularly useful for data enrichment, in the sense that the information in the source table can be fused with information in the target KG, e. g., a large and cross-domain information source like Wikidata. An example of this kind of approach is DAGOBAH [17], whose aim is to use STI to obtain a final KG enriched with information about the entities available in external sources. The focus on one reference KG, the creation of one enriched KG as the output of the enrichment process, and applying STI algorithms to automatically annotate tables are assets shared by most of the work in this domain.

In the following, we discuss how to leverage and take existing work a step forward. We argue that STA can provide a valuable paradigm to support data enrichment, modularly and at scale, in a much wider number of scenarios, including when the final objective is to enrich datasets, and not their transformation into KGs. By *modularly*, we mean

1 https://www.w3.org/TR/r2rml/
2 https://rml.io/specs/rml/

that the paradigm can be implemented by an ecosystem of services that provide access to different KGs to support automatic entity linking and data extensions, under the assumption that entity linking is the crucial task that requires automation. Indeed, automation is a key factor in managing large volumes of data and reaching the *at scale* dimension under certain assumptions.

In this Chapter, we describe a methodological proposal where data workers can perform the data enrichment as an interactive table manipulation and annotation process [21]. Applying STI to an input table plays a fundamental role in linking information to external data sources and supporting data fetching from these sources. In this methodology, entity linking algorithms bridge the gap between the source table and external data sources, while the assisted schema-level annotation drives the selection of data extension services available for these entities.

We then discuss how the semantics (especially KGs) can be employed as a facilitator of the enrichment process, thus filling an existing gap between technologies available today to support data enrichment at scale. To demonstrate the suitability of the proposed methodology, we designed a system supported by general-purpose and specialized services for Linked Data (for instance, for geographical toponyms, weather, and events), which support industry-driven analytic projects that motivated our work. We also report on the architectural choices to put in place to achieve the desired level of scalability, which enables the apply enrichment pipelines on massive datasets. The architectural solution we proposed to implement our methodology has some limitations, mainly due to some suboptimal design choices, which hinder us from exploiting sophisticated STI approaches in our application.

We developed and demoed a prototype application that implements our methodology and relies on the specialized services for Linked Data mentioned above [29]. The proposed methodology has also been employed in a real-life scenario, where we enriched a large dataset of a digital marketing company [30].

4.1 FROM SEMANTIC ANNOTATION TO DATA TRANSFORMATIONS

In a typical analytics project, data enrichment is addressed as part of the data preparation stage. As a result, a *data transformation pipeline* is developed manually, containing the logic to combine different sources of information. We briefly recall the motivating example we provided in Chapter 1, where a company is interested in enriching a source dataset containing digital marketing campaigns data with additional features about weather forecasts. In our scenario, different operations have to be achieved by a data worker in order to enrich the dataset: (i) change the date format, (ii) fetch additional data from the reference

KG (i. e., GeoNames), and (iii) fetching the feature of interest from an external provider (i. e., ECMWF).

The data worker has to deal with three different sources of information: (i) the source dataset containing the company data, (ii) the GeoNames KG, and (iii) the weather dumps, which are provided in a specific format. Figuring out how to combine these datasets is a cumbersome task, especially because in almost all the cases the external data sources (e. g., GeoNames and ECMWF) are unknown to the data worker. Indeed, the data worker faces the following challenges while joining the involved datasets:

- Investigate how to reconcile locations in the table to GeoNames, i. e., to look for a service that meets this requirement (suitable for users familiar with programming languages), or to check out the GeoNames KG and build an ad-hoc reconciliation service (suitable for users experienced in semantics and the geospatial domain);

- Manage to query the ECMWF service, i. e., look for the API documentation (usually suitable for users familiar with programming languages, less applicable to data workers and domain experts);

- Propose a scalable architectural solution able to efficiently enrich datasets, also when they contain a massive amount of data. Meeting possible time constraints is crucial because the enrichment process will very likely query external services (e. g., ECMWF), thus an effective solution for architectural issues (e. g., network latency) is required to overcome any bottlenecks due to a large number of issued API requests.

As a result, the data worker operates at the transformation level and directly designs a data transformation pipeline, like the one depicted in Figure 4.1, where several datasets (or services) are loaded (or queried),[3] then joined with the source dataset.

We argue that the semantics can be exploited to support the data enrichment, allowing the user to do the above operations transparently: the focus becomes to declare the operations needed, while artificial agents (e. g., services) solve the actual problem of data integration, also modeling the final solution as a sequence of transformation steps. An example of this scenario is depicted in Figure 4.2, where a simple data enrichment pipeline is automatically converted into a set of data transformations, generated by transformation services that act as agents; the data worker has thus to invoke the needed transformation services interactively. The transformation steps are then automatically packed into a pipeline and deployed in a generic production environment suitable for managing massive amounts of data.

3 We assume the result is already in tabular format, or it can be transformed into this format.

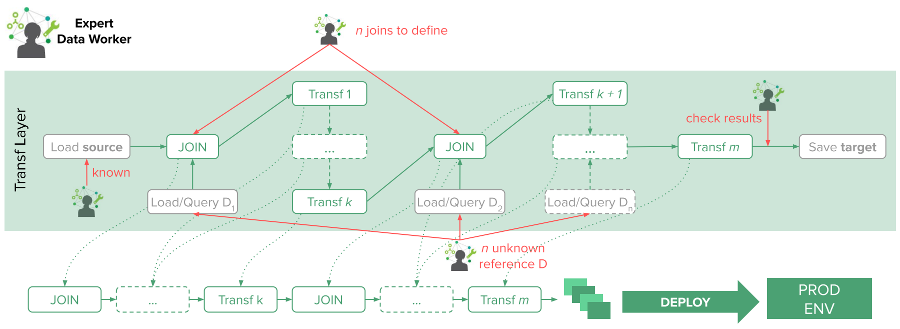


Figure 4.1: A data transformation pipeline to enrich a source dataset with data from n reference datasets.

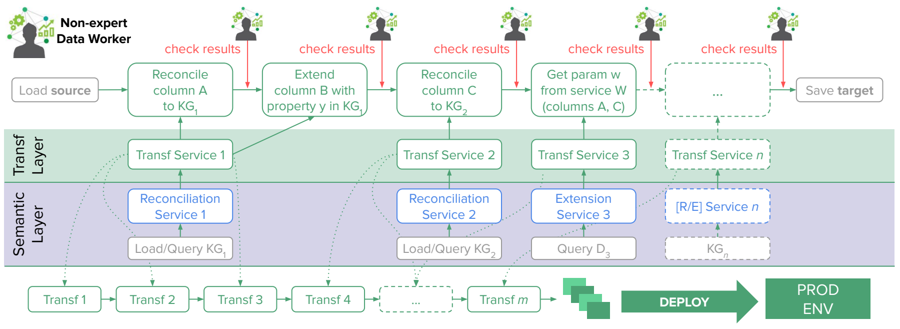


Figure 4.2: A declarative and interactive approach to the semantic data enrichment, supported by n different services.

The following functionalities are crucial in actually supporting a data worker in enriching a dataset at scale:

- Manipulation over table elements (i. e., cell values, column headers, rows, and columns) to clean data anomalies, e. g., removing rows with missing values, or transform the data, e. g., creating a new column that contains dates in a different format. These steps remain fundamental, since they are the building blocks of transformation pipelines.

- Values reconciliation against a reference KG, e. g., matching the locations adopted in the source dataset against the system of spatial identifiers adopted by the external dataset or service (e. g., GeoNames identifiers or WGS84 coordinates).

- Data extension based on the reconciliation results, which represent the bridge between the source dataset and a target KG or service. The extension could add one or more columns to the original dataset, depending on the set of desired features.

These functionalities have to be packed into an approach able to support the development of an interactive environment to design the reconciliation and extension process, and an automatic and scalable platform to execute the process on massive input datasets.

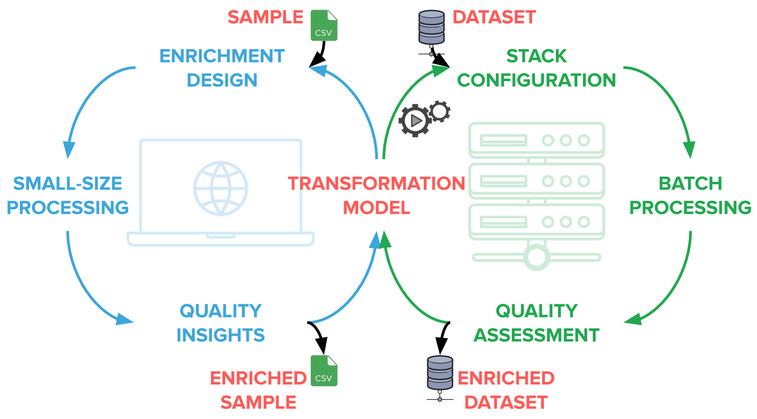


Figure 4.3: The small-scale design/large-scale execution principle at a glance.

In the rest of this Chapter, we will introduce Asia (Assisted Semantic Interpretation and Annotation), a comprehensive approach to provide data workers with suitable tools to interactively design enrichment pipelines on datasets in tabular format, alongside a scalable solution to deploy and run such pipeline against large datasets. We emphasize that most of the related work in this field addresses the problem of automatically inferring annotations that encode the semantics of a table. However, the contribution we discuss in this Chapter is related to implementing reconciliation and extension mechanisms to support interactive data enrichment on small tabular datasets and their automatic execution on massive workloads.

4.2 SMALL-SCALE DESIGN/BIG-SCALE EXECUTION

The approach we propose in this Chapter is built on a *small-scale design/full-scale execution* principle (also referred to simply *design/execution* from now on), whose driver is to separate the full data enrichment process in two logic phases, as also depicted in Figure 4.3:

- The *design phase*, where the user designs the data enrichment pipeline by working on a sample of the source dataset; the output of this phase is a *transformation model* that embeds all the operations needed to enrich the dataset.
- The *processing phase*, which executes the pipeline against the original dataset and produces its enriched version.

Both phases harness the semantics to support the reconciliation and extension tasks, thus rely on external data sources and services.

The rationale behind adopting the design/execution principle is that in a data analytics pipeline, removing the human control during the data preparation stage in favor of a fully automated approach is unsuitable, due to the very high-quality standards they have to meet. In a process where the semantics supports the matching phase, the contribution of a domain expert's knowledge and experience is relevant to the final dataset quality. At the same time, requiring user contributions while processing large datasets is unpractical. Based on this observation, following this principle allows the user to fully control the definition of the enrichment, while the resulting process is fully automated.

Within the design phase, the user is in charge of defining the steps needed to enrich the dataset. The user is expected to work with a sample dataset when the full dataset is too big to be considered and controlled (e. g., a dataset that can be manually revised). The user is supported throughout the definition by computer-aided tools offered as a service in such a way to reduce the need for strong programming skills. Subsequently, the processing phase applies the enrichment pipeline defined by the user, also considering the choices the user made for increasing the dataset quality (e. g., revision actions put in place by the user). The actual transformation is run in batch mode, supporting possibly larger datasets.

Data management efficiency and confidentiality issues also benefit from the design/execution principle. When a dataset contains confidential information, the design/execution principle reduces the amount of data needed to design the transformation, reducing the risk of data leaks. Moreover, the user in charge of defining the enrichment pipeline can work on regular hardware (e. g., data are handled in a Web application accessible via Web browsers), without the need to move the data (e. g., download a full dump); the massive workload is instead processed on-premise, exploiting potentially larger corporate infrastructures.

Finally, the two phases have different time requirements and potentially work on datasets of different sizes, thus need different execution strategies: within the design phase, the system is expected to work on a smaller dataset and to be responsive so that the user can provide feedback interactively; on the contrary, the processing phase processes larger dataset, but can last several hours. The design/execution principle allows us to built an architecture where two main logical components manage these phases, supported by a pool of shared services.

DESIGN PHASE As illustrated in Figure 4.3, the design phase takes a sample dataset as input and iterates over three steps:

- Enrichment design: the user designs an enrichment pipeline that features transformation steps, supported by a graphical interface that facilitates the interactions with reconciliation and extension services;
- Small-size processing: each step in the pipeline is applied to the sample dataset;

- Quality insights: the user can now check the resulting sample dataset, collect statistics about the overall quality of the result (e. g., the number of new missing values).

The iterative process is also interactive, i. e., it is executed every time the user edits the pipeline (e. g., adding a new step). When the required quality standards are met, the process terminates returning two outputs: the enriched sample dataset, and an executable transformation model, which contains the steps of the pipeline and the user actions. The two outputs serve two different scenarios: (i) in a small-scale scenario (e. g., a data journalist that would like to enrich a small dataset), the dataset size is reasonable (e. g., a few thousand rows), and the sample dataset corresponds to the full dataset, thus the full design/process approach should stop at this point (the enriched table produced as output is already the final enriched dataset); (ii) in a large-scale scenario, the executable transformation model is used as the primary step in the downstream processing phase. We assume that when the user needs to enrich large volumes of data (i. e., data too large to be interactively managed), the design phase is carried out using a representative sample of the original dataset.

PROCESSING PHASE Given as input the executable transformation model generated in the previous stage, the processing phase implied three iterative steps:

- Stack configuration: the data flow is defined to support the execution of the enrichment pipeline; the data flow is mainly composed of standard pre- and post-processing steps that can be customized.
- Batch execution: the pipeline is actually executed, possibly in parallel;
- Quality assessment: the user can lastly evaluate the overall quality of the resulting dataset.

In the processing phase, if the result does not meet the desired quality level (e. g., the number of reconciliation mismatches is above a given threshold), the user must go back to the design phase and change the enrichment pipeline, eventually working a different sample dataset (e. g., by adding the set of rows with mismatches). The goal is to reduce the number of overall trials, by limiting as much as possible the trial-and-error approach to the design phase.

In the batch execution, the choice of using external services implementing the reconciliation and extension functionalities represents the main scalability limitation, which we discuss in the following sections.

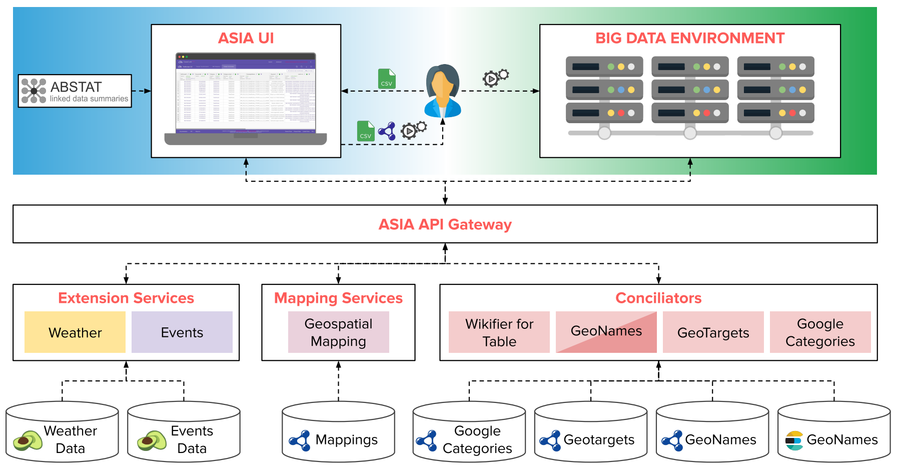


Figure 4.4: Detailed architecture of Asia.

4.3 SEMANTIC ENRICHMENT OF TABULAR DATA AT SCALE WITH ASIA

In this Section, we introduce Asia, our open source solution that employs the design/execution principle to support the enrichment of tabular data at scale. We describe in the following the decisions and the strategies we put in place so that to ensure a suitable level of scalability, while also maintaining a high level of modularity and loose coupling between components, which simplify future extensions.

The resulting overall architecture is presented in Figure 4.4 and features the following main logical blocks:

USER INTERFACE The Asia frontend is represented by a single-page Web application fully integrated into DataGraft,[4] an existing data management platform. The application provides the functionalities and widgets to declare the schema-level annotation and match values in columns against different KGs. These functionalities are supported by a set of backend services that the user can invoke interactively. The application also supports the user in discovering new sources of information: once a column is reconciled against a specific KG, a set of external services that supports the extension from that KG is proposed to the user. More details about the frontend application are given in Section 4.3.1.

BIG DATA ENVIRONMENT This component is the Big Data counterpart of the Asia frontend, and is responsible for the orchestration and execution of the enrichment pipeline at a larger scale. Alongside high-level interfaces to configure and monitor the different data flows, this component provides the needed scalability: the workload is thus distributed to machines, which apply different *replicas* of the enrichment pipeline to different chunks of

4 https://datagraft.io/

the same dataset. We will provide the reader with more details about this component in Section 4.3.2.

BACKEND The core of ASIA is an ecosystem of services orchestrated by the ASIA API Gateway, which provides a unified view of the ASIA backend. Services expose different functionalities, based on their category:

- *Conciliators*: services devoted to the reconciliation of value against a specific KG. Conciliators implement the *OpenRefine Reconciliation and Extension APIs*,[5] which optionally allows for reconciled entities to be also exploited for fetching additional information from the same KG (e. g., once locations have been reconciled with the GeoNames conciliator service, additional properties can be fetched from the service, like the population of the locations). When the extension is available, we say that the conciliator service is also a *KG-based extension* service. Services actually available in ASIA are represented in Figure 4.4 as pink (reconciliation only) and pink-and-red (reconciliation and KG-based extensions) blocks. Different services may rely on different data structures (e. g., full-text indexes), depending on the specific implemented reconciliation strategy and the considered KG dimension.
- *Mapping Services*: services in charge of finding and exposing links between KGs. The primary purpose of these services is to enable the user to map entities seamlessly in different KGs to obtain URIs from a different conciliator. The mapping service (purple block in Figure 4.4) currently available in ASIA is built on links between locations represented in Google Geotargets, GeoNames, DBpedia, and Wikidata.[6]
- *Extension Services*: these services are suitable for extending the input dataset with information from external data sources. Each external data source can be accessed from its specific extension service. Currently, two extension services are available in ASIA: a weather extension service (based on the ECMWF service) and an event extension service (based on the EventRegistry),[7] represented with yellow and blue blocks in Figure 4.4, respectively. In principle, extension services would directly fetch the data from their respective external services; however, to overcome network latency issues, the two services work on regularly downloaded

5 https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Developers

6 We found links between Geotargets and GeoNames using a link discovery tool (i. e., Silk) after converting the Google Geotargets CSV file into an RDF graph. We relied upon existing `sameAs` links between DBpedia and Wikidata and between GeoNames and DBpedia to connect the whole graph.

7 https://eventregistry.org/

dumps of data, which are incrementally stored in local storages.

We remark that since the backend provides common functionalities to both the frontend (to support the definition of the enrichment pipeline) and the Big Data Environment (to support the actual reconciliation and extension steps), its deployment must be tailored to support a specific scenario: in the frontend, we have to support DataGraft in a multi-tenant mode,[8] while in the Big Data Environment, a dedicated deployment is necessary (e. g., with replicas of services) to ensure scalability and efficiency.

4.3.1 *Supporting the Interactive Design in* Asia

A core component of Asia is the application to assist the user in defining enrichment pipelines. This component is essential because it is the one in charge of allowing the user to transparently access the Web of data, so that to connect values in tables with existing KGs, enabling the data extension.

The application is fully integrated within the DataGraft and its data manipulation tool Grafterizer [108], which provides the transformation pipeline abstraction needed to support the transformations over table values, also in batch mode. We chose Grafterizer as our base application because it features a tabular-to-linked-data generator and a compiler to produce portable and repeatable data manipulation pipelines. By inheriting its capabilities, we automatically provide solutions for transforming tabular data to KG and executing many times the same transformation over different datasets that share a common schema (e. g., different temporal snapshots of the same dataset). Asia extends Grafterizer and provides the user with new functionalities to support the schema- and instance-level annotations.[9]

Concerning the schema-level annotation, Asia provides features to streamline this task by supplying cross-lingual vocabulary suggestions services based on data profiling systems, which provide information about the usage of vocabularies in existing data. Currently, suggestions and autocomplete functionalities for types and properties are powered by two services, ABSTAT [91] and Linked Open Vocabularies (LOV). The user can annotate each column with multiple types, a datatype, and a property. At the same time, the application ensures that the annotation is valid, i. e., it enables a proper tabular to RDF conversion (e. g., a user cannot annotate a column with a datatype, without specifying the property that links the column to another column

8 A DataGraft deployment that includes Asia is available online at https://datagraft.io.

9 Demo videos that showcase Asia functionalities are publicly available online at https://youtube.com/playlist?list=PLy7SznldqqmezwdL4QcxQYy2Fz1HV0wMS.

of the same table). The schema-level annotation functionality of Asia has been exploited to annotate the French Business Register dataset (Sirene) schema, enabling its publication as Linked Data [57].

