

Easing Information Extraction on the Web through Automated Rules Discovery



Stefano Ortona

St John's College

Supervisor: *Professor Georg Gottlob*

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2016

*“Science is not only a disciple of reason but,
also, one of romance and passion.”*
— *Stephen Hawking*

Dedicated to Giancarlo Ortona
(20/3/1956 - 23/12/1998)

Acknowledgements

“Once you stop learning, you start dying.”

— *Albert Einstein*

This thesis has a special meaning for all that it represents. It summarises more than three years of hard work, moments of joy and (thankfully few) moments of desperation. It has been a great honour to study at such a prestigious university, and I consider myself extremely lucky to have had such an incredible opportunity. Sometimes we take life and things in it for granted, but we should never forget how other people will never have those privileges that we consider “normal”. When the moment of writing this final piece of the thesis has come, I finally realised how long this journey has been. And despite being a lonely journey where I found most of the strength in myself, this would have not been possible without the help and the support of many people. I would like to thank few of them here, but this list is by no means exhaustive.

Undoubtedly, the two persons who have helped me, inspired me, and guided me the most through this difficult journey are my two thesis advisors, Dr. Tim Furche and Dr. Giorgio Orsi. I am deeply thankful to them for all the long discussions and the extremely useful feedback. The research I will present in this thesis would have not been the same without their constant push and support. They are those who taught me how beautiful and exciting doing research can be, and I will always be thankful because of that. A special thanks goes also to my official supervisor, Professor Georg Gottlob. He has always been a great source of inspiration and it is because of him that I decided to apply to Oxford University. He gave me the freedom to lead my DPhil in the way that I wanted.

Spending this much time at Oxford was a never ending learning experience. I particularly enjoyed collaborating with professors Michael Benedikt and Dan Olteanu. They have always been so kind to provide great feedback on my work, and the numerous constructive discussions we had gave a precious touch to this thesis. I also need to thank all the DIADEM/VADA group: Giovanni, Julia, Xiaonan, Cheng, Christian, Jinsong, Emanuel, Omer, and

Andrey. Going every day for the past four years to our office was always a pleasure, and sharing lunches and coffees with you made my time fun and entertaining. A special note also for Luying Chen and Marcello Buoncristiano, who gave a significant contribution in developing ROSeAnn and WADaR respectively.

A warm thought goes to my family, who never doubted that eventually I would have made it this far. This is particularly true for my mum, who has always been on my side despite constantly pushing me on finding a “real” job. I will always be grateful for all the sacrifices she made for me and for allowing me to follow my dreams. Thanks to my brother, Michele, his wife Valentina, and those two little angels Aurora and Serena. The warmth and the affection you gave me was a fundamental stone for my stability. Thanks to my grandpa, Pierugo, from whom I learned many life lessons.

A special paragraph must be reserved for my friends, who are a strong pillar in my life. First of all I want to thank my Oxford family: Andreas, Lisa, Rita, Anna, Ryan, Miles, and Christos. You are the only reason why I did not go crazy in desperate moments and I could not ask for better and smarter friends to experience my DPhil with. I would never have thought that I could find a second family in Oxford. Secondly, a tender thank you goes to my friends in Rome, “*A-miciii*”. Your proudness and happiness with my choices were a great encouragement to always pursue the best. Leaving Rome was difficult and heartbreaking, but I could have not made such a difficult decision without your love and support. A special credit goes to my eternal friend Virgi. She helped me to get through my most difficult moments and our countless Skype calls kept me focus on achieving my dreams. Last but not least, I want to thank a special group of friends here in Oxford, the “*Fellas*”. Thanks for always reminding me that “work hard, play hard” is the way we should live our lives.

Finally, let me conclude by thanking all the people who were patient enough to collaborate with me. The last part of this thesis is the product of a stimulating collaboration with Dr. Paolo Papotti. I want to thank him for hosting me at Arizona State University, and for all the help he recently gave me in figuring out what to do next. A special thanks to my professors in Rome: Paolo Merialdo, Valter Crescenzi, and Paolo Atzeni. You made me fall in love with Computer Science and I would not be here today without you.

Publications

Some charts, contents and figures in this thesis have appeared previously in the following publications:

1. Luying Chen, Stefano Ortona, Giorgio Orsi, and Michael Benedikt. ROSeAnn: Reconciling opinions of semantic annotators. *Proceedings of the VLDB Endowment* 6, no. 12 (2013): 1238-1241.
2. Luying Chen, Stefano Ortona, Giorgio Orsi, and Michael Benedikt. Aggregating semantic annotators. *Proceedings of the VLDB Endowment* 6, no. 13 (2013): 1486-1497.
3. Stefano Ortona, Luying Chen, and Giorgio Orsi. ROSeAnn: taming on-line semantic annotators. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pp. 591-594. International World Wide Web Conferences Steering Committee, 2014.
4. Stefano Ortona. An analysis of duplicate on web extracted objects. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pp. 1279-1284. International World Wide Web Conferences Steering Committee, 2014.
5. Stefano Ortona, Giorgio Orsi, Marcello Buoncristiano, and Tim Furche. WADaR: Joint wrapper and data repair. *Proceedings of the VLDB Endowment* 8, no. 12 (2015): 1996-1999.
6. Stefano Ortona, Giorgio Orsi, Tim Furche, and Marcello Buoncristiano. Joint repairs for web wrappers. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pp. 1146-1157. IEEE, 2016
7. Stefano Ortona, Vamsi Meduri, and Paolo Papotti. Discovery of Positive and Negative Rules in Knowledge-Bases. *Currently submitted to VLDB 2017*.

Abstract

The advent of the era of big data on the Web has made automatic web information extraction an essential tool in data acquisition processes. Unfortunately, automated solutions are in most cases more error prone than those created by humans, resulting in dirty and erroneous data. Automatic repair and cleaning of the extracted data is thus a necessary complement to information extraction on the Web.

This thesis investigates the problem of inducing cleaning rules on web extracted data in order to (i) repair and align the data w.r.t. an original target schema, (ii) produce repairs that are as generic as possible such that different instances can benefit from them. The problem is addressed from three different angles: replace cross-site redundancy with an ensemble of entity recognisers; produce general repairs that can be encoded in the extraction process; and exploit entity-wide relations to infer common knowledge on extracted data.

First, we present **ROSeAnn**, an unsupervised approach to integrate semantic annotators and produce a unified and consistent annotation layer on top of them. Both the diversity in vocabulary and widely varying accuracy justify the need for middleware that reconciles different annotator opinions. Considering annotators as “black-boxes” that do not require per-domain supervision allows us to recognise semantically related content in web extracted data in a scalable way.

Second, we show in **WADaR** how annotators can be used to discover rules to repair web extracted data. We study the problem of computing joint repairs for web data extraction programs and their extracted data, providing an approximate solution that requires no per-source supervision and proves effective across a wide variety of domains and sources. The proposed solution is effective not only in repairing the extracted data, but also in encoding such repairs in the original extraction process.

Third, we investigate how relationships among entities can be exploited to discover inconsistencies and additional information. We present **RuDiK**, a disk-based scalable solution to discover first-order logic rules over RDF knowledge bases built from web sources. We present an approach that does not limit its search space to rules that rely on “positive” relationships between entities, as in the case with traditional mining of constraints. On the contrary, it extends the search space to also discover negative rules, i.e., patterns that lead to contradictions in the data.

Contents

1	Introduction	1
1.1	Thesis Novelty and Motivation	5
1.2	Thesis Contributions	7
1.3	Thesis Organisation	10
I	Semantic Annotations	11
2	Named Entity Recognition	12
2.1	Annotator Ecosystem	13
2.1.1	Software Framework	14
2.1.2	Web Service API	14
2.2	Vocabulary Alignment	16
2.2.1	Ontology	17
2.3	Conflicts	19
2.4	Annotator Evaluation	21
2.4.1	Dataset	21
2.4.2	Precision and Recall	23
2.4.3	Results	24
2.4.4	Observed Conflicts	27
2.5	The Need for Integration	28
3	ROSeAnn: Reconcile Opinions of Semantic Annotators	30
3.1	Unsupervised Aggregation	31
3.1.1	Atomic Score	31
3.1.2	Weighted Repair	33
3.1.3	Implementation	35
3.1.3.1	Ontology	37
3.1.4	Tractability	38
3.1.5	Bootstrapping	41
3.1.6	Overlapping Span	42
3.2	Supervised Algorithm	44
3.3	Experiments	45

3.3.1	Comparison with Individual Systems	45
3.3.2	Supervised vs Unsupervised	47
3.3.3	Comparative Evaluation	49
3.3.4	Bootstrapping Evaluation	50
3.3.5	Scalability	50
3.3.6	Summary	51
3.4	ROSeAnn: an Available System	52
3.4.1	Graphical Interface	52
3.4.2	Java API	54
3.4.3	Web Service	55
3.5	Related Work	55
3.6	Towards an Alternative to Redundancy	59

II Data Cleaning and Repair 61

4	Web Data Extraction: the Need for Cleaning	62
4.1	Web Wrapper Induction	63
4.2	Web Data Repair: New Challenges	65
4.3	Data Cleaning	66
4.4	Table Segmentation	67
5	WADaR: Joint Wrapper and Data Repair	69
5.1	Preliminaries	73
5.2	Maximal Joint Repair: Atomic Misplacement	76
5.3	Maximal Joint Repair: NP-Completeness	80
5.4	Approximating Wrapper and Data Repair	82
5.4.1	Annotation	83
5.4.2	Segmentation	83
5.4.3	Induction	89
5.4.4	Optimality	91
5.4.5	Complexity	92
5.5	Experiments	93
5.5.1	Dataset	93
5.5.2	Wrapper Generation Systems	94
5.5.3	Metrics	94
5.5.4	Global Accuracy	95
5.5.5	Per Attribute Accuracy	97
5.5.6	Large Scale Evaluation	98
5.5.7	Comparative Evaluation	99
5.5.8	Ablation Study	100

5.5.9	Thresholding	101
5.5.10	Annotator Accuracy	101
5.6	WADaR: an Online Data Repair Platform	103
5.7	Related Work	105
5.8	Annotators: Take the Bad with the Good	107
III	RDF Knowledge Bases	109
6	RuDiK: Affordable Rule Discovery in Knowledge Bases	110
6.1	Preliminaries and Definitions	113
6.1.1	Language	114
6.1.2	Rule Coverage	115
6.1.3	Weight Function	117
6.1.4	Problem Definition	118
6.2	Rule Generation	119
6.2.1	Input Examples Generation	122
6.2.2	Literals and Constants	125
6.3	A* Greedy Algorithm	126
6.3.1	Marginal Weight	126
6.3.2	A* Graph Traversal	127
6.4	Experiments	131
6.4.1	Rule Discovery Accuracy	133
6.4.1.1	Positive Rule Discovery	134
6.4.1.2	Negative Rule Discovery	135
6.4.2	Comparative Evaluation	136
6.4.3	Machine Learning Application	141
6.4.4	Internal Evaluation	143
6.5	Related Work	145
6.6	Main Takeaways	148
7	Conclusion	150
7.1	Future Directions	151
	Bibliography	153

List of Figures

1.1	Ideal Information Extraction on Two Web Sources	2
1.2	Actual Information Extraction on Two Web Sources	4
2.1	OPENCALAIS Online Annotations Tool	13
2.2	ALCHEMYAPI Web Service Response	15
2.3	Web Annotators Disagreement and Semantic Interconnections	16
2.4	Example Snippet of Merged Ontology	19
2.5	Types of Annotator Conflicts	20
2.6	Performance of Individual Annotators	25
3.1	Annotated Span with Atomic Scores	35
3.2	Conflicts in Annotated Spans	42
3.3	Individual Annotators vs Aggregators	46
3.4	Comparative Evaluation.	49
3.5	Aggregation Performance	51
3.6	ROSeAnn GUI	52
3.7	ROSeAnn Innovative Components	53
4.1	Web Data Extraction on <code>waterstones.com</code>	64
4.2	Web Data Extraction on <code>waterstones.com</code> with ROADRUNNER	64
5.1	Data Repair and Alignment on ROADRUNNER	70
5.2	Max-Flow Network vs Markov Chain	88
5.3	Impact of repair.	96
5.4	Comparison with WEIR (Redundant values)	99
5.5	Comparison with WEIR (all values)	100
5.6	Impact of individual components	100
5.7	Impact of t_{flow} and t_{regex} thresholds	101
5.8	Annotator recall drop - Fixed threshold	102
5.9	F_1 -Score variation with a threshold value of 0.1	102
5.10	WADaR Entry Point	103
5.11	WADaR Annotations and Max-Flows Computation	104
5.12	WADaR Repaired Relation	104
6.1	Graph Portion of DBPEDIA	120

6.2	Predictions Accuracy on YAGO	138
6.3	Predictions Accuracy on DBPEDIA	139
6.4	DeepDive Application 1K articles	142
6.5	DeepDive Application 1M articles	142

List of Tables

2.1	Description of Testing Corpora	23
2.2	Conflicts Statistic of Individual Annotators	27
2.3	Span conflicts.	28
5.2	Dataset characteristics.	93
5.3	Wrapper induction errors.	95
5.4	Attribute-level evaluation.	97
5.5	Accuracy of large scale evaluation.	98
6.1	Dataset characteristics.	134
6.2	Positive Rules Accuracy.	134
6.3	Negative Rules Accuracy.	135
6.4	Interesting Output Rules.	136
6.5	AMIE Dataset characteristics.	137
6.6	Negative Rules vs AMIE.	139
6.7	Run Time vs AMIE.	140
6.8	Rules Accuracy without Literals on DBPEDIA.	143
6.9	<i>maxPathLen</i> Parameter Impact on DBPEDIA.	143
6.10	Effect of Negative Examples Generation Strategy on DBPEDIA.	144

Chapter 1

Introduction

The Web has become the largest repository of data ever built in human history. Recent advances in web technologies have made data publishing on the Web an every-day task that is increasing at an unprecedented scale. In the US only, it is estimated that there are over one hundred thousand online shopping web-sites with revenues greater than \$10,000 [112]. This scenario of ever growing information is often defined as the era of *big data*. Despite many different definitions, everyone seems to agree on the 4 V's that characterise big data:¹ *Volume* – the scale in terms of size of data; *Velocity* – the speed of change in the content of data; *Variety* – the difference in flavour and format of data; *Veracity* – the uncertainty that rules such complex data.

The amount of information available on the Web has attracted the interest of a significant body of work in Information Extraction (IE). The main goal of IE systems is to extract relevant data from un(semi)-structured sources and turn such data into some sort of structured format. Data currently available on the Web can be grouped into two main categories: (i) *semi-structured data*, such as HTML pages designed for humans, and (ii) *unstructured data*, such as free text documents. Extracting data from the former is often called *web data extraction* [32, 34, 51, 82, 156]. Extracting data from the latter is defined as *open information extraction* (OIE) [21, 135, 140], where open means that the schema for target relations to extract does not need to be specified in advance. Figure 1.1 shows an example of web IE from two different sources. The first scenario (Figure 1.1a) showcases an example of web data extraction to store relevant data about cameras in a relational database with a well-defined schema, while in the second scenario (Figure 1.1b) the goal is to extract relationships between pairs of entities from a Wikipedia text document, a classic OIE task. The focus of this work is mostly on web data extraction. However, most of the points we outline in this introduction apply equally to OIE as well. In the last part of the thesis (Chapter 6), we will exploit some cleaning techniques to enhance the quality of knowledge bases that may be generated through OIE, by human supervision or through web information extraction or a variant of it.

¹<http://www.ibmbigdatahub.com/infographic/four-vs-big-data>

Nikon Coolpix L340 20.2 MP Digital Camera with 28x Optical Zoom and 3.0-Inch LCD (Black)

by Nikon

★★★★★ 2,201 customer reviews | 707 answered questions

Available from these sellers.

Style: 28x Optical Zoom and 3.0-Inch LCD

28x optical zoom	28x Optical Zoom
from \$109.95	from \$139.00
28x Optical Zoom and 3.0-Inch LCD	Certified Refurbished
from \$139.00	\$139.00

Brand	Model	Zoom	Price	Price	Colour
Nikon	Coolpix L340	28x Optical Zoom	From \$139.00	3.0-Inch LCD	Black

(a) Web Data Extraction on Amazon

Article Talk

<Clint Eastwood, born, San Francisco> Read Edit

Clint Eastwood

From Wikipedia, the free encyclopedia

For other uses, see *Clint Eastwood (disambiguation)*.

Clinton "Clint" Eastwood Jr. (born May 31, 1930) is an American actor, film director, producer, musician, and political figure. He rose to international fame with his role as the *Man with No Name* in Sergio Leone's *Dollars* trilogy of spaghetti Westerns during the 1960s, and as antihero cop *Harry Callahan* in the five *Dirty Harry* films throughout the 1970s and 1980s. These roles, among others, have made Eastwood an enduring cultural icon of masculinity.^{[9][10]}

Eastwood was born in San Francisco, California, the son of Clinton Eastwood Sr. (1906–1970) and his wife, Margaret Ruth (*née* Runner) Eastwood (1909–2006). He was nicknamed "Samson" by the hospital nurses because he weighed 11 pounds 6 ounces (5.2 kg) at birth.^{[12][13]} He has a younger sister, Jeanne (born 1934).^[14] Eastwood's widowed mother later married a retired lumber executive, John Belden Wood (1913–2004), who became stepfather to Clint and Jeanne.^[15]

Eastwood married Margaret Neville "Maggie" Johnson (then working for an auto parts suppliers company^[262]) months earlier on a blind date in Los Angeles,^[263] although Eastwood subsequently had a serious relationship

<San Francisco, located, California> **<Maggie Johnson, married, Clint Eastwood>**

(b) Open Information Extraction on Wikipedia

Figure 1.1: Ideal Information Extraction on Two Web Sources

It is worth keeping in mind that accurate extraction of data on the Web is becoming vital to data-driven decisions. Web data extraction, also known as *web scraping*, is often the only viable data collection method for websites that offer no APIs or do not expose structured data. With the advent of big data, the need for accurate extraction has only increased, with a particular interest in analytics for competitive price intelligence or intelligent supply chain management. The revitalised interest in this area is also evident from a growing number of web scraping startups, e.g., IMPORT.IO, DIFFBOT, SCRAPINGHUB, LATTICE.IO, as well as recent acquisitions,

e.g., BLACKLOCUS by HOME DEPOT, LIXTO by MCKINSEY.

Among the 4 V's of big data, the one that is arguably posing the hardest challenges for web scraping is a combination of Volume and Variety. This is because the amount of useful information on the Web has been growing at an unexpected pace in recent years. More specifically, nowadays data on the Web w.r.t. a specific domain is divided into a *heavy head*, with a few large aggregator websites that publish most of the data, and a *long tail*, where several hundreds of thousand smaller websites contribute in some small way to publish remaining data. Despite being a small portion, the long tail cannot be ignored if we wish to construct a comprehensive body of information [57]. Given the amount of data and sources we need to deal with, semisupervised extraction approaches [23, 55], which require users to supervise each site by identifying relevant data, are not appropriate. Fully automated unsupervised solutions at the scale of hundreds of thousands of sources are the only practical means for web scraping in the era of big data [51, 52, 56, 82].

Automated techniques are based on the key assumption that data published on the Web is originally stored in a behind-the-scene database, and then some scripts gather the data from the database and transform such data in semi-structured formats for presentation. Essentially extracting data from the Web is a reverse engineering process that aims at reconstructing the *hidden* database behind a website. This assumption is backed up by the size of modern websites and their structures. In fact nowadays semi-structured websites publish thousands (millions) of individual records, where each of them typically has a dedicated webpage with a layout similar to other pages of the same source. For example, Amazon was offering roughly 500 millions products in 2015 in the US only.² The only feasible solution for publishing such a big amount of data is making use of scripts that read the data from an unknown data source that is not exposed to the final user.

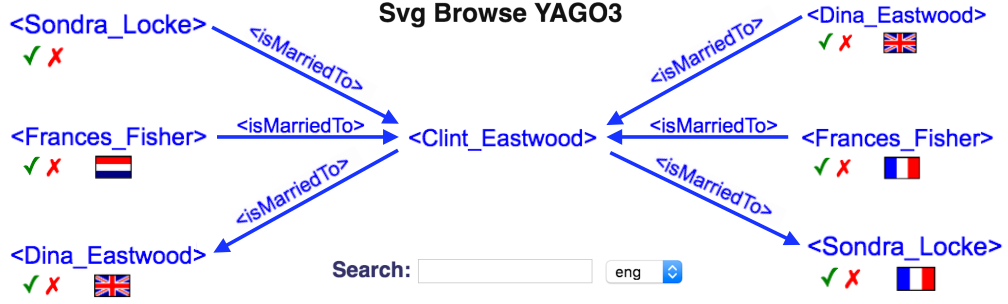
Despite the assumption of having repeated similar structures within a single source, fully automated techniques are still more error prone than those created by humans or with semi-supervised approaches, resulting in dirty data. Eliminating human supervision comes at a price, and this price usually translates to missing data, data misaligned w.r.t. a desired schema, or erroneous data. If Figure 1.1 represents the ideal outcome of IE on the Web from two different sources, Figure 1.2 illustrates what happens in reality. In the case of Amazon, Figure 1.2a shows the output of ROADRUNNER [51], one of the most popular web data extraction systems. Attributes are over-segmented due to the inability of the system to separate different values within the same DOM text node, and there are missing column types due to the lack of domain knowledge. Figure 1.2b instead shows some values currently stored in the online version of YAGO,³ a widely

²<https://export-x.com/2015/12/11/how-many-products-does-amazon-sell-2015/>

³https://gate.d5.mpi-inf.mpg.de/webyago3spotlx/SvgBrowser?entity=%3CClint_Eastwood%3E&relation=%3CisMarriedTo%3E

Attribute_1	Attribute_2
Nikon Coolpix L340 20.2 MP Digital Camera with 28x Optical Zoom and 3.0-Inch LCD (Black)	28x Optical Zoom and 3.0 Inch LCD from \$139.00

(a) Web Data Extraction with ROADRUNNER on Amazon



(b) Open Information Extraction in YAGO

Figure 1.2: Actual Information Extraction on Two Web Sources

used knowledge base built from Wikipedia and Wordnet, to store relationships among real-world entities. For example, according to YAGO, the actor Clint Eastwood married three different women. However, two of them (actresses Frances Fisher and Sondra Locke) never married him, they were just his partners. Moreover, Clint Eastwood’s first wife (Maggie Johnson) is missing, despite the fact of their marriage appearing on Wikipedia (Figure 1.1b).

Data shown in Figure 1.2 is clearly unusable. The presence of errors and mixed up values pose serious challenges on the usability of web extracted data, since noise and missing information could heavily bias business decisions. There is an evident need for a *cleaning* step before any application can get real benefits from such data. A significant body of work has addressed the problem of cleaning erroneous databases [43, 48, 88, 124, 154] and inducing cleaning constraints [17, 41, 97, 153], ranging from manually defined cleaning rules to advanced learning algorithms. Data cleaning techniques induce data-dependent strategies to detect and repair inconsistencies from a single data source. The task is considered solved once the data has been repaired, and different data instances require different runs of the induction process. Unfortunately, web data acquisition is an ongoing process, through which data sources are constantly inspected and data is constantly updated. This is particularly true in the era of big data, where one of the recurring characteristics is Velocity, i.e., the rate at which sources change their content. Moreover, classic database techniques are often not suitable to deal with data extracted from the Web. The size, the uncertainty in the data, and the lack of a pre-defined schema introduce unforeseen challenges that cannot be neglected.

In this thesis we investigate the problem of discovering rules to enhance the quality of web extracted data. Unlike classic database cleaning systems, our goal is to discover rules that are *as general as possible* such that they can be applied multiple times and

on multiple instances, thereby reducing the need of repeat data cleaning and easing integration [73, 111]. W.r.t. Figure 1.2a, a general rule would state that the colour of the product is what appears in round brackets in `Attribute_1`, which is a true statement for most of the products on Amazon. On the other hand, a good rule to solve inconsistencies in Figure 1.2b could state that if two people are labelled as partners on Wikipedia, then they are not married (otherwise Wikipedia would label them as married). The discovery of such rules ease IE on the Web from different angles. First, by applying such rules on the data, we can repair inconsistencies and improve the quality of the data. Second, inducing general rules does not require the re-discovery at every new extraction. Inducing rules is an expensive process, and the more we can avoid it the better. Third, rules can improve the data extraction process itself with the identification of repeating mistake patterns that can be directly encoded back in the extraction process. Most of the errors in web extracted data are introduced in the extraction process [62], hence solving them at the origin would entail producing clean data at every future extraction. Last, rules help domain experts in maintaining and curating big data industrial systems [87].

1.1 Thesis Novelty and Motivation

Annotators as an Alternative to Redundancy. In the first part of this work (Part I) we will focus on a pure IE task: *semantic annotations*. Semantic annotations label snippets of documents according to elements of particular semantic categories (e.g., people, locations, organisations). Such a task is often defined as *named entity recognition* [47, 67, 102, 121]. Given the abundance of research and industrial systems for semantic annotations, we will not design yet another named entity recogniser. On the contrary, we will exploit services that expose semantic annotations as “black-boxes” (e.g. `ALCHEMYAPI`, `OPENCALAIS`), where the user does not have to worry about tuning the service. On the other hand, concept classes can be modified by the service host outside the control of clients.

In this part we present `ROSeAnn`, a novel system for *semantic annotation integration*: middleware that produces a unified annotation on top of eleven semantic annotators. We will show that annotation integration brings substantial benefits both in terms of quality of the result, and in terms of a much richer vocabulary available in an integrated vocabulary. `ROSeAnn` investigates the problem in a novel way, by searching for integration methods that are *ontology-aware* – making as much use as possible of semantic relationships that are available between different concept classes. Unlike previous integration works that use machine learning techniques to determine the optimal combination of annotators [64, 130, 138, 136], we designed an *unsupervised* approach for on-line aggregation in the absence of training data.

Despite being an interesting research direction per se, the main motivation that drives the interest in semantic annotations is *redundancy*, one of the major techniques used in web data extraction. Instance-level and cross-site redundancy has been used in different contexts to detect and repair inconsistent data extracted from the Web [32, 43, 44, 65]. Unfortunately, redundancy-based methods often require the instance-level quadratic comparison of a substantial number of sources that have, so far, limited their scalability. This is hardly negligible in the era of big data, where the number of sources is in the order of hundreds of thousands. Moreover, it is not always possible to leverage sufficient redundancy in every domain. Therefore we decided to encode redundancy in an ensemble of semantic annotators that require (almost) no user effort.

Jointly Repair Web Data and Wrappers. In the second part (Part II) we will use ROSeAnn to tackle the main problem of this thesis: repair web extracted data, along with the programs that generated that data. Web data extraction techniques typically work by inducing a *wrapper* for each target source. A wrapper is a program that turns web content into structured data using techniques ranging from visual analysis of the rendered page to DOM tree mining [51, 82, 94, 156]. As mentioned earlier, simply repairing the extracted data is often not sufficient, since in many cases data acquisition is an ongoing process and thus wrappers will likely be executed many more times in the future. This is particularly the case with data integration [32, 44], in which future executions of the repaired wrappers generate relations already aligned with the target schema, thereby reducing the need for repeated data cleaning.

To solve the problem of jointly repairing web extracted data and their wrappers we present WADaR, a novel system to induce regular expression-like rules to firstly repair and align data to a target schema, and then encode back such rules adjusting the wrapper where possible. Future extraction executions can therefore benefit from the repairs applied to the wrapper, thus reducing the need for a further cleaning step. We avoid the use of redundancy and we encode existing background knowledge with an ensemble of semantic annotators. We then use such annotators to (1) discover hidden attribute structures in the annotated relations through the elimination of incorrect annotations, and (2) construct relation-wide repairs that can clean both the target relation and the input wrapper.

Tools for cleaning and repairing the induced wrappers, as well as the extracted data, are still primarily manual or supervised, especially in industry.⁴ Scalable tools for continuous repair of both the induced wrappers and the generated data are still largely missing, which is the main motivation for our investigation efforts in WADaR.

Scalable Rule Discovery in Knowledge Bases. In the last part of this work we move the attention to OIE on the Web, and we focus in particular on the curation and cleaning of knowledge bases (KBs). KBs store information about real-world entities and relationships among them (Figure 1.1b), usually through RDF triples. Given their

⁴<https://www.trifacta.com/>

flexibility and applicability, KBs have recently attracted the attention of the database community, both in academia ([21, 29, 35, 135, 140]) and industry ([59, 60]). KBs are typically created by inspecting the Web as a source of facts and different candidates are combined to create extended descriptions of entities. Due to their sizes and the lack of schemas, inconsistencies [141] and missing information [60, 84] are very common. Moreover, in the era of big data, KBs pose new challenges in terms of size that classic database techniques cannot handle.

In this part we investigate the problem of discovering logical rules on RDF KBs. We present RuDiK, a scalable disk-based solution to discover first-order Horn Rules constrained by some language biases. We tackle the problem following two main novel directions. (1) We design an algorithm that is general enough to induce both *positive* and *negative* rules. Positive rules enrich the KB by discovering new true facts (e.g., if an entity a has two children, b and c , then b and c are siblings); negative rules discover inconsistencies in the data (e.g., if a person a is born after a person b , then b cannot be the child of a). Previous works on KBs deal either with the positive scenario [15, 38, 84], or the negative one [75]. (2) We propose a scalable solution that could work with any size KB with limited computing resources. Previous approaches for rule discovery in KBs have been based on the assumption that the KB can be entirely loaded in main memory. Unfortunately, modern KB sizes can easily exceed hundreds of GBs and such an assumption is not always feasible.

We will show that errors and missing information are extremely common in KBs built from web sources, and how inducing *universally correct* rules can significantly improve the quality and the coverage of the data.

1.2 Thesis Contributions

This thesis provides the following original contributions to the academic study of IE on the Web.

- (1) We deeply evaluate 11 popular semantic annotators through their web service APIs. We therefore provide a clear overview of the state-of-the-art systems in named entity recognition, including both academic and industrial services (see Chapter 2). We present experimental evidence indicating that the most widely-used individual annotators suffer from serious accuracy deficiencies, with a great variation in performance ranging from below 10% in *F-score* to above 90% (Section 2.4). We also show tremendous variation in performance depending on concepts, with no single annotator having a clear advantage overall. We will show that different concepts imply different authoritative annotators, and surprisingly many of the concepts which exhibit bad performances are not obviously the most semantically complex. We also show significant differences in vocabularies, with

high-level concepts that are supported by multiple annotators but very few specialised entities recognised by individual systems.

- (2) We present in **ROSeAnn** a fully unsupervised approach to reconcile annotators' opinions logically. Our integration method is ontology-aware – making use of an ontology that expresses semantic relationships among different concepts. The ontology was *manually built* by merging concepts from individual vocabularies, adding constraints of **subclass** and **disjointness** to capture compatibility and conflict among different entities. The final version of the ontology contains almost 400 different concepts (Section 2.2.1). The ontology is then used to spot and resolve inconsistent annotations in a weighted repair aggregation algorithm that assigns a score to annotations based on the explicit opinions of individual systems (an annotator tags a snippet with a class), implicit support (an annotator tags with a subclass), and implicit opposition (an annotator tags with a disjoint class). The approach we propose looks for an aggregation which is consistent w.r.t. the ontology and maximises the aggregated score of individual opinions. As of this writing, **ROSeAnn** is the most accurate highly automatised approach to aggregate semantic annotators (see Chapter 3). The creation of the ontology is the only human effort required, which is performed one time only by non domain experts.
- (3) We use **ROSeAnn** to replace supervision and redundancy to repair web extracted data. Inspired by knowledge-based wrapper induction that employs a set of annotators to recognise semantically related content [56, 82], we propose **WADaR**, a novel system that is capable not only of repairing data extracted from the Web, but also of repairing the original wrappers that generated that data (see Chapter 5). This has advantages such as enabling per-source (and therefore per-wrapper) repairs, while retaining a broad applicability across domains. The repair rules we induce in **WADaR** belong to the family of regular expressions, we will thus show how such rules can be directly encoded in **XPath** wrappers to produce clean data in future extractions. We propose an approach that is domain independent and wrapper generation system independent, showing an improvement in the quality of the extracted data by 15%–60%, while fully repairing wrappers in more than 50% of the cases (Section 5.5).
- (4) Despite being unsupervised, **WADaR** uses semantic annotators to identify relevant content in the extracted relations. Unfortunately, entity extractors are often noisy and incomplete (as noted also in [56, 82] and Chapter 2), we therefore propose an approach that is specifically tailored to tolerate these deficiencies by eliminating noise and ambiguity and generalising the underlying structure to cope with incompleteness. More specifically, we inspect one relation at time to identify the *most likely sequence* of attributes that appear within an extracted relation. Such

a sequence is then enforced by removing all those annotations that do not respect the induced attributes order (see Section 5.4). This decouples the quality of the repair from the quality of the annotators, as we can deliver high quality repairs even with extremely noisy annotators. We will show how decreasing the quality of involved annotators has (almost) no effect until the quality is reduced by 80-90% (Section 5.5). Furthermore, annotators can directly benefit from the identification of repeating errors, as such errors can be used directly to tune the quality of the annotations.

- (5) In the last part of the thesis, we drop annotators supervised input and we focus on discovering rules on RDF KBs. We introduce **RuDiK**, a *fully unsupervised* system that discovers first-order Horn Rules to repair and complete KBs built from the Web. To the best of our knowledge, **RuDiK** is the first system capable of discovering both positive and negative rules using the same algorithm (see Chapter 6). We propose an approach for rule discovery that targets one predicate at time, and we characterise a predicate with a set of positive and negative examples. While positive examples can be directly retrieved from the KB, negative examples are not straightforward since KBs are ruled by an *open world assumption*, where everything that is not stated in the KB is to be considered *unknown* rather than false. In such a scenario, we introduce a novel technique to generate negative examples that builds on a partial closed world assumption [60, 84]. The partial closed world assumption assumes that if the KB contains one or more object values for a given subject and predicate, then it contains all possible values. Following this assumption, given a predicate p , we consider a pair of entities (a, b) a negative example for p if two constraints are met: first, a (resp. b) is connected via p to an entity b' (resp. a') that is different from b (resp. a); second, a and b are connected via a predicate that is not p . We will show how this technique provides qualitative and representative negative examples, and how these are used to validate positive rules and to generate negative rules (Section 6.2).
- (6) Characterising a predicate with two sets of examples allows **RuDiK** to significantly shrink the search space. We present a procedure that discovers rules by inspecting just the examples provided plus those entities that occur up to a given distance from them. The problem is encoded as navigating all possible paths in a graph between a given set of pairs of entities, such that these paths are not longer than a maximum length. Such an encoding of the problem is extremely beneficial since we do not inspect the whole KB, but we retrieve connections from the KB *on demand* whenever navigating a certain path is necessary. As a result, **RuDiK** can compute rules on very large KBs in a reasonable time with limited resources (see Section 6.3). Furthermore, we propose a novel problem definition which differs from previous techniques that require threshold/support definitions [84, 142]. We

formally define the problem as a natural mapping from the well-known weighted set cover, where the goal is to find a set of rules that covers all positive examples, and as few negative examples as possible. We eventually give a logarithmic approximated solution to the size of the set of all possible rules. We will show how this formulation has the beneficial effect of limiting the output size, since RuDiK includes a rule in its solution only when it is extremely confident that the rule is well supported. This results in the discovery of a concise and non-overlapping set of rules for each predicate, with a very high accuracy (Section 6.4).

1.3 Thesis Organisation

The thesis is structured as follows: Part I presents our work on semantic annotations. Chapter 2 overviews the semantic annotators ecosystem with an in-depth evaluation of 11 different annotators. Chapter 3 presents ROSeAnn, our tool for semantic annotations integration. We include in this chapter relevant technical aspects, an extended evaluation, and we conclude with an overview of a prototype system. Part II focuses on how to repair web extracted data. Chapter 4 recalls some preliminary concepts about web data extraction and why classic cleaning approaches are not suitable, while Chapter 5 introduces WADaR. We first formalise the problem, give an approximate solution, and empirically evaluate our proposed solution. Eventually we describe the implementation of WADaR as a web platform at the end of Chapter 5. The last part of the thesis (Part III) describes RuDiK in Chapter 6. We first recall some basic preliminaries about KBs, and then describe in detail the disk-based solution we propose in RuDiK, which is then experimentally validated. Chapter 7 concludes the thesis with some future directions.

Part I

Semantic Annotations

*“Even if you are a minority of one,
the truth is the truth.”
— Mahatma Gandhi*

Chapter 2

Named Entity Recognition

In the first part of this thesis work we investigate the problem of integrating semantic annotation: we design middleware that produces a unified annotation on top of diverse semantic annotators. We will show how such a layer brings multiple benefits in terms of a richer vocabulary and more accurate annotations, and how these annotations can replace cross-site redundancy to enhance the quality of web extracted data. In this chapter, we briefly overview the state-of-the-art systems in named entity recognition, while the main technical contributions to annotation integration are outlined in the next chapter.

A growing number of resources are available for enriching documents with *semantic annotations* – annotations that label document snippets as referring to either specific entities (e.g., Barack Obama, the Mayor of Rome) or to elements of specific semantic categories (e.g., singers, commercial organisations). These resources, commonly defined as *named entity recognisers* or more simply *annotators*, are capable of recognising a sequence of tokens as belonging to an entity class of a given vocabulary, and in some cases of linking the identified entity to a real world resource (e.g., Wikipedia page). While originally focused on a few high-level standard classes of annotations (e.g., people, places, organisations), the annotator ecosystem is becoming increasingly diverse. There are high-level and specialist concepts that are supported by multiple annotators and, although their vocabularies are often quite distinct, it is possible to recognise many semantic interconnections among them. There are annotators that deal with very common entity classes (e.g., **Person**), and those that focus on very specialised entities (e.g., **NuclearWeapon**) to support applications domains such as terrorism or biomedicine [36].

Semantic annotators take as input text documents and produce various types of semantic annotations. The most common ones are named entity recognition, entity linking and disambiguation, topic classification, and relation extraction. As an example, consider Figure 2.1. The figure shows the online annotations tool powered by OPENCALAIS [7], a freely web annotations service. Once we paste the desired text into the tool, we can use the service to tag entities on the input text. Thus “*Barcelona*” is classified as **City**, “*Sagrada Familia*” as **Facility**, and “*Antonio Gaudi*” as **Person**.



Figure 2.1: OPENCALAIS Online Annotations Tool

The tool offers additional semantic features, such as topic classification and language recognition. Moreover, OPENCALAIS is also capable of identifying relations among the identified entities (not shown in the example), such as companies acquisition or entities location.

Among the available semantic features, named entity recognition is by far the most mature and popular, it is hence our focus in this work. Named entity annotations (or simply annotations) consist of a contiguous set of tokens of a document – a *span* – along with a *class* (or *entity type*, or *concept*), such as **Continent** or **City** shown in Figure 2.1. For the purpose of this thesis, we define an annotation as a pair of the form $\langle [i, j), \mathbf{A} \rangle$. The interval $[i, j)$ is the span of the annotation, meaning the annotation starts at the character i of the input text (included) and ends at the character j of the input text (excluded), with $0 \leq i < j \leq \text{length}(\text{input_text})$. $\mathbf{A}$ is the class (concept) of the annotation, coming from a given vocabulary. In Figure 2.1 the annotation on the text span “*Barcelona*” is represented by the pair $\langle [0, 9), \text{City} \rangle$.

Roadmap. This chapter is organised as follows: Section 2.1 gives a brief overview of the current available annotators, divided into two major subcategories. In Section 2.2 we propose a technique to align different annotator vocabularies, based on a manually built ontology. Section 2.3 defines a way of qualifying disagreement on annotator opinions, while Section 2.4 evaluates the accuracy of 11 existing annotators and quantifies the disagreement between these systems. Eventually Section 2.5 points out conclusions of annotators behaviour.

2.1 Annotator Ecosystem

Named entity recognisers exploit several techniques in order to produce textual annotations, ranging from natural language processing and standard regular expressions or gazetteers, to learning appropriate models from manually annotated examples. More recent approaches use ontological, background knowledge to locate the entities of interest in a text document [46, 47, 102]. The state-of-the-art annotators can be grouped into two main categories, according to the manual tuning effort required to the end user: *software framework* annotators and *web service* annotators.