The reconciliation is supported by the conciliator services available in the backend. The user specifies a column to reconcile against one of the available services. The reconciliation includes a validation step that involves the user. The validation step is crucial because a wrong reconciliation leads to wrong extensions downstream. Asia provides statistics to help the user understand the quality of the results (e. g., number of unique values reconciled vs. not reconciled, and number of entities reconciled with a score higher/lower than a threshold) and modify them when needed: the user can alternatively choose a different URI from a list of candidate entities for a cell, or manually inserting the right URI when missing. As a result, a new column with URIs is appended to the table, and the tool automatically annotates the new column with the types of reconciled entities when possible (i. e., when a most common type is identifiable, given the set of reconciled entities). Reconciled columns enable the extension from the same conciliator service (i. e., KG-based extension) used for the reconciliation; by using this type of extension, the user can query different properties of the KG. When the selected property is an object property, the target values are KG entities (e. g., the property `parentADM1` in GeoNames returns GeoNames locations as objects), and the new appended column is considered as an already reconciled column, enabling, in turn, new extensions. The new appended column is automatically annotated with a type only when the selected property has a specified `rdfs:range` in the KG ontology.

Extension from specific extension services are enabled as soon as a column matches one of the criteria defined for the service: e. g., the weather extension service based on ECMWF requires a column to be (i) reconciled against the GeoNames reconciliation service, or (ii) annotated as a column with datatype `xsd:date`. In both cases, a dedicated widget mediates the interaction with the user, which can specify the desired properties to include in the dataset. For example, when the weather extension is activated from a reconciled column, a specific widget allows the user to select the observation dates (that can be kept from another column - Asia recognizes the most common date formats) and the day offset x, i. e., the weather forecast for the next x days using the observation date as base. The user also has to select which aggregation function to apply to the daily weather observations (avg, min, max, cumulative).

At any point of the design phase, users can apply transformation steps to the pipeline (e. g., to transform the result of an extension step). When the design is complete, the application provides the user with different functionalities: (i) to download the enriched dataset in CSV format, or (ii) to generate a new RDF graph, or (iii) to download an executable version of the pipeline to perform the same enrichment steps on a different (possibly larger) dataset, which share the schema with the dataset used in the design phase.[10] The executable pipeline also encodes the validation phases addressed by the user when reconciling columns, so that manual fixes and thresholds can be reused in further executions.

4.3.2 *Supporting the Enrichment at Scale in* ASIA

In order to efficiently support the data enrichment at scale, we adopted specific techniques and strategies in the ASIA design phase. In the following, we discuss our choices and report also about current limitations.

To enable scalability, we rely on a Big Data Environment that promotes parallelism by distribution (horizontal scalability): different computation nodes can work independently and in parallel on different dataset chunks. To maximize the parallelism, a desirable feature is that the enrichment pipeline can be executed both independently and in parallel, on different chunks of the same dataset, thus following the so-called *shared-nothing approach* [107]).

ASIA has been integrated into a real Big Data Environment implemented as a private cloud in our scenario. In this environment, the enrichment pipeline steps are compiled into a chain of dependent processing units (i. e., to follow the original sequence of the pipeline steps), each of which is assigned to a different machine. The dependency between units, and thus their synchronization, is managed by reading/writing intermediary results from/to a shared file system (we point the interested reader to [34] for details on the setting up of the Big Data Environment). Figure 4.5 shows a pipeline featuring three steps, which are compiled into three different processing units, each of which can be replicated on different nodes within a cluster; then, all the machines process the first step on a chunk of the original dataset, resulting in an intermediate result stored in the shared file system; then, all the machines apply the second pipeline step to the intermediate result, and so on until all the enrichment steps are completed. While the different processing units are dependent on each other (due to the pipeline chaining), the replicas of a single processing unit have to be as much independent as possible to maximize the parallelism.

Reconciliation and extensions are single steps of an enrichment pipeline, thus they are compiled to processing units. In order to independently execute such operations, ASIA conciliator services have been implemented following the OpenRefine Reconciliation and Extension APIs; basically, each conciliator receives a list of labels or URIs and

10 The executable is provided as a JAR file, requiring a Java Runtime Environment to be executed. The JAR file takes as input a CSV file and allows the user to produce both the CSV and the RDF version of the enriched dataset.

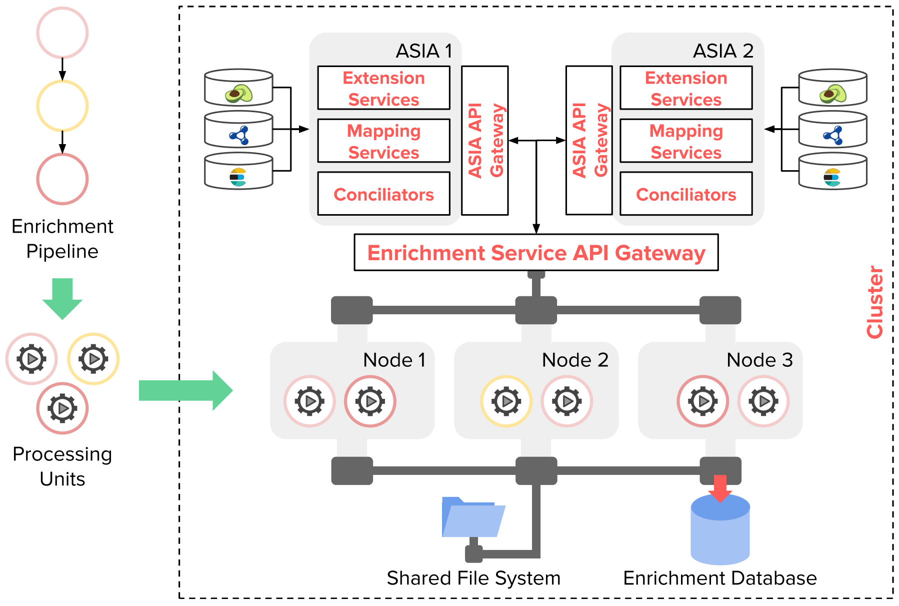


Figure 4.5: An overview of the proposed Big Data Environment.

returns one or more resulting entities (for reconciliation) or values (for extensions). In this way, Asia guarantees a *stateless* execution, which easily enables horizontal scalability: in fact, Asia services can be deployed alongside the cluster nodes, which can invoke the services needed for the enrichment. Since Asia services are stateless, the full system can be replicated on many nodes, distributing the workload on different Asia instances. The service replication is achievable because the involved datasets (i. e., KGs and indexes) are used in read-only mode, thus they can be duplicated, also on-demand, without consistency issues. The workload is distributed by a load balancer (i. e., the Enrichment Service API Gateway in Figure 4.5), which dispatches the enrichment requests across the Asia replicas.

At the same time, Asia extension services (e. g., weather and events) rely on external service providers, which are outside the Big Data Environment. While directly rely on such external providers (i. e., making queries directly against their services) is suitable when working in a small-scale asset (e. g., when using the Asia UI), the same strategy hinders scalability in the Big Data Environment due to the network latency and the high number of needed invocations.

We thus decided to make local the life-cycle of enrichment data. In practice, dumps of external datasets are downloaded in advance from their respective providers (e. g., the weather information are downloaded daily from the ECMWF). This strategy also allows us to preprocess the downloaded dump with information useful to speed up the enrichment process: for example, in this stage, GeoNames URIs are attached to locations available in the downloaded dump, speeding up the weather enrichment based on GeoNames identifiers. However, this solution is suitable for datasets with a known refresh rate, and the refresh frequency depends on the application domain and the nature of data. From the architectural point of view, locally managing these datasets allows us to better size the resources to allocate in terms of computation nodes (e. g., for big KGs, we avoid duplicating the instance, while smaller datasets can be replicated, accordingly with the expected workload).

However, we recognize that the data locality principles are currently partially exploited. In fact, the processing units are deployed on different nodes and perform the transformation pipeline on a physically separated dataset (data are stored in the distributed file system). The main disadvantage of this choice is that the average read/write times double because each node reads a data chunk and writes an enriched data chunk to the file system. To apply the data locality principles as much as possible, a better choice is to directly onboard data chunks on the computation nodes; in this way, the processing units can directly read/write the chunks without transmitting them over the network.

Lastly, it should be noted that values in the same table column may be duplicated, potentially leading to a high number of identical reconciliation/extension requests. To address this issue, we implemented a hierarchical caching system in which each processing unit directly manages the first level of the hierarchy (i. e., two equal values are queried only once), while the ASIA services and databases manage the other levels (i. e., a cached response is returned to identical queries, coming from different processing units). However, the adopted solution has a drawback. Each ASIA replica has its local cache, thus it could be the case that identical requests can be assigned to different replicas of ASIA, causing preventable cache misses. A distributed cache system can be adopted to improve this solution (e. g., Ehcache [122]), so as to share a common cache among multiple replicas of ASIA.

4.4 EVALUATION OF ASIA PERFORMANCE

To test the flexibility and scalability of the proposed solution, we developed a pilot implementation of ASIA, relying on an existing Big Data Environment implemented by SINTEF.[11] The evaluation we hereby describe has been facilitated by JOT Internet Media,[12] a Spanish company operating in the digital marketing domain, which provided us with a real dataset that we exploited for performing three experiments of increasing scale.

The experiments analyzed in the following are based on a subset of the services available in ASIA: the geospatial reconciliation and extension service GN (based on GeoNames) and the weather extension service W (based on ECMWF weather forecasts). GN takes only one value as input (e. g., a toponym listed in a single column of the table)

11 https://www.sintef.no/en/
12 https://www.jot-im.com

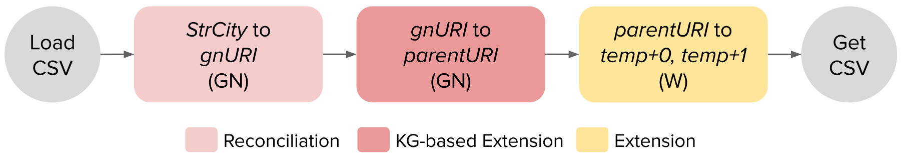


Figure 4.6: The pipeline used in the experimental campaign.

and can be used to generate one (reconciliation) or more (extension) columns, while W takes two attributes as input (location and date), and appends as many columns as the number of desired weather features.

The dataset we used contains 21 columns and describes digital marketing campaign performance indicators collected for three years in Germany and Spain. The experiment has been designed to reflect the real needs of JOT: given a row, the enrichment goal is to add weather information about the region where the city mentioned in the *StrCity* column is located.

To address such a problem, we designed a pipeline like the one sketched in Figure 4.6, which execute the following steps:

- Reconciliation of the column *StrCity* against GN to obtain GeoNames URIs of city toponyms; as a result, a new column *gnURI* containing GeoNames URIs is appended to the dataset.

- KG-based extension of the toponyms contained in the new column *gnURI* to fetch their corresponding first-level administrative division from GN (i. e., regions): given a GeoNames URI, GN returns its `parentADM1` location, which roughly corresponds to the region where the city is located.[13] As a result, the new *parentURI* column is appended to the dataset.

- Extension with weather information about regions in column *parentURI*, i. e., the temperature for the specific date in column *Date*, and the following one, appending two new columns to the dataset: (*temp+0* and *temp+1*).

4.4.1 *Experiment 1: Testing Hierarchical Cache*

The first experiment has been designed to inspect the real performance boost brought by the introduction of the hierarchical cache. We create a scenario where a user designs and executes an enrichment pipeline on a commodity machine over a dataset with 200k rows. An instance of Asia has been installed on a multi-tenant machine.[14]

13 Different countries use different nomenclature to refer to their highest administrative level; for example, Spain has *Autonomous Communities*, while Germany has *States*. We adopt the general term *region*.

14 A server with 4 Intel Xeon CPUs (2.20GHz) and 125GB RAM.

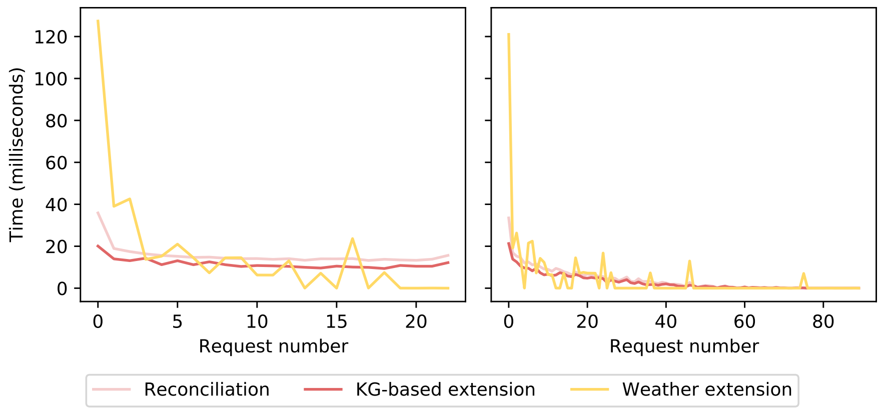


Figure 4.7: Request execution time in milliseconds for the second experiment without duplicates (left-hand side) and with 4 duplicates (right-hand side).

We truncated the pipeline in Figure 4.6 to its first step (reconciliation) for this test,[15] obtaining a short pipeline with only one step. We executed the short pipeline without using the caching strategy: we reconciled the 2,227 different locations available in the dataset, measuring an average time per row of 12.92ms.

We then repeated the same test, but this time we enabled the second level of caching, which is implemented at the reconciliation service level. The same short pipeline ran five times faster, achieving an average processing time of 2.56ms per row. When the first cache layer has been enabled (i. e., the processing units avoid executing the same query twice), the pipeline ran ~770 times faster than the first run without any caching mechanism (0.017ms/row on average). This result is mainly due to the reduced amount of requests over the network, which avoids network latency whenever possible.

4.4.2 *Experiment 2: Testing Cache Over Time*

A second experiment was designed, based on the full pipeline in Figure 4.6, to analyze the behavior of the cache over time.

This pipeline was used to enrich a dataset derived from the one used in the first experiment, filtering out duplicates in the reconciliation target column (i. e., each value occurs at most once), resulting in 2,227 unique cities (and rows). In the first run of this experiment, the cache did not improve the performance because it was built but never used (the table contains unique values only); results are depicted in Figure 4.7 (left-hand side).[16] Afterward, we created a synthetic dataset

15 All the ASIA services implement the same cache mechanism, and we can assume that caching brings the same performance changes to all the services. We can thus safely test only one service.
16 Initial spikes are due to the system startup (e. g., database connectors initialization).

where each line from the previous one was replicated four times, so that to increase the usage of the local cache. The same pipeline has been run to execute the full pipeline, obtaining the results reported in Figure 4.7 (right-hand side); reusing cache speeds up the process progressively (4x on average) and reduces the execution time (we observe a trend that tends to the pure cache access time).

4.4.3 *Experiment 3: Testing Scalability*

We conducted a final experiment to observe the system scalability. We started by running experiments on a single physical machine, where a single instance of the ASIA backend has been deployed.

We sampled the JOT original dataset to obtain datasets of different sizes: 100MB, 1GB, 5GB, and 10GB.[17] Each dataset has been split into 10 chunks of equal size, assigned to 10 processing nodes.[18] The performance is reported in Figure 4.8 (blue points); we can observe that our architecture achieves a linear trend, thus highlighting the scalability of the proposed solution.

Finally, the enrichment of a 100GB dataset was performed on the Big Data Environment deployed on a private cloud. The Big Data Environment has an 8-node cluster of heterogeneous hosts and a shared files system.[19] The processing units of the enrichment pipeline have been deployed on the most powerful hosts in the cluster. The Enrichment Service API Gateway allows each processing unit to access 10 load-balanced replicas of ASIA backend services.

The performance observed in the last run is reported in Figure 4.8 with a red dot. The linear trend with R^2=0.998 is also maintained for the red data point,[20] despite the different contexts in which the experiments have been executed (commodity machine vs. Big Data Environment). This result is mainly due to similar access and reconciliation times between the two experimental configurations.

4.5 SUMMARY

In this Chapter, we discussed a methodology that addresses the efficient enrichment of massive datasets. We targeted a new type of application that is crucial to support analytics workflows at scale: the *semantic enrichment of tabular data* to help users analyze their proprietary data once enriched with third-party data sources. We identified the main challenges in the data science field, where involved data

17 For this dataset, 10MB corresponds to ~50k rows.
18 On a single machine, processing nodes are single processes, which run the enrichment model independently.
19 Five out of eight nodes have 4-core CPUs and 15.4GB RAM, the other three nodes onboard 12-core CPUs and 64GB RAM. The distributed file system manages 6 HDDs (3TB each).
20 The axes in Figure 4.8 uses a base-10 log scale.

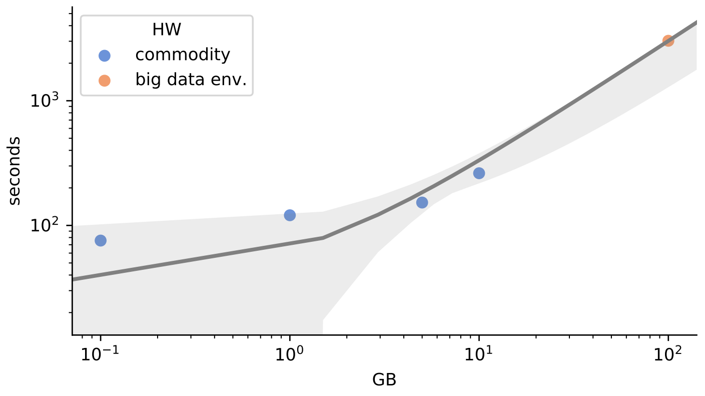


Figure 4.8: Total execution time (in seconds) and linear regression curve, for datasets of different sizes, and two experimental setups.

workers need to integrate large datasets but often have limited programming expertise. We motivated how many applications of this semantic enrichment task can be found in real-world data analytic projects, and we report on a real-life domain such as digital marketing.

Moreover, we proposed ASIA, an open-source solution that modularly integrates different capabilities to assist the user in different phases of the enrichment process, from the design to the execution. The design phase is mainly supported by the ASIA UI, an application that interactively assists users in transforming and enriching datasets, making KGs accessible to non-expert users. The application is sustained by an orchestrated pool of backend services, which support the reconciliation and extension phases, as well as provide users with useful suggestions for completing the schema-level annotation of the dataset. The execution phase is driven by an executable model, which encodes the user-designed enrichment pipeline, and enables the enrichment at a larger scale. At the same time, the executable model supports the repeatability of the task, since the same model can be executed on several different datasets that share a common data schema.

We tested the scalability of an ASIA prototype in an experimental campaign conducted in a real-world asset, thus solving a real enrichment problem for a company. The results we observed highlighted promising performance in terms of scalability, showing that ASIA linearly scales with the dataset dimension, also depending on specific architectural choices we made. We indeed completed the experimental campaign by enriching a 100GB dataset (counting ~500 million rows) with weather information.

ASIA represents a novel system in the literature, which differs from tools like OpenRefine because it also supports the batch execution of enrichment pipelines and provides support to define the schema-level annotation of tables. Compared with traditional data integration tools, Asia exploits the KGs to support the enrichment step.[21] Concerning the schema-level annotation, Asia does not employ sophisticated STA approaches like the ones integrated into Karma [46] but drives the manual schema-level annotation by providing the user with useful suggestions about the usage of shared vocabularies.