2.1.1 Software Framework

Software framework annotators allow developers to design and tune entity recognisers for application-specific purposes. These softwares offer basic natural language processing (NLP) functionalities (such as tokenisation, part of speech tagging, gazetteer-based entities recognition) that can be employed and assembled together into a customised pipeline. Developers can write their own rules on top of existing ones, and the whole process is facilitated by an intuitive expressive language and by a user interface.

Among the available software frameworks, GATE [53] and UIMA [77] are the most widely used. GATE allows the development of annotators through the inclusion of gazetteer and the definition of regular expressions. The most powerful tool offered by GATE is the possibility of combining different rules, through the definition of *Jape Rules*. We can therefore import a gazetteer that contains all the currency symbols, define a regular expression that identifies numbers, and create a jape rule that extracts an annotation of type **Money** anytime that a currency symbol is followed by a number. GATE also offers an intuitive graphical interface, that can be used to test whether a generated rule is working as expected on sample documents. UIMA, other than classic NLP functionalities, contains some predefined annotators that ease the developers' job. UIMA has also some preliminary features for image and sound processing, since often the target information cannot be found on text documents.

Other framework softwares can recognise some predefined high-level concepts, without any tuning or developing effort. STANFORD NER [78], ILLINOIS NER [127] and LINGPIPE [5] are able of extracting annotation of types **Person**, **Location** and **Organization**. These tools use some forms of machine learning techniques and employ learning models that have been trained on news articles, mostly because news documents usually contain a high coverage of the target concepts.

One of the main advantage of using this kind of annotators is the possibility of exporting the extraction rules in your application. In fact most of them offer a programming API (usually Java), through which we can export the created annotators and invoke them inside our application as best as we want. The popularity of these softwares is also evident from the growing number of applications that employ them, both in industry [19, 76], and in academia [82, 132]. On the other hand, manually developing and tuning annotators is a tedious and expensive process. The process usually requires one or more domain experts, as well as the presence of positive and negative examples that can be inspected to derive extraction rules or to train suitable classifiers.

2.1.2 Web Service API

On the other side of the spectrum, web service-based annotators have become extremely popular in the last decade, as evident from a growing number of commercial compa-

nies offering such services [3, 7, 12, 13], as well as recent acquisitions (OPENCALAIS-THOMSON REUTERS, ALCHEMYAPI-IBM, SAPLO-STROSSLE). These services are annotators that may focus on a particular subset of the concepts, and usually offer a web API that can be queried with some limitations (limitations that can be removed with commercial subscriptions). Instead of being application specific “white-boxes” that can be directly modified by developers, web annotators are “black-box” services, where the user does not have to implement her own rules but, on the other hand, concept classes can be added or refined by the service host outside the control of clients. They make use of their own gazetteers and hand-tailored concept-specific heuristics, allowing these to remain proprietary, and at the same time freeing applications from having to constantly upload or update whenever the annotator is tuned. On the other hand, this “black-box” model poses new challenges, since the accuracy of the annotators, and even their vocabularies, may be highly dynamic.

Web annotators offer named entity recognition functionalities via a web service, implemented as a REST API. APIs can be queried specifying a various number of parameters (including the text document to be analysed), and return an answer (typically a JSON object) containing semantic annotations identified on the input document. ALCHEMYAPI [3], a company that uses deep learning for entity recognition and recently acquired by IBM, allows users to specify several parameters in the input request, including: (i) *text*, the input text document to be analysed; (ii) *outputMode*, the desired response format (e.g., JSON, XML, RDF); (iii) *maxRetrieve*, maximum number of entities to retrieve; (iv) *sentiment*, whether to enable entity-level sentiment analysis. Figure 2.2 shows a (JSON) response of ALCHEMYAPI on the input text “Barack Obama is the new president of United States.”. The response shows some general information about the input document (A – language, input text, response status), the set of identified entities (B), and additional attributes for each entity (C). Attributes include the type (concept) of the entity, possible subclasses of the main type, the entity relevance, the sentiment associated to the entity, the number of instances, the text span, and some other optional attributes such as DBpedia links. ALCHEMYAPI

```

{
  "status": "OK",
  "language": "english",
  "text": "Barack Obama is the new president of United States.",
  "entities": [
    {
      "type": "Person",
      "relevance": "0.33",
      "sentiment": {
        "type": "neutral"
      },
      "count": "1",
      "text": "Barack Obama",
      "disambiguated": {
        "subType": [
          "Politician",
          "President"
        ]
      },
      "dbpedia": "http://dbpedia.org/resource/Barack_Obama"
    },
    {
      "type": "JobTitle",
      "relevance": "0.33",
      "sentiment": {
        "type": "neutral"
      },
      "count": "1",
      "text": "president"
    },
    {
      "type": "Country",
      "relevance": "0.33",
      "count": "1",
      "text": "United States"
    }
  ]
}

```

Figure 2.2: ALCHEMYAPI Web Service Response

limits requests to 1,000 daily transactions, however this limit can be upgraded (until unlimited transactions) through commercial subscriptions.

Web-based annotators can only be used as external tools to plug in, and they cannot be adapted to user-specific needs. The vocabulary is fixed and cannot be extended with custom entities, as well as the internal implementation is unknown, therefore it is impossible to tune these systems to tackle particular scenarios. Moreover, being controlled by proprietary companies, the performance and the vocabulary are very dynamic and may change outside the control of clients. Despite this, web-based annotators have many practical advantages, above all they do not need any manual tuning nor implementation. In this thesis we aim at minimising manual supervisions and user efforts, thus we will target web service-based annotators.

2.2 Vocabulary Alignment

Web annotators come in many forms, especially w.r.t. their vocabularies. The vocabulary is the set of concepts that a given annotator is capable of extracting on a given text. Despite the differences in vocabularies, with both high-level and specialist concepts, annotators also have many semantic interconnections. Concepts coming from different vocabularies may seem unrelated, however they might share common semantic meanings. Consider the example shown in Figure 2.3a, a snippet of text from a news article, mentioning an American naval base in the Philippines and classified by four web-based annotators. Annotations include some high-level concepts (i.e., **Location**, **Person**), as well as some more specific ones (i.e., **Facility**). Notice that the annotation of type **City** is only returned by CICEROLITE [4]: indeed, many annotators simply cannot identify this concept because they do not have it in their vocabularies. Also note that annotators have different classification opinions on the same text fragment – one annotator classifies “*Subic Naval Base*” as **Location**, while others classify it as **Facility**. The figure exemplifies disagreement among annotators (clearly the same text snippet cannot be a facility and a person at the same time), as well as hidden semantic

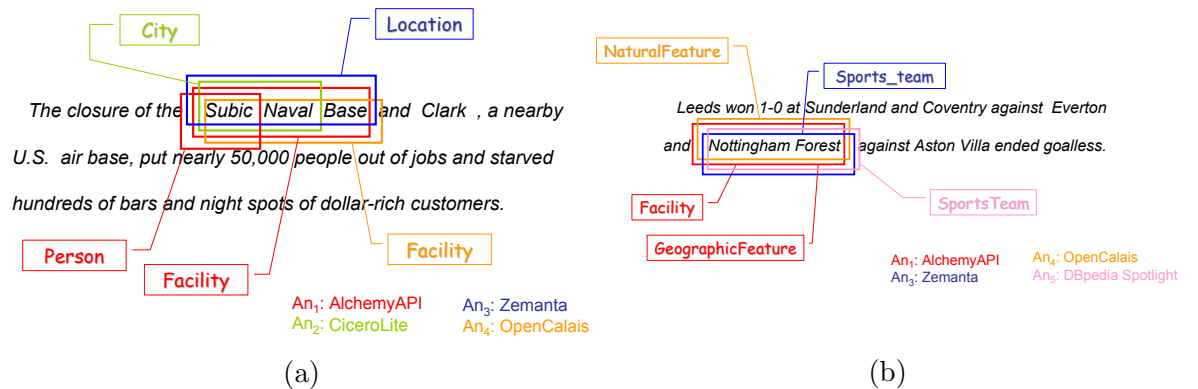


Figure 2.3: Web Annotators Disagreement and Semantic Interconnections

interconnections (the concepts **Location** and **City** are clearly not in contrast with each other, but one identifies a more fine grained entity than the other). Another example of annotators disagreement is shown in Figure 2.3b. Here we see several scenarios of differences (and agreement) in annotator opinions. “*Nottingham Forest*” is labelled as **NaturalFeature** by OPENCALAIS, and as **GeographicFeature** by ALCHEMYAPI – clearly these outputs are compatible, since these two concepts have the same meaning. On the other hand, the concepts **Facility** and **SportsTeam** are both incompatible with the prior classifications, as well as with each other. In this example we notice not only inter annotator differences, but also intra annotator discrepancy: ALCHEMYAPI labels the same text snippet with two different concepts.

Figure 2.3 shows the main challenges of dealing with annotators having different vocabularies: concepts with the same meaning might have different names in different vocabularies (**GeographichFeature** and **NaturalFeature** have the same semantic), two annotators might label the same text snippet with different levels of granularity (**City** is a more specific class of **Location**), two different concepts may still be compatible (a person can be at the same time both a **Politician** and an **Actor**), and eventually some concepts may be in contrast with each other (“*Nottingham Forest*” is either a **SportsTeam** or a **NaturalFeature**, it cannot be both). On top of these scenarios, vocabularies have different coverages – a class **C** might be identified only by a subset of the involved annotators. Some annotators, such as OPENCALAIS, have mostly high-level concepts in their vocabulary – e.g. **Organization**; others such as CICEROLITE, are capable of a much finer classification – e.g. **NonProfitOrg**. One of the visions of this work is to exploit semantic annotations provided by several web services. In order to do so, we need to define a way to compare and align different vocabularies.

2.2.1 Ontology

As shown in the examples above, web annotators do not have a flat vocabulary – vocabularies have semantic interconnections, as well as semantic conflicts. To capture the relationships in different vocabularies, we define an ontology Ψ which states the relationships between entities from different vocabularies. The ontology needs to capture the general-specific entities relationship, as well as the compatibility of different entities. In this thesis, an ontology consists of a finite set of concepts (classes), including the two special types $\perp$ (empty concept) and $\top$ (universal concept), along with constraints restricting or enforcing the co-occurrence of concepts. Our ontology supports three types of constraints:

- Equivalent constraints $C = D$, stating that an entity of type **C** is an entity of type **D** and vice versa (**GeographicFeature** is a **NaturalFeature**, as well as **NaturalFeature** is a **GeographicFeature**).

- Subclass (subsumption) constraints $C \sqsubseteq D$, stating that an entity of type C is also an entity of type D (**City** is also a **Location**).
- Disjointness constraints $C \sqcap D \sqsubseteq \perp$, stating that C and D have an empty intersection (**SportsTeam** and **NaturalFeature** do not have any common instances).

The constraints defined above are enough to capture compatibility and disagreement among concepts: if two concepts are in a disjoint relationship, then these concepts are incompatible. If two concepts are not in a disjoint relationship (they are either in a subclass or equivalent relationship or they are not tied by any relationship), then the two concepts are compatible and the same snippet of text could be labelled with both of them. The constraints defined in our ontology are supported by the major semantic web languages, such as OWL or its fragment OWL2-QL.

Some annotators already use an external ontology – DBPEDIASPOTLIGHT [114] is based on DBpedia and ZEMANTA [13] partially covers concepts in Freebase and DBpedia. Other annotators do not come with any hierarchy of concepts, therefore the meaning of the concepts had to be recovered from the documentation. In some cases concepts could be mapped to Freebase or DBpedia concepts, but in other cases we had to expand our ontology introducing more classes. We believe that as these annotators gain popularity, they will increase the use of standardised vocabularies, making the use of rules as “hard facts”, as we do throughout this thesis, appropriate. A first step in this direction is the SCHEMA.ORG¹ initiative. We are focusing in this thesis on *exploiting* annotator outputs to recover errors on web extracted data, not on the very different problem of mapping/aligning concepts and ontologies (in other terms, “exploiting the ABox”, not “aligning the TBox”). In other words, this thesis focuses on using semantic annotations that work on top of a global ontology, and it is not a novel technique to merge and align existing ontologies. We have thus created a merged ontology manually, aligning the relevant fragments of Freebase and DBpedia within it, expanding with rules that are clear from the documentation and by validating them against SCHEMA.ORG knowledge base when applicable. Our starting point is to import constraints from DBpedia and Freebase, whenever possible, and then to refine the ontology by adding new concepts introduced by the annotators, along with new constraints affecting these concepts. DBpedia and Freebase contain only subsumption relations, while disjointness relationships are essential for determining whether two annotations are compatible. Therefore we manually added disjointness relationships by inspecting the documentation and examples of annotations.

In this work we created a global ontology that merges concepts from 11 web annotator vocabularies. For each concept in each vocabulary, we added a class in our ontology (with a suitable namespace identifying the original annotator). We imported subclass relationships from DBpedia and Freebase for each concept that could be mapped to one

¹<http://schema.org>

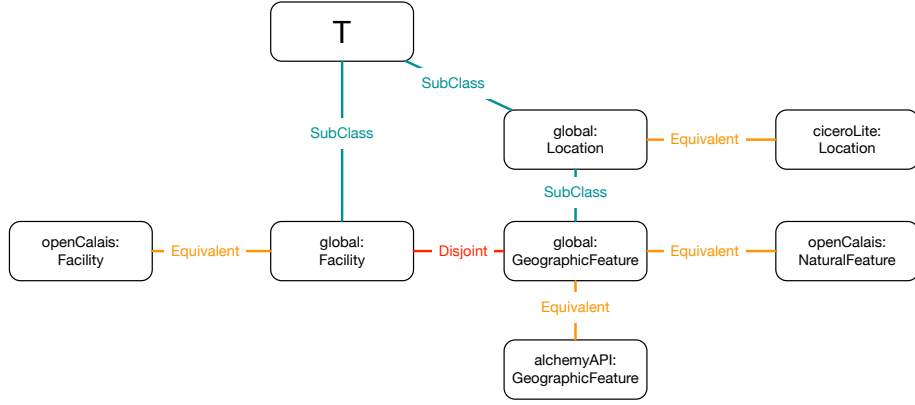


Figure 2.4: Example Snippet of Merged Ontology

of these ontologies, and we added further subclass relationships and disjoint relationships by inspecting the documentation of the annotators. For each group of concepts representing the same semantic entity, we added equivalent constraints and we introduced a *global* concept that represents the semantic entity. Figure 2.4 gives a snippet of our merged ontology that involves four concepts imported from **ALCHEMYAPI**, **CICEROLITE** and **OPENCALAIS**. For each concept we created a new entity class in the ontology, and a corresponding global concept is created for each semantic entity. The subclass relationship between **GeographicFeature** and **Location** comes from DBpedia, while the other relationships have been manually inserted. We mapped the concepts **GeographicFeature** from **ALCHEMYAPI** and **NaturalFeature** from **OPENCALAIS** as equivalent classes of a global concept **GeographicFeature**, while the concept **Location** from **CICEROLITE** is mapped as a superclass, and the concept **Facility** from **OPENCALAIS** as disjoint class. The full merged ontology described in this thesis is accessible online.² It has been created with the Protégé³ ontology editor tool and it has been published according to the standard OWL-RDF language. From now until the rest of the thesis, whenever we refer to a class **C** of the ontology, we refer to the class named **C** and having the global prefix.

2.3 Conflicts

The merged ontology is a means for comparing semantic annotations coming from different annotators. As shown in Figure 2.3, annotator disagreement is common and very rarely two different annotators label exactly the same text snippet with the same concept (see Section 2.4.4 for a detailed analysis). Moreover, even if two concepts have the same name, it does not necessarily imply that the two concepts have the same semantic. On the other hand, the presence of annotations with different labels does not always mean disagreement, as two classes could be in a equivalent or subclass

²<http://diadem.cs.ox.ac.uk/diadem/roseann/MergedOntology.owl>

³<http://protege.stanford.edu/>

relationship. Furthermore, disagreement may happen at span level rather than at semantic level: Figure 2.3a not only shows discrepancy in semantic, but also shows annotations having different spans.

A question we want to answer is to what extent the semantics of the annotations can be useful to determine errors in the annotations. In order to qualify annotators disagreement and detect potential errors in the annotated spans, we define *annotator conflicts*. In particular, we consider a conflict whenever two annotations from two different annotators either do not agree on the annotated concepts or on the annotated spans. Based on the semantic dimension and on the span dimension, we define four different types of conflicts according to our merged ontology Ψ . Figure 2.5 summarises the four types of annotator conflicts.

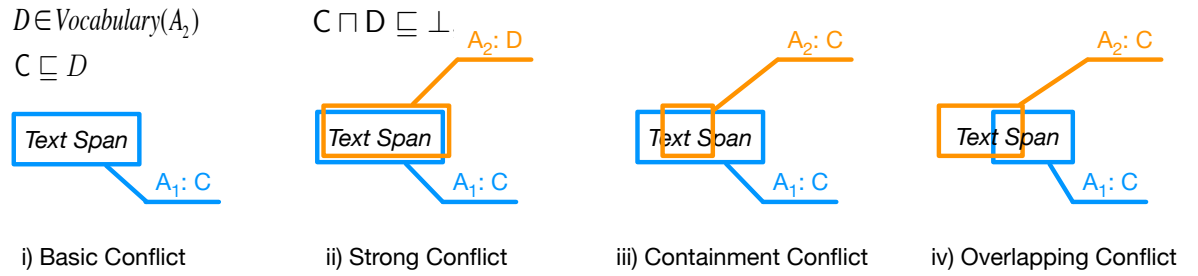


Figure 2.5: Types of Annotator Conflicts

Logical Conflicts. The first type of conflicts captures incoherences among annotation concepts. We distinguish two different logical conflicts.

- i) **Basic Conflict.** A basic conflict occurs when one annotator annotates the span with a concept C , and another annotator which has (a superclass of) C in its vocabulary fails to annotate the same span with it. Clearly, this represents a situation where the annotators have differing opinions. In particular, basic conflicts are useful to identify low recall in annotators, since they highlight *missing* annotations. For annotators with low recall, a basic conflict may be a weak indicator of a problematic annotation. On the other hand, if the annotator that did not label the span is an expert on that specific concept, a basic conflict may be a signal that the found annotation is wrong, improving the precision.
- ii) **Strong Conflict.** Strong conflicts denote situations when two annotators annotate the same span with disjoint concepts C and D . Differently from basic conflicts, a strong conflict aims to improve annotators precision, since C and D are disjoint classes, one of them must come from a wrong annotation (if not both).

Span Conflicts. The second type of conflicts captures instead disagreement among annotation spans. Also here we have two different span conflicts.

- iii) **Containment Conflict.** A containment conflict happens when two annotations agree on the concept, but the span of one annotation is strictly contained in the span of the other annotation. Formally, given two annotations $\langle A, [i, j] \rangle$ and $\langle A', [i', j'] \rangle$, we say that they are in a containment conflict iff $A = A' \wedge i \leq i' \wedge j' \leq j \wedge (j - i) > (j' - i')$.
- iv) **Overlapping Conflict.** Two annotations $\langle A, [i, j] \rangle$ and $\langle A', [i', j'] \rangle$ are in overlapping conflict iff $A = A' \wedge i < i' \wedge j > i' \wedge j < j'$. In other words, overlapping conflicts refer to annotations having the same concept but overlapping spans, where spans are not strictly contained.

Conflicts can be used as a signal to verify the correctness of annotations. In the next chapter we will show how this evidence can be used to detect erroneous annotated spans, and how to use such information to semantically reconcile inconsistent (w.r.t. the ontology) annotated spans. In the next experimental Section we will show that the conflicts defined above are the only ones occurring with web-based annotators.

2.4 Annotator Evaluation

In this work we target semantic annotations coming from black boxes web-based annotators. One of the first question we aim to answer is how *good* these services are. In particular we want to assess the performance of individual annotators both in terms of precision (percentage of correct annotations over all the retrieved annotations), and in terms of recall (percentage of correct annotations over all possible correct annotations). The evaluated systems use various techniques for named entity recognition, such as deep learning, heuristic rules or neural networks. We evaluated 11 different web annotators without considering the internal implementation, just observing the “external behaviour”.

2.4.1 Dataset

In order to evaluate the performance of those individual annotators, we conducted experiments over four benchmark corpora:

1. The **MUC7** dataset [1], which contains 300 newswire articles for evaluating named entity recognition, annotated with standard high-level concepts such as **Person**, **Organization**, **Location** and **Money** – the corpus comes in form of XML documents, where each annotation is identified by a specific tag opened at the beginning of the annotated span and closed at the end of the annotated span.
2. The **Reuters** corpus (RCV1) [2], a general information retrieval corpus of English newswire documents, subdivided into topics.

3. The corpus used by the **FOX** [138] entity extractor and consisting of 100 short texts from news headlines annotated with three concept classes, namely **Person**, **Location**, and **Organisation** – the documents in this corpus are rather short (not more than 3 sentences), however they are very dense with several annotations per document.
4. The corpus used by the **NETagger** [128] entity extractor and consisting of text sampled from 20 web pages mostly about academics and annotated with the same three concepts of the **Fox** corpus plus a **MISC** concept for annotating miscellaneous concepts. We do not perform evaluation on the **MISC** concept due to its ambiguity. For instance, the string “Computer Science” is labelled as **MISC** in the corpus, but other similar strings, e.g., “Biomolecular Computation”, are not. This corpus also allowed us to see the performance of the annotators in a domain other than news.

MUC7, **Fox** and **NETagger** corpora consist of a set of documents, together with a set of annotations coming from a high-level vocabulary (these vocabularies do not contain fine grained concepts of our merged ontology). Since **Reuters** corpus does not come with any annotation with it, we decided to use this corpus and annotate it with concepts from our ontology. The original **Reuters** corpus is very large (810,000 articles), we chose five of the most common topics – *Entertainment&Sports*, *Financial&Economics*, *Healthcare&Social*, *Products* and *Tourism&Travel* – and randomly sampled 250 documents, with 50 documents per topic. For this subset of **Reuters** documents, we manually tagged the texts by using the most specific classes from the global ontology that we introduced before. The manual labelling has been conducted by three different persons, and we used a majority voting strategy to solve eventual conflicts on same text spans. During manual annotation we eliminated a few concepts whose meaning was unclear from the documentation, such as **FieldTerminology** and **IndustryTerm**. The documentation for these concepts is not specific, and we could not find patterns in inspecting example annotations labelling these concepts.

Details on the coverage of the testset are shown in Table 2.1 (Entities indicates the total number of annotations in the corpus). Our gold standard annotation of the **Reuters** corpus is available online from [8]. Our merged vocabulary contains almost 400 concepts, however we found instances of just 215 concepts in the **Reuters** corpus and we could not evaluate the systems on the remaining concepts (the remaining concepts come only from the vocabulary of **CICERO**LITE and are very specific concepts such as **ChemicalWeapon**). Note that for the other corpora, we directly use the original gold standard provided with each corpus.

Table 2.1: Description of Testing Corpora

Test Corpus	Documents	Covered Types	Entities ($\approx$)
MUC7	300	7	18,700
Reuters	250	215	51,100
Fox	100	3	395
NETagger	20	3	624

MUC7 and **Reuters** have been chosen because they are benchmark corpora in information retrieval and named entity recognition. Furthermore, since **Reuters** was not originally annotated, we could chose the granularity of the annotations and we could include specific concepts from our merged ontology. **Fox** and **NETagger** corpora have been included primarily for comparative purposes against systems that aggregate semantic annotations (see Chapter 3 for more details), however their diversity w.r.t. **MUC7** and **Reuters** allowed us to see the behaviour of the annotators both on a different domain and on different documents sizes (**Fox**’s documents are much shorter than the others).

2.4.2 Precision and Recall

Since our gold standard has been created with the most specific concepts of our merged ontology, we needed a way to perform an evaluation that takes into account both the vocabulary coverage, and also the granularity of the gold standard annotations. Our merged ontology can help us in defining new ways of computing precision and recall, to take into account concepts hierarchies. Suppose the text “Mick Jagger” is annotated as **Artist** in our gold standard, and an annotator labels that text snippet as **Person**. Clearly, an artist is also a person, therefore the annotation **Person** is not wrong. On the other hand, if the annotator contains the concept **Artist** in its vocabulary, this situation should be counted as a miss for **Artist** since the annotator fails to recognise the snippet with that concept (if the annotator’s vocabulary instead does not contain **Artist** then the annotator should not be penalised since it is incapable of identifying that concept).

We measure *Precision* and *Recall* in a way that is *ontology-aware*: for example, given our ontology Ψ , if an annotator declares a given snippet to be a **Person** while our gold standard states that it is an **Artist**, then this annotator will be penalised in recall for **Artist** (since it missed its identification), but not in precision for **Person** (since **Person** is a superclass of **Artist** and therefore an artist is also a person). As far as we know, there is no a standard solution to compute precision and recall in this way. Euzenat [68] has introduced semantic precision and recall measures in the area of ontology alignment – but the goal there is to assign static measures to an alignment, which approximates the precision and recall one would get from an “ideal” solution of the problem. In our case, we can estimate the precision and recall by comparing two sets of instances.

More precisely, the precision of an annotator AN for a concept C is defined as:

$$Precision_{\Psi}(C) = \frac{|Inst_{AN}(C^+) \cap Inst_{GS}(C^+)|}{|Inst_{AN}(C^+)|}$$

with $Inst_{AN}(C^+)$ denoting all instances annotated as (a subclass of) C by AN , and $Inst_{GS}(C^+)$ denoting all instances determined to be (a subclass of) C in the gold standard. Therefore, a span annotated as a **Person** will increase the precision of **Person** if the gold standard labels the same span as a subclass of **Person** (e.g., **Artist**).

In computing the intersection of instances, we use the notion of “relaxed” span matching, which requires only the spans overlapping (i.e., two spans match if they share at least one character). We use a relaxed matching mostly because we are interested in evaluating the accuracy of the semantic, rather than the accuracy of the span. Here and throughout the remainder of the thesis, when we mention that the concept A is a subclass of a concept B , we mean that our merged ontology states that A is a subclass of B , possibly improper (that is, A may be equivalent to B). Similarly, when we talk about superclasses of a concept A , we include all the superclasses of A derivable with the ontology, plus the concept A itself.

The recall of an annotator AN for a concept C is defined in an analogous way, using the “relaxed” notion of matching for the intersection also in this case:

$$Recall_{\Psi}(C) = \frac{|Inst_{AN}(C^+) \cap Inst_{GS}(C^+)|}{|Inst_{GS}(C^+)|}$$

In the example of **Artist** in the gold standard and an annotation of type **Person** defined above, the recall of **Artist** is penalised by the annotation while the recall for **Person** is increased since an artist is also a person.

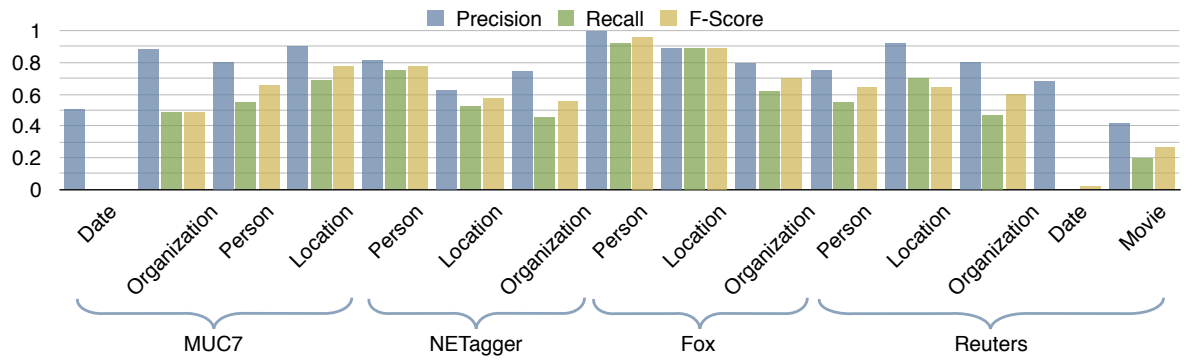
We define the F -score for a concept C and an annotator AN in the standard way, using the measure of precision and recall defined above:

$$F\text{-score}_{\Psi}(C) = \frac{2 \times Precision_{\Psi}(C) \times Recall_{\Psi}(C)}{Precision_{\Psi}(C) + Recall_{\Psi}(C)}$$

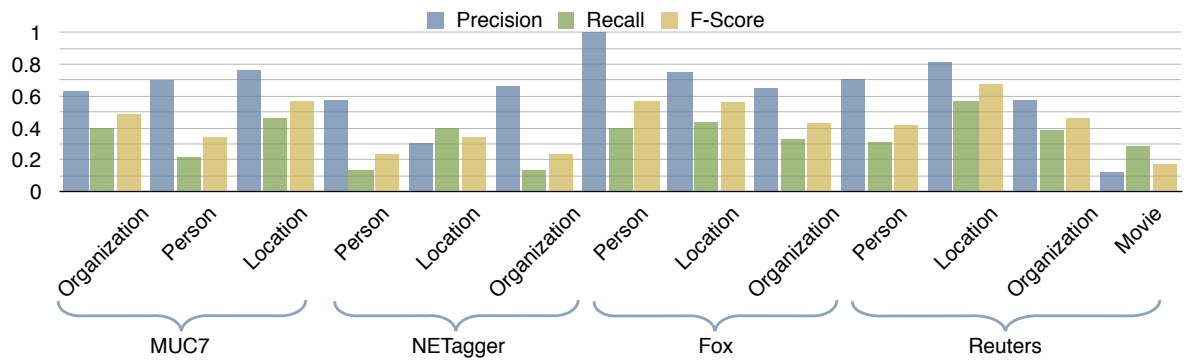
We compute $Precision_{\Psi}$, $Recall_{\Psi}$ and $F\text{-score}_{\Psi}$ at *attribute-level*, i.e., we compute the three measures for each concept C in the vocabulary. To be fair against the evaluated annotators, we evaluate each annotator on its own vocabulary. For each annotator, we compute $Precision_{\Psi}$, $Recall_{\Psi}$ and $F\text{-score}_{\Psi}$ for each concept C in the vocabulary of the annotator. If an annotator does not contain **Artist** in its vocabulary, then it will not be penalised for not identifying that concept.

2.4.3 Results

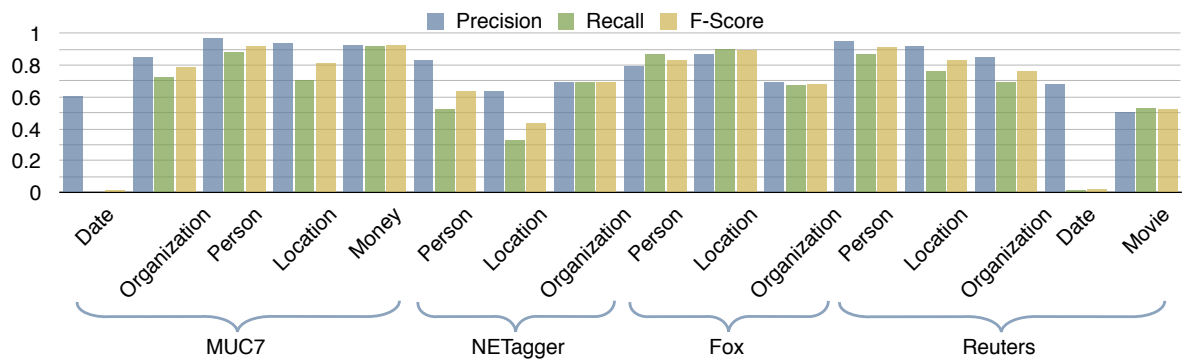
Our evaluation tested each of the 11 systems over the datasets described above using the merged ontology Ψ as a reference schema. We computed Precision, Recall and F -score via an ontology-aware scorer. We present selected results in Figure 2.6. Here we present



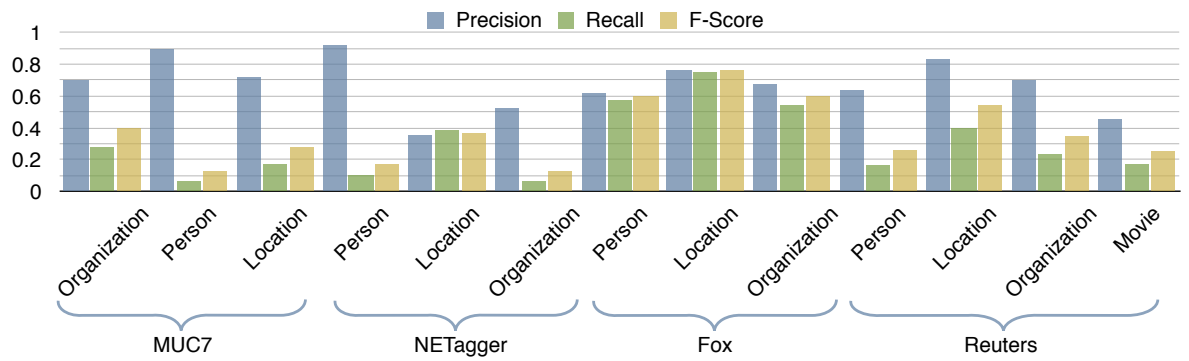
(a) ALCHEMYAPI



(b) DBPEDIA SPOTLIGHT



(c) OPENCALAIS



(d) ZEMANTA

Figure 2.6: Performance of Individual Annotators

only the most relevant results, coming from 4 (out of 11) annotators and a subsets of the concepts of the merged ontology. We found similar results with other annotators and other concepts, and the full set of results including all annotators evaluated on the entire ontology can be inspected at the online appendix [8]. The online appendix also contains the source code of the ontology-aware scorer, as described in Section 2.4.2.

For each annotator A_N , results include all of the **MUC7**, **Fox**, and **NETagger** concepts that are in the vocabulary of A_N . For each corpus, we first mapped the annotations of individual annotators into entity types in the corpus and then computed the measures (i.e., if A_N returns an annotation of type **Artist** and the corpus gold standard contains only **Person**, **Artist** is mapped to **Person**). For the **Reuters** dataset we present a subset of concepts according to the following principles: **(i)** concepts with sufficient support (we filtered down to concepts with instances occurring in all of the five topics and with at least 10 different occurrences;) **(ii)** concepts with diversity in performance (for each annotator A_N , we chose the first half of the concepts with best F -score and the last half of the concepts with worst F -score); **(iii)** concepts with diversity within the ontology (within the subsets above, we deleted some concepts to avoid having more than two subclasses of any given concept). Some of the selected concepts are not reported for some of the annotators because their vocabularies do not contain those concepts (**CICEROLITE** is the only one evaluated on **Percentage** from **MUC7** because it is the only annotator able to recognise percentages, the others do not have the concept in their vocabularies).

The figure shows great variations in performance for each annotator, ranging from below 10% to above 90% in F -score. If we focus on concepts that can be identified by multiple annotators, we also see tremendous variation in performance, with no single annotator having a clear advantage above others. **OPENCALAIS** has a higher performance on most of the concepts from its vocabulary, but it has a substantial deficit on a few shared concepts, and on many important classes, such as **Date**, it can identify only few subclasses (according to its documentation, **OpenCalais** can identify only dates that represent holidays, and not any possible date). Also, **OpenCalais**'s vocabulary is quite limited, since it contains only 39 concepts (against almost 300 recognised by **CICEROLITE**). If we inspect the performance within an annotator, also here differences are quite dramatic, and surprisingly many of the concepts which exhibit very low performances are not the most semantically complex. As an example, **Movie** would seem something easy to recognise by using an online reference such as **IMDB**, but the recall numbers for **DBPEDIASPOTLIGHT**, **ALCHEMYAPI**, and **ZEMANTA** were only 20%, 28%, and 17% respectively. Also, performances on very standard concepts, such as those in **MUC7**, were generally higher than the ones on more specific concepts. However the results of **ZEMANTA** show that this is not always the case.

Overall, **ZEMANTA** and **DBPEDIASPOTLIGHT** show the worst performances, also in those generic concepts where the other annotators perform fairly well (e.g., **Person**). If

we analyse performances over each corpus, every annotator performs better on the **Fox** corpus. Indeed the **Fox** corpus contains only 3 high-level concepts and the instances of these concepts are very popular and easy to recognise – for instance, the instances of **Person** are mostly politicians that can be found in many newswire documents. The performance on this corpus also highlights that the length of the document is not important. Usually web annotators do not use the context of the document to extract entities, therefore the length of the document does not impact on the quality of the annotations.

As we can observe from the results, none of the annotators is an expert on every topic of its vocabulary. Each annotator may perform very well on some concepts (**OPENCALAIS** and **CICEROLITE** are quite expert in recognising persons), but performances may drop significantly on other concepts (**OPENCALAIS**’s performance is almost zero in **Date**, while **CICEROLITE** is not able at recognising **Time** with the same quality of other concepts). Another interesting observation is that all annotators contribute to recall, i.e., each annotator contributes some annotations that are not produced by any of the others. This is true both at instance level, where each annotator capable of identifying a concept **C** contributes with instances of **C** that other annotators do not identify, and at attribute level, where each evaluated annotator contains one or more concepts that are not recognised by any of the other annotators (except **STANFORD NER** and **ILLINOIS NER** that are capable of identifying only **Person**, **Location** and **Organization**).

2.4.4 Observed Conflicts

In Section 2.3 we introduced the concept of annotator conflicts. As we can observe from the evaluation results, annotators can vary radically in accuracy and a second question we want to answer is to what extent the semantics of the annotations can be useful to determine erroneous concepts. We use annotator conflicts as a strong signal for errors in annotations.

Table 2.2: Conflicts Statistic of Individual Annotators

Test Corpus	Annotated Span	Basic Conflicts	Basic Span	Strong Conflicts	Strong Span
MUC7	30,689	36,756	26,174	3,501	2,483
Reuters	29,162	21,639	15,654	2,937	1,981
Fox	798	943	605	185	68
NETagger	1,493	1,486	1,195	179	83
Total	62,142	60,824	43,628	6,802	4,615

Table 2.2 reports the number of basic and strong conflicts in each datasets. The table contains both the number of annotated spans that contain at least one type of conflict (basic and/or strong), and the total number of conflicts for each type. An

annotated span is a span that contains at least one annotation, while a basic span (respectively a strong span) is a span that contains at least one basic conflict (respectively strong conflict). Note that the same span may contain multiple basic and/or strong conflicts, since a conflict involves only two annotations.

As introduced in Section 2.3, we also track conflicts in spans, e.g. annotators agreeing on a concept, but disagreeing on the text snippet. Table 2.3 reports, for each corpus, on the number and type of span conflicts, in particular: **(i)** the total number of annotations, **(ii)** the number of annotation pairs with same span, **(iii)** the number of containment conflicts (annotations having one span strictly contained into the other), **(iv)** the number of overlapping conflicts (annotations having overlapping spans but not strictly contained), and **(v)** the total number of conflicting spans (i.e., either contained or overlapping).

Table 2.3: Span conflicts.

Test Corpus	Total Annotations	Same Span	Containment	Overlap	Conflicts
MUC7	96,111	205,289	43,262	958	44,220
Reuters	87,666	198,745	37,743	898	38,641
Fox	2,730	8,107	2,012	38	2,050
NETagger	5,430	7,029	4,719	263	4,982
Total	191,937	419,170	87,736	2,157	89,893

Although one might expect that conflicts are rare, the results show that they occur very frequently. A chief source of conflict is that annotators usually do not use context to disambiguate. “Reuters” sometimes refers to a news company, and sometimes to a person; “Jaguar” can refer to an animal or to a car; “Tottenham” can refer to a London borough, a train station, or a sports team. Annotators usually label a text snippet by just inspecting the snippet itself, without considering the surrounding context. Moreover, the number of conflicts is limited by the small overlap in the vocabularies and the low recall of the annotators. For example, it is very unlikely for three annotators to simultaneously annotate the same span, and therefore it is very rare to have three annotations being mutually in strong conflict. For the conflicts in span, we observe that most of the conflicts are due to strictly containment, and very rarely to overlapping. This is because some annotators tend to favour *long* annotated span, while others tend to prefer *short* ones (“President Barack Obama” is annotated as **Person** by some annotators, while others annotate only the span “Barack Obama” as **Person**).