However, different extension alternatives may be evaluated in future work to improve Asia: (i) to further improve the system efficiency, we propose to address the drawbacks discussed in Section 4.3.2, i. e., to adopt the data locality principle completely and to replace the hierarchical caching with a distributed one; (ii) to support more reconciliation scenarios, we plan to integrate or extend more sophisticated STI approaches, which is particularly challenging when dealing with large tables [28]; however, integrated STI approaches must support the interactive approach adopted in Asia; (iii) to exploit better the user feedback within the reconciliation step, we plan to exploit learning mechanisms to *propagate* the user choices to other ambiguous cases, not directly evaluated by the user, or to combine the feedback of possibly more than one user as proposed for analogous tasks [26].

21 A detailed comparison can be found as a resource at `https://ew-shopp.github.io/eswc2019-tutorial/`, the tutorial's page where Asia has been presented and compared with other tools, including OpenRefine.

5 EVALUATING ENTITY LINKING FOR TABLES

In this Chapter, we focus on the evaluation of entity linking approaches. As outlined in Chapter 4, STI approaches can be exploited to support data enrichment. However, existing STI approaches are targeted to Web table because the main objective is to support querying the Web. As a result, benchmark datasets to test existing approaches have been built by collecting thousands of Web tables, which are very small (a few hundred rows), but that allow the user to test the algorithm capability to scale to the dimension of the Web. We argue that in the data enrichment context, interpreting Web tables does not cover many real-world scenarios; in fact, company tables are generally a few, but they count millions of rows. In such a setting, STI approaches that consider the full table at once will encounter scalability issues. Existing benchmark datasets do not allow users to test this aspect.

To bridge this gap in the literature, we introduce a novel benchmark dataset that features larger tables, with reference to the dimension of the tables contained in existing datasets [28]. The dataset resembles many situations that occur in not well-maintained data, like the presence of typos, and also increases the level of ambiguity in the dataset, making more challenging the interpretation task. The dataset is mainly targeted to evaluating the CEA task, which is crucial in data enrichment applications. However, the dataset has been extended to cover also the CTA task evaluation. A limitation of our approach is that many situations have been artificially recreated, thus they can be reverse-engineered and easily fixed.

5.1 SEMANTIC TABLE INTERPRETATION EVALUATION

Tables are one of the most used formats to organize data. Every day, data workers and business people have to handle tables that have been extracted from databases of sales, pricing, and more. These tables represent a great source of information for building or populating KGs [103], as well as supporting query answering [110], but firstly the source data have to be manipulated, interpreted within a graph-based schema (e. g., an ontology), transformed, and linked to a reference KG. This operation is usually addressed by solving the STI task (defined in Chapter 2) and mainly consists of linking table elements (i. e., cells and columns) to reference identifiers (e. g., URIs) that are used in larger KGs.

Given an STI algorithm and a benchmark dataset, it is often difficult to understand the shortcoming of the algorithm and how difficult the dataset tables are to annotate. For example, is the algorithm we are evaluating able to annotate tables that contain homonymic names of people?

We argue that different aspects have to be balanced when defining the matching strategy of an STI algorithm: the string matching is a core mechanism, but giving too much importance to its evidence may cause the algorithm to fail in recognizing nicknames and different names for things; also, a too simple string matching strategy (e. g., because a clean text is expected) will fail to identify misspelled entities; conversely, allowing too much fuzziness in the search matching increases the number of candidate entities, i. e., a pool of entities that are selected as potential links, making the disambiguation step more difficult and prone to errors; finally, the popularity of entities is an indicator used to disambiguate entities, but as a consequence, the most popular entities are always preferred by the algorithms, failing to recognize homonymic entities.

Separately evaluating the aspects mentioned above eases the evaluation of the real power of different STI methods in handling the different challenges that data in the tables present. Moreover, building a dataset that enables the evaluation of all these aspects requires collecting non-artificial tables; in fact, if we use a generator for building a dataset (e. g., tables are created by querying a SPARQL endpoint), we have the advantage of creating a multitude of different tables quickly, but at the same time we are prevented from creating tables with new content (i. e., with facts missing in queried KG), or we have to add artificial noise to include some challenges, e. g., typos.

In the last decade, some STI benchmark datasets have been proposed in the literature. As discussed in Section 3.1.4, the most important and used are T2D [102],[1] Limaye [69],[2] and W2D [38], which are all targeted to test STI algorithms with a specific focus on Web tables. Furthermore, we observed that the datasets feature a different amount of tables (~200 tables in T2D, ~485k in W2D), which are relatively small (the average number of rows per table is 123 for T2D, 29 for Limaye, and only 15 for W2D, according to [38]), thus being particularly suitable to benchmark STI algorithms that annotate many small tables, e. g., algorithms to support question answering over tables published on the Web [110].

In our opinion, tables considered for the population of KGs are usually different from Web tables, being more similar to larger legacy tables, which are not represented in the datasets above. We can shortly summarize the difference between these two kinds of tables as follows:

1 The T2Dv2 revised version has been released (`http://webdatacommons.org/webtables/goldstandardV2.html`). However, in this Chapter, we refer to the original version.

2 Different versions of this dataset has been released. In this Chapter, we refer to the one described in [38].

- Legacy tables usually have many rows, while tables in existing benchmark datasets are small. Large tables hinder the usage of heuristics that consider the full table at once (e. g., infer the column type by looking at the whole column), demanding different strategies, e. g., sampling.

- Legacy tables, especially when exported as CSV files, are usually de-normalized tables, counting several columns; this aspect is not well represented in the existing benchmark datasets (each table contains ~1.77 columns with entities on average).

- Because of the de-normalization, legacy tables contain many columns with entity matches, while tables in existing benchmarks are mostly focused on "entity tables", i. e., tables where each row represents only one entity (one-row-per-entity assumption). In some cases (e. g., T2D), additional entity columns, when available in the table, are disregarded and not annotated.

- Entities in Web tables are usually mentioned using their canonical name, while in legacy sources, we find acronyms, abbreviations, misspelled words that considerably increase the ambiguity of the table. For example, the misspelling of drug names is a significant problem in the health domain [50, 52].

In 2019, the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab) was introduced to unify the community efforts towards the systematic evaluation of table annotation algorithms [54], and new benchmark datasets have been proposed. We reported in Section 3.1.4 that some unsolved challenges emerged from the SemTab evaluation, including the need for tables with non-canonical mentions (e. g., nicknames and aliases), misspelled entries (noisy data), missing data, and without false positives that negatively affect the performance of the algorithms (some tables in SemTab inherit annotation errors from T2D and W2D). Moreover, the existing benchmark datasets, as well as the new datasets provided in SemTab, do not provide enough fine-grained information about the achievement of a specific performance score (e. g., if the algorithm fails to annotate a specific type of tables), requiring a manual inspection of the error patterns (some tools have been developed to ease this task, like StilTool [23], which highlights the main error patterns).

5.2 TOUGH TABLES

To address the challenges described above, we constructed 2T (Tough Tables), a novel manually curated dataset designed to evaluate STI approaches, and, in particular, to evaluate specific aspects of the CEA task. While the focus is on CEA because of the central role it plays in data enrichment, 2T has been extended to provide also CTA annotations (more details will be given in Section 5.3).

The annotations in 2T have been manually revised, i. e., we checked the annotations with the following question in mind: "Would a human annotator be able to disambiguate values in this table?". Considering the intrinsic ambiguity of references appearing in tables, we want to ensure that the dataset includes those tables that can be effectively disambiguated by a human annotator, based only on the information available in the table. Indeed, in some cases also the human annotators found it very hard to match cells in some tables due to the high ambiguity of their content (e. g., for a table containing the list of bank failures since 2000, it was not always clear if a bank *A* has been acquired by another bank *B*, or by the *B*'s holding company, which are different entities with very similar names). When the correct link was hard to be decided based on the table content (the context supporting the disambiguation), we decided to leave the annotation empty, preserving the quality of the dataset.

Compared to the previous benchmark dataset, 2T has the following distinguishing features, which make it a precious resource for evaluating the CEA task:

1. *Real Tables*, useful for testing how STI algorithms deal with the knowledge gap due to novel facts. It can often be the case that some cells in a table refer to entities that are described in the reference KG, for which the algorithm is expected to link the correct entity, and some cells refer to entities not described in the reference KG, for which the algorithm should decide not to link any of the existing entities.

2. *Tables with Ambiguous Mentions*, useful for testing the algorithms' capability to handle the ambiguity and link to non-popular entities (tail entities).

3. *Tables with Misspelled Mentions*, useful for testing the weights of lexical features used by the algorithms. We used the misspelled words as a generator to add controlled noise to other tables.

4. *Tables from Various Sources*, useful for understanding the kind of table that is the most difficult to deal with for an algorithm.

5. *Manually Verified Tables*, useful to prevent false positives while evaluating the algorithms; 2T is of high quality, and all the annotations have been manually verified.

2T has been shaped in such a way to allow the users to know which aspects of the entity linking task are handled better/worse by different approaches (see Figure 5.1). Indeed, 2T comes with two main categories of tables:

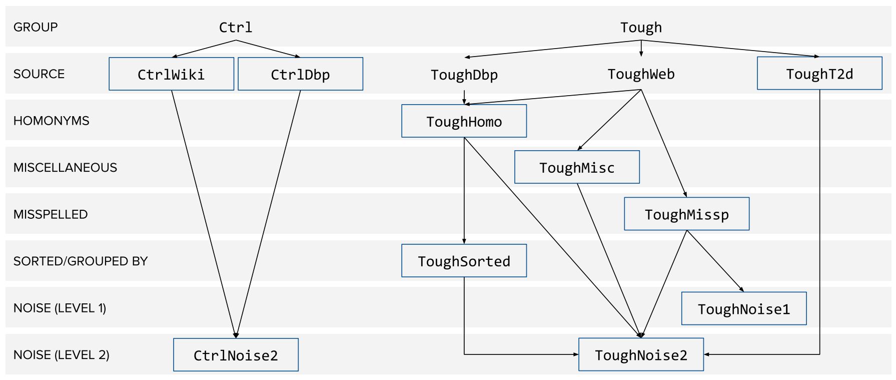


Figure 5.1: 2T profile at a glance. Boxed categories are those considered during the evaluation phase. Each category is composed by some of/all the tables from the parent categories.

- The *control* group (`Ctrl`), which contains tables that are easy to annotate; an STI algorithm should at least annotate these tables with relatively high performance.
- The *tough* group (`Tough`), which features only tables that are hard to annotate.

A complete STI algorithm should properly annotate tables in both the categories because otherwise (i) solving only the `Ctrl` group means that the algorithm is able to cope with obvious entities only, and (ii) solving only the `Tough` tables highlights that the algorithm is too complex and cannot deal with the more straightforward cases.

5.2.1 *Dataset Profile*

Figure 5.1 provides an overview of the different table flavors included within 2T. The `Ctrl` group has been built by collecting tables from Wikipedia (`CtrlWiki`) and querying the DBpedia SPARQL endpoint.[3] Tables in `CtrlWiki` have been manually revised and annotated (further details about this procedure are available in Section 5.3), while tables in `CtrlDbp` are mostly left as returned by the SPARQL endpoint. Entities in such tables are pretty explicit and easy to annotate since most of the mentions refer to DBpedia entities with just slightly different labels from the ones contained in the KG.

The `Tough` group contains mainly tables scraped from the Web. This group contains a small subset of T2D tables (`ToughT2d`), which we re-annotated considering the entities appearing in all the columns (in the original dataset, only the "subject" column is annotated, usually the left-most column). In addition, we collected tables from the Web that contain nicknames or homonyms (`ToughHomo`) and misspelled

3 We used the online version at `http://dbpedia.org/sparql`, which is based on the 2016-10 release.

words (ToughMissp). The ToughHomo category has been extended by adding a few tables generated via ad-hoc SPARQL queries (e. g., using properties that point to entity aliases). Alongside these tables, we included other Web tables (ToughMisc), like non-cleaned Wikipedia tables and tables available as Open Data (in a limited quantity, due to motivations stated in Section 5.3).

To increase the difficulty of the dataset, we selected some specific tables from the ToughHomo category and sorted them in a specific order (ToughSorted). In this way, the task of detecting the column type may be harder; as an example, if we are creating a table describing athletes, we would reasonably organize them by category, having the soccer players in the first part, followed by the basket players, then by the golf players, and so on. Sorting table values poses a problem for those algorithms that infer the column type by only looking at the first *n* rows (with *n* usually small), and then use the inferred type as a filter for the entity linking.[4]

NOISY TABLES Starting from the collected tables, we generated additional noisy Tough tables. Tables in the ToughMissp category are then used to generate a new category of tables, i. e., ToughNoise1, by adding a *level-1* noise: for each table, 10 additional tables with noise have been generated, where each of them contains an incremental percentage of misspelled mentions (increasing by 10% at a time). This noise resembles real noise since we use lists of real-world misspelled words to add noise in tables.

We processed tables in the Ctrl and Tough categories (excluding the ToughNoise1 category) in a similar way, thus creating two new categories (CtrlNoise2 and ToughNoise2) by adding a *level-2* noise, i. e., random noise that changes the labels of randomly selected columns and rows (e. g., it randomly duplicates a symbol). Tables in this new category feature a noise that is random and artificial, thus it does not always resemble a real-world scenario.

NOVEL FACTS One of the main applications of STI is the KG population, where new facts described in a table have to be included in a KG. In data integration pipelines, entity linking and new triples generation play an equally important role. Novel facts detection is not considered in the standard CEA evaluation as defined by SemTab,[5] but we outline that our dataset can be used to test algorithms in finding new facts. 2T tables contain 3,292 entity mentions across 42 tables without a corresponding entity in DBpedia 2016-10. In the CEA asset, an STI algorithm is expected to decide to no link such cells to any entity (e. g., using the *NIL* annotation as in Named Entity Linking/Recognition).

4 This strategy might look naive, but it is the one implemented in OpenRefine, where the first 10 rows are used to infer the possible types of the current column.

5 The SemTab 2019 challenge provided the target file with the full list of cells to annotate with an entity, disregarding novel facts.

In more general assets, like KG construction or population, we expect the algorithm to exploit the annotations to generate a new triple (e. g., using the discovered column type to create a triple with the `rdf:type` property). Depending on the context, such particular cells might be used in the future to test novel knowledge discovery algorithms.

OVERVIEW Benchmarking an STI algorithm using 2T enables a clear understanding of the overall performance of the algorithm. Indeed, by looking at the results achieved in specific categories, we can conclude that:

- If the algorithm performs well only in `Ctrl`, then it relies too much on the performance of simple string matching strategies like label lookup (i. e., it looks only for exact matches or considers only the canonical name of entities);
- If the performance is excellent also in `CtrlNoise2`, then the algorithm adopts a kind of fuzziness in its lookup phase (e. g., small edit distance), but this strategy is not suitable to solve the `Tough` tables;
- If the algorithm performs well only on some `Tough` tables, then we can investigate the weaknesses of the algorithm by looking at the performance on different categories of tables:
  - If the `ToughSorted` tables are annotated in the wrong way, then the algorithm is constrained by the column type inferred by looking at the first n rows, with n too small;
  - If homonyms or nicknames have been wrongly matched, then the algorithm employs popularity mechanisms (e. g., PageRank), or it is based on a lookup service that returns the most popular entities first (e. g., DBpedia Lookup).[6] Annotating nicknames requires the algorithms to cover aspects of semantics that go a bit beyond simple heuristics;[7]
  - If `ToughNoise1` tables are not correctly annotated, then the algorithm cannot deal with real-world noise (that can be trickier than the artificial level-2 noise);
  - If the annotations are wrong for the `ToughHomo` tables, it might be the case the algorithm only focuses on the canonical names of the entities.

6 We point out that some homonyms are very easy to solve using DBpedia. For example, cities in U.S. are easy to find, since the concatenation of the city name with its state name points directly to the right city, e. g., the Cambridge city in Illinois is `dbr:Cambridge,_Illinois` in DBpedia).

7 Note that it is possible to solve this problem using a mapping dictionary if available, but this is not the desired solution because it will not make the algorithm *smart*; the same is valid for looking up on Google Search.

Table 5.1: Detailed statistics for 2T. Values are formatted as *avg* ± *st.dev. (total, min, max)*.

Category	Cols	Rows	Matches	Entities	Cols with Matches	Tables
ALL	4.46±1.90 (802, 1, 8)	1,080.21±2,805.31 (194,438, 5, 15,477)	3,686.98±10,142.60 (663,656, 6, 61,908)	435.92±1,241.93 (15,997, 6, 7,032)	3.00±1.18 (540, 1, 6)	180
`Ctrl Wiki`	5.73±1.28 (86, 4, 7)	66.00±81.54 (990, 10, 263)	241.73±333.57 (3,626, 20, 1,040)	155.53±223.18 (1,904, 15, 769)	3.47±1.41 (52, 2, 6)	15
`Ctrl Dbp`	4.40±0.91 (66, 3, 6)	708.60±718.31 (10,629, 120, 2,408)	2,507.60±2,575.55 (37,614, 360, 7,820)	336.20±212.39 (4,872, 68, 613)	3.53±0.64 (53, 3, 5)	15
`Ctrl Noise2`	5.07±1.28 (152, 3, 7)	387.30±599.25 (11,619, 10, 2,408)	1,374.67±2,141.00 (41,240, 20, 7,820)	245.87±232.95 (6,606, 15, 769)	3.50±1.07 (105, 2, 6)	30
`Tough T2d`	5.73±1.85 (63, 3, 8)	78.09±77.26 (859, 6, 232)	170.00±151.05 (1,870, 6, 464)	95.27±78.92 (1,001, 6, 247)	2.18±1.08 (24, 1, 4)	11
`Tough Homo`	3.36±1.12 (37, 2, 5)	1,648.73±3,272.16 (18,136, 13, 8,302)	6,421.18±13,169.01 (70,633, 25, 33,208)	1,465.45±2,720.49 (8,254, 23, 7,032)	3.00±0.77 (33, 2, 4)	11
`Tough Misc`	6.50±1.31 (78, 4, 8)	122.25±162.86 (1,467, 11, 561)	366.25±416.77 (4,395, 22, 1,214)	220.25±261.07 (2,344, 16, 769)	3.67±1.44 (44, 2, 6)	12
`Tough Missp`	3.50±1.29 (14, 2, 5)	4,175.50±7,549.48 (16,702, 52, 15,477)	16,379.50±30,385.95 (65,518, 178, 61,908)	201.75±347.73 (763, 11, 723)	3.25±0.96 (13, 2, 4)	4
`Tough Sorted`	3.50±2.12 (7, 2, 5)	4,215.00±5,779.89 (8,430, 128, 8,302)	16,732.00±23,300.58 (33,464, 256, 33,208)	3,602.00±4,850.75 (7,201, 172, 7,032)	3.00±1.41 (6, 2, 4)	2
`Tough Noise1`	2.50±1.13 (100, 1, 4)	2,000.30±3,701.62 (80,012, 5, 14,008)	5,735.40±11,198.09 (229,416, 15, 42,024)	201.75±304.98 (763, 11, 723)	2.25±0.84 (90, 1, 3)	40
`Tough Noise2`	4.97±1.97 (199, 2, 8)	1,139.85±3,183.54 (45,594, 6, 15,477)	4,397.00±12,774.58 (175,880, 6, 61,908)	695.55±1,824.20 (11,479, 6, 7,032)	3.00±1.22 (120, 1, 6)	40

Tables 5.1 and 5.2 show statistics about the profile of 2T and existing benchmark datasets.[8] Comparing 2T to existing datasets, we observe that 2T has a higher average number of rows per table, pushing the size of individual tables towards the size of real legacy tables; the number of matches is slightly greater than the number available in ST19-R2 and ST19-R3, also if 2T comes with a number of tables that is up to two orders of magnitude smaller. Since some tables in 2T are built starting from the same core table, we observe a small number of unique entities. Finally, 2T tables have a lower average number of columns per table, but the highest number of columns with at least a match: this aspect helps in having more columns to annotate in the CTA task, and it is also a starting point for future extensions of 2T, i. e., covering the CPA task.