2.5 The Need for Integration

Overall, the results show both the need for annotator integration and the possibility of using conflict and agreement signals in constructing superior aggregate annotations

($\sim 25\%$ average improvement in accuracy). The results are *not* meant to give a general picture of who is competent on which concepts or who is more trustable. Results may vary significantly on different datasets; furthermore, the internal implementation of the annotators is frequently modified, with the modification impacting both their vocabularies and their performance. Thus the goal of the next chapter will not be to come up with an aggregation algorithm that solves conflicts on this particular dataset (e.g., by identifying which annotator is more reliable on which concepts), but to come up with a general dataset-independent approach for aggregating semantic annotations, without relying on hand-crafted rules or heuristics.

Both the overlap and the diversity in annotator vocabularies motivate the need for semantic annotation integration: *middleware* that produces a unified annotation on top of diverse semantic annotators. On the one hand, the diversity of vocabulary allows applications to benefit from the much richer vocabulary available in an integrated vocabulary. For example, recent work on automated wrapper induction (e.g., [56, 134]) uses semantic annotations to lower the need for human interaction. On the other hand, annotation integration can also have an impact in terms of quality of the result. We will show that aggregating annotators brings benefits both in annotators recall (as different annotators contribute with some annotations that are not produced by others), and in annotators precision (as conflicting signals help in deleting incorrect annotations). In the next chapter we present an aggregation algorithm that resolves conflicting spans by inducing annotations that are *logically consistent* w.r.t. our merged ontology, i.e., it deletes and adds concepts to annotated spans in order to eliminate conflicts.

Chapter 3

ROSeAnn: Reconcile Opinions of Semantic Annotators

In this chapter we present an approach to *reconcile logically inconsistent semantic annotations*. One major sign for erroneous annotations is a logical conflict (basic or strong): two annotations occurring in a strong conflict imply the incorrectness of one of the annotations, while a basic conflict implies either an erroneous annotation or a loss in recall by one or more annotators. Our approach targets primarily logical conflicts, and will try to eliminate annotations (as well as introducing new annotations) in order to resolve all the logical conflicts within a document, thus making the document logically consistent w.r.t. the ontology Ψ . We will also present some basic heuristics to solve span conflicts. Since most of the problems are due to logical conflicts (as pointed out in Section 2.4.4), our efforts concentrate mostly on repairing those.

Our approach to integration exploits several unique aspects of semantic capabilities provided by web annotators. A first distinctive aspect is the space of concepts annotated in document snippets: annotations do not form a flat space, but rather identify a meaning for the snippet within an highly hierarchical space — as previously shown, where two annotations with different concepts may thus still be correct. We utilise ontological rules as a precise means of capturing compatibility or incompatibility of the opinions given by different annotators. Therefore we propose integration methods that are *ontology-aware* – making as much use as possible of available semantics relationships.

A key ingredient of our approach is the black-box model used for the annotators. An approach based on hand-defined integration rules would be problematic in this scenario, since we may notice certain patterns of errors at one point, and later such patterns may change as the underlying algorithms are modified. Thus our main goal is to achieve aggregation methods that are as *domain-independent* as possible, and do not rely on hand-crafted heuristics, treating each annotator as an unknown “black-box” that may change both in the vocabulary and in the extraction patterns.

Roadmap. In this chapter we first introduce an unsupervised approach that exploits semantic interconnections expressed in our ontology to solve strong conflicts

(Section 3.1), along with a practical implementation and simple heuristics on how to deal with span conflicts. Section 3.2 then introduces a supervised algorithm, that will provide a comparative upper-bound. We experimentally validated our unsupervised approach both against a supervised algorithm, and against two available systems that aggregate semantic annotations (Section 3.3). Eventually Section 3.4 describes ROSeAnn, an available system that implements our approach, while Section 3.5 reviews some of the related works in semantic annotations and aggregating annotator opinions. We conclude the chapter with Section 3.6, that introduces how semantic annotations can be used as an alternative to redundancy for cleaning web data.

3.1 Unsupervised Aggregation

In this section we introduce an aggregation approach that requires no supervision and no information on the reliability of the base annotators.

Aggregation of conflicting data in the presence of integrity constraints is a long-standing topic in AI, semantic web, and database research. In the case where the input consists of a set of opinions from multiple “experts”, there are a number of *judgement aggregation* methods that have been studied – for example, based on variations of *voting* [96]. These algorithms have been examined not only from the point of view of guaranteeing consistency, but from the perspective of social choice theory – e.g. can they be considered to be fair to the parties involved ([91]). When, on the contrary, the input is seen as a single inconsistent dataset, a common approach in the database and semantic web community is based on *repair*: finding a “minimal” (in some metric) set of update operations to the initial data so that the updated data is consistent [131].

3.1.1 Atomic Score

Our unsupervised solution will use ideas from both voting and repair. For the moment we focus on conflicts occurring on the same span, without considering conflicts in spans (we will come back on span conflicts later on in this chapter). The input for our repair procedure is a single sequence of tokens entirely annotated with different concepts by multiples annotators. We call the input tokens a single span $\hat{s}$ that is tagged by several annotators. In this case, concepts can be identified with atomic propositions, and the ontology relationships can be considered as propositional constraints – e.g. if concepts C and D are disjoint, our logical theory includes the constraint $C \rightarrow \neg D$. Thus we can translate axioms of the ontology Ψ into a propositional theory T_Ψ . One component of our solution is a voting-based *AtomicScore* function $\text{AtomicScore}(C)$, assigning an integer to each concept C . This represents the degree of support for or opposition to C by annotators. We will always consider an annotator to support a concept C if it tags the span $\hat{s}$ with a subclass of C – each such supporter will add to $\text{AtomicScore}(C)$. $\text{AtomicScore}(C)$ will drop for each annotator that tags $\hat{s}$ with a class disjoint from C . We

allow in addition the possibility of “opposition by omission” – $\text{AtomicScore}(C)$ drops when an annotator fails to tag $\hat{s}$ with a superclass of C that is in its vocabulary. Such a penalty is justified under the assumption that annotators have *high recall* (i.e., they rarely miss on identifying concepts that are in their vocabularies). If, on the contrary, annotators have low recall, then the penalisation for omission may be too strong, since missing annotations occur too frequently. The formal definition of $\text{AtomicScore}(C)$ for a concept C is given below.

$$\begin{aligned} \text{AtomicScore}(C) = & \sum_{A \in \text{Anns}} (\sum_{D \sqsubseteq C \in \Psi} \text{Support}(A, D) \\ & - \sum_{D \sqcap C = \perp \in \Psi} \text{Support}(A, D) \\ & - \sum_{C \sqsubseteq D \in \Psi} \text{Omit}(A, D)) \end{aligned}$$

Anns denotes the set of annotations over the span $\hat{s}$, $D \sqsubseteq C \in \Psi$ indicates that from the axioms of ontology Ψ one can derive D is a subclass of C , and $D \sqcap C = \perp \in \Psi$ indicates that Ψ implies disjointness of D and C . $\text{Support}(A, D)$ is 1 if annotator A labels the span $\hat{s}$ with D , and is 0 otherwise. $\text{Omit}(A, D)$ is 1 iff D belongs to the vocabulary of A , but A failed to tag span $\hat{s}$ with D . Note that a term in the sum above will only be present if the corresponding type of axiom is asserted (or inferred) in the ontology. The above sum considers two ways for an annotator can “oppose” a concept C , by tagging with a concept disjoint from C or by omitting to tag with (a superclass of) C in its vocabulary. Tagging the span with a disjoint concept implies a strong conflict, while omitting a superclass of the concept means a basic conflict. This is where our approach exploits information from conflicting annotations to derive the correct answer. Limiting our ontology to subsumption and disjoint constraints plays a vital role in the efficiency of our algorithm, as we will prove in Section 3.1.4. If our ontology language was rich enough to have (e.g.) rules saying that C and D partition all entities, we would also have an additional term representing “support by omission”: an annotator omitting D is supporting C (because an entity must be either C or D). Since we have no such rules in our example ontology, we do not examine this setting.

A straightforward approach is to assume annotators to have low recall, and thus annotator A failing to tag with a concept D gives no signal that the entity is not in class D – this corresponds to setting each $\text{Omit}(A, D)$ to 0 for each A and for each D , eliminating one of the terms above. In our experiments in Section 2.4 with wide-spectrum web-based annotators we showed that recall of the chosen systems is indeed low, and hence we utilise this approach in our implementation (see Section 3.3 for details). In other words, considering the omitting support resulted in a too strong constraint, since we found cases of missing annotations in more than 95% of the annotated spans in our experiments. This would end up in considering almost all of the annotations incorrect, since in almost every case one or more annotators would not annotate the given span even if they know an applicable concept. However, for domain-specific annotators,

other approaches, such as those described in Section 3.1.5 based on bootstrapping, may be appropriate.

We will refer to this simplified version as *simple atomic scoring*, and under this policy $\text{AtomicScore}(\mathbf{C})$ becomes:

$$\text{AtomicScore}(\mathbf{C}) = \sum_{\mathbf{D} \sqsubseteq \mathbf{C} \in \Psi} \{\mathbf{A} \mid \mathbf{A} \text{ annotates with } \mathbf{D}\} - \sum_{\mathbf{D} \sqcap \mathbf{C} = \perp \in \Psi} \{\mathbf{A} \mid \mathbf{A} \text{ annotates with } \mathbf{D}\}$$

Example 1. Assume the simple atomic scoring model, and consider a span annotated by annotator A_1 with **City**, annotator A_2 with **Country**, and annotator A_3 with **NaturalFeature**., and suppose that all of these concepts are disjoint with each other in the ontology Ψ . Since these concepts are disjoint in the ontology, under the simple scoring model each of **City**, **Country**, and **NaturalFeature** will have $\text{AtomicScore} = +1 - 1 - 1 = -1$, since each is supported by one annotation and opposed by the other 2. Suppose also that all of the three concepts are a subclass of **Location**. The AtomicScore for **Location** is thus 3, since each of the involved annotation supports the concept **Location**.

The atomic score aims to capture, for each concept $\mathbf{C}$, how many annotators agree with $\mathbf{C}$ and how many disagree with $\mathbf{C}$.

3.1.2 Weighted Repair

Given the atomic scores, a *score combination function* is responsible for calculating a boolean combination of concepts that is consistent with the ontology, based on the scores of constituent literals. For the combination function we propose a declarative approach, denoted as *weighted repair* (WR). We begin with the union of all concepts returned by any annotator, which can be considered as an initial formula ϑ_{init} containing the conjunction of concepts. We then try to modify the initial formula with repair operations, consisting of either a deletion of a concept occurring as a conjunct term within ϑ_{init} or an insertion of a concept that is absent from ϑ_{init} as a conjunct term. The application of a single repair operation Op to ϑ_{init} is a new formula. A deletion of class $\mathbf{C}$ implies removing every conjunct corresponding to a subclass of $\mathbf{C}$ while adding the negation of $\mathbf{C}$, while an insertion implies conjoining with a proposition corresponding to $\mathbf{C}$ (we will see later in Lemma 3 that once we insert a concept $\mathbf{C}$, we will also insert all its superclasses). We define a set of repairs as *internally-consistent* if no two operations conflict: we do not delete a class $\mathbf{C}$ and also insert a subclass of $\mathbf{C}$. Given an internally-consistent set of repairs $S = \{\text{Op}_1 \dots \text{Op}_n\}$, the application of the repairs on ϑ_{init} , denoted with $S(\vartheta_{\text{init}})$, is defined to be the result of applying every $\text{Op}_i \in S$ in any order – one can easily check that this gives a unique result. We call an internally-consistent repair set S such that $S(\sigma_{\text{init}})$ is consistent with T_Ψ a *solution*. We define the set of concepts that are either not deleted or inserted in S the *survivors* of S . In

other words, the survivors of a solution is the set of concepts (resulting from deleting and/or adding concepts from the initial annotations found over the span) such that no two concepts are in a disjoint relation. Thus the modified annotated span will be freed from strong conflicts.

Example 2. Let us return to Example 1. ϑ_{init} is **City** $\wedge$ **Country** $\wedge$ **NaturalFeature**. The repair set $\{\text{Del}(\text{City}), \text{Del}(\text{Country})\}$ is internally-consistent (we do not delete a concept **C** and add a concept that is a subclass of **C**), and its application to ϑ_{init} gives the formula **NaturalFeature** $\wedge$ $\neg$ **Country** $\wedge$ $\neg$ **City**. Since this is consistent with the ontology (the survivors set contains only **NaturalFeature** and does not have disjoint classes), the repair set is a solution. The repair set $\{\text{Del}(\text{City}), \text{Del}(\text{Country}), \text{Del}(\text{NaturalFeature})\}$ is another solution, yielding the formula $\neg$ **NaturalFeature** $\wedge$ $\neg$ **Country** $\wedge$ $\neg$ **City** – in this case the survivors set is empty.

One can notice that deleting all the concepts (but one) from the initial formula ϑ_{init} always produces a solution, since we cannot have disjointness within a set with zero (one) concepts. Our goal is to find a solution S such that its *aggregate score* is maximal among all solutions, where the aggregate score of S is:

$$\sum_{\text{Ins}(\text{C}) \in S} \text{AtomicScore}(\text{C}) - \sum_{\text{Del}(\text{C}) \in S} \text{AtomicScore}(\text{C}) + \sum_{\neg \text{Del}(\text{C}) \in S} \text{AtomicScore}(\text{C})$$

That is, an insertion of a concept **C** incurs **AtomicScore**(**C**) as a penalty, an operation that deletes a concept **C** incurs the penalty **AtomicScore**(**C**), while a concept **C** that is not deleted from the initial formula increases the aggregate score of **AtomicScore**(**C**). Based on the above definition, there are three possible ways of increasing the aggregate score: (i) delete a concept **C** with negative atomic score – this will increment the aggregate score of **AtomicScore**(**C**) and of the atomic scores of all the subclasses of **C** (since when we delete **C** we delete also all its subclasses); (ii) insert a concept **C** with positive atomic score – this will increment the aggregate score of **AtomicScore**(**C**); (iii) do not delete a concept **C** with positive atomic score – this will increment the aggregate score of **AtomicScore**(**C**). Unlike in other database repair contexts, repair operations can have positive score (net benefit), negative score (net penalty), or zero score (cost neutral, but perhaps needed to gain consistency). Among the possible set of concepts that are consistent with Ψ , we choose the one that maximises the aggregate score.

There can be many repairs that achieve the maximal score, and it is natural to impose other criteria to prune them: (i) Given two solutions with the same score and different numbers of operations, we prefer the one with smallest number of repairing operations. (ii) Given two solutions $S_1 = S' \cup \{\text{Ins}(\text{C}_1)\}$ and $S_2 = S' \cup \{\text{Ins}(\text{C}_2)\}$, with C_2 a subclass of C_1 , we prefer S_2 – that is we prefer those that insert deeper (that is, more specific) classes (if we have a solution that inserts **City** and another solution that inserts **Location** with the same aggregate score, we choose the one that inserts **City** since it is a more specific class). An *optimal* solution is the one that maximises the aggregate score and respects the criteria defined above.

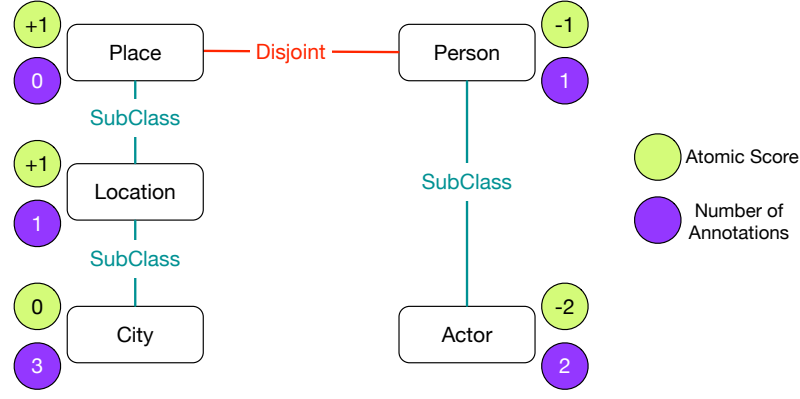


Figure 3.1: Annotated Span with Atomic Scores

Example 3. Consider a span with 3 annotations of type **City**, 1 **Location**, 2 of type **Actor** and 1 **Person**. Figure 3.1 shows the ontological view of the concepts, the number of annotations for each concept and also the atomic score for each of the involved concept. **Location** and **Person** both belong to the initial formula, but since they are disjoint concepts, one of them must be deleted (along with its subclasses). Therefore the plausible solutions consistent with Ψ are:

$$\begin{aligned}
 S_1 &= \{\text{Del}(\text{Person}), \text{Del}(\text{Actor})\} & S_2 &= \{\text{Del}(\text{Location}), \text{Del}(\text{City})\} \\
 S_3 &= \{\text{Del}(\text{Person}), \text{Del}(\text{Actor}), \text{Del}(\text{City})\} & S_4 &= \{\text{Del}(\text{Location}), \text{Del}(\text{City}), \\
 S_5 &= \{\text{Del}(\text{Person}), \text{Del}(\text{Actor}), \text{Ins}(\text{Place})\} & & \text{Del}(\text{Actor})\} \\
 S_6 &= \{\text{Del}(\text{Person}), \text{Del}(\text{Actor}), \text{Del}(\text{City}), \text{Ins}(\text{Place})\}
 \end{aligned}$$

Among those solutions S_5 and S_6 have the highest aggregate score of 5 – deleting **Person** and **Artist** contributes +3, not deleting **Location** is +1, inserting **Place** a further +1, while deleting or keeping **City** does not make any difference, since its atomic score is 0. Among S_5 and S_6 we choose S_5 , since the set of operations is smaller than S_6 .

3.1.3 Implementation

The problem of finding the solution that maximises the aggregate score is an optimisation problem, and we can solve it by a reduction to integer linear programming (ILP). We have boolean variables $X_{\text{Del},C}$ for each concept C that can be deleted, with a value of 1 denoting deletion and 0 denoting no deletion. Similarly we have boolean variables $X_{\text{Ins},C}$ for each concept that can be inserted. Each disjointness or subsumption relation in the ontology corresponds to an integer constraint. In particular, $\forall C, D \in \Psi \mid C \sqcap D \sqsubseteq \perp$ (disjoint concepts), we derive the following constraints:

- (i) $X_{\text{Del},C} + X_{\text{Del},D} \geq 1$. If there are two disjoint concepts in the initial solution, at least one of them must be deleted.
- (ii) $X_{\text{Ins},C} + X_{\text{Ins},D} \leq 1$. We cannot insert two disjoint concepts.
- (iii) $X_{\text{Ins},C} \leq X_{\text{Del},D}$. If we insert a concept C , then we must delete the disjoint concept D from the initial solution.

Similarly, $\forall C, D \in \Psi \mid C \sqsubset D$ (subclass relationship), we derive the following constraints:

- (i) $X_{\text{Del},C} \geq X_{\text{Del},D}$. If we delete a concept D , then we must delete all its subclasses C .
- (ii) $X_{\text{Ins},C} \leq X_{\text{Ins},D}$. If we insert a concept C , then we must insert all its superclasses D .
- (iii) $X_{\text{Ins},C} + X_{\text{Del},D} \leq 1$. We cannot insert a concept C and delete its superclass D .

Some variables have a fixed assignment, therefore we do not have to compute variable values for each concept in the ontology. Suppose C is the set of the concepts in the initial solution. We cannot insert a concept that is in the initial solution, as well we cannot delete a concept that is not in the initial solution – $X_{\text{Ins},C} = 0 \ \forall C \in C$, and $X_{\text{Del},C} = 0 \ \forall C \notin C$. Moreover, we notice that every concept in the ontology that is not in the initial solution and that is not a superclass of one of the concept in the initial solution has an atomic score of 0. These concepts cannot increase the aggregate score, therefore they will never be inserted in the solution – $X_{\text{Ins},C} = 0 \ \forall C \notin C \mid \nexists D \in C \mid D \sqsubset C$. This reduces the number of variables to be assigned to $k + l$, where k is the number of initial annotated concepts, while l is the number of superclasses of the concepts in the initial solution that are not included in the initial solution.

The scoring policy described above is easily translated to maximisation of a corresponding linear objective function:

$$\begin{aligned} \max \{ & (k + 1) \cdot (\sum_{C \in \Psi} \text{AtomicScore}(C) \cdot X_{\text{Ins},C} - \sum_{C \in \Psi} \text{AtomicScore}(C) \cdot X_{\text{Del},C} \\ & + \sum_{C \in \Psi} \text{AtomicScore}(C) \cdot (1 - X_{\text{Del},C})) - (\sum_{C \in \Psi} X_{\text{Del},C} + X_{\text{Ins},C}) \} \end{aligned}$$

where k is the total number of concepts initially annotated. Note that the last summation is necessary to choose between two solutions with the same aggregate score – because of the last summation, given two solutions with the same aggregate score, the maximisation function favours the one with smallest number of operations (insert and delete). k is necessary to make the aggregate score the dominant selection criterion – between two solutions with different aggregate score, the one with the highest score will always be chosen, no matter what the number of operations is.

Example 4. Consider again the annotated span described in Example 3. The concept **Place** is not initially annotated, thus it is the only one which does not belong to the initial solution. The concepts in the initial solution cannot be inserted, therefore $X_{\text{Ins},C} = 0 \ \forall C \in \{\text{Actor, City, Location, Person}\}$, while **Place** cannot be deleted – $X_{\text{Del},\text{Place}} = 0$. Since these are the only concepts involved in this annotated span,

$X_{\text{Del},C} = 0$ and $X_{\text{Ins},C} = 0$ for every other concept C in the ontology. The remaining unknown variables are $X_{\text{Del},\text{Artist}}$, $X_{\text{Del},\text{City}}$, $X_{\text{Del},\text{Location}}$, $X_{\text{Del},\text{Person}}$, and $X_{\text{Ins},\text{Place}}$. Note that **Place** is the only concept in the example eligible to be inserted in the final solution. For each pair of disjoint concepts in the initial solution we add an integer constraint to ensure that at least one of them will be deleted. For instance we add the constraint $X_{\text{Del},\text{Location}} + X_{\text{Del},\text{Person}} \geq 1$ to guarantee that one between **Person** and **Location** is deleted from the solution, or both. We also add the constraint $X_{\text{Ins},\text{Place}} \leq X_{\text{Del},\text{Person}}$, to ensure that if we insert **Place** then we delete **Person**. An example of subclass constraint is $X_{\text{Del},\text{Actor}} \geq X_{\text{Del},\text{Person}}$, stating that if we delete **Person** then we must delete **Actor** as well. Eventually we try to assign a value to all the unknown variables by maximising the objective function defined above. As we have seen in Example 3, the optimal solution corresponds to the assignment $X_{\text{Del},\text{Artist}} = 1$, $X_{\text{Del},\text{City}} = 0$, $X_{\text{Del},\text{Location}} = 0$, $X_{\text{Del},\text{Person}} = 1$, and $X_{\text{Ins},\text{Place}} = 1$.

This optimisation problem requires (in particular) checking consistency of various boolean combinations with respect to the propositional theory T_Ψ . For rich enough ontological rules (enough to give an arbitrary propositional theory), is **NP-hard**, and the use of a robust ILP solver is needed. In our implementation of the ILP-based approach, we used IBM’s ILOG CPLEX¹ as a solver. For each annotated span a collection of a few hundreds constraints is produced. We showed that we can prune these by considering only super-classes of concepts that are returned by one of the annotators on the span; given that the depth of the concept hierarchy is in single digits, this results in a drastic reduction in the number of variables. We also generate constraints corresponding to disjointness axioms from the top of the ontology hierarchy downward, allowing us to suppress redundant disjointness constraints, further reducing the constraint set. Note that the ILP reduction also gives an **NP** upper bound for the decision problem corresponding to finding an optimal repair (we can check the existence of a repair with a given score in polynomial time).

3.1.3.1 Ontology

In order to query the ontology, we need a semantic reasoner that is able to infer logical consequences from a set of axioms. If our ontology contains axioms stating that **Artist** is a subclass of **Person**, and **Person** is disjoint with **Location**, the semantic reasoner must be able to infer that **Artist** is disjoint with **Location**, even if this axiom is not explicitly stated in the ontology. Or similarly if **Actor** is a subclass of **Artist**, and **Artist** is a subclass of **Person**, the reasoner must infer that **Actor** is a subclass of **Person** as well. Such reasoning is offered by most of the semantic engines available, and we chose OWLIM-Lite² to store and query the ontology (because of its popularity and availability).

¹<http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

²<http://graphdb.ontotext.com/display/OWLIMv54/OWLIM-Lite>

However, since our ontology is stable and the types of queries are limited (superclasses, subclasses and disjoint classes of a concept), we created a *materialised version* of the ontology which consists in a set of textual files, where each file contains a specific relation R (e.g., superclass). The materialised version is created through the OWLIM-Lite reasoner, and this optimisation turned out to be much faster, improving the overall performance of two orders of magnitude (see Section 3.3 for details).

3.1.4 Tractability

The integer linear programming implementation described above may be intractable in complex cases, and the computation of an optimal solution might be too expensive for large enough ontologies. We will now show that in our specific settings, where the ontology has only disjoint and subclass constraints, computing an optimal solution is always tractable.

We recall that given a solution S , we define survivors the set of concepts that are either inserted or not deleted in S . Given the definitions of a solution and optimal solution we can derive the following:

Lemma 1. *If S is a solution and $C \in \text{survivors of } S$, then for each D superclass of C , $S' = S \cup \{\text{Ins}(D)\}$ is also a solution.*

Proof. Suppose the lemma is false: given a solution S and a concept C that belongs to survivors of S , and a concept D superclass of C , the set of operations $S' = S \cup \text{Ins}(D)$ is inconsistent with T_Ψ . This means that there exists $E \in \text{survivors of } S$ such that E is disjoint with D . But since D is a superclass of C , if D is disjoint with E , then C is disjoint with E , therefore S is not a solution. $\square$

Lemma 2. *An optimal solution S will never insert a concept C with $\text{AtomicScore}(C) \leq 0$.*

Proof. Let us consider the solution S that inserts C with $\text{AtomicScore}(C) \leq 0$. $S' = S \setminus \{\text{Ins}(C)\}$ is still a solution (removing a concept from the survivors set will never make the survivors set inconsistent), with a greater or equal aggregate score and with less operations. S is not the optimal solution. $\square$

Lemma 3. *For any optimal solution S that inserts a concept C , S also inserts all the superclasses of C .*

Proof. If S is the optimal solution and $\text{Ins}(C) \in S$, then for Lemma 2 $\text{AtomicScore}(C) > 0$. If $\text{AtomicScore}(C) > 0$, then $\text{AtomicScore}(D) > 0$ for any superclass D of C (for the definition of atomic score). $S' = S \cup \{\text{Ins}(D)\}$ is also a solution for Lemma 1, and the aggregate score of S' is greater than the one of S , therefore S must insert D to be the optimal solution. $\square$

Lemma 3 proves that if the optimal solution inserts a concept C , then it will also insert all the superclasses of C .

We now prove that, in the special case of disjointness and subsumption constraints only, we can find at least one optimal solution tractably, without appeal to ILP.

Lemma 4. *Given two disjoint concepts C_1 and C_2 at most one can have a positive **AtomicScore**, and if one does, the other must have negative **AtomicScore**.*

Proof. Suppose C_1 is supported by s_1 annotations and C_2 is supported by s_2 annotations – we have s_1 annotations that are subclass of C_1 and s_2 annotations that are subclass of C_2 . We have that $\text{AtomicScore}(C_1) = s_1 - s_2 - a_1$ and $\text{AtomicScore}(C_2) = s_2 - s_1 - a_2$, where a_1 (resp. a_2) is the number of all the other annotations that are disjoint with C_1 (resp. C_2) and are not subclasses of C_2 (resp. C_1). Note that the positive contributions to the two atomic scores are already included in s_1 and s_2 . Now if $s_1 > s_2$, then $\text{AtomicScore}(C_2) < 0$, if $s_2 > s_1$, then $\text{AtomicScore}(C_1) < 0$, and if $s_1 = s_2$, then both atomic scores are less than or equal to 0. $\square$

Theorem 1. *Assuming only disjointness and subsumption constraints, an annotation set corresponding to some optimal repair set can be achieved by inserting every non-annotated concept of positive **AtomicScore**, deleting every concept of negative **AtomicScore** and deleting some concepts with score 0.*

Proof. The proof of Theorem 1 now follows easily: it suffices to consider any repair set S_0 that yields a boolean combination containing all concepts with positive **AtomicScore**, and which deletes all concepts with **AtomicScore** ≤ 0 . Using Lemma 4 we see that S_0 leaves the database consistent (since there are not concepts with negative scores, there are not disjoint concepts). Now consider S an arbitrary repair set of maximal score, whose application yields a consistent formula. First, suppose the application of S deletes a concept C of positive **AtomicScore**. Thus S contains $\text{Del}(C')$ for C' a superclass of C . Consider the repair S' formed by removing $\text{Del}(C')$ from S . It is easy to see, using the definition of **AtomicScore**, that $\text{AtomicScore}(C')$ must also be positive. If S' were inconsistent, this means that C' was disjoint from some class D either inserted or left intact by S' . But using Lemma 4 we see that any such D that was inserted must have had negative **AtomicScore**, as would any D left intact. Thus a modification to S by deleting or not inserting any such D , as appropriate, would result in a repair set that yields consistency and has higher score. Now suppose that the application of S fails to insert a concept C of positive **AtomicScore**. We can argue, as above, that the application $S \cup \{\text{Ins}(C)\}$ gives a consistent formula (of higher score). Moreover, Lemma 2 proved that the optimal solution will never insert a concept of **AtomicScore** ≤ 0 . We proved so far that the optimal solution insert any concept of positive atomic score, and deletes any concept of negative atomic score, and does not insert any concept with atomic score less than or equal to 0. The last scenario we need to consider is a concept C

in the initial solution with $\text{AtomicScore}(\mathbf{C}) = 0$. After deleting all the concepts with negative atomic score, we have two possible outcomes: either $\mathbf{C}$ is not disjoint with any concept in the solution, or $\mathbf{C}$ is still disjoint with some concepts in the solution. If the former happens, the solution is consistent and we do not delete $\mathbf{C}$ (since deleting it would not increase the aggregate score but would increase the number of operations). If the latter happens, we still have to delete some concepts from the solution to make it consistent. It is easy to see that in this case there can only be one concept $\mathbf{D}$ disjoint with $\mathbf{C}$ (if there is more than one these concepts would have a negative atomic score and they would have been already deleted from the solution), therefore we randomly delete one between $\mathbf{C}$ and $\mathbf{D}$ to make the solution consistent. $\square$

Theorem 1 gives a simple algorithm for finding an annotation that results from an optimal repair set: start with inserting every non-annotated concept of positive **AtomicScore**, deleting all concepts with a negative atomic score, and for each remaining pair of disjoint concepts with zero atomic score, randomly delete one of them. Applying this to the initial formula, one gets a formula that is consistent, since it cannot contain disjoint concepts with **AtomicScore** 0, and also cannot contain disjoint concepts where one has positive **AtomicScore**. By the theorem above, this gives an optimal result. The last part of the algorithm is where we do not have enough evidence to make a deterministic decision. If we have a span annotated as **Person** and **Location**, we cannot decide which one of the two is correct by just looking at the annotated span, therefore we delete one of them – in this case two solutions have the same aggregate score and the same number of operations.

Example 5. Returning to Example 3, an optimal score can be obtained by a repair set that (i) deletes all the concepts with negative atomic score (**Person** and **Actor**), and (ii) inserts **Place** since it has a positive atomic score. The algorithm does not touch **City** since after the deletion step the solution is already consistent. This matches the intuitive solution S_5 anticipated in Example 3.

Despite the above algorithm makes the ILP formulation of the problem redundant, we decided to keep the original problem formulation and its internal ILP implementation for two main reasons: first, the ILP formulation gives an elegant presentation of the problem and a immediate intuition of the characteristics for a good solution; second, the ILP formulation is general enough that can support more complex ontological axioms such as part-of. With the internal ILP implementation, introducing different ontological axioms would require only the translation of these axioms into appropriate ILP constraints. If instead we use the simplified algorithmic version, we would need to change entirely the algorithm to support each of the new axioms.

3.1.5 Bootstrapping

The atomic score function defined at the beginning of the section does not assume any prior knowledge on the annotators. It treats every annotator equally, assuming that every annotator is equally competent on each concept, both in terms of precision and recall. We then made the assumption that all annotators have low recall, and we eliminated the omission penalty. In this section we generalise our approach to take into account prior knowledge on the involved annotators.

The general form of our scoring function allows the possibility of tuning the assumptions about both precision and recall, as well as support for more general concept constraints than simply subsumption and disjointness. If we know a priori for a concept C how much an annotator is trustworthy when identifying C (precision), and how often an annotator misses C (recall), we can modify the atomic score function as follows:

$$\begin{aligned} \text{AtomicScore}(C) = & \sum_{A \in \text{Anns}} (\sum_{D \sqsubseteq C \in \Psi} \text{SupportWeight}_{A,D} \cdot \text{Support}(A, D) \\ & - \sum_{D \sqcap C = \perp \in \Psi} \text{SupportWeight}_{A,D} \cdot \text{Support}(A, D) \\ & - \sum_{C \sqsubseteq D \in \Psi} \text{OmitWeight}_{A,D} \cdot \text{Omit}(A, D)) \end{aligned}$$

where $\text{SupportWeight}_{A,D}$ and $\text{OmitWeight}_{A,D}$ are non-negative $[0, 1]$ -valued weights that indicate how much weight the tagging of A with D or the omission of D by A should have. In other words, $\text{SupportWeight}_{A,D}$ is the quantification of the annotator's precision, while $\text{OmitWeight}_{A,D}$ is the quantification of the annotator's recall. If we can always trust an annotator both in precision and recall (we are confident that an annotator always tags with the correct concept and never misses an annotation), then we can set its $\text{SupportWeight}_{A,D}$ and $\text{OmitWeight}_{A,D}$ to 1, and the atomic score function becomes equal to the one previously defined in the section. The above formula is a definition of AtomicScore , which generalises the specific case of having perfect annotators.

We now consider how the $\text{SupportWeight}_{A,D}$ and $\text{OmitWeight}_{A,D}$ parameters can be computed. We have experimented with a *bootstrapping approach* setting all the weights to be initially uniform at 1, and then iterating:

$$\begin{aligned} \text{SupportWeight}_{A,D}^{k+1} &= \text{AVG}_{\text{Support}(A,D,s)} \text{NormAtomicScore}^k(D) \\ \text{OmitWeight}_{A,D}^{k+1} &= \text{AVG}_{\text{Omit}(A,D,s)} (1 - \text{NormAtomicScore}^k(D)) \end{aligned}$$

where $\text{Support}(A, D, s)$ indicates that s is a span which A tags with D , $\text{Omit}(A, D, s)$ indicates that A has D in its vocabulary but omitted it on s , and $\text{NormAtomicScore}^k(D)$ is a version of the $\text{AtomicScore}(D)$ calculated based on SupportWeight^k and OmitWeight^k , but normalised to be in $[0, 1]$ (e.g., by shifting the scores and dividing by their maximum). The intuition is that if A has high precision for D , spans that it annotates with D should get high support in round k , and hence $\text{NormAtomicScore}^k(D)$ should be close

to 1, making its **SupportWeight** high in round $k + 1$. Similarly, if **A** has high recall for **D**, the spans for which it omits **D** should get low support in round k , which will make **OmitWeight** ^{$k+1$} high.

The set of documents over which the spans are calculated obviously will have a large impact on the results. We have tested the bootstrapping approach using the entire test corpus from Section 2.4.1. We have found that it works well for very general concepts, such as **Person**, **Place**, and **Organization**, where examples are plentiful, but hurts on a more specific (and less frequent) classes (see Section 3.3 for detailed explanation).

3.1.6 Overlapping Span

The integration procedure described so far assumes that annotations are related only if they have the same span, i.e., they insist on the same tokens of text. The repair is therefore computed one span at a time. On the other hand, annotations often have nested or overlapping spans, as in Figure 2.3. We therefore consider a more careful definition of annotation span that takes into account overlaps – other than semantic conflicts, we take into account containment and overlapping conflicts.

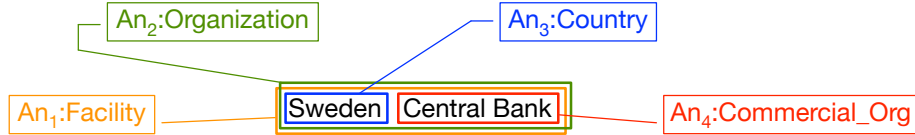


Figure 3.2: Conflicts in Annotated Spans

Example 6. Consider the annotations in the example of Figure 3.2. **AN₁:Facility** and **AN₂:Organization** have the same annotation span. The span of **AN₄:Commercial_Org** is strictly contained in **AN₂:Organization**, and **Commercial_Org** is a subclass of **Organization**. **AN₃:Country** is contained in **AN₁** and **AN₂**, and it is disjoint with these two concepts, while its span is disjoint with the span of **AN₄**.

Overlaps. Let T be a document represented as a sequence of characters $T = \langle c_1, \dots, c_n \rangle$ and $\underline{a} = \langle l_{\omega_i}, [i, j] \rangle$ an annotation of a sequence $\langle c_i, \dots, c_{j-1} \rangle$, $1 \leq i < j \leq n + 1$, of characters of T provided by an annotator AN_i using a label l_{ω_i} from its local vocabulary $\mathcal{L}_{\omega_i}$. We denote by $\text{type}(\underline{a})$ the label l_{Ψ} from the vocabulary $\mathcal{L}_{\Psi}$ of the global ontology to which l_{ω_i} is mapped. We also denote by $\text{intv}(\underline{a})$ the interval $[i, j]$ of characters covered by $\underline{a}$ in T .

Given two annotations $\underline{a}_1, \underline{a}_2$ we say that they are *disjoint* iff $\text{intv}(\underline{a}_1) \cap \text{intv}(\underline{a}_2) = \emptyset$, they *overlap* iff $\text{intv}(\underline{a}_1) \cap \text{intv}(\underline{a}_2) \neq \emptyset$, in particular they *coincide* iff $\text{intv}(\underline{a}_1) = \text{intv}(\underline{a}_2)$. Moreover, we say that $\underline{a}_1$ is *contained* in $\underline{a}_2$ iff $\text{intv}(\underline{a}_1) \subseteq \text{intv}(\underline{a}_2)$.

Orthogonal spans. A way of dealing with overlapping spans is to consider distinct spans as mutually independent, i.e., annotations conflict or support one another only

if their spans coincide. In this case, if an annotation $\underline{a}_1$ is contained in an annotation $\underline{a}_2$, we treat these as two separated spans and we apply our weighted repair algorithm independently on the two spans. This obviously leads to a loss of precision in the aggregation, since overlapping spans do often refer to the same entity – suppose $\underline{a}_1$ annotates the snippet “Mr. Barack Obama” and $\underline{a}_2$ the contained snippet “Barack Obama”, these two annotations are not independent from each other and they should be resolved together.

Coalesced spans. A completely opposite strategy is to take the connected component of spans under the overlap relation, considering each component as a span having all of the annotations associated with its component. We found that in practice these connected components are quite small – at most four tokens. But considering a disjoint pair of annotations connected in this way as being in conflict can lower the recall: e.g. in the case of nested annotations such as “Sweden Central Bank” where the whole string is annotated as **Organization**, e.g., by `OPENCALAIS`, and the string “Sweden” is annotated as **Country**, e.g., by `ALCHEMYAPI`. If we apply our algorithm to all the annotations having overlapping spans, the algorithm deletes one between **Organization** and **Country**, even though they are both correct.