5.3 DATASET CONSTRUCTION

The 2T dataset has been built using real tables, meaning that we look for tables, also artificially built, which resembles real tables. Examples

8 The statistics we reported in this thesis are for 2T v1.0, thus slightly differ from the one provided in [28], which refer to 2T v0.1-pre. Moreover, the header rows were considered while counting rows in [28], while here are discarded.

Table 5.2: Comparison with existing benchmark datasets. Statistics for Limaye and W2D are from [38].

Dataset	Cols (avg)	Rows (avg)	Matches	Entities	Cols with Matches (avg)	Tables
T2D	1,153 (4.95)	28,333 (121.60)	26,124	13,785	233 (1.00)	233
Limaye	- (3.79)	8,670 (29)	5,278	-	-	296
W2D	- (5.58)	7,437,606 (15)	4,453,329	-	-	485,096
ST19-R1	323 (5.05)	9,089 (142.02)	8,418	6,222	64 (1.00)	64
ST19-R2	66,734 (5.60)	300,794 (25.23)	463,796	229,280	15,335 (1.29)	11,920
ST19-R3	9,736 (4.51)	152,753 (70.69)	406,827	174,338	5,762 (2.67)	2,161
ST19-R4	3,564 (4.36)	51,249 (62.73)	107,352	53,007	1,732 (2.12)	817
2T	802 (4.46)	194,438 (1,080.21)	663,656	15,997	540 (3.00)	180

of such tables are "list of companies with their market segment", or "list of Italian merged political parties", which look like results of queries made by a manager or a journalist against a database. We also included artificial tables because obtaining only real tables (e. g., tables from real databases) is very difficult because of some factors: (i) having access to real databases and making the data public is reasonably impractical; (ii) many database dumps are available as open data, which effectively represent a great source of tables; however, such tables usually contain aggregated data (e. g., statistics) to preserve company data, making it difficult to annotate them with entities from a general KG like DBpedia. When the data are fine-grained enough to enable the annotation, almost all the mentioned entities are not available in a general KG. For example, we annotated a table containing the list of bank failures from the U.S. Open Data Portal:[9] only 27 over 561 failed banks are represented in the considered DBpedia KG.

5.3.1 *CEA Ground Truth Contruction*

In this Section, we describe the specific processes we adopted to collect real tables, or build artificial tables that resemble real ones, and how we annotated them.

DBPEDIA TABLES We used the DBpedia SPARQL endpoint as a table generator (SPARQL results are tables). We run queries to generate tables that include:

- Entity columns: columns with DBpedia URIs that represent entities.
- "Label columns": columns with possible mentions for the corresponding entities in the entity column. Given an entity column,

9 https://www.data.gov/

the corresponding label column has been created by randomly choosing between `rdfs:label`, `foaf:name`, or `dbo:alias` properties.

- Literal columns: other columns, with additional information (e. g., average annual incomes of companies).

The tables are almost left as returned by the endpoint; only a few tables required manual fixes because of obvious errors, e. g., the entity `Kazakhstan` has an empty name (“”). The collected URIs are used as the right annotations of the labels listed in the label column.

WIKIPEDIA TABLES We browsed Wikipedia looking for pages containing tables of interest (e. g., list of presidents, list of companies, list of singers). We generated different versions of the collected Wikipedia tables, applying different cleaning steps. The following steps have been applied to Wikipedia tables in the `ToughMisc` category:

- Merged cells have been split into multiple cells with the same value.
- Multi-value cells (slash-separated values, e. g., Pop / Rock, or multi-line values, e. g., Barbados <br> United States, or in-line lists, e. g., <ul>, <li>) have been exploded into several lines. If two or more multi-value cells are on the same line, we exploded all the cells (as the cartesian product of all the values). If a cell contains the same information in more languages (e. g., anthem song titles), we exploded the cell in two or more columns (creating new lines will basically produce duplicates).

Wikipedia tables in the `CtrlWiki` group underwent the next additional cleaning steps:

- “Note”, “Description”, and similar long-text columns have been removed.
- Cells with “None”, “null”, “N/A”, “Unaffiliated”, and similar values have been cleared.
- Columns with only images (e. g., list of U.S. presidents) have been removed.
- All HTML tags have been deleted from cells (e. g., country flag icons).
- Notes, footnotes, and any other additional within-cell information (e. g., birth year and death year of U.S. presidents) have been removed.

Most of the values in tables are already mapped to their Wikipedia page via hyperlinks. We used the hyperlinks as the correct annotations (we trust Wikipedia as a correct source of information since it is maintained and verified by the community), following these criteria:[10]

- If a cell content has several links, we took the most relevant annotation, given the column context (e. g., in the table about U.S. presidents,[11] the "U.S. senator from Tennessee" cell in the "Prior office" column contains two annotations: wiki:U.S._senator and wiki:Tennessee; in this case we took only the wiki:U.S._senator annotation, as the column is about prior offices, not about places).

- Sometimes, if the same value appears several times in the same column (e. g., music genres), only one instance has the hyperlink to the Wikipedia page. In these cases, we copied the same hyperlink to all the instances.

- When the hyperlink is missing (e. g., "Hard Rock" labels in the table of best-selling music artists),[12] we manually added the right link by visiting the main entity page (e. g., wiki:Led_Zeppelin) and looking for the missing piece of information (e. g., under the "Genre" section on the Led Zeppelin page, we can find "Hard Rock" linked to wiki:Hard_rock). In case when the information is missing on the main page (e. g., in the same table, Michael Jackson genres include "Dance", while on his Wikipedia page the genre is Dance-pop), we manually annotated the value with the most related entity in Wikipedia (in this case, the music genre Dance wiki:Dance_music).

Finally, we converted the Wikipedia links to their DBpedia correspondent links by replacing the prefix wiki: with dbp: in the decoded URI (e. g., wiki:McDonald%27s is replaced with dbp:McDonald's), if available, otherwise, we manually looked for the right DBpedia URI (e. g., wiki:1788–89_United_States_presidential_election is replaced with dbp:United_States_presidential_election,_1788–89). If this attempt also failed, we left the annotation blank (no annotations available in DBpedia).

OTHER TABLES Finally, tables in 2T have been collected from other sources (e. g., Open Data portals, domain-specific website, T2D). We manually annotated entities mentioned in such tables and revised them by ensuring that the table content was enough to properly disambiguate the entities (i. e., without querying the Web to obtain additional information). For example, we had to remove the "Florida,

10 In the following, wiki: and dbp: stand for https://en.wikipedia.org/wiki/ and http://dbpedia.org/resource, respectively.

11 https://en.wikipedia.org/wiki/List_of_presidents_of_the_United_States#Presidents

12 https://en.wikipedia.org/wiki/List_of_best-selling_music_artists#250_million_or_more_records

New York, NY" entry from a table because it is impossible to disambiguate without obtaining extra information: in fact, according with the Florida disambiguation page in DBpedia,[13] there are a town and a village named Florida in two different counties of New York, thus it is impossible to disambiguate the cell without knowing the county of the mentioned entity.

5.3.2 *CTA Ground Truth Construction*

The 2T dataset focus is mainly on evaluating CEA because this task is crucial in semantic data enrichment applications, but also it is the core component of many STI approaches, as seen in Chapter 3: indeed, good performance in CEA enables to approximate the CTA task. We exploited this observation to automatically construct the CTA ground truth starting from the CEA one, which we trust as correct. We derived the CTA ground truth employing a majority voting strategy: for each annotated column, we collected its annotated entities from the CEA ground truth and retrieved the most specific type for all the entities.[14] Finally, we annotate the column with the most specific common type, i. e., the lowest common ancestor of all the types. To find the most specific common type, we explored the DBpedia 2016-10 ontology.[15]

5.4 EXPERIMENTS

In this Section, we describe the experiments we conducted to evaluate the toughness of our dataset, as well as its capability to spot the weaknesses of STI algorithms. We recreated the same environment used in SemTab 2019 for evaluating , i. e., target cells are known, and extra annotations, if any, are disregarded. We evaluate the STI algorithms using the evaluation code used in SemTab 2019,[16] thus computing the macro Precision (P), Recall (R), and F1-score (F1) metrics defined as follows:[17]

$$P = \frac{|CorrectAnnotations|}{|SystemAnnotations|} \quad R = \frac{|CorrectAnnotations|}{|TargetAnnotations|}$$
$$F1 = \frac{2 \times P \times R}{P + R}$$

We introduced two simple baselines that annotate a cell by exploiting external lookup services, queried with the actual cell content. The baselines are DBLookup and WikipediaSearch, which query the corre-

13 https://dbpedia.org/page/Florida_(disambiguation)

14 Specific types of entities in DBpedia 2016-10 are available in the instance type dump (http://downloads.dbpedia.org/2016-10/core-i18n/en/instance_types_en.ttl.bz2).

15 http://downloads.dbpedia.org/2016-10/dbpedia_2016-10.nt

16 https://github.com/sem-tab-challenge/aicrowd-evaluator

17 $SystemAnnotations$ = cells annotated by the algorithm; $TargetAnnotations$ = ground truth cells.

sponding online lookup service.[18] Both the baselines annotate a cell using the first result returned by the online service without further processing.

Alongside the baselines, we searched for algorithms to test among the ones that participated in SemTab 2019. We contacted the authors of MTab [85], CSV2KG [115], and Tabularisi [112], the top-3 systems, but they reported that their systems were not ready to be released. We found that the source code of MantisTable [22], awarded with the *outstanding improvement* award at SemTab 2019 (CEA task), is publicly available.[19] Since the scores obtained by all the tools in the CEA task were similar to each other, we considered MantisTable as a good representative STI algorithm for our evaluation.

In addition, we consider other state-of-the-art algorithms that did not participate in the challenge. These algorithms are FactBase, EmbeddingOnGraph, HybridI and HybridII, which have been published in [38].[20] In this way, we tested 2T with general algorithms, which are not tailored to the specific SemTab challenge.

We run all the algorithms on 2T, obtaining the results depicted in Figure 5.2. We underline here that our dataset adopts the same standard format defined in SemTab 2019, thus it is compatible with all the systems that participated in the challenge.[21]

RESULTS In some cases, we experienced scalability issues using MantisTable, which failed to annotate some tables (e. g., it failed to annotate tables with thousands of rows). In our opinion, the scalability issues are due to the several heuristics computed within the matching algorithm, which lead to processing errors when dealing with big tables. Moreover, the complexity of the matching algorithm increases the computational costs: MantisTable took more than 24 hours to process all the 180 tables of 2T. However, this aspect does not compromise our evaluation, proving again that MantisTable can effectively annotate tables as shown in the SemTab 2019 challenge. The reader should consider that the reported results may not reflect the full performance of MantisTable as they represent the output of a dry-run test without the involvement of the developers. Nevertheless, the results suggest that tables in 2T are effectively difficult to annotate for state-of-the-art algorithms.

18 We used the WikipediaSearch online service available at `https://en.wikipedia.org/w/api.php`, while we recreated the DBpedia Lookup service on a dedicated virtual machine.

19 A fork of the original code repository is available at `https://bitbucket.org/vcutrona/mantistable-tool.py`.

20 The original source code is not available. We re-implemented the algorithms from scratch, as described in [38]. More details about the algorithms and their re-implemented versions are available in Chapter 6.

21 The standard format introduced in SemTab 2019 is directly derived from the T2D one, thus the number of algorithms that can be tested is potentially more significant.

Table 5.3: Results for the considered algorithms over 2T. Best results in **bold** (**P**, **R**, **F1**).

Algorithm		ALL	Ctrl Wiki	Ctrl Dbp	Ctrl Noise2	Tough T2d	Tough Homo	Tough Misc	Tough Missp	Tough Sorted	Tough Noise1	Tough Noise2
WikipediaSearch	P	0.60	0.79	**0.93**	0.86	**0.75**	0.55	0.61	0.67	0.54	0.57	0.51
	R	**0.51**	**0.75**	**0.87**	**0.79**	**0.70**	0.55	0.54	0.51	0.54	0.42	**0.46**
	F1	0.55	**0.77**	**0.90**	**0.82**	**0.72**	0.55	0.57	0.58	0.54	0.48	**0.48**
DBLookup	P	0.66	0.76	0.85	0.83	0.74	0.61	0.70	**0.75**	0.61	**0.67**	0.57
	R	0.50	0.70	0.74	0.57	0.68	**0.61**	**0.59**	**0.52**	**0.61**	**0.46**	0.40
	F1	**0.57**	0.73	0.79	0.67	0.70	**0.61**	**0.64**	**0.62**	**0.61**	**0.54**	0.47
MantisTable	P	**0.76**	**0.81**	0.90	**0.87**	0.66	**0.69**	**0.72**	0.67	**0.67**	0.49	**0.72**
	R	0.10	0.67	0.48	0.43	0.61	0.06	0.50	0.02	0.03	0.03	0.07
	F1	0.18	0.73	0.62	0.58	0.64	0.10	0.59	0.05	0.05	0.06	0.12
FactBase	P	0.36	0.71	0.80	0.77	0.62	0.36	0.62	0.36	0.42	0.07	0.37
	R	0.19	0.48	0.58	0.48	0.49	0.29	0.36	0.15	0.28	0.03	0.18
	F1	0.25	0.57	0.67	0.59	0.55	0.32	0.45	0.21	0.34	0.04	0.24
EmbeddingOnGraph	P	0.29	0.54	0.68	0.59	0.51	0.31	0.51	0.35	0.33	0.12	0.27
	R	0.24	0.51	0.64	0.54	0.49	0.31	0.47	0.25	0.33	0.09	0.24
	F1	0.27	0.53	0.66	0.56	0.50	0.31	0.49	0.29	0.33	0.10	0.26
HybridI	P	0.28	0.59	0.71	0.64	0.56	0.31	0.56	0.27	0.31	0.04	0.30
	R	0.23	0.58	0.67	0.60	0.54	0.31	0.52	0.19	0.31	0.03	0.26
	F1	0.25	0.58	0.69	0.62	0.55	0.31	0.54	0.22	0.31	0.04	0.28
HybridII	P	0.30	0.55	0.68	0.59	0.51	0.31	0.51	0.35	0.33	0.13	0.27
	R	0.25	0.53	0.65	0.55	0.50	0.31	0.48	0.25	0.33	0.09	0.24
	F1	0.27	0.54	0.66	0.57	0.51	0.31	0.49	0.30	0.33	0.11	0.26

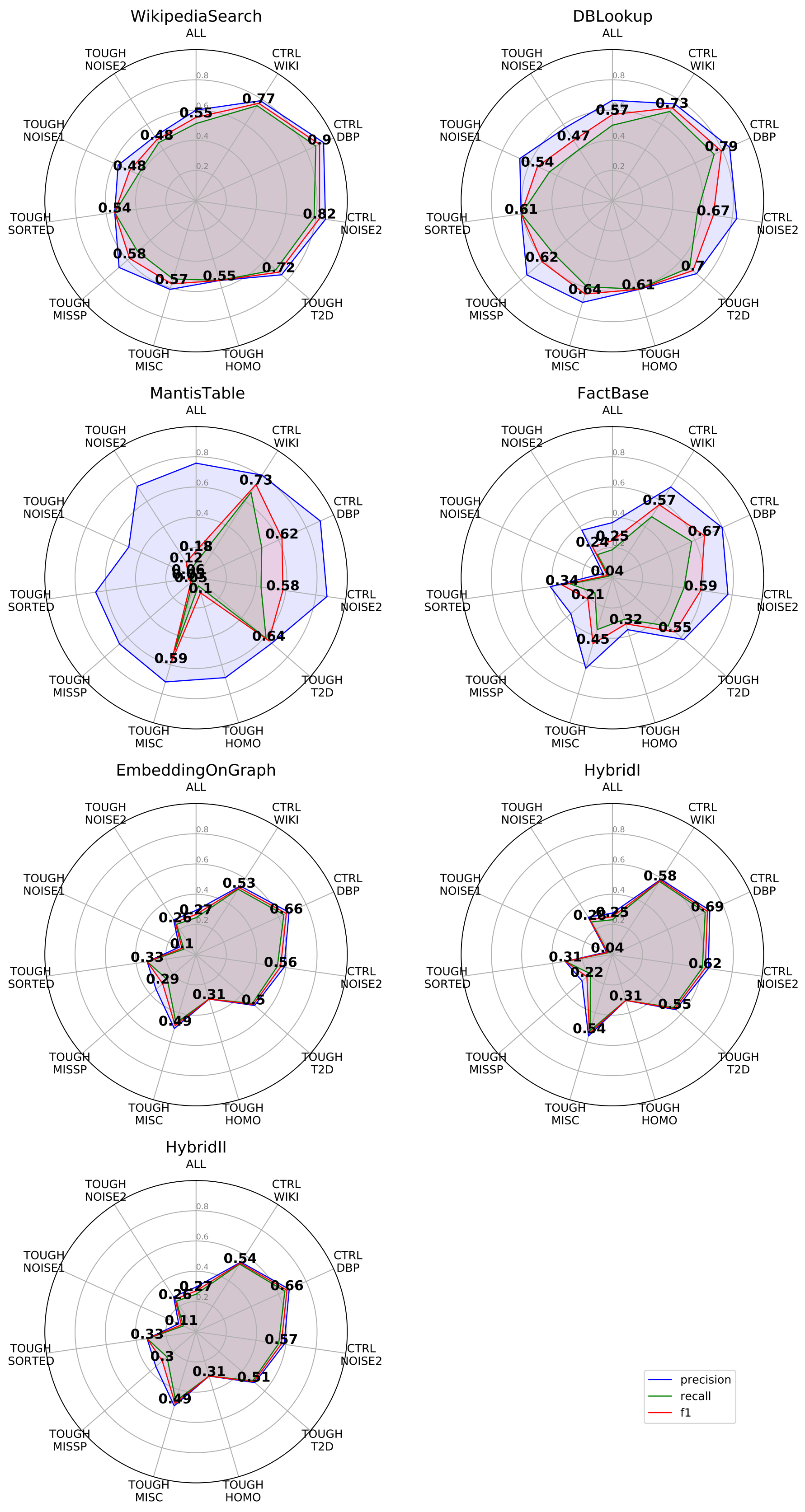


Figure 5.2: CEA performance footprints of the considered algorithms over 2T. F1 score in **bold**.

The precision and recall of the baseline algorithms are quite similar since the online lookup services almost always return a result, and we set it as the annotation without further processing; focusing on the F1, we observe that WikipediaSearch reaches 0.83 F1 on the `Ctrl` group, which is high compared with state-of-the-art models, considering that the process only relies on the lookup service. The performance of DBLookup is good as well but decreases due to the `CtrlNoise2` subcategory, reaching an average F1 of 0.73 on the `Ctrl` group. Both the baseline algorithms are not able to annotate tables in the `Tough` group, with DBLookup doing a bit better; this might be because some of the tables in 2T have been built with SPARQL queries, giving some advantages to the DBLookup service. In general, the performance is low as expected (0.63 overall F1 for both the baselines), given that these algorithms do not use any sophisticated semantic techniques.

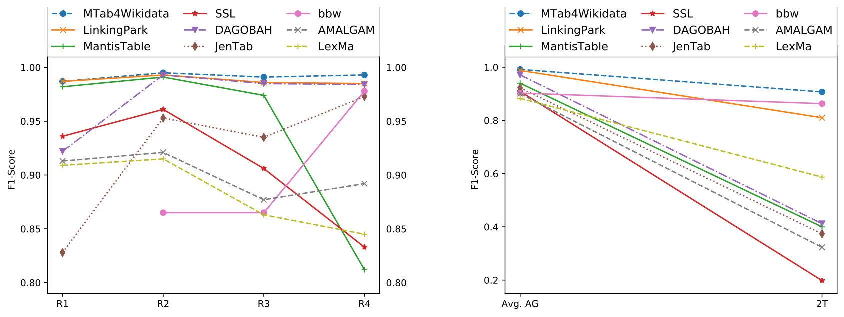
Looking at results of MantisTable, we see that the tool focuses on those cells that can be more easily linked to the KG. The semantic techniques employed in the algorithm push the precision on the `Ctrl` group, but due to the low recall, the algorithm performs worse (0.32 overall F1) than the previous baselines. Since the precision on `CtrlDbp` tables is higher than `CtrlWiki` ones, we can assume that the lookup phase of the algorithm heavily depends on results from DBpedia (as a lookup service or SPARQL endpoint). The low recall confirms the same on the `ToughNoise1` tables, which are the ones with real-world misspelled mentions. For the subset of T2D tables that we chose and re-annotated, we spot an F1 score lower than the 0.98 obtained by MantisTable during the Round 1 of the SemTab challenge;[22] this confirms that the T2D tables are focused on obvious entities, disregarding the more difficult ones. The F1 score drastically decreases on the misspelled tables, highlighting that this aspect is still not fully covered in sophisticated state-of-the-art approaches like MantisTable.