Hybrid spans. In practise we found that if an annotation is contained into another and the concepts are disjoint, then the annotations may still be correct. On the other hand, if the two annotations refer to concepts in a subclass relation, then it is likely that the two annotations refer to the same entity. To circumvent the above problem, we introduce a *hybrid annotation span* defined as follows. Given an ontology Ψ , we define two annotations $\underline{a}_1, \underline{a}_2$ (with $\underline{a}_1 \neq \underline{a}_2$) as belonging to the same annotation span if either:

1. $\underline{a}_1$ is contained into $\underline{a}_2$ (resp. $\underline{a}_2$ is contained into $\underline{a}_1$) and it is not the case that $\Psi \models \text{type}(\underline{a}_1) \sqcap \text{type}(\underline{a}_2) \sqsubseteq \perp$.
2. $\underline{a}_1$ overlaps with $\underline{a}_2$ and neither $\underline{a}_1$ nor $\underline{a}_2$ is contained into the other.
3. $\underline{a}_1$ coincides with $\underline{a}_2$.

Condition 1 relates overlapping annotations that are possibly redundant. This is the case, e.g., when one annotation is contained in another with types in superclass relationship – we purposely excluded those cases of (strictly) contained annotations with logically conflicting types because the annotations may be compatible. Condition 2 captures the case of annotations with ambiguous spans, i.e., they overlap on one or more tokens but neither contains the other. In this case, no matter what is the relation among the involved types, the annotations always belong to the same span that needs to be resolved. Eventually, Condition 3 captures the general (and most frequent) case, where annotations have exactly the same span. For the case show in Example 6, the repair procedure will operate on $\text{AN}_1\text{:Facility}$, $\text{AN}_2\text{:Organization}$, and $\text{AN}_4\text{:Commercial.Org}$ as they all belong to the same span due to conditions 1 and 3, leaving out $\text{AN}_3\text{:Country}$ as it belongs to a different hybrid span. The procedure will eventually keep all the

annotations, since **Facility**, **Organization** and **Commercial_Org** are not disjoint with each other.

For any document T , we first compute annotations belonging to the same hybrid span, then we apply the weighted repair algorithm on every hybrid span. Our experiments have shown that the use of hybrid spans increases the recall but lowers the precision w.r.t. coalesced spans. However, the change in performance is below 1% and the overall F-score remains practically unchanged ($\pm 0.1\%$) on all datasets. In the following we will stick to hybrid spans since containment among annotations is very frequent in practice. An interesting future work is how to extend the weighted repair algorithm to resolve conflicting spans.

3.2 Supervised Algorithm

In Section 3.3 we benchmark our repair-based approach against several other systems, including all freely-available annotation aggregators. But a second natural comparison point is against what can be accomplished with supervision. We have thus implemented our own supervised approach to see to what extent one can leverage the existence of training data to find patterns that indicate (in)correctness of annotations. Can supervision be used not just to look for patterns in reliability of annotators (e.g. from tracking the recall or precision of annotators on concepts), but to detect and exploit more complex patterns in the behaviour of annotators?

We build on maximal entropy models, which have gained wide popularity for a variety of individual extraction tasks in recent years [24]. We thus use a Maximal Entropy Markov Model (MEMM) [113], which combines features of maximal entropy models with the sequential approach of a Markov model. A MEMM has a set of states, an input and output alphabet, and a probabilistic transition and emission functions that capture the system dynamics when a new input item is consumed. The inputs consist of sets of features token by token. As with other maximal entropy models, they allow for feature functions that overlap, without requiring strong independence assumptions. In our model a state is a potential annotation – a label from the alphabet that starts at the character i of the document and ends at the character j . The input alphabet is the sets of all the annotations returned by the involved annotators, while the output alphabet is identical to the state space. The transition probabilities are then computed by considering as features the set of annotations occurring on the same span (on the training data), and adjusting the weight parameters via the *generalized iterative scaling (GIS)* [113]. Eventually a *Markov sequence* is constructed by processing the input evidence, and we use a standard Viterbi algorithm for the computation of the most likely sequence. Please refer to [37] for a detailed explanation of MEMM.

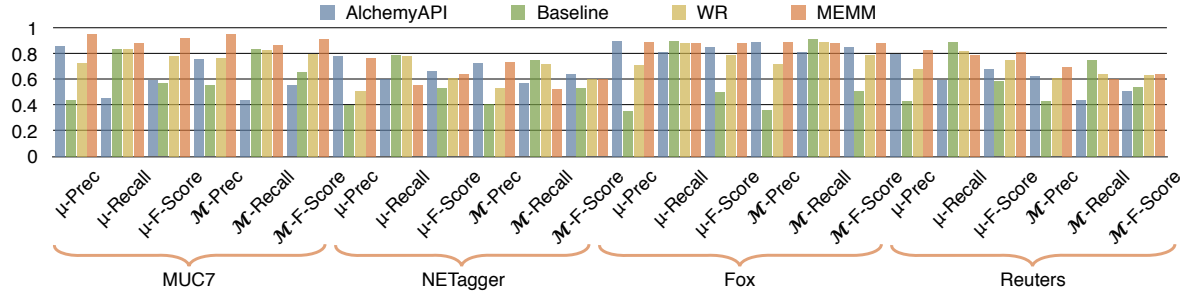
3.3 Experiments

We carried out an extensive experimental evaluation of our aggregation method. A first set of experiments aims to demonstrate and quantify the benefit of aggregating annotations over individual systems. A second set of experiments compares our aggregation approach with various aggregation techniques, including the supervised MEMM, and those described in state-of-the art aggregators such as FOX [138] and NERD [130]. Eventually we demonstrate that the bootstrapping approach is not beneficial, and also that our approach scales well with the size of the document and the number of annotators involved.

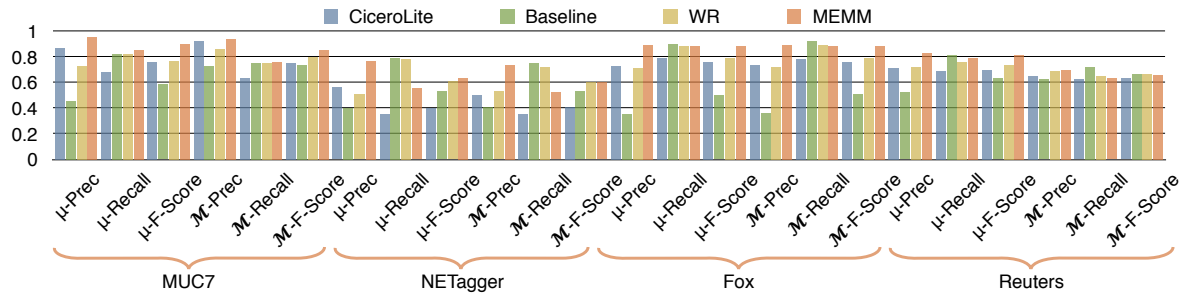
The evaluation presented in this section uses the same datasets and experimental setting which have been described in Section 2.4, along with the same way of calculating precision, recall, and F -score. In this thesis we report the most relevant results, however the full set of experiments containing every possible combination can be examined online at [8].

3.3.1 Comparison with Individual Systems

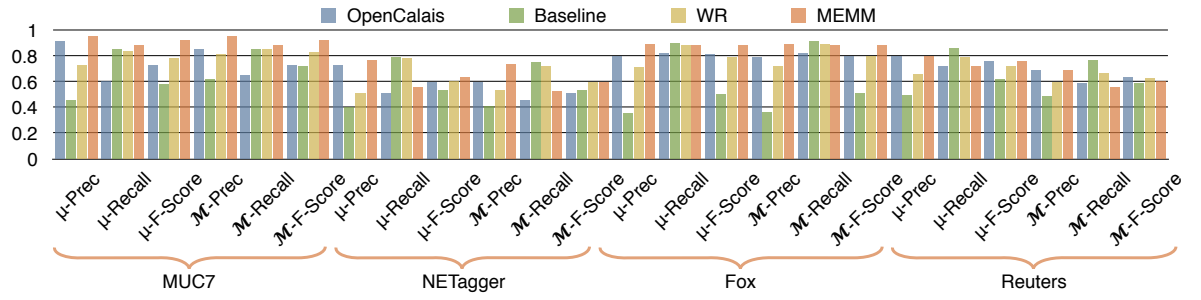
In our first set of experiments, we compared the performance of WR against individual annotators and against the supervised approach MEMM. We also report the performance of a naïve Baseline aggregation that simply gathers all annotations from individual systems and, after mapping the annotated concepts to the global ontology, returns the union of all annotations regardless of consistency. MEMM has been trained on gold standards described in Section 2.4.1, using a 10-fold cross-validation to split the dataset into training and testing data. Figure 3.3 reports the 10-fold cross-validation performance of the two aggregation approaches against each of the individual annotators. In this section we report only the most relevant results for 4 web-based annotators, we found similar results for other annotators that can be examined online. The main focus here is on accuracy (rather than on vocabulary coverage), therefore results against an annotator are computed *only* w.r.t. the concepts that are in the annotator’s vocabulary – thus a different set of concepts is evaluated for each annotator. Indeed the evaluation aims to test the accuracy of the annotators on those concepts where they are supposed to be competent. As in Section 2.4, we computed measures for each concept of the annotators’ vocabulary. However, to present an overall picture of the performance, we give averages of the precision, recall, and F -score, measured in the ontology-aware way explained in Section 2.4. We provide: a *macro-average* ($\mathcal{M}$) that first averages over each concept, and then over annotations, and a *micro-average* (μ) that averages over each annotation (thus concepts that have highest coverage in the testset are given more importance).



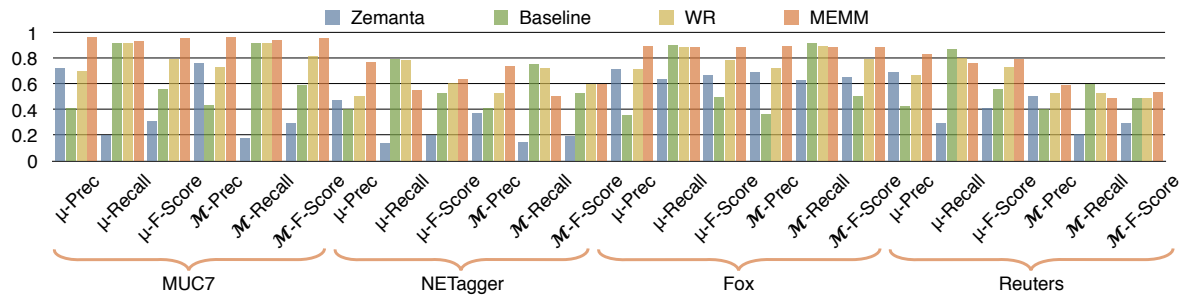
(a) vs ALCHEMYAPI



(b) vs CICEROLITE



(c) vs OPENCALAIS



(d) vs ZEMANTA

Figure 3.3: Individual Annotators vs Aggregators

Our first observation is that all three aggregation methods consistently outperform individual annotators in F -score except for **OPENCALAIS** and **ALCHEMYAPI**. **OPENCALAIS** and **ALCHEMYAPI** performs slightly better in F -score (i.e., 3% better on average) than any of the aggregators on the **NETagger** and **Reuters** corpus respectively. However, it is worth noticing that their concepts coverage is very limited and their vocabularies cover only 18% of all concepts in the corpus. On average, **WR** reports an aggregation benefit of 15%, while **MEMM** of 22% with a significant peak of 66% improvement over **ZEMANTA**. As expected, the **Baseline** aggregation delivers the worst performance w.r.t. the 3 aggregation techniques, with a significant improvement in recall but a drastically loss in precision, worse than any of the individual annotators. The high recall of the **Baseline** is expected, since this method never deletes an annotation. If its recall does not reach 100% is because in some cases *none* of the involved annotators returns the correct answer. Surprisingly, **MEMM** shows a better recall than the **Baseline** in some cases. As we will explain later, this is because **MEMM** sometimes is capable of deriving the correct concept even if the answer is not provided by any of the base annotators.

3.3.2 Supervised vs Unsupervised

As expected, the supervised aggregation is the best among the aggregators. We see from the charts that for **MUC7**, with rich training examples over a small number of target concepts, **MEMM** can bring the *micro-average* and *macro-average* F -scores above 90%. We can observe a small degradation in *macro-average* scores in the cases against **OPENCALAIS** and **EXTRACTIV** on **Reuters** corpus. The decrease is caused by some annotated concepts which have a low number of instances (i.e., small number of training support) in the corpus. For example, **PhoneNumber** has only 36 occurrences in the news-oriented **Reuters** corpus. This causes the examples to be extremely sparse in the training set, and **MEMM** is incapable of deriving suitable patterns (reporting only 5% in F -score). Instead base annotators attain around 80% for this concept.

It is unsurprising that individual annotators can recognise with good accuracy concepts like phone numbers, given that they can be mostly extracted via simple regular expressions. **MEMM** may therefore suffers on concepts that have few instances in the training set, while **WR** is not affected by the coverage.

The major capability of **MEMM** is to learn which annotator can be trusted at a concept-level through global training. Consider, e.g., the key concept **Person**: individual annotators report very sparse performance on this concept in terms of F -score – that is from 14% to 93% on **MUC7** (see Figure 2.6); 26% for **ZEMANTA**, 43% for **SPOTLIGHT**, 63% for **ALCHEMYAPI**, 75% for **EXTRACTIV** and 91% for **OPENCALAIS** on **Reuters**. **MEMM** is able to understand that **OPENCALAIS** is an expert on identifying **Person**, therefore it trusts the system and achieves a comparable high accuracy (94% and 89% on the two datasets respectively). The extreme side of this spectrum is when

MEMM is capable of doing better than all annotators put together: a striking case is the concept **CommercialOrg**, where MEMM improves the accuracy by more than 20% against base annotators and more than 10% against the the unsupervised WR. This is due to an interesting pattern that MEMM is able to infer: annotators often correctly label the concept **Organization**, and in the majority of the case the probability of an **Organization** being a **CommercialOrg** is high; MEMM understands such patterns from the training data and can annotate **CommercialOrg** even when none of the base annotators provide this concept. This is a typical example of the capability of MEMM of deriving the correct concept, even when the concept is not returned by any of the involved annotators. Obviously, this kind of pattern depends mostly on the training data, and this pattern may not longer holds when expanding the dataset to different sources (and may not be derived by MEMM with sufficient training examples). But the result does show the capability of MEMM of deriving patterns within the data or the annotators, when such patterns do exist.

WR shows an overall high precision, competitive with MEMM. This is because it considers a concept **C** correct only when the majority of the “judges” who are expert about **C** tagged the span with (a subclass of) **C** – indeed, such a signal turns out to be a strong indicator of correctness. When the majority of the base annotators is competent over a certain concept, WR delivers also good performance in terms of recall, in some cases even higher than individual annotators. For example, most of the annotators are consistently competent on concept **Country**, with F -score above 75% (only ZEMANTA delivers significantly lower accuracy). In this case WR achieves a precision of 90% while still having recall comparable with OPENCALAIS, the most competent annotator for this concept. However a more common scenario is when the recall of individual systems is rather low (annotators have a concept in their vocabulary, but are not able to identify most of the instances). In this case WR cannot achieve competitive recall, since most of the instances lack sufficient support to be considered correct – WR does not have enough evidence to induce the correct annotation. For the concept **Person** discussed above, the only annotators with high recall are OPENCALAIS and EXTRACTIV, hence the F -score of WR is much lower w.r.t. other cases.

The major advantage of WR w.r.t. supervised approaches is its resilience to sparsity in the dataset (in particular in the training set). For the example of **PhoneNumber** discussed above, where MEMM performs quite poorly, WR has a score comparable with the best individual annotators. Indeed WR does not suffer on the number of instances of a concept, it suffers mostly on the inaccuracy of the involved annotators. Eventually, for what concerns the **Baseline**, the results are as expected: a very good recall (always equal to the sum of base annotators’ recall), but an extremely poor precision (which reflects the incorrectness of the base annotators).

3.3.3 Comparative Evaluation

In this set of experiments we evaluated WR (as well as MEMM) against the FOX [138] and NERD [130] NER aggregators on all datasets. FOX combines annotators’ opinions via neural networks, in order to determine the optimal combination of annotators. It does not use any ontological information. NERD is a more recent approach, where background knowledge is exploited for the integration, along with machine learning techniques to capture the best combination of annotations. NERD uses an ontology that does not contain disjointness constraints, hence conflicts cannot be automatically detected.

Even though we focus mostly on web-based annotators, in order to compare our approach against FOX we aggregated opinions of two software-based annotators, namely the STANFORD NER and ILLINOIS NER, as well as 4 others web-based annotators that are not reported in the individual experiments – namely SAPLO [9], WIKIMETA [11], LUPEDIA [6], and YQL [12] (though individual results about these annotators can be examined online at [8]). Figure 3.4 summarises the results and clearly shows that both aggregation techniques discussed in this chapter are at worst comparable with FOX and NERD and in average outperform the other two aggregators of more than 15% in *F*-score on the MUC7, Fox, and Reuters datasets. The NETagger corpus represents an interesting case with high data sparsity. In fact, not only does NERD show a better behaviour, but also MEMM is regularly outperformed by WR on every concept of this dataset. On the other hand, the difference in performance is less evident on the Fox corpus where FOX and MEMM behave very similarly, but they are often outperformed by WR (again due to data-sparsity issues in the dataset). Since they both use machine learning techniques, FOX and NERD suffer as well from data sparsity, just like MEMM. Moreover, FOX is capable of recognising only high-level concepts (Person, Organization, Location, Date, Time), while NERD’s vocabulary is larger (around 50 concepts). The vocabulary of our aggregation methods consist of the union of the vocabularies of

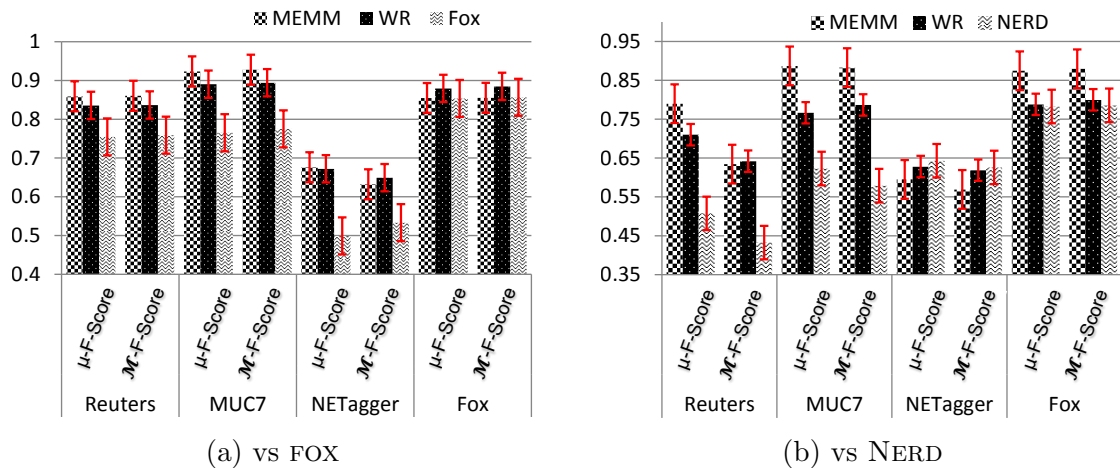


Figure 3.4: Comparative Evaluation.

all the annotators involved, which at current includes almost 400 concepts. For **WR**, expanding the vocabulary is basically no effort: we just include a new annotator and map its concepts to the merged ontology. For **MEMM** instead adding a new annotators implies re-labelling the training dataset with new annotations.

The full suite of test results (including evaluation on 11 different annotators), together with all the text resources, the annotated gold standards, scripts to compute ontology-aware precision and recall, and the merged ontology, is available at the online appendix [8].

3.3.4 Bootstrapping Evaluation

As mentioned in Section 3.1.5, we can also use an hybrid unsupervised-supervised approach. In an initial bootstrapping (supervised) phase we first compute weights for each annotator (weights that express a quantification of the precision and the recall of the annotators), and then we use these weights when aggregating the annotators' responses. The set of documents over which weights are calculated obviously will have a large impact on the results. We have tested two ways of computing weights: *(i)* the bootstrap approach described in 3.1.5, using the entire test corpus as training set; *(ii)* the weights computed by **MEMM** in its training phase. As previously discussed, supervised approaches work pretty well for concepts that have density in the training data (**Person**, **Place**, and **Organization**), while they are not able to generalise correct weights for concepts with a limited number of instances (see the **PhoneNumber** example discussed above). This reflects in the computation of weights: the bootstrap approach works well for general concepts with high density, but its performance are extremely lower on more specific concepts (those with a limited number of instances).

3.3.5 Scalability

In Section 3.1.4 we proved that an optimal solution can be computed efficiently. In the last set of experiments we empirically study the amount of resources required to compute a solution for the integration, both for the unsupervised approach and the supervised one. Figure 3.5 shows the (per-document) average computation time w.r.t. an increasing number of annotators (2 to 11) and for all four different corpus. As expected, **WR** is very fast and it can compute an optimal solution for a single document on the order of 300 milliseconds in the worst-case. **WR**'s performance is directly correlated with the number of annotated spans to be processed, and that is considerably higher in the **MUC7**, **NETagger** and **Reuters** corpora than in the **Fox** corpus (hence why the differences in performances). With the supervised approach instead, the time to compute an optimal solution is affected by the number of concepts to be taken into account and on which **MEMM** has been trained. This is why the **Reuters** corpus takes significantly longer than other corpora, where fewer concepts are involved.

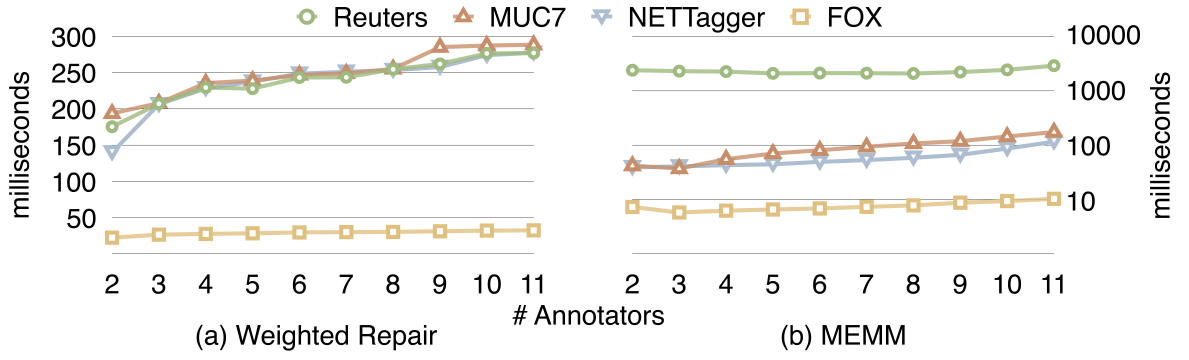


Figure 3.5: Aggregation Performance

The training phase in **MEMM** takes between 68 milliseconds (2 annotators and 3 concepts on the **Fox** corpus) and 2 minutes (11 annotators and 215 concepts on the **Reuters** corpus). Overall, both in the supervised and unsupervised setting, the time required to compute an optimal solution is dominated by at least one (or two) orders of magnitude by the time required to send the various requests to the online annotation endpoints and collect the responses. As an example, **CICEROLITE** takes about 60 seconds to annotate a medium-sized (i.e., 3-4 Kb) document. Generally, none of the studied annotators is capable of producing a response in less than 2-3 seconds. Also, both **MEMM** and **WR** have negligible memory consumption when compared with the size of the raw annotations.

Moreover, since we consider each span at time independently and we do not use any cross-spans information, both **WR** and **MEMM** could run their integration algorithms in parallel for each span. Therefore, we do not report the running times w.r.t. the size of the document, that can be easily reduced by parallelising the computation.

3.3.6 Summary

We can summarise the finding of our experiments as follow:

1. Aggregating semantic annotations of individual annotators is always beneficial in terms of recall. In terms of overall accuracy, aggregation shows consistent benefit except few cases where the majority of annotators contribute wrong annotations. If the majority of annotators provide the correct answer, then the aggregation is also capable of deriving the correct answer.
2. Although supervised aggregation delivers overall higher accuracy, unsupervised aggregation is to be preferred when no training data is available. Moreover, adding a new annotator and expanding the vocabulary has zero cost in the unsupervised setting.
3. Our proposed unsupervised approach consistently outperforms existing annotation aggregation approaches, as well as the supervised algorithm in some cases.

4. Aggregation does not impact the scalability of annotating a single document, since it is always dominated by one or two orders of magnitude by the annotation time of individual web-based annotators.

3.4 ROSeAnn: an Available System

We implemented our middleware for aggregating semantic annotators in an online available system: ROSeAnn (Reconcile Opinions of Semantic Annotators). ROSeAnn offers two main functionalities: on the one hand it is an entry point for the invocation of several web-based annotators (and easily extensible with additional annotators), on the other hand it aggregates opinions to give a unified and consistent annotation on top of several (possibly incorrect) annotators. ROSeAnn is implemented almost entirely in Java, with few additional technologies for specific purposes. In the next 3 sections we explain in details how to use ROSeAnn: (i) as a graphical user interface, (ii) as a programmable API, (iii) as a black box web service.

3.4.1 Graphical Interface

The first and most straightforward way of using ROSeAnn is through its GUI. One can download an executable file (both for Windows and UNIX environments) to launch the GUI ([8]) and graphically use ROSeAnn's functionalities.

The GUI is shown in Figure 3.6 (left-hand side). The top part of the GUI (A) consists of a menu bar for basic operations, such as open a new document, annotate a document, and save an annotated document. With the last button we can configure ROSeAnn: we can specify which annotators to invoke, annotator-specific parameters and whether we want to use the aggregation algorithms or not. After we open a new document, the text is displayed on the main text area of the tool (B). Documents can be annotated via the green arrow button of the menu bar, and once annotated, the

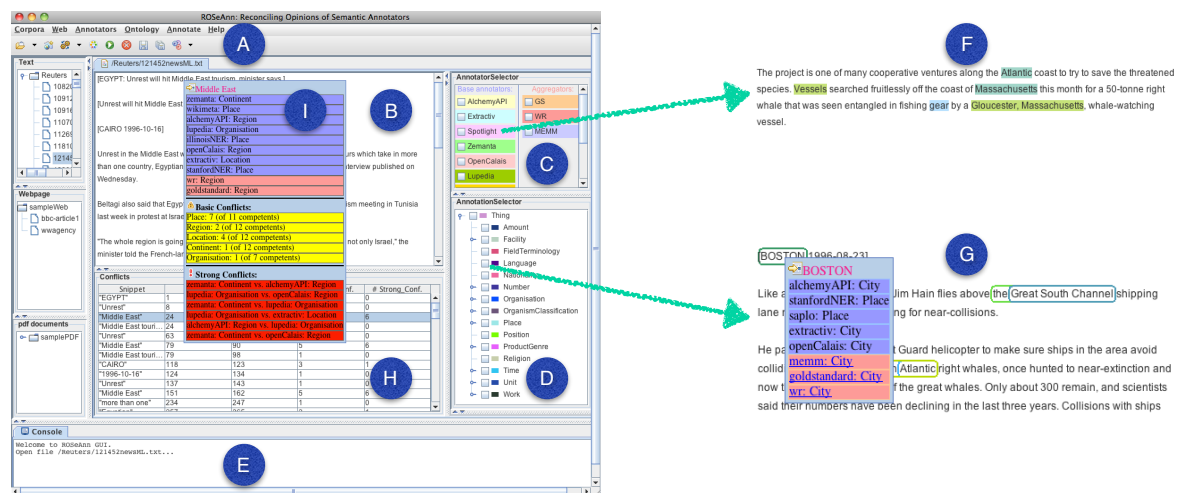
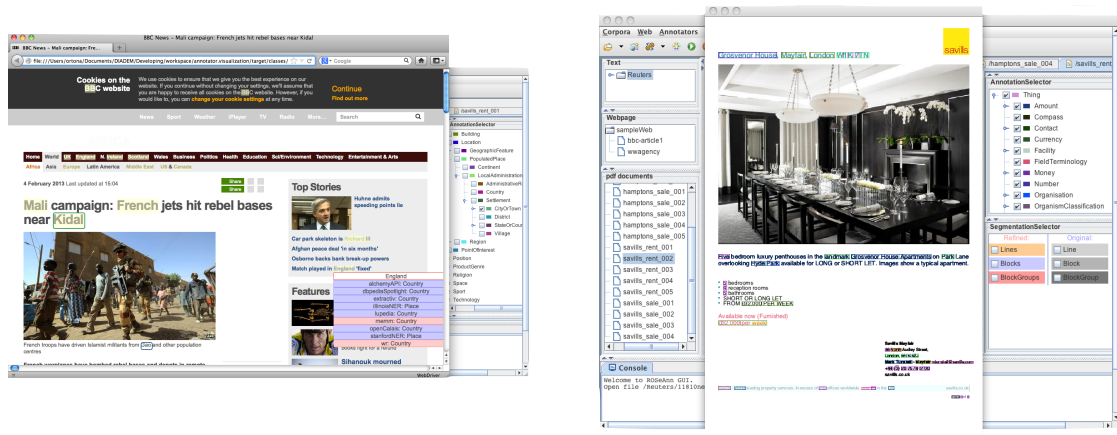


Figure 3.6: ROSeAnn GUI

right-hand side of the tool shows specific boxes for the list of annotators (as well as aggregators) returning at least one annotation (**C**), and the list of concepts retrieved over the document (**D**), organised in a hierarchical representation according to subclass relations in the ontology. The bottom part of the GUI eventually displays some explanation messages for each operation (**E**). We can tick one or more annotators (as well as aggregator methods), and the text snippets annotated by the ticked annotators (aggregators) are shown on the text with the relative background colour (**F**). We can as well tick one or more concepts, and the text snippets annotated as those concepts are highlighted on the text with a coloured bounding box (**G**). Hoovering the mouse over a highlighted text span triggers a pop-up window that contains all the opinions of the annotators and the aggregators on the given span (**G**). Eventually the box right below the main text area shows information about semantic conflicts. For each conflicting span it reports the text of the span, the start and end character, and the number of (strong and basic) conflicts (**H**). We can click on a conflict, and the corresponding text snippet will start blinking for three seconds on the main text area. If we hover the mouse over that text area, a pop-up window shows information about the identified conflicts other than individual opinions (**I**). If the user is satisfied with the result, she can save the annotated document along with the semantic information. The document and the annotations are saved in a XML format, similar to the one used in GATE.

Annotation and visualisation over text documents is a functionality offered by most of the annotation tools. An innovative component in ROSeAnn is the possibility of annotating web pages, as well as pdf documents. Through the menu bar of the GUI we can open a Mozilla Firefox browser and navigate to a target web page, as well as loading a pdf document from the file system. Annotating a web page (pdf document) triggers three different actions: (i) the extraction of visible text parts of the document; (ii) the annotation (and aggregation) of the extracted text; (iii) the mapping of semantic information from the text to the original document. Figure 3.7 shows an annotated



(a) Web Pages Annotation

(b) Web Documents Annotation

Figure 3.7: ROSeAnn Innovative Components

web page (3.7a) and a pdf document (3.7b). All the actions described above are available also on web pages and pdfs. Web navigation and visualisation over web pages are provided via the Firefox Selenium API,³ while for pdfs we used the Apache PDFBox library.⁴ Due to the limitations of the Selenium library, web annotations are currently available only on Linux platforms. The extension of web annotations to any platform is a current ongoing work.

3.4.2 Java API

The most powerful way of using **ROSeAnn** is to embed it into user-defined applications via its Java API [8]. The API offers all the functionalities described in this chapter, plus user-friendly functions to access semantic information retrieved over input documents. In order to use **ROSeAnn**, the user must first configure few parameters. These include the base annotators to be invoked, annotator-specific parameters, the aggregation algorithms and the input ontology. A mandatory parameter to be specified is the license key for each base annotator that requires it, otherwise the annotator cannot be queried. **ROSeAnn** has been designed to be highly extendible, therefore adding a new base annotator or invoking a different aggregation algorithm is just a matter of writing Java classes that implement specific interfaces.

As mentioned in Section 3.1.3.1, **ROSeAnn** is based on an external ontology that is implemented as a set of materialised files. If the user wishes to use a different ontology rather than our merged one, she first needs to create a materialised version of it. **ROSeAnn** offers APIs that, given an OWL file containing the ontology axioms, creates the set of materialised files that can be directly queried by **ROSeAnn**. The OWL file is processed with the OWLIM-Lite reasoner.

After setting all the required parameters and the desired ontology, the user can invoke **ROSeAnn** with text documents, web pages, and pdf documents. Documents can be annotated and aggregated, and the user can access all the retrieved information such as annotations, attributes of a specific annotation, conflicts, type of conflicts, aggregation algorithms output. When annotating web documents we can also retrieve information about the web page, such as the DOM node of a specific text snippet. Pdf documents offer similar features, and the user can retrieve visual coordinates of specific parts of the original document. The GUI described in the previous section can also be launched via the API, and the user can interact with it via specific GUI functions.

The Java API is released via Maven integration, and does not require the download of any specific file. Other than Maven, the API can also be invoked by downloading a single Jar file. Eventually, if the user needs to add or modify some of the offered features, we released all the source code of **ROSeAnn**. She can download the entire suite as a single archive and modify/extend the source code.

³<http://www.seleniumhq.org/>

⁴<https://pdfbox.apache.org/>

3.4.3 Web Service

As a last option, if the user wishes to integrate ROSeAnn in her application faster than using the Java API, she can invoke the web service.⁵ Similarly to web annotator endpoints, ROSeAnn can be invoked through a REST web service. We offer two endpoints: (i) */text*, to annotate textual content provided as input parameter; (ii) */web*, to annotate web pages providing the url of the page as parameter. Both endpoints can be queried via a GET and POST request. In the GET request the user can specify only basic parameters, such as the text to annotate or the annotators to invoke; in the POST request she can specify more detailed parameters, such as annotator-specific values. An example of GET request on the text endpoint is given below. The request

```
curl -i -X GET http://163.1.88.61:9091/roseann/text?  
text=Obama+Barack&timeout=8000&aggregator=true&annotators=illinoisNER&lupedia&stanfordNER&yahooAPI
```

specify the text to be annotated, a timeout of 8 seconds (if an annotator takes more than this time to produce an answer its output is ignored), whether to invoke or not the aggregation algorithms, and the list of base annotators to be invoked. All the parameters are optional, except the text parameter in the text endpoint and the url parameter in the web endpoint (the web page to be annotated). For each parameter a default value is provided if not specified (e.g., the default value for the timeout parameter is 60 seconds).

The response consists in a JSON object, and it is the same for both endpoints. It contains the status of the response, the set of annotations, a set of attributes for each annotation, the set of conflicts and the set of aggregation results (if requested). If for some reasons the request cannot be processed or some problems are encountered during the computation, an appropriate HTTP error code is returned.

The web service offers a limited set of functionalities w.r.t. the Java API, however it is the fastest way of embedding ROSeAnn into user applications. It can be queried at anytime, and it is capable of handling 30,000-50,000 requests per second.

3.5 Related Work

The main area of the aggregation technique described in this chapter is information extraction, but takes inspiration from truth-finding, data integration, and inconsistency-tolerant reasoning. In this section we summarise the most relevant related works.

Named Entity Recognition. Research on Named Entity Recognition (NER) systems can be currently grouped into three strands, according to the sources of information that are leveraged in order to produce textual annotations. *Language-driven*

⁵<http://diadem.cs.ox.ac.uk/diadem/index.php/component/content/article/88-prototypes/80-roseann-web-api.html>

approaches represent the tradition in NER and exploit knowledge about the natural language to derive linguistic patterns – typically in the form of rules – that, once applied to the text, extract instances of the entities of interest. A notable system in this sense is KNOWITALL [67], that uses eight pre-defined extraction patterns to generate candidates for extraction that are then statistically validated over a corpus. The extraction patterns in this work are manually defined. Pre-defined linguistic patterns work well on edited text documents, such as journal articles and books, where the sentences are well structured, but are less effective on irregular documents, e.g., classifieds and tweets, where the sentences are poorly structured. Self-supervised learning approaches have therefore been proposed that use additional information such as POS-tagging labels to identify the most likely candidate for the extraction. This can be done, e.g., by training a suitable classifier as in TEXTRUNNER [21]. TEXTRUNNER, despite leveraging on a machine learning algorithm, cannot be classified as a supervised method, since positive and negative examples are automatically generated via obvious candidate entities.

Knowledge-driven systems use unstructured, i.e., gazetted, or structured, i.e., ontological, background knowledge to locate the entities of interest in a text document. Many research efforts have tried to leverage on an external background ontology, we usually refer to such approaches as *ontology-based information extraction* (OBIE). Ontologies can be used to: (i) drive the construction of meaningful linguistic-patterns as in LEARNINGPINOCCHIO [47], CROSSMARC [101], and 2PP [108]; (ii) refine the extraction rules as in PANKOW [46] and NAMIC [22], or (iii) logically validate the result of the extraction as in SOBA [33] and Maedche et. al. [110] or, simply, (iv) link the extracted entities to a reference ontology as in DBPEDIASPOTLIGHT [114] and KIM [102].

As a last group of NER systems, most commercial entity extraction tools are *data-driven*, i.e., they replace sophisticated annotation techniques and background knowledge with very large amounts of data (usually from the web) and relatively straightforward machine learning techniques. These systems obtain relatively high accuracy in an open domain by restricting the type of entities to be extracted, e.g., to persons and organisations. Notable representatives of this class of systems are ALCHEMYAPI, CICEROLITE, OPENCALAIS, and ZEMANTA, which are also discussed in this work. The main problem with these tools is their poor performance on documents that do not belong to the training domain, as shown in Section 2.4.

Combining annotators. Usually approaches that tackle the aggregation of multiple NER systems have exploited voting mechanism or variations of it [105]. As also noticed in [136], a voting-based aggregation is heavily biased by the type and number of annotators. In particular, voting mechanisms do not work particularly well with web-based annotators, since the coverage of concepts that are claimed to be recognised is very limited and this obviously affects the quality of the outcome, especially when we do not assume any prior knowledge about individual systems. As anticipated in

Section 3.1.5, a bootstrapping approach that assigns weights to the annotators to reflect the confidence in their annotations is an obvious way of overcoming the voting limitation. In [136] the weights are computed using an exponential model inspired from meta information retrieval systems [20]. A major problem with this model is that weights are determined on a per-annotator basis, without taking into account the performance of an individual annotator on different concepts. In our experimental evaluation we proved that the same annotator can be reliable on a subset of concepts, while its performance may drop significantly on a different subset. To overcome this limitation, rather than assigning general weights to individual annotators, we need to assign weights to pairs annotator-concept. Our bootstrapping procedure works at such fine granularity, producing weights for each annotator-concept pair. Alternative simple voting mechanisms are also often complemented with thresholding [99], that outputs annotations only if they exceed a minimal support. Annotators have very different vocabulary coverage, and there are entities that can be correctly identified by a single system. As a matter of fact, 8 (out of 11) of the evaluated systems contain at least one concept that is not identified by any of the other systems. Thresholding on support would lead to a substantially drop in recall. Take as example the concept **CommercialOrg**, identified only by **CiceroLite**. This concept has at least one instance in each of the document of our corpora, and using an arbitrary threshold on it would decrease the overall accuracy of about 20%. Our repair-based approach, in contrast, is not limited by any arbitrary threshold.

Machine learning techniques is a straightforward way of combining annotators, especially to determine the optimal combination of annotators w.r.t. to their accuracy. Among others, the most popular techniques are neural networks [138], support vector machines [64], classifier *stacking* [80, 147, 152], and conditional random fields (CRFs) [136]. The underlying idea of all these approaches is to combine classifiers in a meaningful way to obtain a general *truth-finding* model. None of the approaches above uses ontological knowledge to determine logical conflicts caused by the aggregation. Being generic, these techniques often fail to take into account specific information about the context or the semantics of the text. A recent approach, namely NERD [130], proved that ontological background knowledge – that already proved beneficial for generating the annotations – can also help in the aggregation of multiple annotators by locating logically incoherent annotations. However, the NERD ontology does not contain disjointness constraints and therefore they must use a different way to automatically detect conflicts. We are aware that NERD uses machine learning techniques for the integration as reported in [143], but such techniques are not explained in the documentation when invoking its web service. Our approach uses directly the ontology (and in particular the disjointness constraints) to automatically detect problematic spans and make them consistent w.r.t. the ontology. As an interesting future work we want to evaluate other machine learning methods than **MEMM**, such as CRFs, and how they

can be adapted to consider ontological rules. There is a wide range of variants for both CRFs and Maximal Entropy Markov models (e.g., in the objective function used to train models [98]). In Section 3.3, our approach experimentally outperformed the main machine learning approach which was freely available, FOX [138]. Florian et. al. [80] reported experiments on the **Reuters** corpus with F -scores very similar to the macro and micro measures of our repair approach. Their system also won a CoNLL competition. Unfortunately we could not find an available implementation of it.