FactBase and EmbeddingOnGraph perform the worst in annotating 2T tables, but differently from the previous algorithms, they do not exploit external lookup services (i. e., the lookup search is performed against a private index). Using a dedicated index makes the annotation process harder because the searching phase (i. e., defining the query to submit to the index) is in charge of the algorithm. Results for `ToughMissp` and `ToughNoise1` categories are the worst; we can conclude that the candidates retrieved by both the algorithms are filtered out using too high thresholds, which often remove the right entity from the candidates set. Furthermore, EmbeddingOnGraph exploits popularity indicators (i. e., entities are scored by observing the in-links and out-links of their Wikipedia pages) that tend to reward popular entities; in fact, we can observe poor performance in the `ToughHomo` category. The recall is higher with respect to MantisTable results, in-

22 https://www.cs.ox.ac.uk/isg/challenges/sem-tab/2019/results.html

Table 5.4: Average F1-score for the top-10 systems discarding outliers [55].

	Automatically Generated (AG)				**Tough Tables (2T)**
	Round 1	**Round 2**	**Round 3**	**Round 4**	**Round 4**
CEA	0.93	0.95	0.94	0.92	0.54
CTA	0.83	0.93	0.94	0.92	0.59



(a) AG Datasets (b) Average AG vs 2T

Figure 5.3: Results in the CEA task with the Automatically Generated (AG) and 2T Datasets [55].

deed, FactBase and EmbeddingOnGraph have annotated much more tables than MantisTable, thanks to their higher scalability.

HybridI and HybridII differently combine FactBase and EmbeddingOnGraph; in general, these methods improve the recall but sacrifice the precision, usually leading to better results in terms of F1.

5.4.1 *Results from SemTab 2020*

In 2020, a new edition of the SemTab challenge was organized, using Wikidata as the target KG and releasing a set of new automatically generated datasets (AG). We converted the ground truth of 2T so that to link values in cells to entities in Wikidata.[23] As a result, we obtained a new version of 2T, which we used as a benchmark dataset for the last round (Round 4) of SemTab 2020 [55].

Collected results are available in Table 5.4. We can recognize a significant complexity brought by 2T if we observe the average results in Round 4, with respect to the previous rounds based on AG. We underline that Round 4 was blind, i. e., the participants were prevented from scoring their submission, thus we evaluated how participant systems were able to deal with the challenges in 2T without the chance to tweak and overspecialize the algorithm parameters. 9 systems have produced results for CEA, while only 8 for CTA.

23 We processed `owl:sameAs` links available for DBpedia and Wikidata, with a few manual fixes when the links were missing. More details are available in the 2T repository at https://github.com/vcutrona/tough-tables.

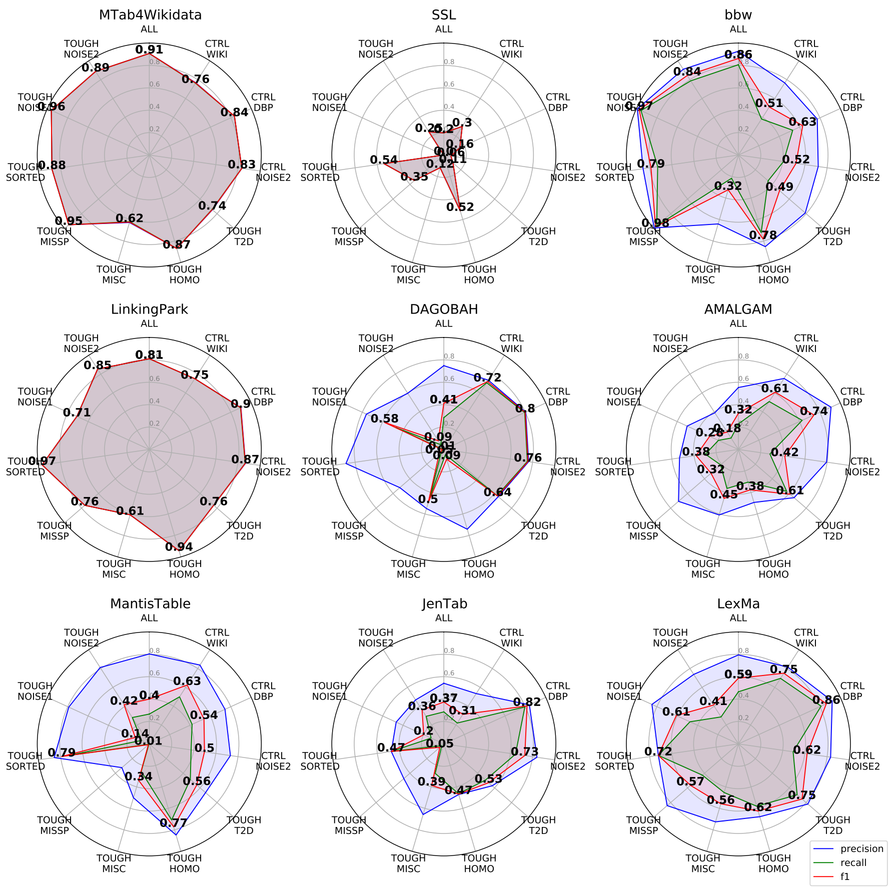
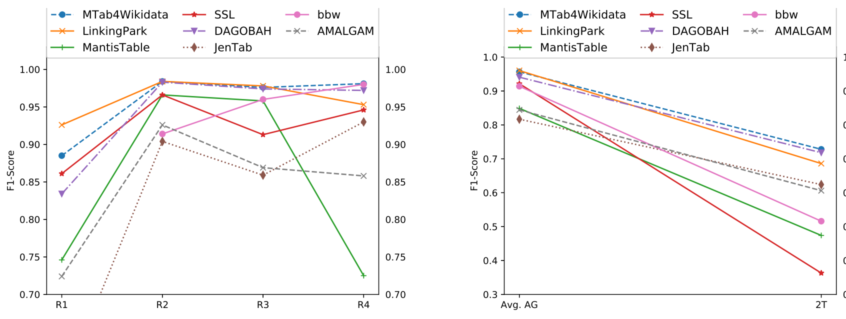


Figure 5.4: CEA performance footprints of the considered algorithms over 2T in SemTab 2020. F1 score in **bold**.



(a) AG Datasets (b) Average AG vs 2T

Figure 5.5: Results in the CTA task with the Automatically Generated (AG) and 2T Datasets [55].

Figure 5.3a shows the overall results for the AG datasets, which were very positive; scores drop over 2T, where only three systems achieved an F1-score higher than 0.8 (Figure 5.3b). By looking at footprints in Figure 5.4, we observe that systems tend to preserve the precision, at the recall cost. In many cases, we observe that precision and recall tend to the same value; this happens when systems always annotate a cell, disregarding the knowledge gap problem, i. e., without deciding whether to annotate a cell or not. Those systems assume that if a cell is listed in the target file (i. e., the file that contains the list of cells to annotate), then that cell surely has a corresponding entity, while in 2T, a small percentage of cells ($< 1\%$) has to be annotated with the *NIL* annotation.

Within SemTab 2020, we had the chance to evaluate several algorithms also in solving the CTA task over 2T. The results in Figure 5.5 show a trend similar to the one observed in CEA, where the average performance against the AG datasets is excellent (Figure 5.5a), but then is dramatically reduced for all the systems when annotating 2T (see Figure 5.5b). Thus, we can conclude that also the CTA task is challenging to solve over 2T, especially for those algorithms that solve the CTA task based on the results of CEA. We can draw the same conclusions as for CEA if we look at the score footprints in Figure 5.6: precision is overly preferred to recall, and systems tend always to annotate a column; by inspecting the submissions files, we spotlighted that some algorithms annotate column with clearly wrong types, by assuming that an empty annotation would always be a wrong annotation.

5.5 SUMMARY

In this Chapter, we presented a novel dataset for benchmarking table annotation approaches on the CEA and CTA tasks. The dataset comes with a mix of real and constructed tables, which resemble many real-world scenarios. We tested our dataset using state-of-the-art ap-

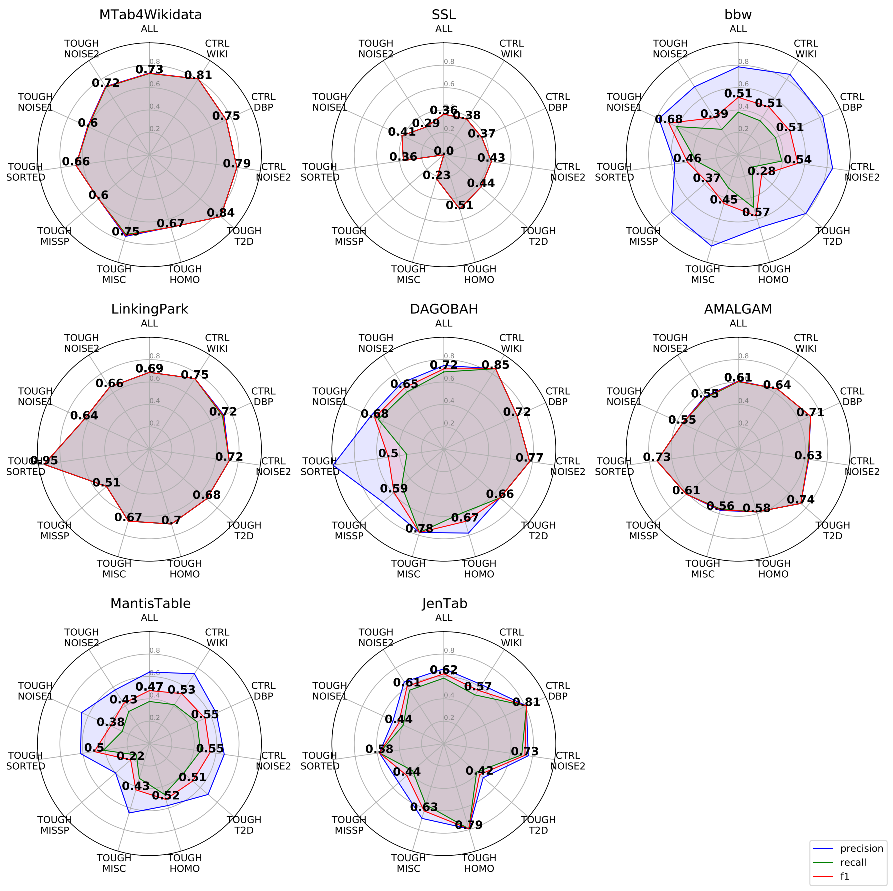


Figure 5.6: CTA performance footprints of the considered algorithms over 2T in SemTab 2020. F1 score in **bold**.

proaches and two baselines. These baselines are represented by online lookup services, which are usually adopted as a building block of many table annotation approaches. We demonstrated that our tables are tough, and solving them requires the algorithms to implement sophisticated mechanisms that consider many semantics aspects. In fact, algorithms focused on precision, like MantisTable, fail to annotate large tables and noisy tables, while baseline algorithms can annotate more cells, but the precision decreases in some cases. Existing scalable solutions like FactBase and EmbeddingOnGraph fail to annotate 2T, achieving worse scores for all the categories. We also used 2T as a benchmark dataset in SemTab 2020, showing that the most recent approaches, when tailored to a specific challenge, an still not cope with the complexity of 2T.

As future work, we plan to extend 2T to cover also the CPA task. We think that the dataset profile makes it suitable to evaluate algorithms in CPA because 2T comes with tables containing many columns with matches (~3 columns per table), making the CPA more challenging: in fact, just moving from 2 to 3 columns to annotate increases the annotation combinations space by 1 degree of freedom.

6 IMPROVING ENTITY LINKING FOR AUTOMATIC AND ASSISTED SEMANTIC TABLE INTERPRETATION

In previous Chapters, we highlighted how many applications nowadays could benefit from matching tables against KGs, including transforming tabular data into KGs, improving the question answering over Web tables, and supporting the KG completion. The community interest in this topic led to a plethora of STI approaches, which automatically support the task at the Web scale. However, we discussed in Chapter 3 that almost all the existing STI algorithms are fully automated, thus are not able to include the information provided by a user in an interactive environment.

In this Chapter, we propose a methodology to improve the entity linking for automatic and assisted STI approaches that rely on the semantic type of entities [31]. The proposed methodology is based on different type enrichment strategies that can be included in existing STI algorithms in a modular fashion. The two approaches we propose are targeted to improve the CEA refinement phase of STI algorithms based on neural type prediction and soft constraints. We will showcase the actual application of such approaches to existing state-of-the-art algorithms, showing promising results on different benchmark datasets. One of the proposed approaches is derived from a particular technique to representing KG types in a continuous vector space, which has been introduced in previous work [7].

A limitation of the proposed approaches is that they are based on a neural model for predicting types, which may be not suitable for predicting types when dealing with some target KGs (we used the DBpedia KG for our study). For example, the Wikidata KG does not have a proper ontology, and the type hierarchy is more fine-grained than the one described in DBpedia, making it potentially more difficult to train a prediction model. In these cases, we propose to use more complex prediction models that should produce better results [74].

6.1 EXPLOITING ENTITY TYPES IN ENTITY LINKING

The literature overview of STI algorithms given in Chapter 3 highlights that the vast majority of proposed algorithms solve the CEA, CTA, and CPA tasks jointly. A widely adopted solution to improve the precision of matching algorithms is to refine the set of candidate entities collected in the candidate retrieval phase by their type in the KG. In this Section, we focus on those algorithms that exploit the evidence collected in CTA (both automatically detected or manually provided by a human) to support the CEA task.

We argue that when an algorithm implements this strategy, detecting the right column type becomes crucial, as well as the type information available in the KG. Column-wise coherence of the entity types is a characterizing feature of tables as opposed to plain texts, but type information explicitly stored in KGs is known to be imperfect and incomplete [74]. For example, in the DBpedia KG,[1] the entity `Arnold Schwarzenegger` is typed as `OfficeHolder`, but not as `Actor`, even if the same person is very famous for his roles in many movies, also available in DBpedia (e. g., `The Terminator` has `Arnold Schwarzenegger` as a starring actor). This aspect is often overlooked by STI algorithms, which assume that KGs are complete [85].

Particular attention has been dedicated to automatic STI, where algorithms must perform the tasks autonomously. Automatic approaches target, in particular, Web tables, which are small and are more likely to describe popular entities. In this context, STI algorithms perform well also when using an incomplete KG, but we reported in Chapter 5 that the performance of STI algorithms drops dramatically as soon as labels are ambiguous and the table size increases [55]. Moreover, with the uprising of interactive tools for transforming tabular data into KGs, like our system Asia illustrated in Chapter 4, and application scenarios where big tables (e. g., tables with thousands of rows) are used to build or enrich KGs, STI algorithms can also work with users-in-the-loop to obtain high-quality annotations and, consequently, high-quality KGs. In these settings, users may define or refine schema-level annotations, while entity linking must be supported by automatic methods due to the different scale of the annotations that must be defined. In such contexts, also assuming a complete KG, the usage of the type information as a hard filter may be inadequate in assisted environments, where non-expert users are taken within the annotation loop, and when there is not a conceptual alignment between the implicit classification in tables and the explicit classification in the KG.

For example, consider Table 1.1 we defined in our motivating example, and DBpedia as our target KG: the implicit classification in the table may hint to annotate the column *City* using the `City` type from the DBpedia ontology. This choice will make it impossible to annotate the cells *Altenburg*, *Ingolstadt*, and *Ulm* correctly if we consider the `City` type as a hard constraint because the corresponding `Altenburg`, `Ingolstadt`, and `Ulm` entities in DBpedia are assigned with the `Town` type. We have a similar situation if we consider the example table given in Figure 2.2; the table is about philosophers of science, thus a non-expert user will annotate the *Name* column with the `Philosopher` type. However, Einstein is represented as a `Scientist` in DBpedia, thus the STI algorithm will discard the right entity.

1 We refer to the 2016-10 version.

On the other hand, expert users may annotate the column with the list of all the relevant types (e. g., `Town` and `City` in the first example), or resort to using a more generic type, which is a superclass of the types associated with all or some entities (e. g., `Person` in the second example). Whatever the applied solution, the underlying STI algorithm will have to deal with a much larger set of candidates; for example, if we set `Person` in the second example and query DBpedia with the term "Einstein", we will end up with 34 entities of type `Person`, 2 of which are of type `Scientist`.

We found two patterns used by STI algorithms available in the literature that can be improved by better handling entity types:

- *Filtering by type*, where types associated with a column are used as hard constraints to filter out candidate entities having different types;
- *Ranking by distributed entity representations similarity*, i. e., entity embeddings, where embeddings are used to compute the similarity between candidates for different labels in order to support the disambiguation.

We can see these patterns as core mechanisms in some state-of-the-art algorithms, such as FactBase, EmbeddingsOnGraph, and their hybrid combinations HybridI and HybridII, which provide a comprehensive solution to address both the disambiguation of ambiguous labels and scalability [38], but also in other approaches [102, 127], also tailored on the SemTab challenge [17, 22, 85].

In this Chapter, we propose to use neural models for type prediction and type representation to improve the mechanisms mentioned above and discuss type enrichment strategies that can be used in existing algorithms in a modular fashion:

- *Type enrichment for filtering by type*, which uses neural type prediction algorithms to enrich the types of candidate entities with types predicted by a neural network;
- *Type enrichment for ranking by distributed entity representations similarity*, which instead uses distributed type representations to enrich entity embeddings, making their similarity more aware of their types.

The two approaches can also be combined, and we propose to enrich entity embeddings with the type predicted by a neural model. In fact, we argue that both approaches capture a similar principle in orthogonal ways, i. e., to implement soft type-based constraints to improve entity disambiguation.

We tested our approaches by incorporating them into state-of-the-art algorithms proposed in [38], thus benchmarking the improved algorithms over datasets used in previous work, or which have been published to make disambiguation more challenging and test algorithms at a larger scale. While we run experiments on a selected pool of state-of-the-art algorithms, the novel methodology we propose in the following applies to almost every algorithm that uses a filtering or ranking strategy based on the entity typing. Among the available algorithms, we selected the ones proposed in [38] because of their performance and because they are more prone to handle large tables, general enough to be applied to different settings and not based on specific assumptions tailored to a specific challenge.

6.1.1 *Matching Pipelines in FactBase and EmbeddingsOnGraph*

Before introducing our methodology, we introduce the reader to the complex matching pipelines proposed in [38], providing the sufficient amount of details needed to understand how we improve the filtering by type and the ranking by distributed entity representations similarity strategies adopted in such algorithms.[2]

FACTBASE The algorithm examines all the cells of an entity column at once, i. e., it works column-wise. Two filters are applied to the set of candidates entities: (i) filter by type, where the entities are filtered by considering their *direct* types, i. e., the most specific entity types; (ii) filter by token, where the entities are filtered by representative language tokens. Values in context columns C_j are used to match facts in the KG and expand the set of candidates. FactBase implements a pipeline that consists of a preliminary step and three entity annotation steps:

- *Candidates retrieval and schema-level annotation*. All the labels in the entity column are looked up in an index, which stores the reference KG. For each label, a list of candidates is returned, ordered by a specific lookup score. Then, for each label, the algorithm looks at its top-ranked candidate to extract (from the KG) its direct types and its textual description, which is processed (e. g., stop words are removed) to return a set of tokens. Given all the entity types and description tokens extracted for all the labels in the column, the k-most frequent types (in the original work, $k = 5$) and the most frequent token in the descriptions extracted for all the labels are associated with the column, thus returning a set of k *column types* and one *column token*. The algorithm then uses unambiguous labels, i. e., labels for which one unique candidate has been found, to understand which columns in the table describe facts that are also present in the KG. Given the entity column C_j under evaluation, the algorithm annotates

2 We prefer to describe the details of the algorithms to make this work self-contained and ease the understanding of our methodology and its evaluation. In this way, we also favor the replicability of our work.

a context column C_k with a KG property P that describes the relation between the labels in the entity column and the values in the context columns. When a context column C_k is annotated with a property P also used in the KG, a fact $\mathtt{label}_{i,j}$ P $\mathtt{value}_{i,k}$ can be extracted from the i-th row of the table and used to lookup more candidates, and, in particular, all those entities x for which the fact x P $\mathtt{value}_{i,k}$ is part of the KG, thus expanding the set of candidates for a given label. In order to select the properties to use for annotating some of the context columns, the algorithm picks each unambiguous label in the C_j column and, for each context column C_k, matches the pair $(\mathtt{label}_{i,j}, \mathtt{value}_{i,k})$ against all the indexed facts. A property P is chosen to annotate a context column if it matches facts extracted from at least n different rows with unambiguous labels (the original work heuristically set $n = 5$). If more properties satisfy this constraint, the most frequent property is selected. We refer to this property as to the *context column property*. As a consequence of this step, the entity column under attention is annotated with a set of its k column types and its most frequent token, while some context columns are annotated with a property.

- *Annotation by lookup - for unambiguous labels*. Unambiguous labels found in the previous step are annotated with their unique candidate entity.
- *Annotation by strict lookup for ambiguous labels*: this step refines the set of candidates of the ambiguous labels, i. e., labels for which more than one candidate was found, by filtering out candidates with types that differ from the entity column types *and* a description that does not contain the column token. Then, the label is annotated with the candidate with the highest score.
- *Loose lookup for labels without candidates*: this step looks for new candidates for the labels that are still not annotated; given the context columns annotated in the first step with a property, this step retrieves as candidates all the entities that match some of the indexed facts of the KG, as explained in the preliminary step. The new set of candidates is then ranked based on the edit distance between the entity label in the KG, and the label in the entity column. The first candidate is used to annotate the label in the current cell. If also this lookup fails, the cell is left without any annotation.

EMBEDDINGS ON GRAPH The algorithm can work column-wise, row-wise or table-wise. It does not apply any filter by type mechanism and is based on constructing a disambiguation graph like several approaches also applied to named entity linking [77]. We here describe the column-wise approach tested in the original work,[3] and used in our experiments. A set of candidates is retrieved for each label in the entity column by selecting the best m matches using a char-level trigram similarity with a threshold σ (where $m = 8$ and $\sigma = 0.82$ in the original paper). All the candidate entities for each label represent the nodes of the disambiguation graph; each node has (i) a *prior* probability, which is based on the degree (in-links + out-links) of the corresponding DBpedia page (we refer to the original paper for details), and (ii) an embedding that represent the entity. Each pair of candidates from different sets is connected by an edge weighted by the cosine similarity of their embeddings. Finally, the priors are updated by executing PageRank on the constructed graph; the candidate with the highest score in each set is chosen for the annotation.

HYBRID I/HYBRID II The same work also proposed hybrid models that apply the FactBase and EmbeddingsOnGraph algorithms sequentially; HybridI executes FactBase first, and then applies EmbeddingsOnGraph to annotate cells without an annotation, while HybridII works in the other way around, applying EmbeddingsOnGraph first and FactBase subsequently.

6.2 NEST: FILTERING AND RANKING USING SOFT TYPE CONSTRAINTS

Filtering and ranking candidate entities by their type is a fundamental step to preserve the type coherence of entities in a column, with the filtering step also reducing the search space. However, using the column type as a hard constraint might introduce some errors, mainly due to missing type assertion axioms in the target KG, as well as the inaccurate type information that non-expert users may enter in an assisted annotation environment.

We introduce NEST (NEural Soft Type Constraints), a methodology to replace hard constraints based on entity types adopted in entity linking pipelines. The methodology relies on distributed representations of entities, i. e., entity embeddings, which can be computed with different approaches (we point the reader to a recent survey [51] for a complete overview). We consider two strategies to include soft type constraints, addressing two patterns used in previous work and,

3 The approach used in the original work by the authors was not explicitly stated. The table-wise approach can combine richer information but at the price of scalability for large tables with thousands of rows; the row-wise approach demands embedding that maximizes the relatedness between entities, like the ones used in the original work, while the column-wise approach should exploit embeddings which maximize the type coherence between similar entities. These aspects are not discussed in [38]. Since EmbeddingsOnGraph has been tested only on datasets featuring tables with at most one entity column in the original work, we suppose the algorithm works column-wise.

especially, in the unsupervised state-of-the-art algorithms described in Section 6.1.1:

- The first strategy, *type enrichment for filtering by type*, combines direct types of the candidate entities with types predicted by a neural model to refine the type-based filtering. The neural model relies on distributed representations of entities for type prediction. Given an input vector representing an entity, a neural network returns a probability distribution over the possible entity types. The model is trained only with the more specific types. The distribution can be used to select a list of the most probable types according to the network, which can enrich (or refine) the set of most specific types explicitly specified in the KG, e. g., by predicting the type `Philosopher` for `Albert Einstein`. This strategy will be demonstrated by applying it to the FactBase algorithm.

- The second strategy, *type enrichment for ranking by distributed entity representations similarity*, starts from the consideration that entity embeddings are particularly useful to evaluate entity similarity, and evaluating the similarity between candidate entities in the same column helps entity disambiguation (as described in Section 6.1.1). However, the type-level characterization is not explicitly featured in popular entity embeddings, e. g., RDF2Vec [101]. To feature type-level information more explicitly in the embeddings, we rely on type embeddings [7], i. e., distributed representations of entity types: given the vector of an entity $\vec{e}$ and the vector of its type $\vec{t}$, the two vectors are concatenated, generating a *typed entity embedding* $\vec{e}\frown\vec{t}$, i. e., a representation in a vector space where entities that share the same types are closer [6]. In this way, the vector components representing the entity type in the concatenated vectors induce a *soft* (type) constraint over the selected annotations, increasing the similarity of entities of the same type. For example, if the vector of `Philosopher` is concatenated to the vector of `Albert Einstein`, this candidate entity will be more similar to other entities of type `Philosopher`; otherwise, since `Philosopher` and `Scientist` are similar in the type space, even if we concatenate the vector `Scientist`, the similarity between `Albert Einstein` and other philosophers will not be seriously affected. We will demonstrate this strategy by applying it to the EmbeddingsOnGraph algorithm.

The clear advantage of this methodology stands in its modularity. The two approaches can be used jointly and introduced in different matching pipelines with near zero engineering disruption. Moreover, the strategies can be applied with different entity and type embeddings, and also with different neural type prediction models. The simplicity of the methodology, as well as the reuse of existing re-

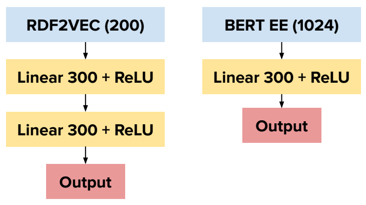


Figure 6.1: The architectures of the DNNs for type prediction. Numbers describe layers size.

sources (e. g., pre-trained embeddings) represent a great added value and make the methodology very modular and easily applicable to different algorithms and KGs.

For type embeddings, we used Type2Vec [7], a model inspired by distributional semantics that does not require expressive and rich ontologies. Type2Vec embeddings are obtained by annotating a text with entities, replacing the entities with their minimal type, and then applying Word2Vec [76] to the text to generate the embeddings.

In the next Section, we will describe two neural prediction models working with different sources of information. In Section 6.2.2, we will provide the reader with details about how we featured these strategies into state-of-the-art algorithms [38] to demonstrate their effectiveness.

6.2.1 *Neural Models for Entity Type Prediction*

We designed two Deep Neural Networks (DNNs) for type prediction that take as input embeddings, possibly generated with different models. The architectures of such DNNs are depicted in Figure 6.1. One network (left-hand side network in Figure 6.1) uses embeddings generated with RDF2Vec [101], which creates a virtual document containing random walks over a KG and then applies Word2Vec on this document. A second network (right-hand side network in Figure 6.1) uses entity embeddings generated from textual descriptions of entities using the Bidirectional Encoder Representations from Transformers (BERT) [35], a powerful language model that has shown strong performance over several downstream tasks in different languages [89]. We generate entity embeddings, which we named BERT Entity Embeddings (BERT EE), as follows: we consider the DBpedia abstracts of entities as their descriptions, thus we use BERT to extract the token embedding of each token in the description,[4] and finally, we average the token embeddings, obtaining an average vector that encodes the entity abstract.

The architectures have been selected with respect to the performance obtained at validation time, and we use early stopping to prevent overfitting (more details about the models training are given

4 BERT accepts a limited amount of input tokens; tokens that exceed the limit are just ignored.

in Section 6.2.2). Both DNNs are trained to reduce the categorical cross-entropy loss, thus solving a multi-class classification task.

To predict the type of an entity, NEST implements a classifier modeled as a straightforward DNN, which learns to map entities with similar embeddings to the same type. As an example, we obtain that the entity `Albert Einstein`, whose entity embedding is similar to the embeddings of other scientists and philosophers, has a high probability of being of types `Scientist` and `Philosopher`. While architecturally simple, the DNNs we designed within NEST can be trained quickly, using already available data, and no sophisticated hardware is required to complete the training. These aspects increase the applicability of the methods to different KG.

6.2.2 NEST-*enriched Algorithms*

In this Section, we illustrate how we modularly applied NEST to the selected state-of-the-art algorithms.

FACTBASE ST The candidate retrieval phase in FactBase mainly filters the candidate entities by a set of acceptable column types and a single column token. The column types are collected based on the direct types of the candidate entities available in the target KG. We propose the enhanced version of this algorithm, namely FactBaseST, which exploits the neural type prediction models, which we trained and executed over DBpedia, to enrich the type information used for filtering. The usage of predicted types in FactBaseST aims to capture two intertwined intuitions: (i) predicted types for all entities in a column can provide additional evidence to determine the correct column types, thus reducing column types used as filters subsequently; (ii) by enriching the set of types associated with a candidate entity (e. g., by adding the type `Philosopher` to `Albert Einstein`), the filters applied to individual candidates are softened and become less sensitive to missing type information or mismatches between the intended conceptualization in the table column and the classification in the KG.

More precisely, we captured the intuitions above as follows:

- The set of column types extracted from the KG is refined by intersecting it with the set of types predicted by the prediction model. The set of predicted types used in this refinement consists of the h most frequent types among all the most confident types predicted for each candidate (we set $h = 5$ as in the original work). If the intersection is empty (e. g., because the model predicts generic types that are not in the current column types set), the previous column types are preserved, discarding the predicted types. In this way, predicted types improve the performance of the algorithm only when they are specific enough to match the column types.

- The types associated with each candidate are extended by also considering the two most likely types predicted for the candidate by the network. We consider only the most two confident predicted types because we found that this reflects the average number of different direct types in DBpedia for the entities with more than one direct type.

By considering the two neural type prediction models described in Section 6.2, we implemented two versions of FactBaseST:

- FactBaseST-R2V, the algorithm that uses the RDF2Vec-based DNN for type prediction. The DNN has been trained using ~200k 200-dimensional vectors of DBpedia entities as input [101], considering only their most specific types.[5] We balanced the training set by down-sampling (we sample 1000 entities per type), and we removed those types with less than 1000 instances to reduce variability (e. g., `BowlingLeague` and `MouseGene`), resulting in 236 different predictable types. In this case, the network does not exploit the textual description of entities, making it applicable to almost every KGs. The trained DNN has shown remarkable performance, surpassing 0.90 in accuracy score both on the training and on the validation set (20% of the data) on the type prediction task. For the training phase, we set an early stopping on the validation loss (patience equal to 4).

- FactBaseST-A2V, which instead exploits the DNN based on BERT EE to predict types. Since training BERT EE requires textual descriptions of entities, this algorithm is suitable only for KGs that include a textual description of their entities (e. g., DBpedia abstracts or Wikidata descriptions). We trained the DNN in FactBaseST-A2V with embeddings generated by feeding a pre-trained large BERT model (1024-dimensional vectors) with abstracts of DBpedia entities. As well as we did in FactBaseST-R2V, we sampled 1000 entities per type, removing those types with less than 1000 instances; however, in this case, we had also to remove entities without an abstract from the training set, resulting in 228 different predictable types. The trained DNN scores slightly above 0.90 in accuracy score on the type prediction task (same configuration as above).

EMBEDDING ON GRAPH ST The EmbeddingsOnGraph algorithm does not exploit the type information to either filter the retrieved candidates or rank them. In fact, the algorithm mainly relies on priors probabilities of entities, and their entity embeddings, which are used for building a weighted disambiguation graph. As a consequence, the algorithm does not preserve the column-wise coherence of entity types, assuming that

5 For DBpedia entity types, see `http://downloads.dbpedia.org/2016-10/core-i18n/en/instance_types_en.ttl.bz2`.

entities with similar vectors are of the same type. However, this is not true in general; for example, in RDF2Vec the entity `Paris` (type: `Settlement`) is very similar to the entities `Architecture of Paris` (type: `Thing`, cosine similarity: 0.84) and `Anne Hidalgo` (type: `Person`, cosine similarity: 0.76).

We thus provide an extension of EmbeddingsOnGraph, namely EmbeddingsOnGraphST, where we introduce the type information in the form of *typed entity embeddings*. The conceptually relevant implication in using typed entity embeddings instead of plain entity embeddings is that entities with similar predicted types have a higher cosine similarity. As a consequence, when PageRank is computed over the disambiguation graph, the weight of edges between entities of the same type increases. At the same time, the weight of edges between entities of different types decreases further, thus implementing the soft constraint over the similarity that we aimed to capture. While the column-wise coherence of entity types is still not guaranteed, this approach tends to favor the selection of candidates of the same type.

HYBRID I ST/HYBRID II ST The algorithms resemble their original implementations by jointly applying FactBaseST-R2V or FactBaseST-A2V, and EmbeddingsOnGraphST, generating the variants HybridIST-R2V, HybridIST-A2V, HybridIIST-R2V, and HybridIIST-A2V.

6.3 EXPERIMENTAL EVALUATION

To study the actual performance gain in integrating NEST into state-of-the-art algorithms, we conducted an experimental campaign by recreating the SemTab 2019 environment for the CEA task [54]. Hence, we used DBpedia 2016-10 as the target KG, and we scored the algorithms using the Precision (P), Recall (R), and F1-score (F1) metrics (all in their *macro* version) defined as follows:[6]

$$P = \frac{|CorrectAnnotations|}{|SystemAnnotations|} \quad R = \frac{|CorrectAnnotations|}{|TargetAnnotations|}$$
$$F1 = \frac{2 \times P \times R}{P + R}$$

In the following, we illustrate the datasets and the actual algorithms we used in our experiments.

DATASETS In our experiments, we considered three datasets (Table 6.1):[7]

- T2D [102], which is the dataset also used in the original work [38]; T2D is a relatively small dataset with only limited contents, but

6 $SystemAnnotations$ = cells annotated by the algorithm; $TargetAnnotations$ = ground truth cells.

7 An extended version of this table is available in Chapter 5; we report this shortened version for the reader's convenience.

Table 6.1: Profiles of the benchmark datasets considered within the evaluation.

Dataset	Cols (avg)	Rows (avg)	Matches	Entities	Cols with Matches (avg)	Tables
T2D	1,153 (4.95)	28,333 (121.60)	26,124	13,785	233 (1.00)	233
ST19-R4	3,564 (4.36)	51,249 (62.73)	107,352	53,007	1,732 (2.12)	817
2T	802 (4.46)	194,438 (1,080.21)	663,656	15,997	540 (3.00)	180

it still represents a reference dataset in the literature. We also exploit it to observe better the impact of the adaptation we made while re-implementing the original algorithms. We did not use the other datasets used in the original work (i. e., Limaye and W2D) because Limaye and T2D have similar profiles, with Limaye featuring smaller tables and fewer columns to annotate, while W2D contains only very small tables (23 cells in average to annotate for each column), and it has been partially included in the SemTab dataset. The challenges in dealing with small tables have been extensively discussed in the original work [38].

- ST19-R4, a novel dataset provided in SemTab 2019 [54]. This dataset is the only one containing only non-trivial cases among SemTab datasets.[8] More importantly, ST19-R4 has been built using a generator, which constructs tables by querying DBpedia. Each table has one class as the main topic, and the other columns are filled with values of a predefined pool of properties of each instance. We highlight that the generator ensures that the type of the entities linked by an object property matches the expected property range in the ontology [54]. Thus, the problem we are addressing in this Chapter, i. e., dealing with incomplete type information, has been artificially removed from ST19-R4. For example, suppose the generator creates a table about `Film`, using the property `starring` to populate a column. In that case, we obtain that the film `The Terminator`, which has `Arnold Schwarzenegger` as a starring actor, is removed from the table because the range of the property `starring` is `Actor`, but `Arnold Schwarzenegger` is typed as `OfficeHolder`. For the above reason, experiments on this dataset are reported for fairness, but also for evaluating if Nest, which uses predicted types and soft filtering, may lead to degraded performance in a setting where these mechanisms are not useful (in this case, by construction). Indeed, we expect Nest not to improve the results of the original

8 The overall performance observed in SemTab 2019 for this dataset was high, also thanks to hard-coded workarounds adopted by the participants [115], which we did not implement in our algorithms.

algorithms on ST19-R4, but we want to measure its negative impact.

- 2T [28], the dataset we presented in Chapter 5, which features ambiguous and noisy tables that resemble real-world cases. In our experimental setting, 2T represents the general scenario where (i) columns with entities "of the wrong type" have not been fixed (as it happened for ST19-R4), and (ii) cells are not obviously linkable (as in T2D).

ALGORITHMS We applied NEST to the algorithms in [38], as described in Section 6.2. Since the original algorithms have not been open-sourced, we had to re-implement all the algorithms from scratch. We tested our re-implemented versions on T2D for fairness because T2D is the dataset used in the original work. However, we observed that our versions performed worse if compared with the original results.[9] We managed to replicate the disambiguation mechanisms employed in the original algorithms, but then we did not focus on optimizing the lookup search. Indeed, in our opinion the performance decrease is due to the following factors:

- The private FactBase index was used in the original work, which includes a 2016 dump of Wikidata, with mappings to DBpedia entities. A dump of this index has not been provided, as well as details on how to properly replicate the indexing phase (e. g., index scoring function and field analyzers). Thus, we generated a new ElasticSearch index containing DBpedia entities, their labels,[10] and their anchor texts from Wikipedia. We preferred not to include Wikidata in our index to reduce the amount of total memory needed to store it.

- Within FactBase, the `schema:description` property of Wikidata entities is crucial for the candidate disambiguation phase since it is used for computing the column token. The same property is missing for DBpedia entities, thus we resembled it by analyzing the DBpedia short abstracts of entities, obtaining different descriptions.[11] Also, the list of stopwords we considered[12] may differ from the one used in the original work.

- The queries to the *FactBase* index have not been published, thus it is not possible to either reuse the same strategies (e. g., fuzzy

9 Results are not directly comparable because we computed the macro precision, recall, and F1-measure, as in SemTab, but their respective micro versions have been computed in [38]. We report here the scores achieved by the original algorithms to help the reader in understanding the gap: FactBase (P: 0.88, R: 0.78, F1: 0.83); EmbeddingsOnGraph (P: 0.86, R: 0.77, F1: 0.81).

10 We also include the labels from the DBpedia Lexicalization datasets.

11 A possible corresponding property in DBpedia would be the `dct:description` one, but it is missing for many entities (e. g., `Milan`).

12 https://www.nltk.org/

match) or apply the same parameters (e. g., max edit distance in fuzzy search, fields boost). As a consequence, the candidate retrieval phase potentially returns different candidate entities.

- For the EmbeddingsOnGraph algorithm, we used the RDF2Vec vectors (uniform model from [101]), which differ from the embeddings used in the original work.