A possible way of extending our repair algorithm is to incorporate deep knowledge about annotators, when such knowledge is available. Michelakis et. al. [116] look at the combination of the annotation rules adopted by each annotator. It makes use of a maximal entropy classifier, but with “evidence vectors” being the output of rule-based annotators. Our approach instead has been designed to treat annotators as unknown “black-boxes”, such that the integration of new systems requires very minimal effort. Moreover, they completely neglect semantic relationships within the vocabulary of annotations. Our unsupervised method takes inspiration from classic judgement aggregation techniques [96], where the final solution captures judgements that are at a minimum aggregate-distance from those of individual experts. However, most of the traditional aggregation methods use techniques that exploit how individual experts reach a conclusion, thus treating experts as transparent “white-boxes”.

Relation to other areas. In a different area – that of asserting the quality and the trustworthiness of web sources – work has been done on trying to estimate the trust and dependence of sources [61, 86] and produce a *truth-finding* model. For example, Dong et. al. [61] use iterative algorithms to determine the level of trustworthiness of each web source. In a bootstrapping step a voting strategy determines an initial value of reliability, and then values are iteratively refined by considering agreeing/disagreeing outcomes on new sources. Here trust is naturally attributed globally to a web source, without a fine-grained granularity that judges sources on specific entities. These techniques are complementary to our aggregation methods since they exploit redundancy across different web sources to determine true assertions (similar to what has been done to compute web wrappers [32]). We could expand our technique with a further validation step, where annotations referring to the same event (e.g., websites publishing same news) are validated via integration algorithms described in these works.

Another complementary body of works studies how to detect and repair inconsistent facts in automatically created ontology, e.g., YAGO [140]. SOFIE [141] uses common-knowledge rules to detect, delete or repair logically inconsistent facts. As mentioned above, these techniques can be incorporated in our repair approach when additional knowledge about annotators is available. If we consider the example shown in Figure 2.3, rules could be used there to restrict the combination of valid concepts that appear in nested annotations. For instance the name of a museum usually con-

tains also the name of the country/city in which it is located, while people citations may contain the position/title of the person.

Our repair-based aggregation is close in spirit to consistent reasoning for inconsistent ontologies [131]. However, such techniques cannot be straightforwardly applied to our setting where inconsistent assertions can be stated multiple times and the logical consistency also relates to the way annotations are contained into each other. We consider scoring of repairs due to the support of the facts they remove. Scoring of repairs also arises for numerical integrity constraints [27, 79]. Unlike in our case repairs can only have non-negative scores, and this impacts the technical development of the solution dramatically.

3.6 Towards an Alternative to Redundancy

In this first part of the thesis we presented ROSeAnn, a named entity recognition tool that produces semantic annotations on top of existing web-based annotators. Web-based annotators are often inaccurate and incomplete, and we showed how these systems are not always reliable in recognising semantic entities. We showed significant differences among systems in vocabularies and performances, and how these differences can be exploited in aggregating different opinions of individual systems. ROSeAnn leverages frequent conflicts among annotators in order to produce a consistent solution that maximises a suitable defined objective function. We proved the effectiveness of combining different vocabularies in a big merged ontology, which is the only manual part required in this work. Since each base annotator is treated as a black-box, adding/removing an annotator is just a matter of adjusting the merged vocabulary. We empirically evaluated ROSeAnn and showed that it outperforms every single base annotator, as well as every aggregation tool available, making it the most accurate unsupervised system for named entity recognition.

Instance-level redundancy across multiple sources is a technique widely used in different areas, such as table/list segmentation [40, 65], data cleaning [43, 73], web data extraction [32, 44]. The underlying idea of these approaches is to compare data from different sources, and divide the information into clusters such that pieces of data belonging to the same cluster are *semantically* coherent and similar. The study on named entity recognition that we presented in this and the previous chapter provides a valid alternative to redundancy. Extracting entities from free text is equivalent to clustering text snippets according to their semantics, where a semantic cluster corresponds to an annotation concept. Annotators encode redundancy in their extraction algorithms and rather than comparing data from multiple sources, data is classified according to their internal rule engines. Redundancy-based approaches face two main obstacles: (i) we cannot always leverage sufficient redundancy for every piece of information to extract; (ii) redundancy suffers of a *scalability* problem – since it requires the comparison of

sources with each other, redundancy methods do not scale with an increasing number of sources. These obstacles can be easily overcome by encoding redundancy in named entity extractors.

In the next part of the thesis we will show how we can replace redundancy with named entity recognition to solve the problem of repairing web data. As a valid alternative to exploit redundant data, we will show an approach that scales well with the number of input web sources, and that requires minimal effort to tune annotators.

Part II

Data Cleaning and Repair

*“Perfection is immutable.
But for things imperfect,
change is the way to perfect them.”
— Owen Feltham*

Chapter 4

Web Data Extraction: the Need for Cleaning

In this second part we focus on the main challenge of this thesis work: inducing rules to clean and repair web extracted data along with the extraction process that generated such data. Rather than using cross-site, instance-level redundancy, we will propose a novel approach to repair web extracted relations that encodes existing background knowledge with the ensemble of entity recognisers presented in the previous part. In this chapter we recall main techniques in web data extraction and why database cleaning solutions are inappropriate in this scenario, while the next chapter describes our original solution to repair data extracted from the Web with regular expression-like rules.

The Web has become the largest repository of data ever built in human history. If we consider the United States for instance, the number of online shopping websites with revenues of more than \$10,000 is estimated of being more than a hundred of thousand [112], with most of the data published in *structured* format – information is published as collection of *records* or *objects*, where each record contains a set of *attributes* that defines specific properties of the record (e.g., a record could be a movie with an attribute title, or an electronic product with an attribute price). A comprehensive knowledge of information on the Web is usually divided into two main portions: **(i)** a *heavy head*, where a few hundreds of large websites provide information about the majority of the records (usually aggregators); and **(ii)** a *long tail*, where the remaining part of the information is scattered over several hundreds of thousand smaller websites. Even though the majority of the information is published by the heavy head, a significant portion of data is only available in the long tail [57].

With the increasing volume of information on the Web, web data acquisition plays an important role in modern organisations and is a strategic business process for data-driven companies such as insurers, retailers, and search engines. The abundance of web data has made *web scraping* (also known as web data extraction or web wrapping) an essential tool in data acquisition processes. Web scraping processes range from expensive manual data collection and purchasing (such as exploiting websites' API to

collect data of interest), to cheaper but often technically challenging methods such as automated extraction and crowdsourcing.

Roadmap. In this chapter we first introduce techniques for web data extraction (Section 4.1), with a particular focus on automated extraction. Automated extraction often suffers of the major problem of producing dirty data, so we revise some of the most relevant techniques to clean such data in Section 4.3 and Section 4.4. Eventually Section 4.2 concludes the chapter with a brief explanation on why these techniques are not suitable for cleaning web extracted data.

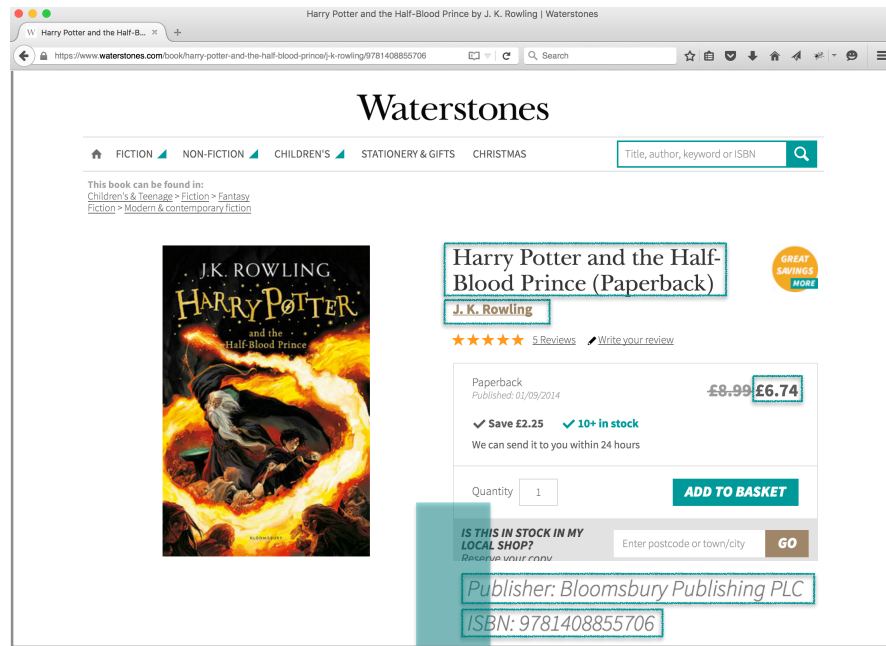
4.1 Web Wrapper Induction

Web data acquisition plays a central role in information extraction due to the abundance of data on the Web. Nowadays we can find almost any kind of information on the Web, information that is mostly available in HTML pages, designed for humans. Sometimes web data is easily accessible for machines, e.g., via web API. As an example, Linkedin offers a web API¹ through which developers can automatically gather information such as the job of a user or her Linkedin skills. However, often web data is not that easy accessible and web scraping is the only viable data collection method for websites.

Web scraping refers to the process of inducing a *wrapper* for a web source in order to extract relevant information from it. A wrapper is a program that turns web content into some form of structured format, so that the extracted information can be easily queried. Consider the example in Figure 4.1, showing a website (`waterstones.com`) publishing information about books. A wrapper for this source should be able to extract all the attributes of the book – namely Title, Author, Cover, Price, Publisher, and ISBN, surrounded with a green bounding box in the figure – in order to turn the web content into a structured format (e.g., the relation table at the bottom of the figure). If we can extract data about all the books published on `waterstones.com` (or even better from any books website), then answering queries such as “Give me all the books of Agatha Christie” or “Is the price of The Lord of the Rings less than £10?” would become a trivial task.

Web scraping typically relies on inducing a wrapper for every source, and in this direction semisupervised approaches have extensively investigated the problem [23, 55]. Semisupervised methods usually produce precise and correct data, however they require users to supervise the extraction by identifying data on each website and providing extraction examples. Clearly, techniques that rely on human supervision do not scale with an increasing number of sources and they are not feasible when we target the long tail – which implies extracting data from hundreds of thousand sources. To overcome this limitation, a number of fully automated techniques for web scraping

¹<https://developer.linkedin.com>



Title	Author	Cover	Price	Publisher	ISBN
Harry Potter and the Half-Blood Prince	J.K. Rowling	Paperback	£6.74	Bloomsbury Publishing PLC	9781408855706

Figure 4.1: Web Data Extraction on `waterstones.com`

have recently emerged, ranging from visual analysis of the rendered page to DOM tree mining [51, 82, 94, 156]. Wrapper induction systems usually inspect a limited set of records from the same source, and based on similarities/differences of the examples, induce extraction rules that are then applied to the entire source. The revitalisation of the area is also evident from a growing number of web scraping startups (e.g., IMPORT.IO, DIFFBOT, SCRAPINGHUB, LATTICE.IO), as well as recent acquisitions (e.g., BLACKLOCUS by HOME DEPOT, LIXTO by MCKINSEY).

Automated web scraping is a powerful data acquisition technique, but unfortunately automation comes at a price, and this price usually consists of errors in the extracted data. Automatically induced wrappers are indeed more error prone than those created by humans, resulting in dirty and incorrect data. As an example, Figure 4.2 shows the outcome of the application of a wrapper generated by ROADRUNNER [51] on the webpage shown in Figure 4.1. This example highlights some of the common issues found in data generated by automatically induced wrappers: (1) *Under/over*

Attribute_1	Attribute_2	Attribute_3	Attribute_4
Harry Potter and the Half-Blood Prince (Paperback)	J.K. Rowling	£8.99 £6.74	Publisher: Bloomsbury Publishing PLC ISBN: 9781408855706

Figure 4.2: Web Data Extraction on `waterstones.com` with ROADRUNNER

segmented attributes refer to the issue of attribute values split over multiple columns or a column containing multiple attribute values. As we can see in the figure, the ROADRUNNER wrapper incorrectly extracts Title and Cover (as well as Publisher and ISBN) together within the same attribute. These are commonly caused by irregularities in the HTML markup, or by the inability of wrapper induction systems to segment different attributes within the same DOM text node. Indeed often websites publish multiple attributes within the same DOM text node, and wrappers are able to extract only entire DOM node values (as it is the case of the webpage of Figure 4.1). (2) *Missing or incorrect column types* happen when the generated wrapper is incapable of assigning a meaningful label to extracted attributes. ROADRUNNER assigns labels only if these are explicitly stated on the website, otherwise columns are just named with progressive numbers (Attribute_1, Attribute_2, etc.). Such errors are associated with the lack of domain knowledge or suitable explicit labels in the source. (3) *Misplaced* and *missing* values (not present in the figure) reflect the inability of the wrapper to extract all the relevant information from the source, or to misclassify some of the extracted data. Irregularities in the source, ambiguities in the content, or a not representative set of examples considered in the induction are some of the typical causes of these errors. (4) *Spurious content* happens whenever the extracted data contains irrelevant information. As we see in the figure, the strings “£8.99”, “Publisher:”, and “ISBN:” are additional content that is not of interest.

Data shown in Figure 4.2 is clearly unusable. The information is somehow represented in the extracted relation, however it is impossible to query. Such data requires a further step of *data cleaning*, in order to overcome errors introduced during the extraction and realign the data to its original source.

4.2 Web Data Repair: New Challenges

Web extracted records contain specific problems that are not common in traditional data cleaning settings. Classic cleaning algorithms are not suitable for web records, especially if we target the long tail (i.e., extraction is performed on hundreds of thousand web sources). Techniques such as manual definition of cleaning constraints or the identification of suitable training examples are too expensive to scale with the size of the Web. Furthermore, relying on external experts or knowledge of information would limit the extraction to only those few domains where such an external knowledge is available. Even more recent techniques such as list segmentation or table induction would fail when applied to web records, given the presence of spurious and misplaced content that traditional algorithms would not be capable to deal with.

On top of the problems mentioned above, data acquisition through wrapper generation is not a static process that is executed once and for all, but rather an ongoing process where the same wrapper will likely be executed many more times in the future.

Due the dynamic nature of the Web, where published data is constantly updated and modified, an up-to-date data extraction strategy must inspect websites continuously, and thus wrappers are likely to be executed monthly, daily, and sometimes even multiple times per day. Therefore just repairing the extracted data is often not sufficient, since re-applying a data repair step for each wrapper execution may be too expensive. Rather than repair the extracted data, a more suitable approach would be to tackle the problem at its origin, by repairing the program that generated the data (the wrapper) such that future executions would produce clean data that does not need to be repaired. Scalable tools for continuous maintenance of both the induced wrappers and the extracted data are still largely missing. In the next two sections we give a brief overview of the state-of-the-art solutions in data cleaning.

4.3 Data Cleaning

Data cleaning, sometimes also referred as *data cleansing* or *data repair*, is the process of detecting and repairing errors and inconsistencies from data in order to improve the quality of data [124]. Data cleaning is a long-standing challenge in the database research, emerged mostly when multiple data sources need to be integrated, e.g., in data warehouses. The process is one step of the more general problem of *data integration* [106], that tackles not only the cleaning problem, but also the elimination of duplicates and the integration of different representations of the same object.

As shown in Figure 4.2, web extracted data via automatically generated wrappers often needs to be cleaned. Even though data cleaning mostly refers to databases, similar techniques can be applied to the web scenario, where sources are websites rather than conventional databases. Research efforts in this direction can be classified into three main categories, according to the underlying techniques used for cleaning.

Rule Based. The first body of work that tries to repair inconsistencies in databases relies on the definitions of rules or constraints. A database is said to be *dirty* if one (or multiple) of its tuples violates a given set of integrity constraints, and the data cleaning process consists in making the database consistent w.r.t. the set of constraints [30, 48, 69, 88, 137]. With respect to Figure 4.1, examples of constraints may be that a price is a number that cannot exceed the value of £1,000 or that ISBN is a 13 digits number. Most of the approaches that address data cleaning via constraints definitions rely on the existence of pre-defined constraints, and derive fast algorithms that make the database consistent minimising some cost functions (e.g., the number of repair operations). Even though the process sometimes can be facilitated via an intuitive graphical interface [54, 85, 125], manually deriving constraints is an expensive process. This is particularly true on the Web, where the information is gathered from hundreds of thousand sources and each source may require a different set of constraints.

Machine Learning. Several cleaning techniques adopt machine learning (ML) algorithms to detect and repair dirty databases. Yakout et. al. [154] propose a cleaning model that maximises the likelihood of the data given the underline data distribution (modelled using standard ML techniques), similar to [104] where an initial examples seed is exploited to iteratively discover attribute outliers (and consider these as erroneous values). Cleaning dirty data with ML can be problematic due to the existence of noise in the data to be processed, which can bias the learnt model with errors in the training data and thus introduce serious negative impacts [161]. In fact the main obstacle when training underlying models is the collection of positive and negative examples, which is an expensive process often slowed down by the limited availability of representative examples. This is particularly true when values to be repaired are collected from thousand of web sources, where each source may require a different set of examples in order to predict different classes of errors.

Collaborative Data Cleaning. A more recent research trend aims to improve the accuracy of the data by using external and additional information, in collaboration with standard data cleaning techniques. Some approaches leverage on external (reliable) knowledge bases in order to automatically detect erroneous information [43], while others rely on the presence of domain experts or master data that provide ground truth values [72, 125, 148]. As an example, a reliable external knowledge of information might state that Frank Sinatra was born on December 12th, 1915, and we can automatically detect (and correct) all the tuples in the database that contradict that information. Other approaches use as an external knowledge of information the crowd [43, 157], where data is validated through the generation of simple questions to be confirmed by an inexpert population.

The main problematic aspect when using an external source of information is that such a source is not always available, especially when data is extracted from vertical web domains that are not widely popular. With this respect, the crowd may be a good alternative solution, however typical crowd-sourced tasks must be carefully designed (usually broken down into very simple pieces [123]) in order to avoid ambiguities and misunderstanding, as well as crowdsourcing approaches cannot be treated as traditional data processing tools (e.g., quality-control mechanisms must be developed) [81].

4.4 Table Segmentation

Most data cleaning algorithms rely on the binary decision that a given attribute value is either correct or incorrect. Incorrect values are then targeted to be replaced (or deleted). As shown in Figure 4.2, web extracted data usually suffers from a different problematic nature. Indeed data is somehow extracted in the relation, however it is not correctly *segmented* and *aligned* to the reference output schema (plus the presence of some spurious content that needs to be removed).

A more relevant body of work close in spirit to web data cleaning is list segmentation/table induction algorithms. These approaches tackle the problem of inducing structured attributes from unstructured (i.e., wrongly segmented) list of values. As an example, TEGRA [40] is a recent work that given a set of strings extracted from web tables (each string is a row of the table), clusters groups of words from each string according to their semantic similarity – each group of words consist of an attribute value. By doing so the authors aim to reproduce the original table semantic (i.e., the original attribute values), since often on the Web column labels are not specified or are semantically ambiguous. Segmentation problems are rooted in other areas such as bio-informatics [93], where subparts of long sequences of characters need to be tagged according to some pre-defined categories (e.g., DNA).

Traditional table induction approaches try to solve the problem by leveraging some form of supervision. Many of these match values to be segmented against an initial seed of correctly segmented records [50, 100, 117], while others require positive and negative examples of valid column values to train a suitable classifier capable of segmenting target strings [111, 149]. Sometimes supervision can be avoided by automatically generating examples via distant supervision through existing knowledge bases [49] or, approaches like TEGRA or [65], overcome supervision by inspecting values to be segmented globally and looking for similar and/or redundant values.

Despite being close in spirit to web cleaning problems, list segmentation algorithms present some limitations that make them unsuitable for the web setting. Indeed the whole content of target strings to be segmented is assumed to be correct, it just needs to be partitioned into a given (or unknown) set of attributes. Web records instead often contain noise and spurious content that needs to be deleted (strings such as “£8.99” or “*Publisher:*” in Figure 4.2). Moreover web values might be misplaced and need to be moved in a different position in the original record – just defining string boundaries is not sufficient to correctly repair the data.

In the next chapter we will present a solution to adapt data cleaning techniques in order to deal with the new challenges posed by web records. We will look for solution that does not repair extracted relations in isolation, but rather to joint repair wrappers and wrapper generated data at the same time.

Chapter 5

WADaR: Joint Wrapper and Data Repair

Recent advancements in web data extraction and wrapper induction have made accurate and fully automated wrapper generation finally possible at the scale of thousands or hundreds of thousands sources. However, as shown in the previous chapter, tools for cleaning and maintaining the induced wrappers, as well as the extracted data, are still primarily manual or supervised. Scalable tools for continuous maintenance of both the generated wrappers and the wrapper generated data are still largely missing. Moreover, most of data cleaning efforts aim just at repairing the final produced data. In web data extraction instead, errors are most likely caused by the extraction process, and extraction errors are far more prevalent than source errors [62, 63]. In this chapter we present an approach to jointly repair wrappers and wrapper generated data, such that future wrapper executions will already produce correct data without the need of a further cleaning step.

Figure 5.1 shows an example of a relation generated by a wrapper created with ROADRUNNER [51] on three webpages similar to the one shown in Figure 4.1. The extracted (source) relation (R_S) is given at the top (supposing the wrapper has been applied to three web pages of `waterstones.com`), the desired (target) schema (Σ) at the bottom. R_S exemplifies common issues found in automated wrapper induction, where correct values are underlined with respective attribute colours. In the example the original source does not contain any error, and all the under/over segmented values, incorrect column types, and spurious contents in R_S are extraction errors, due to the incapability of the wrapper to correctly *align* the extracted data to the target schema.

Repairing the data means aligning the source relation to the target schema – a task that requires the identification of exact values from R_S (underlined colours in Figure 5.1) and their placement under correct attributes in Σ . However, as previously anticipated, just repairing the extracted data is often not sufficient, as in most of the cases data acquisition is an ongoing process and thus the same wrapper will likely be executed many more times in the future. This is particularly the case for data integration [32, 44], in which future executions of the repaired wrappers generate relations

R_S: Source Relation

Attribute_1	Attribute_2	Attribute_3	Attribute_4
Harry Potter and the Half- Blood Prince (Paperback)	J.K. Rowling	£8.99 £6.74	Publisher: Bloomsbury Publishing PLC ISBN: 9781408855706
A Brief History of Seven Killings (eBook)	Marlon james	£7.19	Publisher: Oneworld Publications ISBN: 9781780745886
The Lord of the Rings (Hardback)	J. R. R. Tolkien	£20.00 £15.00	Publisher: HarperCollins Publishers ISBN: 9780261103252

Σ: Target Schema

Title	Cover	Author	Price	Publisher	ISBN
-------	-------	--------	-------	-----------	------

Figure 5.1: Data Repair and Alignment on ROADRUNNER

already aligned with the target schema, thereby reducing the need for repeated data cleaning and easing integration [73, 111, 150]. If we could repair the wrapper other than just repair the generated relation, we would solve the problem at its root. Unfortunately, existing tools typically rely on cross-sources redundancy techniques that do not scale well with an increasing number of sources (e.g., [32, 44]). In principle nothing prevents the design of scalable approaches that use cross-sources redundancy, however the state-of-the-art systems have not addressed this problem yet. Moreover we will show that is not always possible to rely on enough redundancy.

The problem. We therefore investigate in this chapter the problem of computing a *joint wrapper and data repair* for web data extraction systems, so that future wrapper executions can benefit from the repairs applied to the wrapper. The problem takes as input a possibly faulty wrapper (specified in **XPath**), a relation generated by applying the wrapper to some collection of web pages of the corresponding source, and a target relation schema. A solution to the problem is a set of repair expressions that align the generated relation to the target schema, adjusting the wrapper where possible.

Example 7. Consider the (source) relation of Figure 5.1, where we aim to align the underlined values to the target schema Σ , according to respective colours. A possible set of repair expressions is

```

<TITLE, substring-before($, ␣)>
<COVER, substring-before(substring-after($, (, ), ))>
<AUTHOR, substring-before(substring-after($, ), ␣)>
<PUBLISHER, substring-before(substring-after($, sher: ␣), ␣ISBN)>
<DURATION, substring-after($, ISBN: ␣)>

```

specifying, for each attribute: (a) the attribute column label, and (b) a standard **XPath** string-valued expression that extracts the correct value from the concatenation (\$) of the columns of Figure 5.1, in the same left-to-right order as they appear. Note that none of the expressions repair PRICE, since it is not trivial to find a simple regular expression able to extract correct values from the concatenation string \$ for each record (there is not enough syntax regularity).

In this work we focus on a post-processing step of web data extraction, when the extraction process has already taken place and it has produced an extracted relation. This model allows flexibility and generality, as we can use our repair step with any data extraction system that is capable of producing relational tables. In fact we will show how our approach is not coupled with the extraction system, as it is capable of delivering high quality repairs no matter how the data has been extracted, treating the extraction system as an unknown black box. Moreover, usually wrapper induction techniques inspect only a portion of the target source, since the induction process is expensive and it cannot afford to look at every page in the source. We will propose a technique that is fast and scalable, and that will compare all the records extracted from a single source. On the other hand, incorporating the repair step directly in the extraction process has the main advantage of reducing the complexity of the overall pipeline, since the system would already extract clean data that would not need a post-processing step. The solution we propose considers both aspects: we will present a post-processing step that is independent from the extraction system, and that will produce feedbacks for the extraction process in order to improve the quality of the data already in the extraction.

Novelty. Classical *wrapper maintenance* assumes that a correct (usually manually created) wrapper exists and focuses on maintaining the correctness of the wrapper through techniques such as (a) detecting changes on the sources that may affect the wrapper, and (b) minimally (and robustly) adjusting the wrapper to adapt to these changes [39, 107, 115, 122, 126]. Unfortunately, for automated wrapper induction methods, the induced wrappers are often incorrect at the beginning. Tools for cleaning and maintaining the induced wrappers, as well as the extracted data, are primarily manual or supervised. The novelty of our work is in the introduction of a scalable tool for continuous maintenance of both the induced wrappers and the extracted data that can scale well with an increasing number of sources.

Contribution 1. We first formally define the problem of computing maximal joint repairs and we give an optimal solution that relies on the existence of attribute oracles. Oracles are functions that can test whether a given string value belongs to the domain of an attribute (i.e., whether a value can be correct for a given attribute). We show that the problem is **NP-complete** but becomes tractable under a few natural assumptions based on the behaviour of wrapper induction systems: (1) errors in wrapper induced relations are likely to be systematic, as wrappers are often induced from templated websites, where records and webpages follow the same structure (template); (2) wrapper induction systems tend to under-segment and misplace content rather than over-segment it (*atomic misplacements*, see Section 5.2 for details). In practice, this means that we can avoid reordering fields of generated relations in order to reconstruct a correct segmentation and alignment.

Despite being tractable, computing joint repairs remains difficult in practice. The main obstacle is the presence of attribute oracles, functions capable of recognising semantically related content in the relation, e.g., that in Figure 5.1 the tokens after “*ISBN:*” are all book unique identifiers. The recognition of semantically related content has some similarities to knowledge-based wrapper induction, where either an existing set of entity recognisers (like in [56, 82]) or cross-site, instance-level redundancy is employed to induce wrappers [32, 40, 44, 65]. However, in the wrapper induction setting, background knowledge is only compared with a small sample of webpages considered for induction, and the induced wrapper is applied to all the pages. For repairing, on the other hand, we can afford to compare it with the entirety of the extracted data rather than on just few sample webpages, thus discovering more meaningful patterns.

Contribution 2. We follow the inspiration of the former approaches and we propose an approximate solution that replaces attribute oracles with an ensemble of entity recognisers (Chapter 3), encoding existing background knowledge and cross-site redundancy. This has advantages such as enabling per-source (and therefore per-wrapper) repairs, while retaining a broad applicability across domains.

Unfortunately, entity recognisers are often noisy and incomplete (as noted also in [37, 56, 82] and Chapter 2) and our approach is specifically tailored to tolerate these deficiencies by eliminating noise and ambiguity and generalising the underlying structure to cope with incompleteness. The resulting approach is fully unsupervised and computes an optimal approximation of the joint repairs using an ensemble of entity recognisers to (1) discover hidden attribute structures in the annotated relations through the elimination of incorrect annotations, and (2) construct relation-wide repairs for both wrappers and generated relations.

Contribution 3. We show that our method is optimal under certain conditions on the error rate of the entity recognisers and, then, experimentally verify the performance and the generality of our method on a dataset of 100 websites from 10 different domains (Section 5.5). The evaluation uses 4 different wrapper induction systems, namely DIADEM [82], DEPTA [156], VINTs [159], and ROADRUNNER [51], and shows in each case an improvement in the quality of the extracted relation by 15%–60%, while fully repairing corresponding wrappers in more than 50% of the cases. We also show that approaches for unsupervised integration of wrapper-generated data [32, 44] can be used to construct a joint repair but underperform in terms of accuracy and scalability.

Roadmap. This chapter is organised as follows: Section 5.1 introduces the joint data and wrapper repair problem. Section 5.2 provides a polynomial solution for a specific case of the problem, while Section 5.3 gives exact bounds for the general case. Section 5.4 presents a general procedure for the computation of an optimal approximate solution to the problem that is then empirically evaluated in Section 5.5. We implemented the system as explained in Section 5.6, and related works on this topic

are surveyed in Section 5.7. Section 5.8 concludes the chapter with an overview of the main downsides of using annotators to repair data extracted from the Web.

5.1 Preliminaries

A *relation* R is a set of n -tuples $\langle u_1, \dots, u_n \rangle$ where $n = \text{arity}(R)$ is the *arity* of the relation. A *schema* $\Sigma = \langle A_1, \dots, A_m \rangle$ is a tuple representing *attributes* (or *fields*) of the relation, where $m = \text{arity}(\Sigma)$ is the arity of the schema. Each attribute $A \in \Sigma$ takes values from a *domain* $\text{dom}(A)$ extended with the **null** value. Here we do not use the general database notion of domain (e.g., string, date, numeric), in our setting a domain is the set of admissible semantic values for a given attribute. As an example, the domain for the attribute MAKE of a car may contain the values “Ferrari”, “Citroën” and “Ford”. In other words, a domain is a set of possible values for a given semantic domain. We test whether the string representation $T(u)$ of a value u belongs to a given domain A via an *oracle* function ω_A , s.t., $\omega_A(T(u)) = 1$ if $T(u) \in \text{dom}(A)$ or $u = \text{null}$, and $\omega_A(T(u)) = 0$ otherwise. We say that R *agrees* with Σ if $\text{arity}(R) = \text{arity}(\Sigma)$ and that R *satisfies* Σ if it agrees with Σ and $\omega_{A_i}(u_i) = 1$, for each attribute $A_i \in \Sigma$ and corresponding attribute value u_i in any tuple $\langle u_1, \dots, u_n \rangle \in R$.

Web Wrappers. In web data extraction, a wrapper is a program extracting structured data from a collection of (semi-structured) web pages P represented as DOM trees. Differently from, e.g., information extraction, in our setting the pages in P follow the same *template*. In this work we limit ourselves to wrappers specified in XPath [25], which is the most widely used language for web data extraction and wrapper definition.

Definition 1. An XPath wrapper W is a set of pairs $\{\langle L_1, \chi_1 \rangle, \dots, \langle L_m, \chi_m \rangle\}$, where each L_i is a label and each χ_i is an XPath expressions on DOM trees, possibly with functions, e.g., `substring-after(//b[@name='price'], '£')`. The application of a wrapper W to a collection of web pages P is denoted by $W(P)$ and produces, as an output, a relation with schema $\langle L_1, \dots, L_m \rangle$.

For simplicity, we interpret the result of an XPath expression as a single string value. When multiple records occur on a page, e.g., in the case of listings, a wrapper may also specify expressions identifying individual records and listings. More complex languages, such as, OXPath [83], even support actions to interact with specific nodes in the DOM and to specify user defined actions to navigate the website. While our approach is able to deal with such wrappers, in this thesis the focus is on the repair of single attribute expressions, thus limiting to only XPath wrappers suffices for the scope of this thesis. Our approach can be easily adapted to wrappers generating arbitrary nested lists (e.g., list of records, each with an attribute that is a list of records),

however wrapper generators providing such complex structures are extremely rare and the majority of the systems deal only with flat list of attributes.

Segmentation and Partition. Given a tuple $\mathbf{u} = \langle u_1, \dots, u_n \rangle \in R$, we denote by $\Pi(\mathbf{u})$ a tuple obtained by permuting the values of $\mathbf{u}$ and by $T(\mathbf{u})$ the string concatenation (on a suitable separator, e.g., $_$) $T(u_1) _ \dots _ T(u_n)$ ($_$ here stands for the string concatenation operator). Thus, $T(\Pi(\mathbf{u}))$ is the string computed by concatenating permuted values of $\mathbf{u}$ according to the permutation Π . A *segmentation* of a string S is a sequence $(s_1, \dots, s_k)$ of non-overlapping substrings of S . If the concatenation $s_1 _ \dots _ s_k = S$, then we say that $(s_1, \dots, s_k)$ is a *partition* of S . In other words, a partition of a string S is a way of dividing S into some non overlapping parts.

Fitness. Automatically induced wrappers often introduce misalignments between relations and schemas, i.e., *under/over segmented* or *misplaced* values. We therefore require a way to quantify the agreement between a relation and a schema.

Definition 2. The *fitness* for a tuple $\mathbf{u}$ and for a relation R w.r.t. a schema Σ and a set of oracles $\Omega = \{\omega_{A_1}, \dots, \omega_{A_{\text{arity}(\Sigma)}}\}$ are computed as:

$$f(\mathbf{u}, \Sigma, \Omega) = \sum_{i=1}^c \omega_{A_i}(T(u_i)) / d \quad f(R, \Sigma, \Omega) = \sum_{\mathbf{u} \in R} f(\mathbf{u}, \Sigma, \Omega) / |R|$$

where $c = \min \{\text{arity}(\Sigma), \text{arity}(R)\}$ and $d = \max \{\text{arity}(\Sigma), \text{arity}(R)\}$.

The normalisation factors d and $|R|$ are necessary to avoid that two relations with different number of attributes have the same fitness.

Example 8. Consider the following (misaligned) relation and the schema $\Sigma : \langle \text{MAKE}, \text{MODEL}, \text{PRICE} \rangle$:

	MAKE	MODEL	PRICE
u_1		£19k	Audi A3 Sportback
u_2		£43k	Audi A6 Allroad quattro
u_3	Citroën	£10k	C3
u_4	Ford	£22k	C-max Titanium X

Given a set of oracles $\Omega = \{\omega_{\text{MAKE}}, \omega_{\text{MODEL}}, \omega_{\text{PRICE}}\}$, the fitness of R w.r.t. Σ and Ω is 2/12 as only the two values of MAKE are in the correct position, out of 12 possible correct values. The other values are misplaced and/or under-segmented.

Given a tuple permutation Π , we can compute the fitness for a relation on the permutation of each tuple rather than on the original tuples. Formally:

$$f(\Pi(R), \Sigma, \Omega) = \sum_{\mathbf{u} \in R} f(\Pi(\mathbf{u}), \Sigma, \Omega) / |R|$$

Going back to Example 8, given the permutation $\Pi = (1, 3, 2)$ (we switch the second and third elements of each tuple), the fitness of $\Pi(R)$ increases to 8/12 as the four

values of PRICE and two values of MODEL (“C3” and “C-max Titanium X”) are now in the correct position.

Repair. The relation of Example 8 is misaligned and wrongly segmented w.r.t. to the schema but the correct information is somehow represented in the relation. In order to maximise the fitness, just re-adjusting (permuting) the fields is not enough, as two values of the column PRICE contain multiple attribute values. When a relation does not fit a schema, we repair it by re-segmenting it and adjusting the wrapper accordingly.

Definition 3. Given a relation R and a schema Σ , a Σ -repair of R is a pair $\sigma = \langle \Pi, \rho \rangle$, where Π is a permutation of the fields of R and ρ a set of pairs $\langle A_i, \mathcal{E}_{A_i} \rangle$ where $A_i \in \Sigma$, and $\mathcal{E}_{A_i}$ is a regular expression over strings. The *application* of a Σ -repair σ on a tuple $\mathbf{u}$, denoted by $\sigma(\mathbf{u})$, is the tuple $\mathbf{u}^* = \langle \mathcal{E}_{A_1}(\mathcal{T}(\Pi(\mathbf{u}))), \dots, \mathcal{E}_{A_m}(\mathcal{T}(\Pi(\mathbf{u}))) \rangle$, where $(\mathcal{E}_{A_1}(\mathcal{T}(\Pi(\mathbf{u}))), \dots, \mathcal{E}_{A_m}(\mathcal{T}(\Pi(\mathbf{u}))))$ is a segmentation of $\mathcal{T}(\Pi(\mathbf{u}))$. This notion generalises to relations, denoted by $\sigma(R)$, by applying σ to all tuples $\mathbf{u} \in R$ and generating the new relation $\sigma(R)$.

Put simply, a Σ -repair extracts values for an attribute A from a tuple $\mathbf{u}$. This is done by concatenating the (possibly permuted) values of $\mathbf{u}$, and by applying a regular expression $\mathcal{E}_A$. Strings resulting from the application of the regular expressions must be a segmentation of the tuple in order to avoid that same pieces of the tuple are assigned to different attributes.

Example 9. Consider the relation and the schema of Example 8. The following is a possible Σ -repair maximising the fitness of R w.r.t. Σ and Ω . The symbol \$ denotes the argument of the expression (in this case the string constructed concatenating the permuted tuple values), and ‘_’ is the whitespace character:

$$\begin{aligned} \Pi : & (1, 3, 2) \\ \rho : & \langle \text{MAKE}, \text{substring-before}(\$, _) \rangle \\ & \langle \text{MODEL}, \text{substring-before}(\text{substring-after}(\$, _), \text{\textasciitilde{L}}) \rangle \\ & \langle \text{PRICE}, \text{concat}(\text{\textasciitilde{L}}, \text{substring-after}(\$, \text{\textasciitilde{L}})) \rangle \end{aligned}$$

We first permute each tuple by switching the second and third element, and then we apply the regular expressions of ρ on the concatenation of the permuted tuple values. The application of σ , to, e.g., $\mathbf{u}_4$ produces the correct (segmentation and) tuple $\langle \text{Ford}, \text{C-max Titanium X}, \text{\textasciitilde{L}22k} \rangle$.

Σ -repairs are applied to wrappers in a similar way.

Definition 4. The application of a Σ -repair σ to a wrapper $\mathbf{W} = \{ \langle A_1, \chi_1 \rangle, \dots, \langle A_n, \chi_n \rangle \}$, denoted by $\sigma(\mathbf{W})$, produces a modified wrapper $\mathbf{W}' = \sigma(\mathbf{W})$ as follows:

$$\begin{aligned} \mathbf{W}' = & \{ \langle A_1, \mathcal{E}_1(\text{concat}(\chi_{\pi_1}, \dots, \chi_{\pi_n})) \rangle, \\ & \dots \\ & \langle A_m, \mathcal{E}_m(\text{concat}(\chi_{\pi_1}, \dots, \chi_{\pi_n})) \rangle \} \end{aligned}$$

where **concat** concatenates all the strings produced by the **XPath** expressions $\chi_{\pi_1}, \dots, \chi_{\pi_n}$ in the order specified by the permutation $\Pi = (\pi_1, \dots, \pi_n)$.

5.2 Maximal Joint Repair: Atomic Misplacement

Definition 5. Let Σ be a schema, Ω a set of oracles for Σ , R a relation, and W a wrapper s.t. $W(P)=R$ for some collection of web pages P . We say that a Σ -repair σ is a *maximal joint repair* for R and W w.r.t. Σ and Ω , if $\sigma(W(P))=\sigma(R)$ and σ maximises the fitness of R w.r.t. Σ and Ω , i.e., $f(\sigma(R), \Sigma, \Omega)$ is maximum.