The original algorithms assume the one-entity-per-cell, thus annotating rows with entities; however, we need to annotate cells in order to use the benchmark datasets built for the CEA task. We provided a generalization of the algorithms by exploiting their column-wise nature: we can annotate tables with multiple entity columns by considering one entity column at a time, and setting the other as context columns, thus annotating individual cells instead of entire rows.

Finally, we modified the original EmbeddingsOnGraph algorithm to avoid scalability issues in our experiments; in fact, running EmbeddingsOnGraph on a table with 5000 rows leads to the creation of a disambiguation graph with 40k nodes in the worst case (if all the top-8 candidates for each label are distinct) and ~800M edges. The disambiguation graph is k-partite (only candidates from different sets are linked), thus the maximum number of edges is $\frac{n^2(k-1)}{2k}$ in the worst case. In our implementation, EmbeddingsOnGraph splits big tables into partitions of 500 rows each, which are iteratively annotated.

We remark here that a proper comparison between the original algorithms and their respective re-implemented version is unfair because we were prevented from fully-reproduce the indexing and searching steps. Also, different metrics have been adopted for evaluating the algorithms (micro vs. macro versions). However, we believe that our comparison is fair over the different discussed models because we consider the results obtained with the re-implemented algorithms as baselines.

We also report that we would have preferred to test Nest with several algorithms to demonstrate its generality. However, just a few STI algorithms have been open-sourced, with some limitations: e. g., MantisTable [22] does not scale to the dimension of the considered datasets (we reported in Chapter 5 that it took more than 24 hours to annotate 2T), while CSV2KG [115] has a still incomplete repository.[13] We also contacted the authors of MTab [85], Tabularisi [112], and CSV2KG [115] (the top-3 SemTab systems) after the challenge, but they reported that their systems were not ready to be released. Thus, for our experiments, we chose two algorithms from the literature that (i) were at least partially reproducible (i. e., they have been explained in detail), (ii) employ the type-based filtering and ranking strategies

13 `https://github.com/IBCNServices/CSV2KG` - the file csv2kg/annotate.py implements an incomplete matching pipeline.

Table 6.2: Results for different benchmark datasets. Algorithms improved with NEST are identified by the ♣ symbol. Best results in **bold** (for each dataset) and underlined (for each algorithm).

Method	**T2D**			**ST19-R4**			**2T**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
EmbeddingsOnGraph	0.782	0.723	0.751	0.483	0.470	0.477	0.293	0.245	0.267
EmbeddingsOnGraphST ♣	**0.811**	0.751	**0.780**	0.540	0.526	0.533	0.378	0.316	0.344
FactBase	0.791	0.635	0.704	**0.745**	0.465	**0.573**	0.365	0.185	0.246
FactBaseST-R2V ♣	0.789	0.638	0.706	0.731	0.454	0.560	**0.434**	0.241	0.309
FactBaseST-A2V ♣	0.783	0.638	0.703	0.735	0.458	0.565	0.374	0.216	0.274
HybridI	0.756	0.740	0.748	0.530	0.526	0.528	0.275	0.231	0.251
HybridIST-R2V ♣	0.766	0.751	0.759	0.549	0.544	0.546	0.355	0.299	0.324
HybridIST-A2V ♣	0.762	0.746	0.754	0.551	**0.547**	0.549	0.317	0.266	0.289
HybridII	0.758	0.742	0.750	0.488	0.484	0.486	0.295	0.248	0.270
HybridIIST-R2V ♣	0.784	**0.768**	0.776	0.544	0.540	0.542	0.380	**0.319**	**0.347**
HybridIIST-A2V ♣	0.784	**0.768**	0.776	0.544	0.540	0.542	0.380	**0.319**	**0.347**

we need to test with NEST, and (iii) can scale reasonably (i. e., they annotate our datasets in a few hours).

6.3.1 *Results*

To measure the performance gain in introducing NEST in different matching pipelines, we tested the selected state-of-the-art algorithms and our NEST-improved versions on different benchmark datasets. The results we obtained are available in Table 6.2, and they confirm that the use of NEST can improve existing matching pipelines.

As expected, applying NEST to EmbeddingsOnGraph increases the performance in all our tests because using typed entity embedding strengthens the similarity between entities of the same type. As a result, since EmbeddingsOnGraph is a column-wise approach, the typed entity embeddings can guarantee a higher column-wise coherence of entity types, an aspect that was completely overlooked in the original work. The results for 2T are very low, and in our opinion, this is due to different reasons: (i) the candidate retrieval phase is based only on the trigram similarity and uses a high threshold, returning a small set of candidates; (ii) the prior probabilities used to initialize the PageRank algorithm are based on the entity popularity, which rewards popular entities in almost the cases; this choice penalizes the algorithms when annotating 2T, which contains tables with many homonyms, which often do not link to the most popular mentioned entity.

NEST does not improve FactBase when annotating the more straightforward datasets T2D and ST19-R4, while its contribution is more valuable on 2T, which is more challenging to annotate. The scores obtained by FactBaseST-R2V and FactBaseST-A2V on T2D are comparable with the ones obtained by FactBase: the recall slightly increases, while the precision drops a little; this is the expected behaviour since types in T2D are homogeneous in each column; furthermore, tables in T2D are mainly collected from Wikipedia, the same source used to create DBpedia, so there is an overlap between the conceptualization in Wikipedia tables and the entity representations in the KG. If there is a mismatch, as the example of `Albert Einstein` in Section 6.1, NEST increases the performance, but in all the other cases, considering the secondary type of an entity may add noise to the matching pipeline.

This effect is amplified in ST19-R4, where the problem we address with NEST has been artificially removed by construction and we always have a perfect conceptual alignment between the implicit classification in tables and the explicit classification in DBpedia. Results on this dataset show the possible performance degrades due to the application of NEST when it is not useful (e. g., because the extracted types are complete by construction). In all of the cases except for FactBase on ST19-R4, we observe that the performance is not affected. In the case of ST19-R4, the performance drop with respect to FactBase is very limited (-1% in P and R). We believe that the loss on this specific dataset is balanced by a consistent gain in all the other settings where the artificial removal of this problem does not occur. Furthermore, the results highlight that the standard FactBase algorithm underperforms on 2T, mainly because the candidate retrieval step is not able to deal with the values in 2T tables, which are ambiguous and perturbed with typos. However, using NEST to relax the type-based filtering leads to a valuable performance increase, helping the algorithm disambiguate the higher number of candidates.

Similar results for FactBaseST-R2V and FactBaseST-A2V show that for KGs that have, on average, both textual and factual descriptions, type information can be predicted from both these sources with small differences. However, we did not have enough evidence to prefer one source over the other for similar KGs.

As also observed in the original work [38], hybrid methods improve the recall of their primary method but often have a negative impact on the precision. HybridIIST-R2V and HybridIIST-A2V perform better[14] than their respective HybridIST-R2V and HybridIST-A2V methods, thanks to the higher performance of EmbeddingsOnGraphST.

We finally discuss the worse performance observed for the algorithms FactBase and EmbeddingsOnGraph (as well as their NEST-improved versions) compared to SemTab 2019 systems.[15] We believe

14 Results in Table 6.2 show equal performance, but it is an effect due to the rounding.
15 https://www.cs.ox.ac.uk/isg/challenges/sem-tab/2019/results.html

that the results are due to the underlying algorithms, which we avoided on purpose to overfit on the selected datasets and KG. In fact, we advocate that STI algorithms should target the general problem of matching tables to a KG, and not overspecialize over two KGs like Wikidata and DBpedia (although we use them for evaluation). Moreover, the comparison between the selected algorithms and SemTab systems is not completely fair because SemTab systems (i) have been fine-tuned on the SemTab datasets, and (ii) may implement specific workarounds (e. g., to handle exceptional cases of people names as did by CSV2KG[16] and MantisTable[17] in Round 4).

6.4 SUMMARY

In this Chapter, we presented Nest, a methodology to include soft type-based constraints into matching pipelines of STI algorithms. The methodology helps to better deal with inaccurate entity type information, either when it comes from an incomplete KG or when it is entered by a non-expert user within the annotation pipeline. The contribution is useful for different STI approaches because it is modular (it is built on existing entity/language embedding models that can cover most of the entities in a KG) and allows for its integration with nearly zero effort, thus applying at a large scale, with arbitrary KGs and different algorithms.

Our experimental campaign demonstrated how different state-of-the-art algorithms could benefit from Nest usage, testing Nest-improved algorithms on benchmark datasets with different features. We also tested the negative impact of Nest when annotating a dataset where the problem addressed in our work has been artificially removed by construction.

As future work, we plan to explore different ways of improvement: (i) training the type prediction models would be more problematic when the target KG does not have a proper ontology, or its types hierarchy is more fine-grained than the DBpedia one (e. g., as in Wikidata); thus, it would be interesting to investigate which model is better to use for individual entities as a function of the information available for that entity (e. g., a limited number of facts); (ii) in our work, we focused on the indirect evaluation of the type prediction on the CEA downstream task, but we plan to compare the proposed models with other entity types prediction methods [74]; (iii) evaluating if the type prediction models defined in Nest can support the CTA task, e. g., by smoothing the standard aggregation policies (e. g., majority voting) using the confidence scores provided by the DNNs as weights for the aggregation.

16 https://github.com/IBCNServices/CSV2KG/blob/master/csv2kg/util.py - line 50.

17 https://bitbucket.org/disco_unimib/mantistable-tool.py/src/master/mantistable/process/utils/nlp/utils.py - line 56.

7 CONCLUSIONS

Data enrichment is a fundamental step in many data science projects, where the extension of a source dataset with additional features enables different downstream analytics. However, a general methodology to assist data workers in enriching a source dataset with information from external data sources is still missing. As a result, data enrichment represents a cumbersome task to solve, which usually takes most of the time dedicated to a project (up to 80%). Indeed, data workers have to face different challenges, including where to find the right data to use and how to push them into the source dataset.

The Semantic Web principles and technologies simplify the access and reusability of datasets: indeed, URIs support the identification of resources in different data sources, while SPARQL and shared vocabularies ease the access to these resources. Nowadays, the KG represents one of the preferred paradigms to store and organize information, and the availability of several KGs is a valuable factor to support data enrichment. The data enrichment task can benefit from the KGs because their semantics provides a powerful tool to help the data workers find and merge the needed sources of information. We believe data enrichment represents an important application for semantics technologies, which simplify the access to different resources, but this aspect has been underexplored in the literature, with many ad-hoc solutions tailored to specific projects. Moreover, the focus has been on KG like DBpedia and Wikidata, with several STI approaches tailored to these KGs, but we advocate that STI algorithms should target the general problem of matching tables to a KG, thus being applicable in different contexts (e. g., with a custom KG).

In this thesis, we have proposed a general semantic-enhanced methodology to interactively support a data worker in solving the data enrichment task when working with data in tabular format. We built our methodology on top of the pervasive data transformation pipeline paradigm employed by many data integration tools. The paradigm provides the needed scalability to support the task at a large scale, i. e., when dealing with large tables.

The methodology has been implemented within Asia, a system that we designed to effectively support the interactive data enrichment task, also at a large scale; indeed, Asia managed to enrich a 100 GB dataset (~500M rows) with weather features. However, how to better support STI in an interactive environment and at a large scale is still an open problem. The most recent advancements in state of the art [22, 38, 41], as well as the best-performing systems in dedicated challenges [85, 112, 115], are not capable to (i) efficiently annotate a large amount of data, and (ii) integrating the human in the STI loop. ASIA is also useful for users that need an enriched dataset to support downstream applications (e. g., data analytics), without building or enriching a KG.

In order to better evaluate how effective and efficient the STI tools are, we built 2T, a new benchmark dataset that features large tables collected from different sources, enabling a better understanding of the capabilities of STI algorithms. 2T has been built by following the SemTab 2019 specifications, thus it can be used for testing almost all the proposed STI algorithms in the literature.

Finally, we presented NEST, a human-in-the-loop ready methodology to move existing semantic table annotation approaches a step forward. The methodology helps existing approaches in better dealing with the entity typing information used within linking algorithms, which can be missing (i. e., the KGs are incomplete [74]) or wrong (e. g., when a non-expert enters the entity types).

7.1 FUTURE RESEARCH DIRECTIONS

The work presented in this thesis opens up different future research directions, including:

- The ASIA system considers the human in the loop, assisting a data worker with suggestions on how to annotate the source dataset, as well as providing evidence about the table reconciliation process; however, an interesting research topic is how to introduce automatic assistive mechanisms to reuse and propagate the user inputs (e. g., active learning), and how to integrate them within the data transformation process.

- 2T represents the best candidate to test the scalability of existing STI algorithms; the scalability is an aspect that has been overlooked by recent systems (e. g., systems participating in SemTab), but that is crucial to support the semantic data enrichment for large tables, and to enable novel scenarios like the semantic enrichment of *streams* (which can be interpreted as infinite tables).

- 2T represents a great starting point to generate even more challenging benchmark datasets, such as by adding new kinds of noise or increasing the knowledge gap. These aspects are significant for pushing STI towards more realistic scenarios (e. g., business and industry).

- The NEST methodology can be extended to support the type completion task in existing KGs, an issue that we indirectly mitigate for STI algorithms; it is interesting to compare NEST with other type prediction models [74], eventually providing an integrated methodology to address the type completion task for KGs.

BIBLIOGRAPHY

[1] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. "Towards a Hybrid Imputation Approach Using Web Tables." In: *2nd IEEE/ACM International Symposium on Big Data Computing, BDC*. IEEE Computer Society, 2015, pp. 21–30.

[2] Domenico Beneventano and Maurizio Vincini. "Foreword to the Special Issue: "Semantics for Big Data Integration"." In: *Information* 10 (2 Feb. 2019), p. 68.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. "The semantic web." In: *Scientific american* 284.5 (2001), pp. 34–43.

[4] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. "Methods for exploring and mining tables on Wikipedia." In: *Proceedings of the SIGKDD Workshop on Interactive Data Exploration and Analytics, IDEA@KDD*. ACM, 2013, pp. 18–26.

[5] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. "TabEL: Entity Linking in Web Tables." In: *ISWC*. Vol. 9366. Springer, 2015, pp. 425–441.

[6] Federico Bianchi, Matteo Palmonari, and Debora Nozza. "Towards Encoding Time in Text-Based Entity Embeddings." In: *ISWC*. Vol. 11136. Springer, 2018, pp. 56–71.

[7] Federico Bianchi, Mauricio Soto, Matteo Palmonari, and Vincenzo Cutrona. "Type Vector Representations from Text: An Empirical Analysis." In: *Proceedings of the First Workshop on Deep Learning for Knowledge Graphs and Semantic Technologies, DL4KGS@ESWC*. Vol. 2106. CEUR-WS.org, 2018, pp. 72–83.

[8] Christian Bizer. "The Emerging Web of Linked Data." In: *IEEE Intell. Syst.* 24.5 (2009), pp. 87–92.

[9] Bernard Bosanquet. *Logic, Or, the Morphology of Knowledge, Volume 1*. Cambridge University Press, 2012.

[10] Tom B. Brown et al. "Language Models are Few-Shot Learners." In: *NeurIPS*. 2020.

[11] Quyen Bui-Nguyen, Qing Wang, Jingyu Shao, and Dinusha Vatsalan. "Repairing of Record Linkage: Turning Errors into Insight." In: *EDBT*. 2019, pp. 638–641.

[12] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. "Data Integration for the Relational Web." In: *Proc. VLDB Endow.* 2.1 (2009), pp. 1090–1101.

[13] Michael J. Cafarella, Alon Y. Halevy, Hongrae Lee, Jayant Madhavan, Cong Yu, Daisy Zhe Wang, and Eugene Wu. "Ten Years of WebTables." In: *Proc. VLDB Endow.* 11.12 (2018), pp. 2140–2149.

[14] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. "WebTables: exploring the power of tables on the web." In: *Proc. VLDB Endow.* 1.1 (2008), pp. 538–549.

[15] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. "Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks." In: *SIGMOD*. ACM, 2020, pp. 1335–1349.

[16] Dylan Cashman, Shenyu Xu, Subhajit Das, Florian Heimerl, Cong Liu, Shah Rukh Humayoun, Michael Gleicher, Alex Endert, and Remco Chang. "CAVA: A Visual Analytics System for Exploratory Columnar Data Augmentation Using Knowledge Graphs." In: *IEEE Trans. Vis. Comput. Graph.* 27.2 (2021), pp. 1731–1741.

[17] Yoan Chabot, Thomas Labbé, Jixiong Liu, and Raphaël Troncy. "DAGOBAH: An End-to-End Context-Free Tabular Data Semantic Annotation System." In: *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab@ISWC*. Vol. 2553. CEUR-WS.org, 2019, pp. 41–48.

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

[19] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. "Learning Semantic Annotations for Tabular Data." In: *IJCAI*. ijcai.org, 2019, pp. 2088–2094.

[20] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. "An Overview of End-to-End Entity Resolution for Big Data." In: *ACM Comput. Surv.* 53.6 (Dec. 2020).

[21] Michele Ciavotta, Vincenzo Cutrona, Flavio De Paoli, Nikolay Nikolov, Matteo Palmonari, and Dumitru Roman. "Supporting Semantic Data Enrichment at Scale." In: *Technologies and Applications for Big Data Value*. (To appear). 2021.

[22] Marco Cremaschi, Flavio De Paoli, Anisa Rula, and Blerina Spahiu. "A fully automated approach to a complete Semantic Table Interpretation." In: *Future Gener. Comput. Syst.* 112 (2020), pp. 478–500.

[23] Marco Cremaschi, Alessandra Siano, Roberto Avogadro, Ernesto Jiménez-Ruiz, and Andrea Maurino. “STILTool: A Semantic Table Interpretation evaLuation Tool.” In: *ESWC Satellite Events*. Vol. 12124. Springer, 2020, pp. 61–66.

[24] Isabel F. Cruz, Venkat R. Ganesh, Claudio Caletti, and Pavan Reddy. “GIVA: a semantic framework for geospatial and temporal data integration, visualization, and analytics.” In: *SIGSPATIAL*. ACM, 2013, pp. 534–537.

[25] Isabel F. Cruz, Venkat R. Ganesh, and Seyed Iman Mirrezaei. “Semantic extraction of geographic data from web tables for big data integration.” In: *Proceedings of the 7th Workshop on Geographic Information Retrieval, GIR*. ACM, 2013, pp. 19–26.

[26] Isabel F. Cruz, Matteo Palmonari, Francesco Loprete, Cosmin Stroe, and Aynaz Taheri. “Quality-based model for effective and robust multi-user pay-as-you-go ontology matching.” In: *Semantic Web* 7.4 (2016), pp. 463–479.

[27] Vincenzo Cutrona, Federico Bianchi, Michele Ciavotta, and Andrea Maurino. “On the composition and recommendation of multi-feature paths: a comprehensive approach.” In: *GeoInformatica* 23.3 (2019), pp. 353–373.

[28] Vincenzo Cutrona, Federico Bianchi, Ernesto Jiménez-Ruiz, and Matteo Palmonari. “Tough Tables: Carefully Evaluating Entity Linking for Tabular Data.” In: *ISWC*. Vol. 12507. Springer, 2020, pp. 328–343.

[29] Vincenzo Cutrona, Michele Ciavotta, Flavio De Paoli, and Matteo Palmonari. “ASIA: a Tool for Assisted Semantic Interpretation and Annotation of Tabular Data.” In: *ISWC Satellite Tracks*. Vol. 2456. CEUR-WS.org, 2019, pp. 209–212.

[30] Vincenzo Cutrona, Flavio De Paoli, Aljaz Kosmerlj, Nikolay Nikolov, Matteo Palmonari, Fernando Perales, and Dumitru Roman. “Semantically-Enabled Optimization of Digital Marketing Campaigns.” In: *ISWC*. Vol. 11779. Springer, 2019, pp. 345–362.

[31] Vincenzo Cutrona, Gianluca Puleri, Federico Bianchi, and Matteo Palmonari. “NEST: Neural Soft Type Constraints to Improve Entity Linking in Tables.” In: *ESWC*. (Under revision). 2021.