As an example, the Σ -repair of Example 9 is a maximal joint repair as the fitness increases from $2/12$ to (its maximum) 1 (all 12 values are placed under the correct attribute in the new relation $\sigma(R)$).

In this section we establish the complexity of the problem when relations have only *atomic misplacements*, i.e., where all attribute values are either entirely misplaced or, if they have been over-segmented, their fragments are in adjacent fields in the relation (and in the correct consecutive order).

Relations at the bottom of the page show potential errors in wrapper generated relations. In R_1 both MODEL (“A3 Sportback”) and PRICE (“£19k”) are entirely misplaced, therefore this relation has only atomic misplacements. Also R_2 has only atomic misplacements, since MODEL has been over-segmented but its fragments are in adjacent fields and in the correct consecutive order (token “Sportback” appears after token “A3” and they both belong to MODEL). In R_3 instead, the value for MODEL is over-segmented, but its fragments are not in adjacent consecutive order in the relation (there is PRICE in between “A3” and “Sportback”). In this section we consider only relations with errors similar to R_1 and R_2 , while we will face the general case (R_3) in the next section.

Under this restriction, our problem is closely related to the *Table Induction Problem* (TIP) [40, 44, 65, 111], which asks to compute a segmentation of a list of strings in k parts in such a way that the values of each column minimise a given distance function. TIP is shown to be **NP-hard** via a reduction from *Multiple Sequence Alignment* [40]. An optimal approximation can be obtained in polynomial time by decomposing the distance function in such a way that the independent minimisation of each component results in the minimisation of the whole distance function. In our setting, fitness maximisation plays the same role as distance minimisation, while the presence of the oracles reduces the fitness of a relation to an aggregate of the fitness of its tuples (i.e.,

MAKE	MODEL	PRICE
Audi	£19k	A3 Sportback

(a) R_1

MAKE	MODEL	PRICE
Audi	A3	Sportback £19k

(b) R_2

MAKE	MODEL	PRICE
Audi	A3	£19k Sportback

(c) R_3

we can independently maximise the fitness of individual tuples). Therefore we can reduce the problem to maximise the fitness for a single tuple.

The problem is now to determine how hard it is to compute a maximal joint repair of a single tuple. We show that this problem can be reduced to the computation of a partition of $T(\mathbf{u})$ that, encoded as a Σ -repair, produces the maximal fitness. However, since a partition of a string must contain all the subparts of the original string by its definition, restricting to partitions raises the problem of treating “garbage” content – parts of the extracted data that are never accepted as attribute values in any segmentation and typically form part of the formatting or boilerplate code of a web page.

Example 10. Consider the following relation and the target schema $\Sigma : R(\text{BEDS}, \text{TYPE}, \text{CITY}, \text{PRICE})$.

	BEDS	TYPE	CITY	PRICE
$\mathbf{u}$		2 beds apartment	Bradford	£495k

If the correct tuple is $\langle 2, \text{apartment}, \text{Bradford}, \text{£495k} \rangle$, then it is not possible to obtain such a tuple by restricting to partitions of 4 elements of $T(\mathbf{u}) = \text{“2_beds_apartment_Bradford_£495k”}$, as the garbage substring “*beds*” must be part of an element of the partition (by definition of partition). We therefore need a way to identify and eliminate garbage substrings by leveraging the oracles. Since we have to enumerate all possible partitions, this can be done by marking substrings of $T(\mathbf{u})$ that are never accepted by an oracle in any of the enumerated partitions. As an example, consider the following partitions of $T(\mathbf{u})$:

	ω_{BEDS}	ω_{TYPE}	ω_{CITY}	ω_{PRICE}
Φ_1	“2 beds”	“apartment”	“Bradford £”	“495k”
Φ_2	“2”	“beds apartment”	“Bradford £”	“495k”
Φ_3	“2”	“beds”	“apartment Bradford”	“£495k”

If a substring is (part of) an attribute value, it will be accepted by an oracle in some of the enumerated partitions, e.g., “2” will be accepted by ω_{BEDS} in Φ_2 and Φ_3 , while “beds” will never be accepted by any of the oracles in any of the partitions. Once all partitions have been enumerated, we can update them by removing all identified garbage substrings. This will collapse some of the partitions as it is the case for Φ_1 and Φ_2 above, collapsing to (“2”, “apartment”, “Bradford £”, “495k”).

Lemma 5. *Let R be a relation with atomic misplacements, Σ a schema, k the arity of Σ , and Ω a set of oracles for Σ . Given a tuple $\mathbf{u} \in R$, Algorithm 0 constructs a tuple representing a segmentation of $T(\mathbf{u})$ of maximum fitness w.r.t. Σ and Ω in polynomial time.*

Proof. Algorithm 0 first enumerates all possible partitions of $\mathbf{T}(\mathbf{u})$ of k elements. The algorithm first computes all partitions of k elements of $\mathbf{T}(\mathbf{u})$, with $k = \{1, \dots, \text{arity}(\Sigma)\}$ (line 2). As we expect (and allow) **null** values, for each partition we create a tuple $\mathbf{u}_p$ initialised with all **null** values (line 4), and then we assign to the first k elements of $\mathbf{u}_p$ the k elements of the considered partition (lines 5-6). Eventually $\mathcal{U}$ is a set of tuples, where each tuple represents a possible partition of $\mathbf{T}(\mathbf{u})$ (remaining tuple elements are filled with **null** values). A partition of k elements for a string S is computed recursively in Algorithm 1. Since by its definition each partition must contain at least one character, the 1-element partition consists of one element equal to S (lines 1-2). A partition of k elements is computed as follows: we assign to the first element of the partition the first y characters of the string ($y = \{1, \dots, \text{length}(S) - k + 1\}$) (line 5), and computing the partitions of $k - 1$ elements on the rest of the string (line 6). The partition is the union of the first element containing the first y characters with all the partitions of $k - 1$ elements (lines 7-8). Once the partitions have been enumerated and a tuple has been constructed for each of them, algorithm **Naïve-Repair** tests each element of each tuple against the oracles, and stores those (garbage) strings that are never accepted by any oracle in any of the tuples (lines 8-12). Eventually the algorithm computes the tuple that maximises the fitness without garbage substrings and returns it (lines 13-20). The operation $\mathbf{u} \setminus T_{\text{garbage}}$ deletes all the garbage strings in T_{garbage} from any element of $\mathbf{u}$, as illustrated in Example 10. The returned tuple $\mathbf{u}_{\text{max}}$ is the one of maximum fitness representing a segmentation of $\mathbf{T}(\mathbf{u})$.

Under the assumption of atomic misplacements, there is no need to permute values of $\mathbf{u}$ because the correct value of an attribute will, in the worst case, be split across adjacent fields and in the correct consecutive order. In other words the permutation order Π is not important, as we can use the tuple $\mathbf{u}$ without permuting its values. The number of different partitions is therefore equivalent to the number of ways a $\mathbf{T}(\mathbf{u})$ can be partitioned into $k = \text{arity}(\Sigma)$ parts. Since the number of characters per tuple n is usually much larger than k , the number of different partitions is polynomial in n and described by the binomial coefficient $\binom{n+k-1}{k-1} \sim \frac{n^k}{\sqrt{k \cdot k^k}}$ [92]. Therefore, the algorithm runs in polynomial time in $\text{arity}(\Sigma)$ and in the maximum number of characters that can appear in $\mathbf{T}(\mathbf{u})$. $\square$

Once the tuple representing the optimal segmentation has been computed, we still have to construct the expressions matching the correctly segmented content in the relation. This can be done using the values of the tuple elements as positive examples. Indeed an f_{XPATH}^Σ expression that exactly matches the correctly segmented values in each tuple can always be naïvely constructed by: (i) selecting the tuple with the content to be matched, and (ii) matching the desired value using substring functions that use the content before and/or after the correct value as context. The expression segmenting an attribute is then the disjunction of the expressions computed for each tuple.

Algorithm 0: Naïve-Repair.

input : $T(u)$ – string tuple
input : Ω – ensemble of oracles
input : Σ – schema of the relation
output: u_{max} – max segmentation

```

1  $\mathcal{U} \leftarrow \emptyset$ ;
2 foreach  $i \in \{1, \dots, \text{arity}(\Sigma)\}$  do
3   foreach  $\phi_i \in \text{Partition}(T(u), i)$  do
4      $u_p \leftarrow \text{Tuple}(\text{null}, \text{arity}(\Sigma))$ ;
5     foreach  $p_j \in \phi_i$  do
6        $u_p[j] \leftarrow p_j$ ;
7      $\mathcal{U} \leftarrow \mathcal{U} \cup \{u_p\}$ ;
8  $T_{garbage} \leftarrow \emptyset$ ;
9 foreach  $u \in \mathcal{U}$  do
10   foreach  $u \in u$  do
11     if  $\nexists \omega \in \Omega \mid \omega(T(u)) = 1$  then
12        $T_{garbage} \leftarrow T_{garbage} \cup \{T(u)\}$ ;
13  $u_{max} \leftarrow \text{null}$ ;
14  $w_{max} \leftarrow -1$ ;
15 foreach  $u \in \mathcal{U}$  do
16    $w \leftarrow f(u \setminus T_{garbage}, \Sigma, \Omega)$ ;
17   if  $w > w_{max}$  then
18      $w_{max} \leftarrow w$ ;
19      $u_{max} \leftarrow u \setminus T_{garbage}$ ;
20 return  $u_{max}$ 

```

Algorithm 1: Partition.

input : S – string
input : k – number of partition elements
output: Φ – k -partitions of S

```

1 if  $k = 1$  then
2   return  $\{S\}$ ;
3  $\Phi \leftarrow \emptyset$ ;
4  $n \leftarrow \text{length}(S)$ ;
5 foreach  $y \in \{1, \dots, n - k + 1\}$  do
6    $\Phi_{k-1} \leftarrow \text{Partition}(\text{substring}(S, y + 1, k - 1))$ ;
7   foreach  $\phi_{k-1} \in \Phi_{k-1}$  do
8      $\Phi \leftarrow \Phi \cup \{(\text{substring}(S, 1, y), \phi_{k-1})\}$ ;
9 return  $\Phi$ 

```

Example 11. Consider again the relation R and the schema Σ of Example 8. The following are the optimal tuples obtained by executing Naïve-Repair on every tuple of R , and by mapping the elements of a tuple to the corresponding attribute.

ω_{MAKE}	ω_{MODEL}	ω_{PRICE}
$\langle \text{Audi} \rangle$	$\langle \text{A3 Sportback} \rangle$	$\langle \text{£19k} \rangle$
$\langle \text{Audi} \rangle$	$\langle \text{A6 Allroad quattro} \rangle$	$\langle \text{£43k} \rangle$
$\langle \text{Citroën} \rangle$	$\langle \text{C3} \rangle$	$\langle \text{£10k} \rangle$
$\langle \text{Ford} \rangle$	$\langle \text{C-max Titanium X} \rangle$	$\langle \text{£22k} \rangle$

A naïvely constructed f_{XPATH}^Σ expression for, e.g., MAKE is:

```

matches($, £19k_Audi_A3_Sportback) and
substring-after(substring-before($, A3_Sportback), £19k_)
or
...
matches($, Ford_£22k_C-max_Titanium_X) and
substring-before($, £22k_C-max_Titanium_X))

```

The above repairs work well for the input example, however they pivot on the content

of the tuples and they are thus highly dependent on the input data. We will show later on in this chapter how to avoid this overfitting problem.

Given the result of Lemma 5 and the fact that a Σ -repair for a relation with only atomic misplacements can always be constructed in polynomial time from the output of Naïve-Repair as illustrated above, we can now derive the following:

Theorem 2. *Given a relation R with atomic misplacements, a wrapper W that generated R , a schema Σ and a set of oracles Ω for Σ , it is possible to construct a maximal joint repair for R and W w.r.t. Σ and Ω in polynomial time.*

5.3 Maximal Joint Repair: NP-Completeness

The interaction of attribute misplacements and over-segmentation makes computing an optimal solution intractable, since we have to find the right permutation of elements in the input tuple that, once segmented, produces the maximal fitness.

Example 12. Consider the following relation, having the same schema of Example 8, where values have been both wrongly segmented and misplaced.

	MAKE	MODEL	PRICE
$\mathbf{u}$	Ford C-max Titanium	£22k	X

Without permuting the fields it is not possible to maximise the fitness, since the over-segmented and misplaced value “*C-max Titanium X*” cannot be tested against Ω_{MODEL} . In order to re-construct the segmentation of maximal fitness, we need to switch the values of MODEL and PRICE, so that the value “*C-max Titanium X*” is over-segmented but its fragments are in adjacent fields and in consecutive order.

We can easily accommodate non atomic misplacements in Algorithm 0 by enumerating all of the possible permutations of elements of $\mathbf{u}$, and then apply the algorithm on each computed permutation to choose the partition of maximal fitness. This is unnecessary in table induction where the content is expected to be wrongly segmented but in the correct adjacent order. We now prove that the corresponding decision problem is NP-complete.

Theorem 3. *Let R be a relation, Σ a schema, and Ω a set of oracles for Σ . Given a tuple $\mathbf{u} \in R$ and a score w , checking whether there exists a permutation $\Pi(\mathbf{u})$ and a segmentation of $\mathbf{T}(\Pi(\mathbf{u}))$ that produces a tuple of fitness $\geq w$ w.r.t. Σ and Ω is NP-complete.*

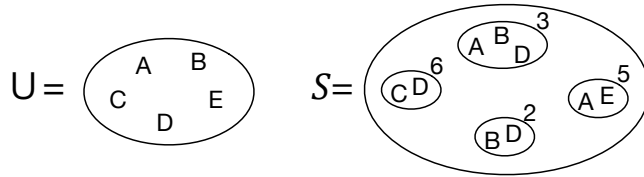
Proof. We prove the NP hardness via a reduction from the *Weighted Set Packing* problem (WSP). Given a set U and a family $\mathcal{S}$ of subsets of U , a packing is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets such that all sets in $\mathcal{C}$ are pairwise disjoint. If we assign a weight $w(S)$ to

each element S of $\mathcal{S}$, the problem of determining if there exists a $\mathcal{C}_w$ of weight $\sum_{S \in \mathcal{C}_w} w(S)$ equal than or greater to w is **NP-complete**. For any instance of WSP, we construct an instance of our problem as follows. Each element of U for WSP is taken as a different element of $\mathbf{u}$. Each set S in $\mathcal{S}$ is mapped to a string x_S by considering the string representation of elements in U and concatenating them (in any order). We then define an oracle ω such that $\omega(x_S) = w(S)$ for each S in $\mathcal{S}$. Note that the fact that oracles return a positive real value does not affect the polynomial of the reduction.

If there is a solution to WSP, then there is a solution to our problem with fitness $\geq w$. Suppose $\mathcal{C}_w$ is the solution to WSP. For each $S \in \mathcal{C}_w$, we add the corresponding x_S as an element to tuple $\mathbf{u}_{max}$ representing a segmentation of $T(\Pi(\mathbf{u}))$. Since elements in $\mathcal{C}_w$ do not share any common element, elements of $\mathbf{u}_{max}$ do not share any element of $\mathbf{u}$, therefore there exists a permutation Π such that $\mathbf{u}_{max}$ is the tuple representing a segmentation of $T(\Pi(\mathbf{u}))$, and the fitness of $\mathbf{u}_{max}$ is equal to the weight of $\mathcal{C}_w$. $\mathbf{u}_{max}$ is a solution to our problem.

We then show the other direction – if there is a solution to our problem, then there is a subfamily $\mathcal{C}$ with weight $\geq w$. Suppose $\mathbf{u}_w$ is the tuple representing a segmentation of $T(\Pi(\mathbf{u}))$ with fitness $\geq w$. For each element $\mathbf{u}$ of $\mathbf{u}_w$, we consider the string representation $x_S = T(\mathbf{u})$. We map back x_S to the element S of $\mathcal{S}$, and we add S to the subfamily $\mathcal{C}$. Since string representation of elements in $\mathbf{u}_w$ are non-crossing ($\mathbf{u}_w$ is a tuple representing a segmentation), then such strings do not share any element of $\mathbf{u}$. Thus elements in $\mathcal{C}$ are pairwise disjoint, and the weight of $\mathcal{C}$ is equal to the fitness of $\mathbf{u}_w$. $\mathcal{C}$ is a solution to WSP.

Membership in **NP** is straightforward. Given a tuple $\mathbf{u}_{input}$ and a weight w , checking whether the string representations of elements in $\mathbf{u}_{input}$ are non-crossing and that the fitness of $\mathbf{u}_{input}$ is w or more can be done in polynomial time. $\square$



Example 13. We now give an example of how WSP can be reduced to our problem. Suppose to have the set U and the family $\mathcal{S}$ of subsets of U (with weights) shown above. A solution to this instance is $\{\{A, E\}, \{C, D\}\}$ as the sum of their weights is 11 and these subsets are pairwise disjoint. From WSP we create an instance of our problem as follows: $\mathbf{u} = \langle A, B, C, D, E \rangle$ and we construct a function oracle w such that $w(\text{'ABD'}) = 3$, $w(\text{'AE'}) = 5$, $w(\text{'CD'}) = 6$, $w(\text{'BD'}) = 2$. w returns 0 for **null** and -1 for any other string. The tuple $\mathbf{u}_{max} = \langle \text{'AE'}, \text{'CD'}, \text{'null'}, \text{'null'}, \text{'null'} \rangle$ is a tuple representing a segmentation of some $T(\Pi(\mathbf{u}))$ and has maximal fitness. Indeed if we consider the

permutation $\Pi = (1, 5, 3, 4, 2)$, this generates the string $T(\Pi(\mathbf{u})) = \text{'AECDB'}$, of which $\mathbf{u}_{max}$ represents a segmentation. The fitness of $\mathbf{u}_{max}$ is 11 and there is not any other tuple with higher fitness. $\mathbf{u}_{max}$ is a solution to our problem.

5.4 Approximating Wrapper and Data Repair

Section 5.2 and Section 5.3 showed the complexity of computing a maximal joint repair. The former showed that an optimal solution can be computed in polynomial time when relations have only atomic misplacements, the latter proved the NP-completeness of the general case. We will show in Section 5.5 that wrappers tend to generate relations with atomic misplacements, hence our focus is only on these types of errors.

Even under the assumption of atomic misplacements, computing a joint repair remains impractical for a number of reasons. Firstly, the assumption of having perfect oracles for every attribute is unrealistic. Named entity recognition is a long standing challenge in information extraction and it has not been perfectly solved, especially for those domain-specific attributes [37, 56]. Moreover, a tuple-by-tuple repair would still require testing $\mathcal{O}(\text{length}(T(\mathbf{u}))^{\text{arity}(\Sigma)})$ alternative partitions against the oracles for each tuple $\mathbf{u}$, making it infeasible on large relations. E.g., in the BOOK domain (Section 5.5) we have roughly 2,000 records per site, 150 characters per record, and 5 attributes, i.e., $2,000 \cdot 150^5$ partitions.

Another problem is the robustness of the Σ -repair, i.e., its ability to match correct values on unforeseen data (or to avoid overfitting). This is of paramount importance for web data extraction, since we do not want to re-compute the repair at every wrapper execution.

Example 14. Consider the relation and the Σ -repair of Example 9, where the repair expression for MAKE pivots on a whitespace character. All the examples for MAKE appear before the first whitespace character, therefore the expression extracts everything that appears before the first whitespace character. If the web source publishes a new value for MAKE that contains one or more whitespaces (e.g., “Aston Martin”), then the expression would extract only a part of the correct value (e.g., “Aston”).

Clearly, the naïve construction of expressions illustrated at the end of Section 5.2 always overfits the examples and would require re-computing the repair at every wrapper execution as the produced expressions are never robust. Our goal is instead to produce a repair for the original wrapper such that future extractions on unforeseen data will produce clean values.

Approach Overview. We now introduce a general framework to *approximate* joint repairs that: (i) Relaxes the notion of oracles to allow errors. In particular, oracles are replaced by an ensemble of entity recognisers (annotators) tagging strings with *types* corresponding to attributes in Σ (Chapter 3). (ii) Under the assumption

of atomic misplacements, computes relation-wide segmentations that can be encoded in f_{XPATH}^Σ and are robust w.r.t. unforeseen data. (iii) Optimally approximates (w.r.t. fitness) maximal joint repairs constructed by **Naïve-Repair**.

Relation-wide repairs require the understanding of the underlying structure of the relation, e.g., to locate misplaced or wrongly segmented attribute values. In this respect, we observe the following: (i) Tuples are extracted from *templated* web pages, i.e., they follow a similar structure within a site modulo optional attributes, i.e., values may be omitted for some of the tuples. (ii) Wrappers make *systematic* errors, i.e., if an attribute is misplaced or wrongly segmented by a wrapper, this happens in a substantial fraction of the relation. (iii) The ensemble of annotators makes *non-systematic* errors. Annotators usually make mistakes, but same mistakes happen in a limited fraction of tuples.

Algorithm 2, given a relation R , a schema Σ and a set of annotators Ω , constructs an approximate joint repair under the assumption of atomic misplacements. The algorithm takes as input a relation R generated by the application of a wrapper W (not an input to the algorithm), and a collection Ω of annotators for values of attributes in Σ . Algorithm 2 works in three phases: *annotation*, *segmentation*, and *induction*.

5.4.1 Annotation

The input for this phase is a relation R and an ensemble of annotators Ω . For each tuple $\mathbf{u} \in R$, $T(\mathbf{u})$ is handed over to annotators in Ω producing, as an output, an annotated tuple (lines 1–4). We represent tuple annotations via a labelling function **anns** associating a tuple to a set of pairs of the form $\langle [i, j], A \rangle$, where $[i, j]$ is an integer interval (with $0 \leq i < j \leq \text{length}(T(\mathbf{u}))$) and $A \in \Sigma$. This phase produces an annotated relation R_Ω , consisting of a set of annotated tuples. We expect it to contain both incorrect and missing annotations, as the result of non perfect annotators.

5.4.2 Segmentation

The segmentation phase takes as input the annotated relation R_Ω . A straightforward way of constructing a Σ -repair of maximum fitness is to directly use the annotated values, since the fitness is determined by the number of times annotators recognise values in R_Ω . Due to noisy annotators, we first have to determine the sequences of annotation types representing the correct structure of the relation. Since extracted tuples follow the same website template and if they contain errors they are likely to contain similar errors due to the systematic behaviour of the wrapper, this phase aims at re-constructing the correct attribute sequence that is to be found on each tuple. Our approach takes inspiration from [28], where a set of XML documents are given as example to re-construct the DTD and schema used to generate these documents. In [28] authors consider each document as a positive example, without errors. In our

setting instead most of the annotations are to be considered correct, with the presence of some non-systematic errors.

Algorithm 2: Repair.

```

input :  $R$  – relation instance, set of tuples
input :  $\Omega$  – ensemble of annotators
input :  $\Sigma$  – schema of the relation
input :  $t_{flow}$  – flow network threshold
input :  $t_{regex}$  – regex induction threshold
output:  $\rho$  – set of pairs  $\langle A, \mathcal{E}_A \rangle$ 
1  $R_\Omega \leftarrow \emptyset$ ; // Phase 1: Annotation
2 foreach  $u \in R$  do
3    $anns(u) \leftarrow \text{annotate}(u, \Omega)$ ;
4    $R_\Omega \leftarrow R_\Omega \cup \{(u, anns(u))\}$ ;
5  $R_{copy} \leftarrow R_\Omega$ ; // Phase 2: Segmentation
6  $cover \leftarrow 0$ ;  $S \leftarrow \emptyset$ ;
7 repeat
8    $\langle V, \lambda, E, w \rangle \leftarrow \text{FlowNetwork}(R_{copy})$ ;
9    $s \leftarrow \text{MaxFlowSequence}(\langle V, \lambda, E, w \rangle)$ ;
10  if  $\nexists a, b \in s \mid a = b$  then
11     $S \leftarrow S \cup s$ ;
12  foreach  $anns(u) \in R_{copy}$  do
13    if  $\text{subSequence}(anns(u), s)$  then
14       $cover \leftarrow cover + 1$ ;
15       $R_{copy} \leftarrow R_{copy} \setminus \{(u, anns(u))\}$ ;
16 until  $cover \geq (|R| \cdot t_{flow})$ ;
17 foreach  $anns(u) \in R_\Omega$  do
18   foreach  $\langle [i, j], A \rangle \in anns(u)$  do
19     if  $\nexists s \in S \mid A \in s$  then
20        $anns(u) \leftarrow anns(u) \setminus \{\langle [i, j], A \rangle\}$ ;
21  $\rho \leftarrow \emptyset$ ; // Phase 3: Induction
22 foreach  $A \in \Sigma$  do
23    $\rho \leftarrow \rho \cup \{\langle A, \text{regexInduction}(A, R_\Omega, t_{regex}) \rangle\}$ ;
24 return  $\rho$ 

```

This can be encoded as the problem of computing max-flows in a network $N(V, E)$ having the following properties:

1. Each node $v \in V$ represents an annotation type $A \in \Sigma$ in a given context. The context identifies the sequence of annotations seen before A in the same tuple. Two special nodes, i.e., SRC (source) and SINK nodes represent the beginning and the end of the tuple respectively.
2. There exists a path $\langle \text{SRC}, v_{A_1}, \dots, v_{A_k}, \text{SINK} \rangle$ in N if and only if there exists a tuple $u \in R_\Omega$ such that we observe a sequence of annotations $(a_{A_1}, \dots, a_{A_k})$ in u , where a_{A_i} is an annotation of type $A \in \Sigma$ and i is the position in the sequence.
3. Given an edge $e(v_{A_i}, v_{A_{i+1}}) \in E$, its capacity is the sum of all the annotations that would be preserved by selecting all the tuples $u \in R_\Omega$ s.t. we can observe a

sequence of annotations $(a_{A_1}, \dots, a_{A_i}, a_{A_{i+1}}, \dots, a_{A_k})$ in $\mathbf{u}$.

Algorithm 3: Flow Network.

```

input :  $R_\Omega$  – annotated relation
output:  $\langle V, \lambda, E, w \rangle$  – flow network
1  $V \leftarrow \{\text{SRC}, \text{SINK}\};$ 
2  $E, \lambda, w \leftarrow \emptyset;$ 
3 foreach  $\mathbf{u} \in R_\Omega$  do
4    $ann\# \leftarrow |\text{anns}(\mathbf{u})|;$ 
5    $A_{seen} \leftarrow (); v_{prev} \leftarrow \text{SRC};$ 
6   foreach  $\langle [i, j], A \rangle \in \text{anns}(\mathbf{u})$  do
7      $v \leftarrow \{v \in V \mid \lambda(v) = \langle A, A_{seen} \rangle\};$ 
8     if  $v = \text{null}$  then
9        $v \leftarrow \text{new Node};$ 
10       $V \leftarrow V \cup \{v\};$ 
11       $\lambda \leftarrow \lambda \cup \{(v, \langle A, A_{seen} \rangle)\};$ 
12       $E \leftarrow E \cup \{(v_{prev}, v)\};$ 
13       $w \leftarrow w \cup \{(\langle v_{prev}, v \rangle, 0)\};$ 
14       $w(\langle v_{prev}, v \rangle) \leftarrow w(\langle v_{prev}, v \rangle) + ann\#;$ 
15       $A_{seen} \leftarrow (A_{seen}, A);$  // identify the context
16       $v_{prev} \leftarrow v;$ 
17   if  $\langle v_{prev}, \text{SINK} \rangle \notin E$  then
18      $E \leftarrow E \cup \{(v_{prev}, \text{SINK})\};$ 
19      $w \leftarrow w \cup \{(\langle v_{prev}, \text{SINK} \rangle, 0)\};$ 
20      $w(\langle v_{prev}, \text{SINK} \rangle) \leftarrow w(\langle v_{prev}, \text{SINK} \rangle) + ann\#;$ 
21 return  $\langle V, \lambda, E, w \rangle$ 

```

Paths on the network are possible attribute structures (sequence) for R . Incorrect annotations lead to networks where multiple nodes are labelled with the same attribute or to paths having a small flow capacity. We say that a path is *valid* if and only if it does not contain two nodes with the same attribute label.

Algorithm 3 constructs the network starting from an annotated relation. It iterates over all tuples (line 3) and their annotations (line 6), creating a new node whenever it observes a new annotation type or an existing one but in a different context. A suitable labelling function λ associates each node to a pair $\langle A, (A_1, \dots, A_k) \rangle$ denoting the type of the node and its context (line 11), context represented as the sequence of types observed in $\mathbf{u}$ before A (line 15). For any annotation A in the tuple, the context is updated as the sequence of annotations seen before A in the tuple (the context is empty for the first annotation in the tuple – line 5). The algorithm checks if the network contains a node with label A and context equal to the current context (line 7), and it creates a new node if the network does not contain such a node (lines 8-13). The weight w of an edge (v_i, v_{i+1}) is the total number of annotations we observe if we consider the sequence $(\text{SRC}, \dots, v_i, v_{i+1})$ (line 4). The algorithm stores a previous node (initialised as SRC at the beginning of each tuple in line 5), and for each current annotation increases the edge weight between the previous node and the node created by the current annotation of the total number of annotations in the tuple (line 14). The

previous node is then updated with the current node (line 16). Eventually the edge weight between the last annotation in the tuple and the node SINK is also incremented (line 20). Each of these edges is first created with a 0 weight if it did not exist in the network (lines 12-13 and lines 18-19). The procedure is repeated for each tuple, where at the beginning of each tuple the context becomes empty and the previous node becomes SRC.

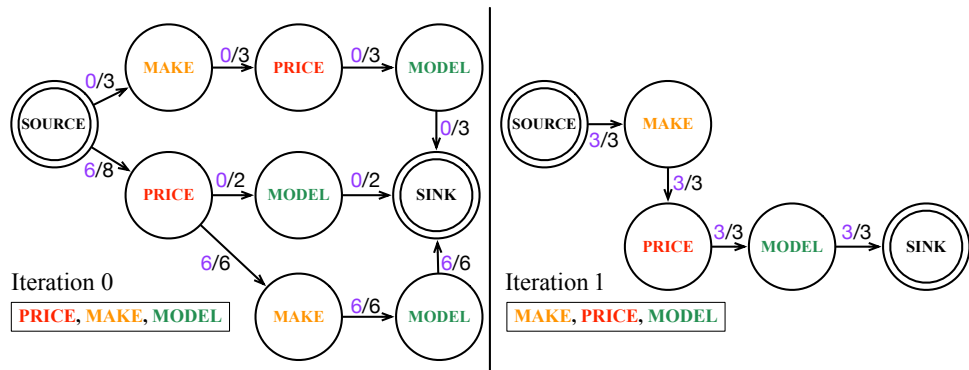
Algorithm 2 uses the computed network to compute the max-flow sequences S . For the max-flow computation, we adopted the Dinitz blocking flow algorithm. This choice is purely to ease the implementation. Other more efficient algorithms can be similarly adapted to our case [90]. We first compute augmenting paths with Dinitz’s algorithm (line 9), then we select only valid paths, i.e., paths where there are not two nodes with the same attribute label (lines 10–11). Dinitz’s algorithm runs in $\mathcal{O}(|V|^2 \cdot |E|)$. The number of computed sequences depends on the number of tuples that are *covered* by each sequence. In particular, we iterate over the following steps: (i) Compute the max-flow sequence s in the network. (ii) Remove from R_Ω all annotations matching s or one of its subsequences and recompute the network (lines 12–15). The iteration continues until the number of tuples that are covered by some computed sequences exceeds a given threshold $t_{flow} \in [0, 1]$ (line 16). The output of the algorithm is a ranking of attribute sequences by number of covered tuples. The max-flows computation on a network aims to reconstruct (i) the original (valid) attribute sequence of the original website, (ii) that maximises the total number of preserved annotations (since the majority of annotations are to be considered correct).

Example 15. Consider the relation of Example 8, with target schema $\Sigma = \{\text{MAKE}, \text{MODEL}, \text{PRICE}\}$ and annotators $\Omega = \{\omega_{\text{MAKE}}, \omega_{\text{MODEL}}, \omega_{\text{PRICE}}\}$. The following is a possible annotated relation R_Ω , having ω_{MAKE} and ω_{MODEL} incomplete, i.e., they miss the values “Citroën” and “Allroad quattro”.

u_1		£19k	Audi A3 Sportback
u_2		£43k	Audi A6 Allroad quattro
u_3	Citroën	£10k	C3
u_4	Ford	£22k	C-max Titanium X

We walk through the construction of the network on each tuple. When analysing tuple u_1 the network contains only nodes SRC and SINK and no edges. v_{prev} is set to SRC and the context to empty. u_1 contains annotations (in order) PRICE, MAKE, MODEL. First annotation is PRICE with context empty. A node v_1 with this label and context does not exist, therefore is created. The edge between SRC (previous node) and v_1 (current node) is created with capacity 3 (number of annotations in u_1). v_1 becomes the current node and the context is now (PRICE). Next annotation is MAKE. A node v_2 with this label and context is created, since it did not exist in the network. The edge between v_1 and v_2 is created with capacity 3. Previous node is updated

to v_2 and context to (PRICE,MAKE). Last annotation is MODEL. A node with label MODEL and context (PRICE,MAKE) is created as v_3 and the edge between v_2 and v_3 is created with capacity 3. Eventually an edge between the last node (v_3) and SINK is created with capacity 3. The iteration on the first tuple is now terminated. The second tuple contains the same annotations sequence of the previous one, therefore each annotation has same label and same context of annotations seen in u_1 . This tuple does not introduce any new node, but every edge created in the iteration of u_1 is incremented of 3 (number of annotations in u_2). u_3 has two annotations: PRICE, MODEL. First annotation is PRICE, with empty context. Such a node already exists in the network (v_1), therefore only the edge between SRC and v_1 is incremented of 2. Next annotation is MODEL with context equal to (PRICE). A node with label MODEL already exists (v_3) but with a different context, therefore a new node is created (v_4). Eventually edges between v_1 and v_4 and between v_4 and SINK are created with capacity 2. The last tuple contains a different annotations sequence from any inspected before, therefore each annotation leads to the creation of a new node and a new edge. The final computed network is shown below on the left. Nodes with same label are due to different contexts. For instance, we have two nodes PRICE since in the first three tuples PRICE is not preceded by any annotation (empty context), while in the last one is preceded by MAKE (context=(MAKE)). The maximum flow is 6 corresponding to the sequence s_1 : (PRICE, MAKE, MODEL) covering the first three tuples in R_Ω . Note that u_3 is also covered since it contains a subsequence of s_1 . If we set $t_{flow} > 0.75$ (the algorithm needs to cover more than 75% of the tuples to terminate), the algorithm then deletes the annotations covered by s_1 , leading to the network on the right. The sequence corresponding to the maximum flow of 3 is now s_2 : (MAKE, PRICE, MODEL). The computation stops here as the set $\{s_1, s_2\}$ covers the entire relation.



After the computation of max flow sequences on the network, Algorithm 2 removes from R_Ω all annotations that do not match any of the computed sequences (wrong annotations), leaving each tuple of R_Ω with only one annotation per attribute (lines 17–20). The output of the segmentation phase is the annotated relation R_Ω , without those annotations that disagree with the computed flow sequences S . This is exactly

where our approach is robust to noisy annotators as we will show in Section 5.5. By re-constructing the original attribute sequence of the input source we can overcome non-systematic erroneous annotations and thus improve the quality of the annotators.

Originally, we attempted to encode the problem as the computation of the most likely sequence in a memory-less Markov Chain. The chain is built by creating a node for each annotation label found on the annotated relation, and the weight from a node with label A to a node with label B is computed as the sum of all the tuples that contain an annotation of label A followed by an annotation of label B. We now give a brief example on why encoding the segmentation problem as a max flow is the correct solution instead of looking at this as a sequence labelling problem.

Assume we have the following (single-column) annotated relation where each letter represents an annotation label (i.e., a concept, e.g., Money, Model). As an example, the content of tuple u_1 is annotated with an annotation with label A, followed by annotations with labels B and C. Moreover, assume that the schema of the relation is $\Sigma = \langle A, B, C, D \rangle$.

u_1	A B C
u_2	A B C
u_3	A B C
u_4	A D
u_5	A D
u_6	B A D
u_7	B A D

We do not represent the actual text content of the tuples since it is irrelevant to the construction of the network. The most frequent sequence that also preserves the highest number of annotations is $\langle A, B, C \rangle$, preserving 9 annotations from the first 3 annotated tuples.

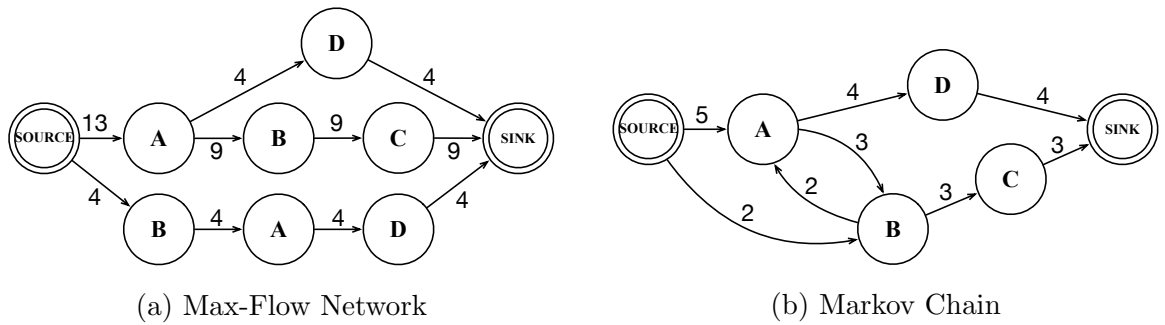


Figure 5.2: Max-Flow Network vs Markov Chain

The maxf-flow network corresponding to the example is shown on the left of Figure 5.2, and the max-flow sequence is (correctly) $\langle A, B, C \rangle$. We also show on the right of Figure 5.2 the corresponding Markov Chain built as explained above. For instance, the weight from SRC to A on the Markov Chain is 5 because 5 tuples start with an annotation of type A. Similarly, the weight of the edge between A and D is 4 because

4 tuples contain an annotation of type D right after an annotation of type A. The most likely sequence in this example is $\langle A, D \rangle$ (computed with a classic Viterbi algorithm), preserving only 4 annotations (tuples $\mathbf{u}_4$ and $\mathbf{u}_5$). Indeed the main problem with Markov Chains is their memory-less property, equivalent to forgetting the context of an annotation, which plays a crucial role in the computation of max flows.

Our max-flow sequence approach recalls in some ways the Viterbi algorithm in CRFs for text segmentation [111]. The main difference is that the Viterbi algorithm, like Markov chains, is a memory-stateless algorithm that decides the most plausible next state based only on the evidence collected in the current state. While such techniques have been widely used in text segmentation, we found them unsuitable in our case because in order to decide what is the most likely next attribute in the sequence, we need to remember which attributes we have seen before. If we need to decide what comes after the attribute MODEL, it makes a difference whether we have seen MAKE or not before MODEL. Max-flow sequences look at a sequence that is globally optimal, while Viterbi algorithm is locally optimal for each traversed state.

5.4.3 Induction

As a final step, the induction phase takes as input the modified annotated relation R_Ω and computes regular expressions segmenting attribute values. For each attribute $A \in \Sigma$, the function `regexInduction` (line 23 of Algorithm 2) produces a regular expression $\mathcal{E}_A$ matching the substrings annotated as A in $T(\mathbf{u})$, for each $\mathbf{u} \in R_\Omega$. The expression $\mathcal{E}_A$ is constructed by generalising the annotated spans.