[32] Dong Deng, Yu Jiang, Guoliang Li, Jian Li, and Cong Yu. “Scalable Column Concept Determination for Web Tables Using Large Knowledge Bases.” In: *Proc. VLDB Endow.* 6.13 (2013), pp. 1606–1617.

[33] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. “TURL: Table Understanding through Representation Learning.” In: *arXiv e-prints* (2020). arXiv: 2006.14806 [cs.IR].

[34] Yared Dejene Dessalk, Nikolay Nikolov, Mihhail Matskin, Ahmet Soylu, and Dumitru Roman. "Scalable Execution of Big Data Workflows using Software Containers." In: *MEDES*. ACM, 2020, pp. 76–83.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *NAACL-HLT*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[36] 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: *KDD*. ACM, 2014, pp. 601–610.

[37] Carina F. Dorneles, Rodrigo Gonçalves, and Ronaldo dos Santos Mello. "Approximate data instance matching: a survey." In: *Knowl. Inf. Syst.* 27.1 (2011), pp. 1–21.

[38] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. "Matching Web Tables with Knowledge Base Entities: From Entity Lookups to Entity Embeddings." In: *ISWC*. Vol. 10587. Springer, 2017, pp. 260–277.

[39] Vasilis Efthymiou, George Papadakis, Kostas Stefanidis, and Vassilis Christophides. "MinoanER: Schema-Agnostic, Non-Iterative, Massively Parallel Resolution of Web Entities." In: *EDBT*. OpenProceedings.org, 2019, pp. 373–384.

[40] Ivan Ermilov and Axel-Cyrille Ngonga Ngomo. "TAIPAN: Automatic Property Mapping for Tabular Data." In: *EKAW*. Vol. 10024. 2016, pp. 163–179.

[41] Yasamin Eslahi, Akansha Bhardwaj, Paolo Rosso, Kurt Stockinger, and Philippe Cudré-Mauroux. "Annotating Web Tables through Knowledge Bases: A Context-Based Approach." In: *SDS*. IEEE, 2020, pp. 29–34.

[42] Daniel Faria, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel F. Cruz, and Francisco M. Couto. "The AgreementMakerLight Ontology Matching System." In: *OTM Conferences*. Vol. 8185. Springer, 2013, pp. 527–541.

[43] José Ferreirós. "The road to modern logic—an interpretation." In: *Bulletin of Symbolic Logic* 7.4 (2001), pp. 441–484.

[44] Cheng Fu, Xianpei Han, Jiaming He, and Le Sun. "Hierarchical Matching Network for Heterogeneous Entity Resolution." In: *IJCAI*. ijcai.org, 2020, pp. 3665–3671.

[45] Tim Furche, Georg Gottlob, Leonid Libkin, Giorgio Orsi, and Norman W Paton. "Data Wrangling for Big Data: Challenges and Opportunities." In: *EDBT*. 2016, pp. 473–478.

[46] Shubham Gupta, Pedro A. Szekely, Craig A. Knoblock, Aman Goel, Mohsen Taheriyan, and Maria Muslea. "Karma: A System for Mapping Structured Sources into the Semantic Web." In: *ESWC Satellite Events*. Vol. 7540. Springer, 2012, pp. 430–434.

[47] Aidan Hogan et al. "Knowledge Graphs." In: *arXiv e-prints* (2021). arXiv: 2003.02320 [cs.AI].

[48] Kevin Zeng Hu, Snehalkumar (Neil) S. Gaikwad, Madelon Hulsebos, Michiel A. Bakker, Emanuel Zgraggen, César A. Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çagatay Demiralp. "VizNet: Towards A Large-Scale Visualization Learning and Benchmarking Repository." In: *CHI*. ACM, 2019, p. 662.

[49] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. "Sherlock: A Deep Learning Approach to Semantic Data Type Detection." In: *KDD*. ACM, 2019, pp. 1500–1508.

[50] Faiza Hussain and Usman Qamar. "Identification and Correction of Misspelled Drugs Names in Electronic Medical Records (EMR)." In: *ICEIS*. Vol. 2. 2016, pp. 333–338.

[51] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. "A Survey on Knowledge Graphs: Representation, Acquisition and Applications." In: *arXiv e-prints* (2021). arXiv: 2002.00388 [cs.CL].

[52] Keyuan Jiang, Tingyu Chen, Liyuan Huang, Ricardo A. Calix, and Gordon R. Bernard. "A Data-Driven Method of Discovering Misspellings of Medication Names on Twitter." In: *MIE*. Vol. 247. Studies in Health Technology and Informatics. IOS Press, 2018, pp. 136–140.

[53] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. "LogMap: Logic-Based and Scalable Ontology Matching." In: *ISWC*. Vol. 7031. Springer, 2011, pp. 273–288.

[54] Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, and Kavitha Srinivas. "SemTab 2019: Resources to Benchmark Tabular Data to Knowledge Graph Matching Systems." In: *ESWC*. Vol. 12123. Springer, 2020, pp. 514–530.

[55] Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Kavitha Srinivas, and Vincenzo Cutrona. "Results of SemTab 2020." In: *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab@ISWC*. Vol. 2775. CEUR-WS.org, 2020, pp. 1–8.

[56] Emilia Kacprzak, José M. Giménez-García, Alessandro Piscopo, Laura Koesten, Luis Daniel Ibáñez, Jeni Tennison, and Elena Simperl. "Making Sense of Numerical Data - Semantic Labelling of Web Tables." In: *EKAW*. Vol. 11313. Springer, 2018, pp. 163–178.

[57] Shady Abd El Kader, Nikolay Nikolov, Bjørn Marius von Zernichow, Vincenzo Cutrona, Matteo Palmonari, Brian Elvesæter, Ahmet Soylu, and Dumitru Roman. "Modeling and Publishing French Business Register (Sirene) Data as Linked Data Using the euBusinessGraph Ontology." In: *Joint Proceedings of the International Workshops on Sensors and Actuators on the Web, and Semantic Statistics, SemStats@ISWC*. Vol. 2549. CEUR-WS.org, 2019.

[58] Evgeny Kharlamov et al. "Ontology Based Data Access in Statoil." In: *J. Web Semant.* 44 (2017), pp. 3–36.

[59] Craig A. Knoblock, Pedro A. Szekely, José Luis Ambite, Aman Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, Mohsen Taheriyan, and Parag Mallick. "Semi-automatically Mapping Structured Sources into the Semantic Web." In: *ESWC*. Vol. 7295. Springer, 2012, pp. 375–390.

[60] Dimitrios Koutsomitropoulos, Spiridon Likothanassis, and Panos Kalnis. "Semantics in the Deep: Semantic Analytics for Big Data." In: *Data* 4 (2 May 2019), p. 63.

[61] Benno Kruit, Peter A. Boncz, and Jacopo Urbani. "Extracting Novel Facts from Tables for Knowledge Graph Completion." In: *ISWC*. Vol. 11778. Springer, 2019, pp. 364–381.

[62] Kostis Kyzirakos, Dimitrianos Savva, Ioannis Vlachopoulos, Alexandros Vasileiou, Nikolaos Karalis, Manolis Koubarakis, and Stefan Manegold. "GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings." In: *J. Web Semant.* 52-53 (2018), pp. 16–32.

[63] Quoc V. Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents." In: *ICML*. Vol. 32. JMLR.org, 2014, pp. 1188–1196.

[64] Oliver Lehmberg and Christian Bizer. "Stitching Web Tables for Improving Matching Quality." In: *Proc. VLDB Endow.* 10.11 (2017), pp. 1502–1513.

[65] Oliver Lehmberg and Christian Bizer. "Profiling the semantics of n-ary web table data." In: *Proceedings of the International Workshop on Semantic Big Data, SBD@SIGMOD*. ACM, 2019, 5:1–5:6.

[66] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. "A Large Public Corpus of Web Tables containing Time and Context Metadata." In: *WWW*. ACM, 2016, pp. 75–76.

[67] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. "The Mannheim Search Join Engine." In: *J. Web Semant.* 35 (2015), pp. 159–166.

[68] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. "Deep entity matching with pre-trained language models." In: *Proc. VLDB Endow.* 14.1 (2020), pp. 50–60.

[69] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. "Annotating and Searching Web Tables Using Entities, Types and Relationships." In: *Proc. VLDB Endow.* 3.1 (2010), pp. 1338–1347.

[70] Steve Lohr. "For big-data scientists, 'janitor work' is key hurdle to insights." In: *New York Times* 17 (2014).

[71] Xusheng Luo, Kangqi Luo, Xianyang Chen, and Kenny Q. Zhu. "Cross-Lingual Entity Linking for Web Tables." In: *AAAI*. AAAI Press, 2018, pp. 362–369.

[72] Christos Makris, Georgios Pispirigos, and Michael Angelos Simos. "Text Semantic Annotation: A Distributed Methodology Based on Community Coherence." In: *Algorithms* 13.7 (2020), p. 160.

[73] José-Lázaro Martínez-Rodríguez, Aidan Hogan, and Ivan López-Arévalo. "Information extraction meets the Semantic Web: A survey." In: *Semantic Web* 11.2 (2020), pp. 255–335.

[74] André Melo, Johanna Völker, and Heiko Paulheim. "Type Prediction in Noisy RDF Knowledge Bases Using Hierarchical Multilabel Classification with Graph and Latent Features." In: *Int. J. Artif. Intell. Tools* 26.2 (2017).

[75] Peter Mika, Edgar Meij, and Hugo Zaragoza. "Investigating the Semantic Gap through Query Log Analysis." In: *ISWC*. Vol. 5823. Springer, 2009, pp. 441–455.

[76] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. "Distributed Representations of Words and Phrases and their Compositionality." In: *NeurIPS*. 2013, pp. 3111–3119.

[77] Andrea Moro, Alessandro Raganato, and Roberto Navigli. "Entity Linking meets Word Sense Disambiguation: a Unified Approach." In: *Trans. Assoc. Comput. Linguistics* 2 (2014), pp. 231–244.

[78] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. "Deep Learning for Entity Matching: A Design Space Exploration." In: *SIGMOD*. ACM, 2018, pp. 19–34.

[79] Varish Mulwad, Tim Finin, Zareen Syed, and Anupam Joshi. "Using Linked Data to Interpret Tables." In: *Proceedings of the First International Workshop on Consuming Linked Data*. Vol. 665. CEUR-WS.org, 2010.

[80] Emir Muñoz, Aidan Hogan, and Alessandra Mileo. "Using linked data to mine RDF from wikipedia's tables." In: *WSDM*. ACM, 2014, pp. 533–542.

[81] Daniele Nardi, Ronald J Brachman, et al. "An introduction to description logics." In: *Description logic handbook* 1 (2003), p. 40.

[82] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. "A survey of current Link Discovery frameworks." In: *Semantic Web* 8.3 (2017), pp. 419–436.

[83] Sebastian Neumaier, Jürgen Umbrich, Josiane Xavier Parreira, and Axel Polleres. "Multi-level Semantic Labelling of Numerical Values." In: *ISWC*. Vol. 9981. 2016, pp. 428–445.

[84] Axel-Cyrille Ngonga Ngomo and Sören Auer. "LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data." In: *IJCAI*. IJCAI/AAAI, 2011, pp. 2312–2317.

[85] Phuc Nguyen, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. "MTab: Matching Tabular Data to Knowledge Graph using Probability Models." In: *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching co-located with the 18th International Semantic Web Conference, SemTab@ISWC*. Vol. 2553. CEUR-WS.org, 2019, pp. 7–14.

[86] Thanh Tam Nguyen, Nguyen Quoc Viet Hung, Matthias Weidlich, and Karl Aberer. "Result selection and summarization for Web Table search." In: *ICDE*. IEEE Computer Society, 2015, pp. 231–242.

[87] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. "Deep Sequence-to-Sequence Entity Matching for Heterogeneous Entity Resolution." In: *CIKM*. ACM, 2019, pp. 629–638.

[88] Kyosuke Nishida, Kugatsu Sadamitsu, Ryuichiro Higashinaka, and Yoshihiro Matsuo. "Understanding the Semantic Structures of Tables with a Hybrid Deep Neural Network Architecture." In: *AAAI*. AAAI Press, 2017, pp. 168–174.

[89] Debora Nozza, Federico Bianchi, and Dirk Hovy. "What the [MASK]? Making Sense of Language-Specific BERT Models." In: *arXiv e-prints* (2020). arXiv: 2003.02912 [cs.CL].

[90] Daniel Obraczka and Axel-Cyrille Ngonga Ngomo. "Dragon: Decision Tree Learning for Link Discovery." In: *ICWE*. Vol. 11496. Springer, 2019, pp. 441–456.

[91] Matteo Palmonari, Anisa Rula, Riccardo Porrini, Andrea Maurino, Blerina Spahiu, and Vincenzo Ferme. "ABSTAT: Linked Data Summaries with ABstraction and STATistics." In: *ESWC Satellite Events*. Vol. 9341. Lecture Notes in Computer Science. Springer, 2015, pp. 128–132.

[92] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. "Schema-agnostic vs Schema-based Configurations for Blocking Methods on Homogeneous Data." In: *Proc. VLDB Endow.* 9.4 (2015), pp. 312–323.

[93] George Papadakis, Georgios M. Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. "Three-dimensional Entity Resolution with JedAI." In: *Inf. Syst.* 93 (2020), p. 101565.

[94] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." In: *EMNLP*. ACL, 2014, pp. 1532–1543.

[95] Dessislava Petrova-Antonova and Rumyana Tancheva. "Data Cleaning: A Case Study with OpenRefine and Trifacta Wrangler." In: *QUATIC*. Vol. 1266. Springer, 2020, pp. 32–40.

[96] Minh Pham, Suresh Alse, Craig A. Knoblock, and Pedro A. Szekely. "Semantic Labeling: A Domain-Independent Approach." In: *ISWC*. Vol. 9981. 2016, pp. 446–462.

[97] Rakesh Pimplikar and Sunita Sarawagi. "Answering Table Queries on the Web using Column Keywords." In: *Proc. VLDB Endow.* 5.10 (2012), pp. 908–919.

[98] Nikiforos Pittaras, George Papadakis, George Stamoulis, Giorgos Argyriou, Efi Karra Taniskidou, Emmanouil Thanos, George Giannakopoulos, Leonidas Tsekouras, and Manolis Koubarakis. "GeoSensor: semantifying change and event detection over big data." In: *SAC*. ACM, 2019, pp. 2259–2266.

[99] Gianluca Quercini and Chantal Reynaud. "Entity discovery and annotation in tables." In: *EDBT*. ACM, 2013, pp. 693–704.

[100] S. K. Ramnandan, Amol Mittal, Craig A. Knoblock, and Pedro A. Szekely. "Assigning Semantic Labels to Data Sources." In: *ESWC*. Vol. 9088. Springer, 2015, pp. 403–417.

[101] Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. "RDF2Vec: RDF graph embeddings and their applications." In: *Semantic Web* 10.4 (2019), pp. 721–752.

[102] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. "Matching HTML Tables to DBpedia." In: *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, WIMS*. ACM, 2015, 10:1–10:6.

[103] Dominique Ritze, Oliver Lehmberg, Yaser Oulabi, and Christian Bizer. “Profiling the Potential of Web Tables for Augmenting Cross-domain Knowledge Bases.” In: *WWW*. ACM, 2016, pp. 251–261.

[104] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. “Finding related tables.” In: *SIGMOD*. ACM, 2012, pp. 817–828.

[105] Yoones A. Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. “Knowledge Base Augmentation using Tabular Data.” In: *Proceedings of the Workshop on Linked Data on the Web co-located with WWW*. Vol. 1184. CEUR-WS.org, 2014.

[106] Vivek R. Shivaprabhu, Booma Sowkarthiga Balasubramani, and Isabel F. Cruz. “Ontology-based Instance Matching for Geospatial Urban Data Integration.” In: *Proceedings of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. ACM, 2017, 8:1–8:8.

[107] Michael Stonebraker. “The Case for Shared Nothing.” In: *IEEE Database Eng. Bull.* 9.1 (1986), pp. 4–9.

[108] Dina Sukhobok, Nikolay Nikolov, Antoine Pultier, Xianglin Ye, Arne Berre, Rick Moynihan, Bill Roberts, Brian Elvesæter, Nivethika Mahasivam, and Dumitru Roman. “Tabular data cleaning and linked data generation with Grafterizer.” In: *ISWC*. Springer. 2016, pp. 134–139.

[109] Dina Sukhobok, Nikolay Nikolov, and Dumitru Roman. “Tabular Data Anomaly Patterns.” In: *International Conference on Big Data Innovations and Applications*. IEEE, 2017, pp. 25–34.

[110] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. “Table Cell Search for Question Answering.” In: *WWW*. ACM, 2016, pp. 771–782.

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

[112] Avijit Thawani, Minda Hu, Erdong Hu, Husain Zafar, Naren Teja Divvala, Amandeep Singh, Ehsan Qasemi, Pedro A. Szekely, and Jay Pujara. “Entity Linking to Knowledge Graphs to Infer Column Types and Properties.” In: *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab@ISWC*. Vol. 2553. CEUR-WS.org, 2019, pp. 25–32.

[113] Goce Trajcevski, Booma Sowkarthiga Balasubramani, Isabel F. Cruz, Roberto Tamassia, and Xu Teng. “Semantically Augmented Range Queries over Heterogeneous Geospatial Data.” In: *SIGSPATIAL*. ACM, 2020, pp. 68–77.

[114] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. "AGDISTIS - Graph-Based Disambiguation of Named Entities Using Linked Data." In: *ISWC*. Vol. 8796. Springer, 2014, pp. 457–471.

[115] Gilles Vandewiele, Bram Steenwinckel, Filip De Turck, and Femke Ongenae. "CVS2KG: Transforming Tabular Data into Semantic Knowledge." In: *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab@ISWC*. Vol. 2553. CEUR-WS.org, 2019, pp. 33–40.

[116] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. "Recovering Semantics of Tables on the Web." In: *Proc. VLDB Endow.* 4.9 (2011), pp. 528–538.

[117] John Villafranco. "Self-Regulation in the Big Data and AI Space." In: *The Judges' Journal* 59.1 (2020), pp. 32–35.

[118] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. "Discovering and Maintaining Links on the Web of Data." In: *ISWC*. Vol. 5823. Springer, 2009, pp. 650–665.

[119] Hongzhi Wang, Mingda Li, Yingyi Bu, Jianzhong Li, Hong Gao, and Jiacheng Zhang. "Cleanix: a Parallel Big Data Cleaning System." In: *SIGMOD Record* 44.4 (2015), pp. 35–40.

[120] Pei Wang, Yongjun He, Ryan Shea, Jiannan Wang, and Eugene Wu. "Deeper: A Data Enrichment System Powered by Deep Web." In: *SIGMOD*. ACM, 2018, pp. 1801–1804.

[121] Mark D. Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship." In: *Scientific Data* 3.1 (2016), p. 160018.

[122] Daniel Wind. *Instant effective caching with Ehcache*. Packt Publishing Ltd, 2013.

[123] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. "Sato: Contextual Semantic Type Detection in Tables." In: *Proc. VLDB Endow.* 13.11 (2020), pp. 1835–1848.

[124] Lu Zhang, Tiantian Wang, Yiran Liu, and Qingling Duan. "A semi-structured information semantic annotation method for Web pages." In: *Neural Comput. Appl.* 32.11 (2020), pp. 6491–6501.

[125] Meihui Zhang and Kaushik Chakrabarti. "InfoGather+: semantic matching and annotation of numeric and time-varying attributes in web tables." In: *SIGMOD*. ACM, 2013, pp. 145–156.

[126] Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. "Novel Entity Discovery from Web Tables." In: *WWW*. ACM / IW3C2, 2020, pp. 1298–1308.

[127] Ziqi Zhang. "Effective and efficient Semantic Table Interpretation using TableMiner$^+$." In: *Semantic Web* 8.6 (2017), pp. 921–957.

[128] Hai Zhuge and Xiaoping Sun. "Semantics, knowledge, and grids at the age of big data and AI." In: *Concurrency Computation* 31 (3 2019).

[129] S. Zillner, E. Curry, A. Metzger, S. Auer, R. Seidl, and (Eds.) *European Big Data Value Strategic Research & Innovation Agenda*. 2017.