The function `regexInduction` is explained in Algorithm 4. It takes as input a set of (positive) examples: $\{\langle T(\mathbf{u}_1), \text{anns}(\mathbf{u}_1) \rangle, \dots, \langle T(\mathbf{u}_n), \text{anns}(\mathbf{u}_n) \rangle\}$ where each example consists of the string representation of a tuple and the corresponding annotated spans. For each attribute A, the function takes the corresponding annotations in the annotated span examples (line 5) and the function `computePrefixSuffix` (line 6) looks for (i) common length prefixes and or suffixes to the annotated content, or (ii) non-content strings that repeat a constant number of times before/after the annotated content. Non-content strings consist of punctuation, whitespace, special characters, and other *un-annotated* content. Among the computed expressions, the algorithm returns the expression matching the highest number of examples (lines 10-12), provided that this number exceeds a given threshold $t_{regex} \in [0, 1]$ (line 13). The algorithm also prefers expressions where the common prefix/suffix does not overlap with annotated content (i.e., it privileges robust expressions). When it fails to produce an expression matching enough examples, the function constructs an expression that matches the disjunction of the annotated values. We call this expression *value-based* (line 14). Value-based expressions are less robust since they depend on the observed attribute values and are similar to expressions produced by `Naïve-Repair` illustrated at the end of Section 5.2. Clearly value-based expressions need to be computed at each wrapper

Algorithm 4: Regex Induction.

input : A – attribute to repair
input : R_Ω – annotated relation
input : t_{regex} – threshold
output: $\langle A, \mathcal{E} \rangle$ – tuple with best regular repair expression for A

```
1  $C \leftarrow \emptyset$ ;  
2  $c_{best} \leftarrow 0$ ;  
3  $\mathcal{E}_{best} \leftarrow \text{null}$ ;  
4 foreach  $(u, \text{anns}(u)) \in R_\Omega$  do  
5    $\langle [i, j], B \rangle \leftarrow \{ \langle [i, j], B \rangle \in \text{anns}(u) \wedge B = A \}$ ;  
6    $\mathcal{E} \leftarrow \text{computePrefixSuffix}(\text{T}(u), [i, j])$ ;  
7   if  $\mathcal{E} \notin C$  then  
8      $C \leftarrow C \cup \{(\mathcal{E}, 0)\}$ ;  
9    $C(\mathcal{E}) \leftarrow C(\mathcal{E}) + 1$ ;  
10  if  $C(\mathcal{E}) > c_{best}$  then  
11     $c_{best} \leftarrow C(\mathcal{E})$ ;  
12     $\mathcal{E}_{best} \leftarrow \mathcal{E}$ ;  
13 if  $c_{best} < (|R_\Omega| * t_{regex})$  then  
14    $\mathcal{E}_{best} \leftarrow \text{valueBased}(A, R_\Omega)$ ;  
15 return  $\langle A, \mathcal{E}_{best} \rangle$ 
```

execution, while normal regular expressions are robust enough to extract correct data even on unforeseen examples. The method is an improved version of the text-range induction algorithm used in [82]. Please refer to the online technical report for a detailed explanation of the regex induction Algorithm [10]. The output of the induction phase (and of the repair algorithm) is the set of pairs computed as described above, consisting of a Σ -repair of R w.r.t. Σ and Ω . The expressions computed in Σ -repair can be used to directly repair the relation (as SQL statements) and can also be encoded in the original wrapper such that future executions would already produce clean data.

Example 16. Consider the relation and the set of max-flow sequences of Example 15. Algorithm 2 computes the following Σ -repair (with $\Pi = (1, 2, 3)$ fixed as we only have atomic misplacements):

$$\begin{aligned} &\langle \text{MAKE}, \text{value-based}(\$, 'Audi \mid Ford') \rangle \\ &\langle \text{MODEL}, \text{substring-after}(\text{substring-after}(\$, _), _) \rangle \\ &\langle \text{PRICE}, \text{substring-after}(\text{substring-before}(\$, k_), _) \rangle \end{aligned}$$

We use the **value-based** syntax as a shortcut for a case statement in XPath. Note that the expressions for PRICE and MODEL are robust, provided that examples are representative of the corresponding domains. In contrast, the expression for MAKE is value-based (similar to the one computed in Example 11) and therefore much less robust – the expression would fail with any value of MAKE different from “Audi” and “Ford”. In this case, the result has a higher fitness than the one computed using directly the

annotations (it would not identify “*Allroad quattro*” as belonging to MODEL because such an annotation is missed).

Value-based expressions are clearly repairs that suffer of overfitting, since they highly depend on the observed data and are not likely to work on different instances. In such a case our approach does not encode these repairs back in the wrapper, but rather it uses them just to repair the extracted relation like a traditional data cleaning algorithm. Value-based expressions are the only repair means when there is not enough regularity in the input source to induce general repairs that would work on different instances. We will show in Section 5.5 that in the majority of the cases our approach is able to produce robust repairs, while it backs up to overfitted value-based expressions only when the extracted data is highly noisy or the annotators make systematic errors.

5.4.4 Optimality

We now show that Algorithm 2 produces good approximations of joint repairs if certain conditions on the error rate of the annotators are satisfied. In particular we show that our solution has always a higher fitness than the one computed using directly the annotations.

Let $\langle \nu_1, \dots, \nu_M \rangle$ be the error rates for the annotators $\langle \omega_1, \dots, \omega_M \rangle$ in Ω . Moreover, let σ_Ω be the joint repair constructed by directly using the values returned by the annotators, and σ be the joint repair produced by Algorithm 2. The following result holds.

Theorem 4. *For every relation R and wrapper W such that $W(P)=R$ for some collection of pages P , every schema Σ and every set of annotators Ω , if $\max_{\omega \in \Omega}(\nu_\omega) < (1 - \sum_{\omega \in \Omega} \nu_\omega)$, then $f(\sigma_\Omega(R), \Sigma, \Omega) \leq f(\sigma(R), \Sigma, \Omega)$.*

Proof. (Sketch) We first consider the case where both t_{flow} and t_{regex} are set to their maximum value, i.e., all max flows sequences are returned, and all examples have to be matched during the induction of expressions. It is easy to see that, returning all max-flow sequences is equivalent to provide examples for the induction that coincide with all the annotated values. Since also t_{regex} is set to the maximum, either all examples are covered, thus inducing the maximum fitness when the repair is applied, or we default to value-based expressions, that, again, match only annotated values. In this case, regardless of the error rate of the annotators, our approach will produce the same fitness as the one produced by σ_Ω .

Consider now the case at the opposite side of the spectrum, i.e., where t_{flow} is set to 0. In such a way only one max-flow sequence is returned, i.e., the one that preserves most of the annotations. In this case the error rates of the annotators can make a difference. Clearly, if an annotator makes at least as many errors as the number of tuples correctly annotated, then the repair may return the wrong sequence. However,

if the annotators find the correct sequence on a large number of tuples (bigger than the maximum number of tuples wrongly annotated by a single annotator), then the correct sequence will always be the one induced first, thus deleting incorrect annotations, and thus producing a higher fitness than the one produced by using directly the annotations (including the incorrect ones). $\square$

In other words, the joint repair computed by Algorithm 2 will always induce a fitness that is not less than the one induced by a joint repair that directly uses the annotations, provided that the number of tuples annotated with correct sequences is more than the maximum number of errors made by an individual annotator on the same relation. Theorem 4 can also be seen as a precise quantification of how many errors can be made by an annotator before being considered systematic.

5.4.5 Complexity

We now show the complexity of Algorithm 2. The algorithm produces a repair by executing the three steps consecutively, therefore we analyse each of them.

In the first phase the algorithm iterates over each tuple to annotate it. This phase is therefore polynomial in the size of the relation.

The second phase builds a flow-network and computes max flows on the network. The network has at most a number of nodes equal to the total number of annotations and a number of paths equal to the number of tuples in R_Ω . In the worst case, where each attribute has a different annotator, the number of annotations is $n \cdot \text{arity}(\Sigma)$, where n is the total number of tokens in R , i.e., each token of each tuple is annotated with all attributes. The network can therefore be constructed in polynomial time. Dinitz's algorithm runs in $\mathcal{O}(|V|^2 \cdot |E|)$, where the number of nodes is at most equal to the number of annotations (each annotation has a different context) plus nodes SRC and SINK, and the number of edges is at most equal to the number of nodes $+ |R|$. The computation of max-flow sequences always terminates since (i) R_Ω is finite, and (ii) every time a max-flow sequence is computed, the corresponding annotations are removed from R_Ω , thus reducing the number of possible valid flows in the network. In the worst case, each tuple produces a new sequence, therefore the number of sequences is bounded by the size of the relation.

In the last phase the algorithm computes regular expressions to segment annotated values. The induction of a regular expression for an attribute A requires the iteration over all the examples and the inspection of tokens occurring before and after the annotated spans. In the worst case we need to inspect each token of each tuple. Since we repeat this procedure for all the attributes, this requires $\mathcal{O}(n \cdot \text{arity}(\Sigma))$ steps, with n the number of tokens in R_Ω .

The overall algorithm is therefore polynomial in the size of the relation, total number of tokens and number of target attributes.

5.5 Experiments

Our repair approach is implemented in Java and SQL, and uses the ROSeAnn unsupervised entity recogniser framework to produce annotated relations (see Chapter 3). Within ROSeAnn we have used all of the 11 web based annotators described in Chapter 3. The vocabulary consists of ~ 350 different entity types (concepts). When an entity type was not available in the ensemble’s vocabulary, we constructed an annotator using GATE [53] framework. In this chapter, we do not focus on the accuracy of the annotators, but rather on evaluating the robustness of our approach to annotator noise.

Our experimental evaluation aims at demonstrating (i) the independence of our approach w.r.t. domains and wrapper induction systems, (ii) the robustness w.r.t. annotation errors, (iii) an increased accuracy and performance w.r.t. redundancy-based and semi-supervised methods.

5.5.1 Dataset

The dataset consists of 100 websites from 10 domains and is an enhanced version of SWDE [94], a benchmark commonly used in web data extraction. SWDE’s data is sourced from 80 sites and 8 domains: **Auto**, **Book**, **Camera**, **Job**, **Movie**, **NBA Player**, **Restaurant**, and **University**. For each website, SWDE provides collections of 400 to $2k$ *detail* pages (i.e., where each page corresponds to a single record). In order to test exhaustively the quality of web extracted data, we complemented SWDE with collections of *listing* pages (i.e., pages with multiple records) from 20 websites of **Real Estate (RE)** and **Auto** domains. SWDE’s data is sourced from American websites, while the two additional domains refer to UK sources. The final dataset consist of more than $120k$ pages, $130k$ records and more than $500k$ attribute values. Table 5.2 summarises the characteristics of the dataset.

Table 5.2: Dataset characteristics.

<i>Domain</i>	<i>Type</i>	<i>Sites</i>	<i>Pages</i>	<i>Records</i>	<i>Attributes</i>	<i>Origin</i>
Real Estate	listing	10	271	3,286	15	UK
Auto	listing	10	153	1,749	27	UK
Auto	detail	10	17,923	17,923	4	US
Book	detail	10	20,000	20,000	5	US
Camera	detail	10	5,258	5,258	3	US
Job	detail	10	20,000	20,000	4	US
Movie	detail	10	20,000	20,000	4	US
Nba Player	detail	10	4,405	4,405	4	US
Restaurant	detail	10	20,000	20,000	4	US
University	detail	10	16,705	16,705	4	US
Total	-	100	124,715	129,326	78	-

SWDE comes with ground-truth data created under the assumption that wrapper induction systems could only generate extraction rules with DOM-element granularity, i.e., without segmenting text nodes. Since modern wrapper induction systems support text-node segmentation, we have refined the ground truth accordingly. As an example, in the **Camera** domain, the original ground truth for MODEL consisted of the entire product title but the text node also provides additional attributes such as COLOUR, PIXELS, and MANUFACTURER. SWDE ground-truth contains the whole DOM-element content where the target attribute is published. We refined it by keeping only the relevant content and deleting all additional garbage.

5.5.2 Wrapper Generation Systems

We generated input extracted relations for our evaluation using four wrapper induction systems: DIADEM [82], DEPTA [156] and ViNTs [159] for listing pages, and ROADRUNNER [51] for detail pages.¹ The output of DIADEM, DEPTA, and ROADRUNNER can be readily used in the evaluation, since these are full fledged data extraction systems, supporting the segmentation of both records and attributes within listings or (sets of) detail-pages. ViNTs, on the other hand, segments rows into records within a search listing and, as such, it is not aware of attributes, only of records. We therefore post-processed its output, typing the content of lines from different records that are likely to have the same semantics. We used a simple string similarity distance metric and grouped under the same attribute the most similar extracted lines. Despite the large amount of work in the area, we were not able to find any other available wrapper-generation systems. Our joint repair approach is bound to the accuracy of the extraction system, i.e., it cannot repair data that has not been extracted.

5.5.3 Metrics

The performance of the repair is evaluated by comparing wrapper-generated relations against the SWDE refined ground truth before and after the repair. The metrics used for the evaluation are **Precision**, **Recall**, and **F_1 -Score** computed at attribute level. Both the ground truth and the extracted values are normalised (e.g., by eliminating whitespaces at the beginning and end of the values or being case insensitive), and *exact* matching between the extracted values and the ground truth is required for a hit. We use an exact matching in order to evaluate the capability of segmenting individual DOM text nodes. Related works ([32, 56, 82]) have usually used a *containment* matching, which is able to capture the quality of the data only at DOM level granularity. In this work we pose particular attention on the ability to segment individual dom nodes, hence we

¹ROADRUNNER can be configured for listings but its configuration has been optimised for detail pages.

do not report numbers on containment matching as these do not capture the desired quality of the output. In this thesis we only present the most relevant results. The full evaluation, together with the dataset, gold standard, extracted relations, the code of the normaliser and of the scorer are available at the online appendix [10]. All experiments are run on a desktop with an Intel quad-core i7 at 3.40GHz with 16 GB Ram and Linux Mint OS 17.

Table 5.3: Wrapper induction errors.

<i>System</i>	<i>Correct (%)</i>	<i>Under Segmented (%)</i>	<i>Over Segmented (%)</i>	<i>Misplaced (%)</i>	<i>Missing (%)</i>
DIADEM	60.9	34.6	0	23.2	3.5
DEPTA	49.7	44	0	25.3	6
VINTs	23.9	60.8	0	36.4	15.2
ROADRUNNER	46.3	42.8	0	18.6	10.4

5.5.4 Global Accuracy

The first two questions we want to answer are: whether joint repairs are necessary and what their impact is in terms of quality.

Table 5.3 reports, for each system, the percentage of: (i) Correctly extracted values. (ii) Under-segmentations, i.e., when values for an attribute are extracted together with values of other attributes or spurious content. (iii) Over-segmentations, i.e., when attribute values are split over multiple fields. As anticipated in Section 5.4, this rarely happens since an attribute value is often contained in a single text node. Hence our approach focuses on under-segmentation and misplacement (i.e., atomic misplacement). In this setting an attribute value can be over-segmented only if the extraction system is capable of splitting single text nodes (only DIADEM among the evaluated systems), but even in this case the splitting happens only when the system can identify a strong regularity within the text node. In our experiments we have never encountered a case of over-segmentation. (iv) Misplacements, i.e., values are placed or labeled as the wrong attribute. This is mostly due to lack of semantic knowledge or overlapping attribute domains. (v) Missing values, due to lack of regularity and optionality in the web source (ROADRUNNER, DEPTA, VINTs) or missing values from the domain knowledge (DIADEM). Note that the numbers do not add up to 100% since errors may fall into multiple categories. These numbers clearly show that there is a quality problem in wrapper induction relations and also support the atomic misplacement assumption.

Figure 5.3 shows the impact of the joint repair on our metrics. Light (resp. dark)-coloured bars denote the quality of the relation before (resp. after) the repair.

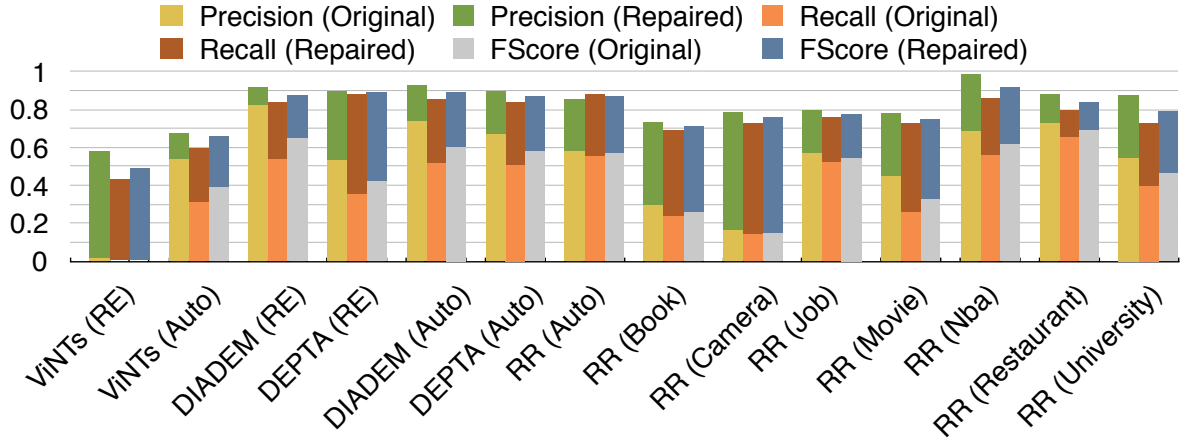


Figure 5.3: Impact of repair.

A first conclusion that can be drawn is that a repair is always beneficial. From 697 extracted attributes, 588 (84.4%) require some form of repair and the average pre-repair F_1 -Score produced by the systems is 50%. Our approach is able to induce a correct regular expression for 335 (57%) attributes, while for the remaining 253 (43%) it produces value-based expressions. We are able to repair at least one attribute in every wrapper and we can repair more than 75% of attributes in more than 80% of the cases.

Among the considered systems, DIADEM delivers, on average, the highest pre-repair F_1 -Score ($\geq 60\%$), but it never exceeds 65%. ROADRUNNER is on average worse than DIADEM but it reaches a better 70% F_1 -Score on Restaurant. Websites in this domain are in fact highly structured and individual attribute values are contained in a dedicated text node. When attributes are less structured, e.g., on Book, Camera, Movie, ROADRUNNER has a significant drop in performance. VINTs delivers the worst pre-cleaning results. This is somehow expected, since VINTs is not a classic web data extraction tool, but it is primarily designed for search engine results.

In terms of accuracy, our approach boosts F_1 -Score between 15% and 60%. Performance is consistently close to or above 80% across domains and, across systems (except for VINTs), with a peak of 91% for ROADRUNNER on NBA Player.

The following are the remaining causes of errors: (i) Not extracted missing values cannot be repaired as our approach can only use the data available in the relation. This mostly affects ROADRUNNER and VINTs. (ii) There are cases where the structure of the content is regular enough to induce (suboptimal) f_{XPATH}^Σ expressions without triggering the use of value-based expressions. This often results in a sub-optimal repair of the relation. This is more evident on domains such as Book, Camera, and Movie where some of the attributes only occur in irregular positions in text summaries or titles. (iii) On very irregular relations (e.g., those generated by VINTs), the repair essentially relies on the accuracy of the annotators, since the relation contains prevalently garbage content and it is difficult to regularly isolate target values.

Another interesting question is whether the ability to segment text nodes already during wrapper induction (as done, e.g., by DIADEM) has an impact on the quality of the repair. It turns out that the impact is very limited: we disabled text-node segmentation in DIADEM and re-run the experiment – the number of values requiring segmentations ranges from 4.7% in *Auto* to 30% in *Real Estate*. The effect is almost identical in all domains and leads to a slightly better precision, as generally the induced regular expressions in our approach are superior in precision to the ones induced during the wrapper induction, where only limited number of page samples are considered. Our approach instead inspect the entirety of the relation and can derive meaningful patterns on a broader set of examples. Such problems may be avoided with a suitable sampling strategy that would identify meaningful and representative examples during the wrapper induction phase.

Note that numbers published in this thesis differ in parts from previously published results (e.g., [32, 51]). This is due to the use of *exact* matches (in this thesis) instead of containment.

5.5.5 Per Attribute Accuracy

Another question is whether there are substantial differences in attribute-level accuracy. The top of Table 5.4 shows attributes where the repair is very effective (F_1 -Score $\simeq 1$ after repair). These values appear as highly structured attributes on web pages and the corresponding expressions repair almost all tuples. As an example, DOOR NUMBER (in *Auto*) is almost always followed by suffixes ‘*dr*’ or ‘*door*’. In these cases, the wrapper induction under-segmented the text due to lack of sufficient examples, but there is enough regularity for perfectly repair values after the extraction.

Table 5.4: Attribute-level evaluation.

<i>System</i>	<i>Domain</i>	<i>Attribute</i>	<i>Original F₁-Score</i>	<i>Repaired F₁-Score</i>
DIADEM	Real Estate	POSTCODE	0.304	0.947
DIADEM	Auto	DOOR NUMBER	0	0.984
DEPTA	Real Estate	BATHROOM NUMBER	0.314	0.973
DEPTA	Auto	MAKE	0.564	0.986
DIADEM	Real Estate	CITY	0	0.59
DEPTA	Real Estate	COUNTY	0	0.728
DIADEM	Auto	ENGINE TYPE	0	0.225
DEPTA	Auto	PRICE	0.711	0.742

For attributes such as CITY and COUNTY, despite a significant boost (59% and 72% respectively) produced by the repair, the final F_1 -Score is still low. These are irregularly structured attributes, often co-occurring with others, e.g., STATE, POSTCODE, in ways

that cannot be easily isolated by regular expressions. Despite not having syntactic regularity, these attributes are *semantically* related, e.g., COUNTY usually appears after CITY and before POSTCODE, and could be captured by extending $f_{\text{XPath}}^{\Sigma}$ with NER capabilities [37] (e.g., by specifying actions that extract everything between CITY and POSTCODE, see Chapter 7).

An exceptional case is ENGINE TYPE, where the value ‘*Petrol*’ is also recognised as COLOUR. This causes a loss of performance as it creates a systematic error in the annotated relation. Another exception is the case of PRICE in relations generated by DEPTA. DEPTA extracts large chunks of text with multiple prices among which the annotators cannot distinguish the target price reliably, resulting in worse performance.

5.5.6 Large Scale Evaluation

To prove the scalability of our approach w.r.t. an increasing number of input records and input sources, we performed a large scale evaluation on real data. We performed an extraction of restaurant chain locations in collaboration with a large social network, which provided us with 210 target websites. Some of these sources were very large, with more than 20k records. We used DIADEM as a wrapper induction system and we then applied joint repair on the generated relations. The accuracy has been manually evaluated by third-party rating teams on a sample of nearly 1,000 records of the 276,787 extracted. Table 5.5 shows **Precision** and **Recall** computed on the sample (values higher than 0.9 are highlighted in bold). Evaluation on sample records has been computed on the final data, after the extraction and the repair. In order to estimate the impact of the repair, we computed, for each attribute, the percentage of values that are different before and after the repair step. These numbers are shown in the last column of Table 5.5. Clearly, the repair is beneficial on all of the cases. For OPENING HOURS and LOCATED WITHIN, where recall is very low, the problem is due to the fact that these attributes were often not extracted at all from the source pages, thus being impossible to repair. The independent evaluation proved that our repair method can scale to hundreds of thousands of non-synthetic records. We run each source in parallel, and the repair took less than 10 seconds on the largest source (more than 20k records).

Table 5.5: Accuracy of large scale evaluation.

<i>Attribute</i>	<i>Precision</i>	<i>Recall</i>	<i>% Modified values</i>
LOCALITY	0.993	0.993	11.34%
OPENING HOURS	1.00	0.461	17.14%
LOCATED WITHIN	1.00	0.224	29.75%
PHONE	0.987	0.849	50.74%
POSTCODE	0.999	0.989	9.4%
STREET ADDRESS	0.983	0.98	83.78%

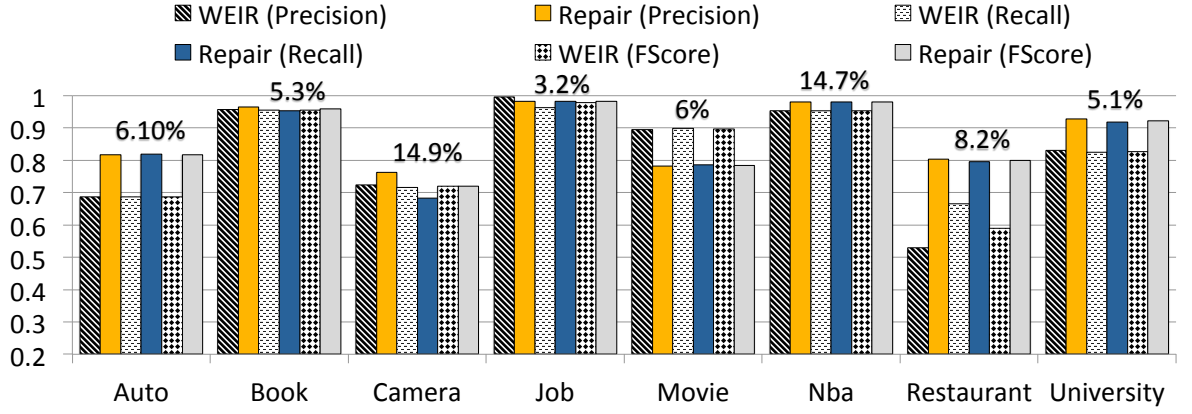


Figure 5.4: Comparison with WEIR (Redundant values)

5.5.7 Comparative Evaluation

Weir. We compare our approach against WEIR [32], a wrapper induction and data integration system that can be used to compute a joint repair of a relation w.r.t. a schema. WEIR induces wrappers by generating candidate expressions using simple heuristics and by filtering them using instance-level redundancy across multiple web sources, i.e., it picks, among candidate rules, those that consistently match similar values on different sources. We compare with WEIR as the only other similar system, TURBOSYNCR [44], is significantly older, and we were not able to obtain an implementation.

WEIR uses only redundant values for rules selection, resulting in relations with missing values (and records). We compared against WEIR on SWDE original dataset, the same one used in their evaluation [32] (using ROADRUNNER as extraction system). We evaluated WEIR and our approach in two separate settings: Figure 5.4 shows the performance of our approach and WEIR on each domain, computed on redundant records only, while in Figure 5.5 we also take into account non-redundant records.

A first observation is that redundant records are a small fraction of the whole relation, thus limiting the recall (shown on top of the bars in Figure 5.4). The results show that, if we limit the evaluation to redundant values only, our approach delivers same or better performance than WEIR. Interesting cases are **Auto**, **Restaurant** and **University**, where our approach outperforms WEIR by more than 10% in F_1 -Score. In particular, WEIR suffers from false redundancy caused by a lax similarity measure and under-segmented text nodes. The only case where WEIR performs better than our approach is in **Movie**, where the presence of multivalued attributes (such as **GENRE**) causes the selection of suboptimal max-flow sequences. If we do not limit the evaluation to redundant values only (Figure 5.5), our approach clearly outperforms WEIR in every domain, with a peak of 36% boost in F_1 -Score in **Camera**. The degradation in performance of WEIR is due to the presence of records that are published on a single source, where WEIR cannot use its redundancy-based techniques.

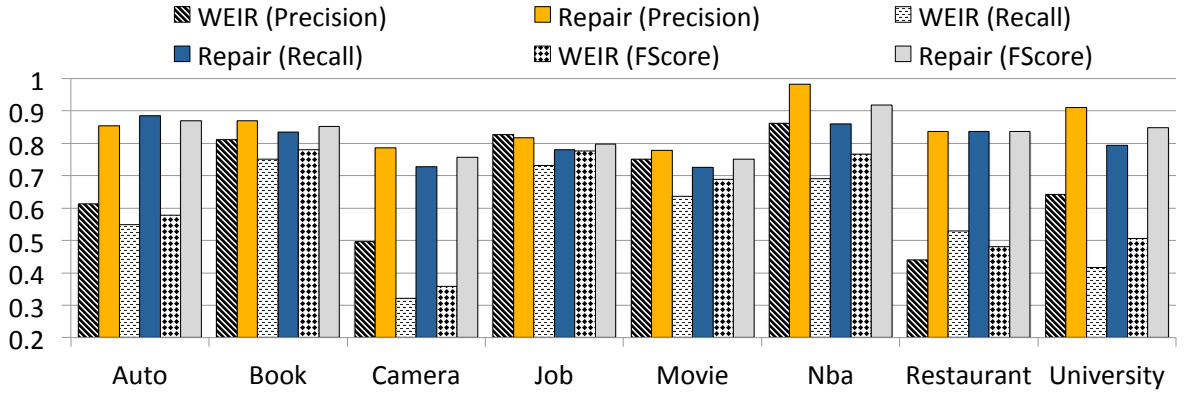


Figure 5.5: Comparison with WEIR (all values)

In terms of running time, WEIR requires an average of 30 minutes per domain whereas our approach repairs a domain in less than 2 minutes. This is due to the way WEIR exploits cross-source redundancy, i.e., instances in a source are compared against instances of all other sources. As a consequence, the running time increases with the number of sources. Our approach instead operates independently on each source, thus the running time is not bounded to the number of input sources.

On the other hand, the instance-level redundancy method by WEIR (and in principle of TURBOSYNCR) can bootstrap the creation of suitable annotators when these are not readily available.

5.5.8 Ablation Study

In this experiment we measured the impact of each phase of the joint repair computation on F_1 -Score for the most relevant scenarios (we found similar results in other scenarios but those can be inspected online).

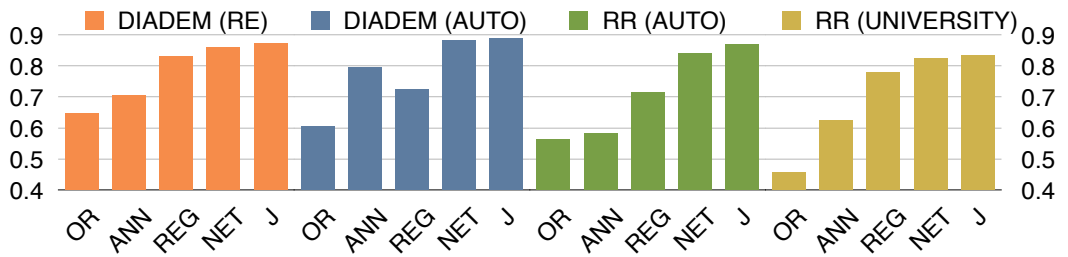


Figure 5.6: Impact of individual components

With respect to Figure 5.6, ORIGINAL (OR) refers to the original (i.e., before repair) quality of the relation, while JOINT (J) is the post-repair quality. ANNOTATOR (ANN) shows the effect of constructing the repair by directly using annotations, REGEX (REG) shows the performance when only regexes are induced (i.e., without value based expressions), NETWORK (NET) shows the repair performance by using only value based expressions computed from max-flow sequences (i.e., no regex induction).

As we can see, the direct use of annotations for repair or regex induction delivers poor results. The major contribution to the quality is the use of max-flow sequences that uncover the underlying structure of the relation and eliminate noisy annotations (column NET). Indeed columns ANN and NET use the same kind of repair (i.e., they use directly annotated values), and their difference exemplifies the improvement in annotations accuracy after the max-flow sequences computation. Regex induction is still beneficial afterwards to recover missing annotations – regex generalises patterns that can be lost by the annotators. The most striking case is STREET_ADDRESS in the Real Estate domain. The attribute is hardly recognised by annotators (accuracy around 50%), however its structure in the relation is very regular and the after-repair accuracy reaches 85%.

5.5.9 Thresholding

Another factor we wanted to measure is the effect of the thresholds t_{flow} and t_{regex} on performance.

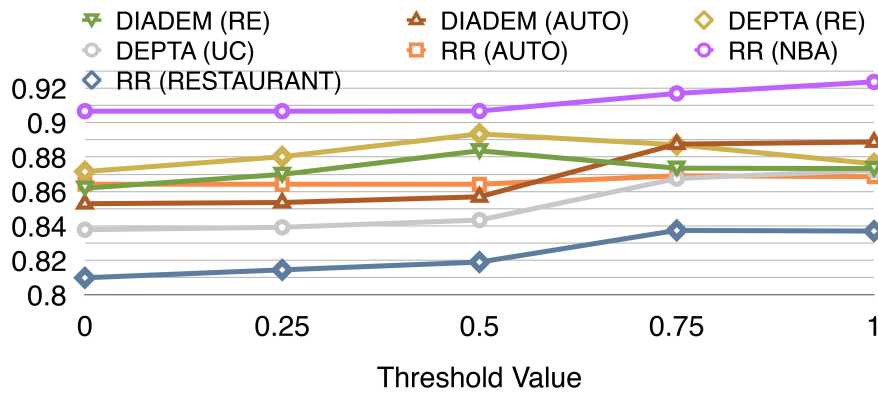


Figure 5.7: Impact of t_{flow} and t_{regex} thresholds

Figure 5.7 shows the variation of F_1 -Score for the most interesting scenarios (other scenarios report similar results). The setting of excessively low thresholds negatively impacts the performance, as it causes a premature induction of regexes that repair only a small number of records. However, there are cases, e.g., DIADEM on Real Estate, where a lower threshold helps to recover missing annotations. In domains where attributes are better structured or the annotator is more accurate, e.g., Auto, the best performance is achieved by setting a high threshold. Overall, the variation in performance is anyway limited ($\leq 3\%$) and the average best performance is obtained with a 75% threshold.

5.5.10 Annotator Accuracy

A last question we wanted to answer is on the influence of the annotators. How much the quality of the annotators can affect the quality of the repair? We gradually

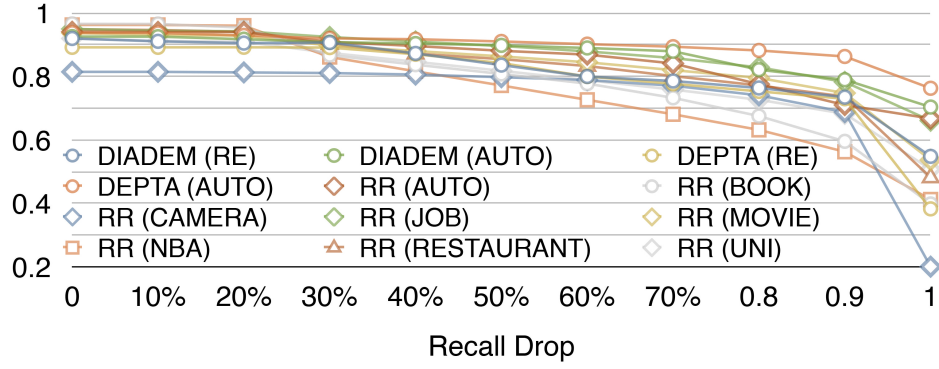


Figure 5.8: Annotator recall drop - Fixed threshold

decreased the **Recall** of our annotators by randomly eliminating a number of annotations and observing the effect on F_1 -Score while keeping a fixed regex induction threshold (0.75). To lower the effect of sampling bias, we ran the experiment 30 times with different annotation sets and took the average performance. The accuracy numbers are limited to those attributes where our approach induces regular expressions, since it is already clear that annotator errors directly reduce the accuracy of value-based expressions. This is still a significant number of attributes, i.e., $\geq 65\%$ in all cases except for ROADRUNNER on **Book** (35%), and ROADRUNNER on **Movie** (46%).

Figure 5.8 shows the impact of a drop in recall (x-axis) on F_1 -Score. As we can see, our approach is robust to a drop in recall until we reach 80% loss, then the performance rapidly decays. This is somehow expected, since the regular expressions compensate for the missing recall up to the point where the max-flow sequences are no longer able to determine the underlying attribute structure reliably.

Figure 5.9 shows the effect on F_1 -Score if we set a low regex-induction threshold (i.e., 0.1) instead. Clearly, in this case our repair approach is highly robust to annotator inaccuracy and we notice a sensible loss in performance only after 80-90% loss in recall. Indeed with a low regex threshold regular expressions are induced even with a small number of examples, and these expressions can recover the loss in recall of the annotators.

In summary, a lower regex-induction threshold is advisable when we know that

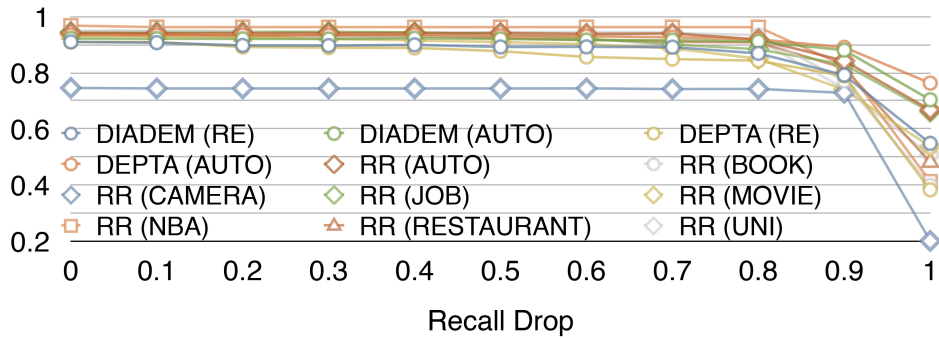


Figure 5.9: F_1 -Score variation with a threshold value of 0.1

annotators have low recall. Even involving annotators with very low accuracy, our approach is robust enough to overcome the errors introduced by such annotators. Thus we can afford to spend minimal time in tuning and implementing annotators, because eventual errors will be recovered in the segmentation phase (precision) and in the induction phase (recall).

5.6 WADaR: an Online Data Repair Platform

We implemented our joint repair approach as an online web platform: WADaR (Wrapper And Data Repair). It will be soon available at the online appendix [10].

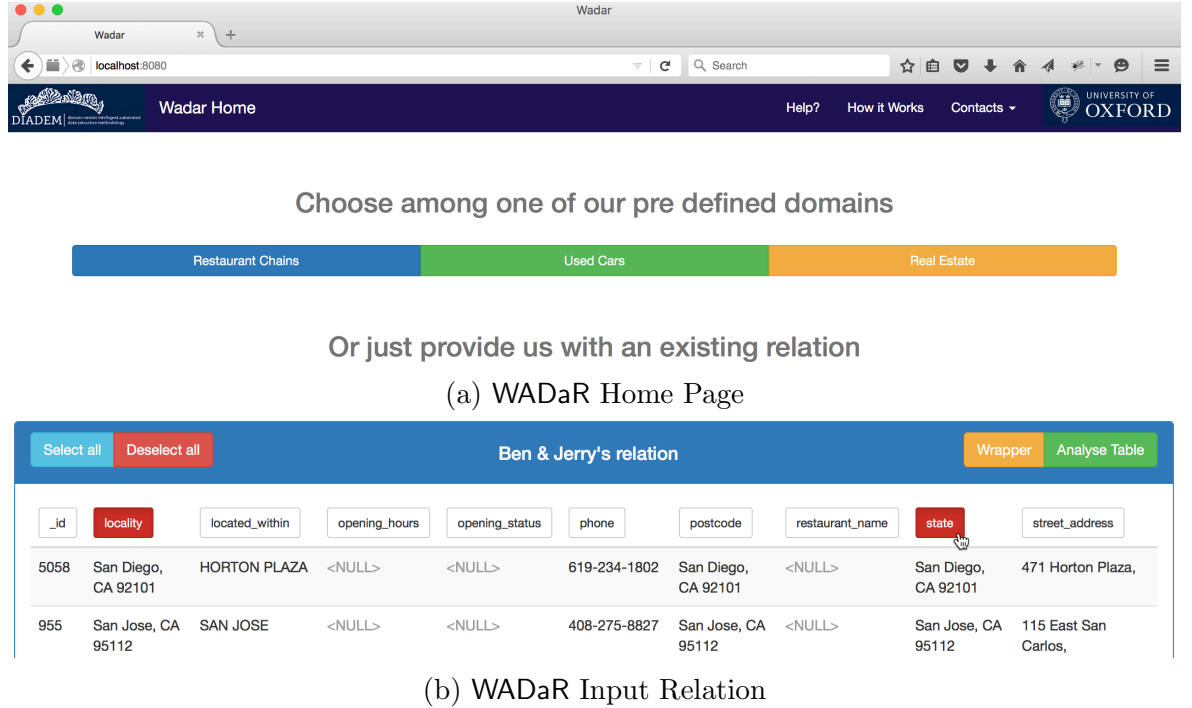


Figure 5.10: WADaR Entry Point

The WADaR home page (Figure 5.10a) is the entry point to load a web extracted relation. Users can load a relation in three different ways: (i) choose a pre-defined example from several domains; (ii) specify a custom relation (RDBs and CSVs are currently supported); (iii) directly extract a relation specifying a web wrapper. The default wrapper language we use in WADaR is OXPath [83], an extension of XPath with actions, i.e., click, fill, and extraction functions (markers).

Figure 5.10b shows a loaded relation after executing a wrapper harvesting locations, contacts and opening hours information from the US chain www.benjerry.com. As we can see from the figure, values for LOCALITY, STATE, and POSTCODE are correct but under-segmented and duplicated – they are extracted all together within attributes LOCALITY and STATE. Moreover STREET_ADDRESS contains some spurious content. The user can select on this page the attributes to be repaired and click the **Analyse Table** button to trigger the repair approach.

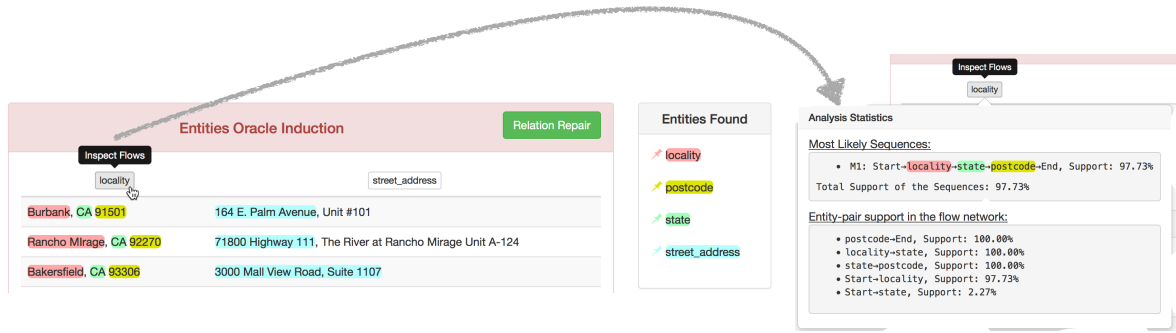


Figure 5.11: WADaR Annotations and Max-Flows Computation

The next page (Figure 5.11) shows the annotated relation after the max-flows computation. At this point WADaR has already computed the max-flow sequences and deleted those wrong annotations that do not respect any of the sequences. The annotated relation highlights annotations with a specific background colour (with a legend on the right side of the table) and users can also inspect which max-flow sequences are the result of the computation by clicking on the attribute name (right side of Figure 5.11). The user can now click the **Relation Repair** button to trigger the final repair based on the computed sequences.



Figure 5.12: WADaR Repaired Relation

The last page of the web platform shows the relation output of the whole repair algorithm (Figure 5.12). The main area of the page shows the repaired relation, where successfully repaired attributes are highlighted in green, others in red (A). If we hover the mouse over red values a small modal gives a brief explanation on why that value could not be repaired (e.g., the computed regular expression was crossing an annotated span). The user can then compare the original table with the repaired one (B), or inspect not repaired values one by one (C). The button **Statistics** (D) shows some numbers on the repair process: number of attributes repaired with regular expressions,

number of attributes repaired with value based expressions, number of values modified, and a chart showing the (predicted) error rate for each attribute before and after the repair. Furthermore, users can analyse the modified wrapper (where blue highlights show modified expressions that are result of the repair) and adjust wrapper expressions if needed (**E**).

At this point, if the user is satisfied with the outcome of the repair, she can commit or export the repaired relation (**F**). Another possibility is to re-extract the data with the modified wrapper and repeat the process until she is satisfied with the result. If the button **Extract Again** is pressed, WADaR extracts the data again with the modified wrapper and it goes back to the input relation page (Figure 5.10b).

5.7 Related Work

Wrapper Induction and Maintenance. Computing joint repairs is one of the many *maintenance* problems faced in web data extraction [39, 107, 115, 122, 126]. Wrapper maintenance techniques have, so far, focused on two problems: (i) Efficient detection of wrapper failures due to changes in the targeted web sources. (ii) Computation of the minimal (and robust) changes required to restore the wrapper to working conditions. However, classical wrapper maintenance has assumed perfect, typically human-created wrappers to begin with, with errors only being introduced over time due to change in the sources. When covering thousands or hundreds of thousands of sources with automatically or semi-supervised wrapper induction this assumption is no longer valid.

Closer in spirit to joint repairs are techniques to generate wrappers from background data [32, 44, 89, 155]. These techniques implicitly align background data and wrappers as part of the generation process. The closest works to ours are TURBOSyncER [44] and WEIR [32], which use instance-level redundancy across multiple sources to compute extraction rules on individual sites that, together, can be used to effectively learn wrappers without supervision. TURBOSyncER works more at intensional/schema level rather than instance level, hence our approach is not not entirely complementing it (differently from WEIR). An advantageous side-effect of these approaches is the construction of “compatible” relations that can be more easily integrated. Differently from TURBOSyncER and WEIR, our approach assumes the existence of an already generated wrapper to be repaired w.r.t. a target schema. From a practical point of view, both TURBOSyncER and WEIR can be adapted to compute joint repairs, however, as shown in Section 5.5.7 with a significantly worse performance than our approach due to their reliance on redundancy. Our approach also eliminates the need for re-induction of wrappers, leading to better runtime performance. It is worth noticing that both WEIR and TURBOSyncER try to solve a significant different problem, however each of them can be adapted to tackle the problem of repairing extracted data and wrappers.

Redundancy. Instance-level redundancy across web sources has been previously used in different contexts to detect and repair inconsistent data extracted from the web [32, 43, 44, 65]. Redundancy-based approaches face two main obstacles: (i) it is not always possible to leverage sufficient redundancy in every domain, (see, e.g., the number of redundant records in SWDE of Figure 5.4), and (ii) redundancy-based methods require access to a substantial number of sources that have, so far, limited their *scalability* (see, e.g., WEIR running time). Encoding the redundancy by other means, e.g., through entity recognisers and knowledge bases, has proven beneficial to circumvent the scalability problems without sacrificing generality of the approaches [43, 82]. Our approach achieves this via an ensemble of entity recognisers (Chapter 3), some of which are trained using redundancy-based methods. Despite relying on entity recognisers, our approach is highly robust to annotators noise and users can spend very little time on tuning such annotators because possible errors will be recovered by our approach (Section 5.5.10). Differently from redundancy-based approaches that usually require a pairwise-source comparison, our approach is independent from the number of sources because each of them can be run in parallel. However, exploiting redundancy-based techniques to bootstrap the construction of the annotators is an interesting future direction that we intend to investigate.

Cleaning, segmentation and alignment. Traditional data cleaning methods focus on the detection and repair of database inconsistencies using statistical value distributions [146, 154], constraints [30, 69, 88, 137], and knowledge bases [43]. Differently from our setting, cleaning methods operate on relation(s) that contain incorrect values but are assumed to be correctly segmented. Moreover, as anticipated in Chapter 4, most of data cleaning algorithms are either too expensive (rule-based and machine learning) or rely on an external source of information that limits the application to popular domains.

A closer problem to repair web extracted relations is tackled by list segmentation/table induction techniques, where the goal is to induce structured records from unstructured (i.e., wrongly segmented) lists of values. These are alternatives to the segmentation based on flow networks used in our approach. Being inspired by tagging problems common in bio-informatics and other areas, these approaches traditionally require some form of supervision. Many require an initial seed of correctly segmented records [50, 100, 109, 117, 158], while others require positive and negative examples of valid field/column values as training data [111, 149], sometimes leveraging existing knowledge bases [49] or, again, instance-level redundancy [40, 65]. One of the first successful approach in this direction is GOOGLE WEB TABLE [144] (GWT). GWT re-constructs the columns semantic by assigning each column to entity types that can be found in Freebase, thus reducing its applicability to “common-knowledge” domains. Our work shows that there is much to gain in terms of generality and performance when

redundant information is synthesised as annotators, e.g., we do not require pairwise comparison of tuples and we can deal with unpopular domains.

A more recent approach similar to GWT is the unsupervised method employed in TEGRA [40], close in spirit to our segmentation strategy. Differently from, e.g., LISTEXTRACT [65], TEGRA looks at the relation globally, circumventing the intractability of the problem by constructing approximated but optimal solutions. The key difference between TEGRA and our approach is in the distance metrics used for alignment. TEGRA’s is essentially equivalent to cross-site instance-level redundancy (carrying all its pros and cons), while our approach encodes it via attributes oracles. Another major difference of our setting w.r.t. approaches like GWT and TEGRA is the presence of misplacements and spurious content in the lists to be segmented. Both GWT and TEGRA only segment the data without eliminating spurious content (this is necessary in wrapper-generated data). It is worth mentioning that the problem of segmenting lists with spurious content was already tackled by [151] but, again, using cross-site, instance-level redundancy.

5.8 Annotators: Take the Bad with the Good

In this second part of the thesis we introduced WADaR, a system to compute joint repairs for web wrappers and generated relations. We first presented the main challenges and problems when dealing with automated web data extraction, and why traditional data cleaning/table segmentation techniques are not suitable for repairing web data and wrappers. Then we proposed an unsupervised framework that is independent on the domain and on the wrapper induction system, and that is capable of delivering high quality repairs for web extracted data and their wrappers. We showed how our approach is not only beneficial for the final quality of the data, but also how it is highly scalable by not relying on instance-level redundancy like most of previous approaches. The only supervised and domain-specific part of our system is the ensemble of annotators used to label the extracted relations.

The annotator ecosystem is becoming extremely rich and diverse, and there are many resources out there that can be used without any tuning effort (Chapter 2). But what happens when there are not available annotators for our target entities? In WADaR we implemented a noise-tolerant sequence labelling algorithm that is robust to annotators’ deficiency. This means that even when the user is forced to implement her own annotators, she can spend very little time in doing so since we do not rely on perfect annotators. On the contrary, we can still deliver high quality repairs with highly noisy annotations. Can we do even better? Can we get rid of the supervised side and repair data without any external help?

Recent efforts in web data extraction are shifting the interest to a different setting. The goal in these new approaches is to extract real-world entities, and relationships

that connect them in real life [21, 60, 135, 140]. For example, target data could store which director directed a particular movie, the foundation year of a company, or which city is capital of which country. Besides accurately describing facts from the real world, this particular kind of data carries a big amount of hidden information. Sometimes the writer of a movie is also the director. On the other hand, a city cannot be the capital of a country c_1 if it is located in a different country c_2 . In the next chapter we will present a fully unsupervised approach to discover declarative rules over entities and relationships among them. Close in spirit to traditional database constraints induction systems [17, 41, 97, 153], we will discover augmenting and repair rules by just looking at the data without any external supervision.

Part III

RDF Knowledge Bases

*“To know that we know what we know,
and to know that we do not know what we do not know,
that is true knowledge”.
— Confucius*

Chapter 6

RuDiK: Affordable Rule Discovery in Knowledge Bases

So far, the focus of this thesis has targeted classic web data extraction: we extract entities from semi-structured web sources, along with their attributes, in order to store them in some structured format. A novel research direction is recently focusing on extracting data from web documents to construct knowledge bases (KBs), in order to structurally store entities and relationships among them. Given the different nature of target data, KBs are usually not stored in relational databases. In this chapter we move the attention from classic web data extraction to KBs built from web sources. More specifically, we consider RDF KBs, the most popular format where data is stored as RDF triples.

Building large RDF KBs has become a popular trend in information extraction. Differently from classic relational databases, KBs store information in the form of triples, where a *predicate* expresses a binary relation¹ between a *subject* and an *object*. KB triples, often referred to as facts, store information about real-world entities, such as “Scott Eastwood is the child of Clint Eastwood”, “Michelle Obama is the spouse of Barack Obama”, or “Larry Page is the founder of Google”. The most popular research efforts in this direction can be currently grouped into two strands, according to the creation technique they rely on: *human crafted* KBs (FreeBase [31], Wikidata [145]), and automated *information extraction* (DBPedia [29], DeepDive [135], Yago [140], NELL [35], TextRunner [21]). Recent advances in KBs have also attracted interest from the industrial world, as is evident from investments of large companies (Facebook Graph,² Google Knowledge Vault [60], Walmart [59]).

KBs are built by extracting information from web sources. Given the abundance of web data, KBs nowadays end up being prohibitively large – the online version of WIKIDATA contains more than 1 billion facts and 300 million different entities.³ Furthermore, usually KBs do not expose a schema that clearly defines instance data. The

¹We will use the two terms “predicate” and “relation” interchangeably in this chapter

²<https://developers.facebook.com/docs/graph-api>

³<https://query.wikidata.org/>

set of predicates is unknown a-priori, and adding new predicates/facts is just a matter of inserting new triples in the KB without any schema/integrity check. As a direct consequence, the amount of incompleteness and inconsistencies in KBs is significantly larger than classic databases [141]. For example, the standard *closed world assumption* (CWA) does no longer hold in KBs [60, 84]. We cannot assume that a fact that is not stated in a KB is false, but rather we should label it as *unknown*. This is known as the *open world assumption* (OWA).

Despite being erroneous and incomplete, KBs carry such a big amount of information that they can be inspected to find additional information or constraints over existing data. In this chapter we investigate the problem of discovering *declarative rules* over RDF KBs. More specifically, we target the discovery of first-order Horn Rules constrained by some language biases. We define two different types of rules: (i) positive rules, used to enrich the KB with new facts and thus make it more complete; (ii) negative rules, used to spot logical inconsistencies and repair erroneous triples.

Example 17.

$\langle \text{Clint_Eastwood} \text{ child } \text{Scott_Eastwood} \rangle.$ $\langle \text{Barack_Obama} \text{ child } \text{Sasha_Obama} \rangle.$
 $\langle \text{Sasha_Obama} \text{ parent } \text{Barack_Obama} \rangle.$ $\langle \text{Barack_Obama} \text{ birthDate } '4/10/1961' \rangle.$
 $\langle \text{Sasha_Obama} \text{ birthDate } '10/6/2001' \rangle.$ $\langle \text{Clint_Eastwood} \text{ birthDate } '31/5/1930' \rangle.$

For example, consider the above KB that contains six triples with information about parent and child relationships and birth dates. A potential positive rule could be:

$$\text{parent}(b, a) \Rightarrow \text{child}(a, b)$$

which states that if a is parent of b , then b is child of a . If the KB states that “Clint Eastwood is the parent of Scott Eastwood”, we can infer the fact that “Scott Eastwood is the child of Clint Eastwood”. On the other hand, a negative rule may state that:

$$\text{birthDate}(a, v_0) \wedge \text{birthDate}(b, v_1) \wedge v_0 > v_1 \Rightarrow \neg \text{child}(a, b)$$

The above rule expresses the concept that b cannot be child of a if a was born after b . By instantiating the above rule on the KB, we may discover erroneous triples where a child is born before her parent. Given a target predicate in input (e.g., **child**), we look for a procedure that discovers both positive and negative rules for the given predicate.

Other than enriching and cleaning KBs, discovering declarative rules brings multiple benefits. We can discover negative facts which are by nature absent in KBs). We can help domain experts in automatically defining rules to maintain and curate Big Data industrial systems [87]. Rules can help in discovering meaningful training examples for Machine Learning tasks [129, 135].

Discovering declarative rules from data sources is a long standing research challenge. Inductive Logic Programming (ILP) was the first discipline to mine Horn Rules from

background knowledge and training examples [120]. Similarly, a significant body of work has addressed the problem of discovering constraints over relational databases. The most popular constraint families are Functional Dependencies [17, 97, 153] and Denial Constraints [41]. However, current KBs pose new challenges that make these approaches unsuitable. First, the size of modern KBs. Classic ILP systems cannot cope with large KBs and adopting database techniques would require the materialisation of all possible predicate combinations into relational tables. Second, the OWA that rules KBs. KBs contain only positive statements, and since we cannot rely on the CWA, we cannot derive negative statements as counter examples – most of classic databases approaches rely on the presence of positive and negative examples or on the CWA. Third, ILP and database techniques usually make the assumption that data is either clean or has a negligible amount of errors. We will show that KBs present a high percentage of errors and we need techniques that are noise tolerant.

Novelty. In this chapter we present RuDiK (Rule Discovery in Knowledge Bases), a novel system for the discovery of positive and negative Horn Rules over KBs. To the best of our knowledge, RuDiK is the first system capable of discovering both positive and negative rules using the same algorithm. Moreover, classic rule discovery systems assume that the KB has to fit in memory to have acceptable performance [15, 38, 75, 84]. We drop such an assumption and propose a disk-based solution that loads into memory only a small portion of the KB. This is a key point not only because we can discover rules on any size KB, but also because we can enlarge the expressive power of the considered class of rules to include smart comparisons among constants and numbers, including inequalities.

Contribution 1. We first formally define the problem of rule discovery over erroneous and incomplete KBs. The input of the problem consists of a target predicate, uniquely identified by a set of positive and negative examples. The ideal solution of the problem is a set of rules that covers all positive examples, and as few negative examples as possible. The problem can be naturally mapped to the well-known weighted set cover problem, which is proven to be **NP-complete**.

In contrast with traditional ranking of rules based on a measure of support [58, 84, 119, 133], our novel problem definition inspired by the set cover allows the identification of a subset of useful rules to be exposed to the user. The main benefit of such an approach is to relieve the user from the definition of a suitable threshold that separates good and bad rules. We will show that properly setting such thresholds is not trivial (see Section 6.4.2), and we incorporate this aspect in the problem definition.

Contribution 2. We give a $\log(k)$ approximation of the problem close in spirit to the greedy algorithm for set cover. k is the maximum number of positive examples covered by a single rule. The algorithm we propose in RuDiK alternates and combines two different phases: (i) it materialises the KB as a directed graph and discovers

rules by navigating valid paths on the graph; (ii) at each iteration it follows the most promising path in the A^* traversal fashion, allowing the pruning of unpromising paths.

Previous approaches for rule discovery assume that the KB has to fit in main memory to deliver acceptable performance [15, 38, 75, 84]. Unfortunately, KBs nowadays have sizes that can easily exceed hundreds of GBs and previous works require powerful resources to handle them. With our rules generation based on graph traversal we materialise only a small portion of the KB that is needed for the discovery, generating nodes and edges *on demand* by querying the disk whenever the navigation of a specific path is required. Moreover, the A^* graph traversal allows the pruning of unpromising paths that are therefore never materialised. Thus we significantly reduce the search space, cutting down the running time by ten times in the best case scenario.

Contribution 3. We experimentally verify the performance of rule discovery in RuDiK using both real-world and synthetic datasets (Section 6.4). The evaluation is carried out on the 3 most popular and widely used KBs, namely DBPEDIA [29], YAGO [139], and WIKIDATA [145]. We show that our algorithm is general enough to deliver both accurate positive and accurate negative rules, and that it clearly outperforms state-of-the-art systems both in final accuracy and running time. Eventually we give an example of application where our rule discovery approach gives a beneficial impact, providing useful examples that can train Machine Learning algorithms (Section 6.4.3).

Roadmap. The rest of the chapter is structured as follows. Section 6.1 introduces basic preliminaries and the discovery problem tackled by RuDiK. In Section 6.2 we present a graph-based solution to generate the universe of all possible rules, while Section 6.3 is the main contribution of the chapter: we outline an approximate solution of the problem along with some optimisation techniques. Section 6.4 empirically evaluates RuDiK, and the most relevant works are surveyed in Section 6.5.

6.1 Preliminaries and Definitions

The work of this chapter focuses on discovering rules from RDF⁴ KBs. An RDF KB is a database that represents common information through RDF triples $\langle s, p, o \rangle$, where a *subject* is connected to a *object* via a *predicate*. Triples are often called *facts*. As an example, the fact that Scott Eastwood is the child of Clint Eastwood could be represented with the triple $\langle \textit{Clint_Eastwood}, \textit{child}, \textit{Scott_Eastwood} \rangle$. RDF KB triples respect the following constraints:

- triple subjects are always *entities*, i.e., concepts from the real world;
- triple objects can be either entities or *literals*. Literals represent primitive types – numbers, dates, and strings;

⁴<https://www.w3.org/RDF/>

- triple predicates specifies real-world relationships between subjects and objects.

If we compare the KB model with object oriented programming languages, entities correspond to complex objects with properties and attributes, while literals correspond to primitive types such as integers, chars and strings.

Differently from, e.g., relational databases, KBs do not have a schema that instance data is forced to follow. The set of predicates is unknown a-priori, and adding new predicates is just a matter of inserting new triples having those predicates. Sometimes KBs include *T-Box* facts in order to define classes, domain/co-domain types for predicates, and relationships among classes. While the T-Box can be useful in order to check integrity and consistency within a KB, very often such information is incomplete or not present at all. Hence our focus in this thesis is only on the *A-Box*, the set of facts that describe instance data. On the one hand this model allows great flexibility, on the other hand the likelihood of introducing errors and inconsistencies is much higher for KBs than traditional schema-guided databases.

6.1.1 Language

Inspired by Inductive Logic Programming [120], the goal of our work is to discover first-order logical formulas in KBs. More specifically, we target the discovery of *Horn Rules*. A Horn Rule is a disjunction of *atoms* with at most one unnegated atom. When written in the implication form, Horn Rules have one of the following format:

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow B \qquad A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow \neg B$$

where $A_1 \wedge A_2 \wedge \dots \wedge A_n$ consists of the *body* of the rule (a conjunction of atoms), while B is the *head* of the rule (a single atom). The head of the rule is either unnegated (left) or negated (right). We call the former *definite clause* or more simply *positive rule*, while the latter *goal clause* or *negative rule*. In the context of KBs, an atom is a predicate connecting two variables, two entities, or an entity and a variable. For simplicity, we represent an atom with the notation $\mathbf{rel}(a, b)$, where $\mathbf{rel}$ is the name of the predicate, and a and b are either variables or entities. In the context of Horn Rules, all variables appearing in a rule are implicitly universally quantified. Given a Horn Rule r , we define r_{body} as the body of the rule, and r_{head} as the head of the rule. We define the variables appearing in the head of the rule as the *target variables*. The first rule of Example 17 shows a plausible positive rule, while the second rule is a potential negative rule. a and b are the target variables in both cases.

Furthermore we extend the language in order to include *literals comparison*. A literal comparison is a special atom $\mathbf{rel}(a, b)$, where $\mathbf{rel} \in \{<, \leq, \neq, >, \geq\}$, and a and b are both literal values except if $\mathbf{rel}$ is equal to $\neq$. In such a case a and b can be also assigned to entities, and if they are both of them must be entities of the same type. We will explain in details later on in the thesis this exception and why it is important.

Given a KB $\mathcal{K}$ and an atom $A = \mathbf{rel}(a, b)$ where a and b are two entities, we say that A *holds* over $\mathcal{K}$ iff $\langle a, \mathbf{rel}, b \rangle \in \mathcal{K}$. Given a KB $\mathcal{K}$ and an atom $A = \mathbf{rel}(a, b)$ with at least one variable, we say that A can be *instantiated* over $\mathcal{K}$ if there exists at least one entity from $\mathcal{K}$ for each variable in A such that if we substitute variables with entities in A , A holds over $\mathcal{K}$. Transitively, given a body of a rule r_{body} and a KB $\mathcal{K}$, we say that r_{body} can be instantiated over $\mathcal{K}$ if every atom in r_{body} can be instantiated.

Eventually, we introduce two language biases in order to avoid the explosion of the search space. Following the biases introduced by other approaches for rule discovery in KBs [38, 84], we introduce such restrictions in order to speed up the discovery process. Language biases introduce a compromise between the expressiveness of target rules and the complexity of the discovery problem. Therefore we define a rule *valid* iff it follows the following constraints.

Connectivity. We say that an atom A_1 can be reached by an atom A_2 of the same rule iff A_1 and A_2 share at least one variable or one entity. The connectivity constraint requires that every atom in a rule must be *transitively* reachable by any other atom in the rule.

Repetition. Every variable in a rule must appear at least twice. Since target variables already appear once in the head of the rule, the repetition constraint limited to the body of a rule requires that each variable that is not a target variable must appear at least twice, while target variables at least once.

Language restrictions obviously limit the output of the discovery algorithm to a subset of all plausible rules. We will show in Section 6.2 how these restrictions can significantly speed up the discovery process through a fast disk-based approach that leverages on these limitations. Besides efficiency purposes, the language biases are also driven by limiting rules to be human readable. In fact a rule that involves two atoms about completely disconnected entities would be hard to judge by an end user.

6.1.2 Rule Coverage

Given a pair of entities (x, y) from a KB $\mathcal{K}$ and a Horn Rule r , we say that r_{body} *covers* (x, y) if $(x, y) \models r_{body}$. In other words, given a Horn Rule $r = r_{body} \Rightarrow \mathbf{r}(a, b)$, r_{body} covers a pair of entities $(x, y) \in \mathcal{K}$ iff we can substitute a with x , b with y , and the rest of the body can be instantiated over $\mathcal{K}$. Given a set of pairs of entities $E = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ and a rule r , we denote by $C_r(E)$ the *coverage* of r_{body} over E as the set of elements in E covered by r_{body} : $C_r(E) = \{(x, y) \in E \mid (x, y) \models r_{body}\}$.

Given the body r_{body} of a Horn Rule r , we denote by r_{body}^* the *unbounded body* of r . The unbounded body of a rule is obtained by keeping only those atoms that contain a target variable and substituting these atoms with new atoms where the target variable is connected to a new unique variable. As an example, given $r_{body} = \mathbf{rel}_1(a, v_0) \wedge$

$\text{rel}_2(v_0, b)$ where a and b are the target variables, $r_{body}^* = \text{rel}_1(a, v_1) \wedge \text{rel}_2(v_2, b)$. While in r_{body} the target variables are bounded to be connected to the same variable v_0 , in r_{body}^* the target variables are not bounded to share the same variable. Given a set of pairs of entities $E = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ and a rule r , we denote by $U_r(E)$ the *unbounded coverage* of r_{body}^* over E as the set of elements in E covered by r_{body}^* : $U_r(E) = \{(x, y) \in E \mid (x, y) \models r_{body}^*\}$. Note that, given a set E , $C_r(E) \subseteq U_r(E)$. In other words, the unbounded coverage of a rule over E always contains all the elements of the coverage over E .

Example 18. Given the negative rule r and the KB $\mathcal{K}$ of Example 17, we denote with E the set of all possible pairs of entities in $\mathcal{K}$. The coverage of r over E ($C_r(E)$) is the set of all pairs of entities $(x, y) \in \mathcal{K}$ such that both x and y have the **birthDate** information and x is born after y , while the unbounded coverage of r over E ($U_r(E)$) is the set of all pairs of entities (x, y) such that both x and y have the **birthDate** information, no matter what the relation between the two birth dates is. In that specific KB, the pair $(Sasha_Obama, Barack_Obama)$ belongs to both $C_r(E)$ and $U_r(E)$, since Sasha Obama was born after his father; on the other hand the pair $(Clint_Eastwood, Barack_Obama)$ belongs only to $U_r(E)$, since Clint Eastwood was born before Barack Obama. Note that the pair $(Clint_Eastwood, Scott_Eastwood)$ does not belong neither to $C_r(E)$ nor to $U_r(E)$, since the birth date for Scott Eastwood is missing.

The unbounded coverage is essential to distinguish between missing and inconsistent information: if for a pair of entities (x, y) the **birthDate** information is missing for either x or y (or both), we cannot say whether x was born before or after y , therefore we cannot state whether the negative rule of Example 17 covers (x, y) or not. On the other hand, if both x and y have the **birthDate** information and x is born before y , we can affirm that the negative rule of Example 17 does not cover (x, y) . Given that modern KBs are largely incomplete [60, 118], discriminating between missing and conflicting information becomes of paramount importance.

We can now define the coverage and the unbounded coverage for a set of rules $R = \{r_1, r_2, \dots, r_n\}$ as the union of individual coverages:

$$C_R(E) = \bigcup_{r \in R} C_r(E) \quad U_R(E) = \bigcup_{r \in R} U_r(E)$$

Our problem tackles the discovery of positive (negative) rules for an input given predicate (target predicate). We uniquely identify a predicate with two different sets of pairs of entities. *G – generation set*: G contains good examples for the target predicate (G contains examples of pairs each composed of a parent and one of its children if we are discovering positive rules for a child predicate). *V – validation set*: V contains counter examples for the target predicate (pairs of people that are not in a child relation). We will explain in Section 6.2.1 how to generate these two sets for a given predicate. For

example, given the target predicate `child` for which we aim to discover positive rules, G contains pairs of parent-child people such as $(Barack_Obama, Sasha_Obama)$ and $(Clint_Eastwood, Scott_Eastwood)$, while V contains pairs of *not* parent-child people such as $(Barack_Obama, George_Bush)$ and $(Clint_Eastwood, Quentin_Tarantino)$.

Note that our approach is not less generic than those for mining rules for an entire KB (e.g., [15, 84]): it is true that we require a target predicate as input, however we can generically apply our setting for every predicate in the KB and compute rules for each of them (see Section 6.4.2).

We can now formalise the *exact discovery problem*.

Definition 6. Given a KB $\mathcal{K}$, two sets of pairs of entities G and V from $\mathcal{K}$ such that $G \cap V = \emptyset$, and a universe of Horn Rules R , a solution for the *exact discovery problem* is a subset R' of R such that:

$$R_{opt} = \underset{R' \subseteq R}{\operatorname{argmin}} f(x) := \{x = |R'| \mid C_{R'}(G) = G \wedge C_{R'}(V) \cap V = \emptyset\}$$

The ideal solution is a set of rules that covers all examples in G , and none of the examples in V . Note that given a pair of entities (x, y) , we can always generate a Horn Rule whose body covers only (x, y) by assigning target variables to x and y .

Unfortunately, since the solution is not allowed to cover any element in V , in the worst case the exact solution may be a set of rules such that each rule covers only one example in G , resulting in a solution that overfits the input data and that is difficult to understand for end users.

6.1.3 Weight Function

In order to allow flexibility and errors in both G and V , we drop the strict requirement of not covering any element of V . However, since covering elements in V is an indication of potential errors, we limit the coverage over V to the minimum possible. We therefore define a *weight* to be associated with a rule.

Definition 7. Given a KB $\mathcal{K}$, two sets of pair of entities G and V from $\mathcal{K}$ such that $G \cap V = \emptyset$, and a Horn Rule r , the weight of r is defined as follow:

$$w(r) = \alpha \cdot \left(1 - \frac{|C_r(G)|}{|G|}\right) + \beta \cdot \left(\frac{|C_r(V)|}{|U_r(V)|}\right) \quad (6.1)$$

with $\alpha, \beta \in [0, 1]$ and $\alpha + \beta = 1$.

The weight is a value between 0 and 1 that captures the *goodness* of a rule w.r.t. G and V : the better the rule, the lower the weight – perfect rules would have a weight of 0. The weight is made of two components normalised by the two parameters α and β . The first component captures the coverage over the generation set G – the ratio between the coverage of r over G and G itself. Note that if r covers all elements in G ,

then this component vanishes. The second component aims at quantifying potential errors of r , or rather the coverage over V . The coverage over V is not divided by total elements in V , because KBs are largely incomplete and we cannot distinguish between a missing information and a conflicting one (we cannot be sure whether elements for which the information is missing are covered or not by r). Thus we divide the coverage over V by the unbounded coverage of r over V . Ideally this number is close to 0. The parameters α and β are used to give relevance to each component. We would set a high β if we want to discover rules with high precision that identify few mistakes, or we would set a high α if we are more interested in recall and the discovered rules should identify as many examples as possible.

Example 19. W.r.t. the negative rule r and the KB $\mathcal{K}$ of Example 17, given two sets of pair of entities G and V from $\mathcal{K}$, the two components of w_r are computed as follows: the first component is computed as 1 minus number of pairs (x, y) in G where x is born after y divided by the total number of elements in G ; the second component is the ratio between number of pairs (x, y) in V where x is born after y and number of pairs (x, y) in V where the date of birth (for both x and y) is available in $\mathcal{K}$. In the specific example, the pair $(Sasha_Obama, Barack_Obama)$ would contribute both to the numerator of the first element and the denominator of the second element, while the pair $(Clint_Eastwood, Barack_Obama)$ would only contribute to the denominator of the second element.

Definition 8. Given a set of rules R , the weight for R is defined as:

$$w(R) = \alpha \cdot (1 - \frac{|C_R(G)|}{|G|}) + \beta \cdot (\frac{|C_R(V)|}{|U_R(V)|})$$

Assigning a weight to one or to multiple rules allows us to take into consideration an important aspect of modern KBs: the presence of errors. We will show in the experimental evaluation that very often universally correct rules have a significant coverage over V , which corresponds to errors in the KB. The exact discovery problem implies the absence of errors in the input KB, however such an assumption is too strong for modern KBs that are usually automatically built from external web sources [60, 135, 139] and thus inconsistencies are common [141].

6.1.4 Problem Definition

We can now state the approximate version of the problem.

Definition 9. Given a KB $\mathcal{K}$, two sets of pair of entities G and V from $\mathcal{K}$ where $G \cap V = \emptyset$, a universe of rules R , and a weight function w for R , a solution for the *approximate discovery problem* is a subset R' of R such that:

$$R_{opt} = \underset{R' \subseteq R}{\operatorname{argmin}} (w(R') | R'(G) = G)$$

We can map this problem to the well-known weighted set cover problem, which is proven to be a **NP-complete** problem [45]. The universe corresponds to G and the input sets are all the possible rules defined in R .

Since we want to minimise the total weight of the output rules, the approximate version of the discovery problem aims to cover all elements in G , and as few as possible elements in V . Since for each element $g \in G$ there always exists a rule that covers exactly only g (single-instance rule), an optimal output covering the whole G is always guaranteed to exist. We expect the output to be made of some rules that cover multiple examples in G , while remaining examples in G to be covered by single-instance rules. In the best-case scenario a single rule covers all elements in G and none of the elements in V .

Section 6.3 will describe a greedy polynomial algorithm to find a good solution for the approximate discovery problem.

6.2 Rule Generation

We introduce RuDiK, a disk-based approach to discover first-order Horn Rules in RDF KBs. The first task RuDiK needs to address is the generation of the universe of all possible rules R . Each rule in R must cover one or more examples of the generation set G .

In this setting, the universe of all possible rules can be generated by just inspecting elements of G . This is a key part of our generation approach: we do not inspect the entire KB, but just entities belonging to G . The smaller the size of G is, the smaller is the search space for rules generation. On the contrary, recent KB rule mining approaches tackle the problem by searching for valid rules over the entirety of the KB. This is typically done by initially looking at all facts that share a common predicate, and then expanding these facts with other predicates that share common entities [38, 84]. Alternatively other approaches leverage on well established relational database techniques such as functional dependencies discovery [75].

The generation problem is simple but presents significant scalability issues. The number of all possible rules increases exponentially with the size of the KB and enumerating all of them means exploring a huge search space. This is the main reason why previous approaches solve the problem by loading the entire KB into main memory and by aggressively indexing triples on subject, object, and predicate. Unfortunately, modern KBs can easily exceed the size of hundreds of GBs, making a memory-based solution unfeasible on common machines. We introduce in RuDiK a novel rules generation technique that inspects only a portion of the KB. On the one hand, we load into main memory only a small fraction of the database. On the other hand, this allows a larger expressiveness of the considered rules than previous works. One could argue that reducing the search space may lead to missing some meaningful rules. We will show

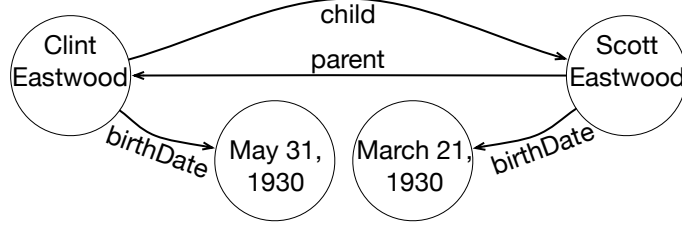


Figure 6.1: Graph Portion of DBPEDIA

in Section 6.4.2 that our generation technique for G creates a representative (reduced) view of the entire search space.

The data structure RuDiK leverages on is a straightforward translation of a KB $\mathcal{K}$ into a directed graph: entities and literals are nodes of the graph, while there is a direct edge from node a to node b for each triple $\langle a, rel, b \rangle \in \mathcal{K}$. Edges are labelled, where the label is the relation rel that connects subject to object in the triple. Figure 6.1 shows a portion of DBPEDIA [29] that connects two persons in a child and parent relationship, along with their date of birth. The graph represents information of four KB triples (two birth dates, one parent and one child relation).

The body of a Horn Rule can be seen as a path in the graph. W.r.t. Figure 6.1, the body $\text{child}(a, b) \wedge \text{parent}(b, a)$ corresponds to the path $\text{Clint Eastwood} \rightarrow \text{Scott Eastwood} \rightarrow \text{Clint Eastwood}$. In Section 6.1.1 we defined valid rules. The body of a valid rule contains target variables a and b at least once, and every other variable at least twice. In a valid body also each atom is transitively connected to every other atom. If we allow navigation of edges in any direction (no matter what the direction of the edge is on the graph), we can easily translate bodies of valid rules to valid paths on the graph. Given a pair of entities (x, y) , a valid body corresponds to a valid path p on the graph that meets the following criteria:

1. p starts at the node x ;
2. p touches y at least once;
3. p ends in x , in y , or in a different node that has been touched before.

In other words, given the body of a rule r_{body} , r_{body} covers a pair of entities (x, y) iff there exists a valid path on the graph that corresponds to r_{body} . Therefore, given a pair of entities (x, y) , we can generate bodies of all possible valid rules by simply computing all valid paths from x with a standard BFS. Note how the transitive connection between atoms is always guaranteed by the construction property of a path p : for each node n in p there always exists a subpath that connects n to every other node in p . The key point is the ability of navigating each edge in any direction, which basically means turning the original directed graph into an undirected one.

Despite the navigation over an undirected graph, we still need to keep track of the original direction of the edges. This is essential when translating paths to Horn Rule

bodies. In fact, if we navigate an edge `rel` between a and b , an original edge direction from a to b produces the atom `rel(a, b)`, while a direction from b to a produces the atom `rel(b, a)`. These two atoms are different, where the position of variables is determined by the original direction of the edge.

One should notice that for two entities x and y , there might exist infinite valid paths starting from x since every node can be traversed multiple times. Thus we introduce the *maxPathLen* parameter that determines the maximum number of edges in the path. When translating paths to Horn Rules, *maxPathLen* determines the maximum number of atoms that we can have in the body of the rule. This parameter is essential to avoid the discovery of rules with infinite body length.

The main advantage of inspecting just the generation set G is the capability of loading only a small portion of the graph that is currently needed. Given a pair of entities (x, y) , we retrieve from the KB all those nodes at distance *maxPathLen* – 1 or less from x or y , along with their edges. Retrieving such nodes and edges can be done recursively: we maintain a queue of entities, and for each entity in the queue we fire a SPARQL query against the KB to get all entities (and edges) at distance 1 from the current entity – we call these queries *single hop queries*. We then add the new found entities to the queue iff they are at distance less than *maxPathLen* – 1 from either x or y and they have not been visited before. The queue is initialised with x and y . By doing so we retrieve a small portion of the entire KB, and namely only that needed to discover rules that cover (x, y) . We will show in the experimental section that SPARQL engines are very fast at executing single hop queries.

The generation of the universe of all possible rules for G is therefore straightforward: for each element $(x, y) \in G$, we construct the portion of the graph as described above and compute all valid paths starting from x . Computing paths for every example in G implies also computing the coverage over G for each rule. The coverage of a rule r is simply the number of elements in G where there exists a path that corresponds to r_{body} . Our discovery technique will also generate single-instance rules: rules that cover only one example $(x, y) \in G$ by instantiating target variables a and b in the rule with x and y . Once the universe of all possible rules has been generated (along with coverages over G), computing coverage and unbounded coverage over V is just a matter of executing two SPARQL queries against the KB for each rule in the universe. We will show in Section 6.3 how some queries can be avoided, as well as there is no need of enumerating all possible rules in the universe as some of them will never be part of the final solution.

Clearly, the size of G has a direct impact on the search space and hence on the running time. Since we generate all valid rules for each example in G , the search space grows roughly linearly with the size of G . If we could know a-priori the minimum subset of examples that lead to the generation of all valid rules, then we could use only those few examples. A future direction we are working on exploits exactly this point:

how to select a small number of representative examples in order to reduce the size of G , without significantly affecting the quality of the output.

6.2.1 Input Examples Generation

A crucial role in our approach is played by the two input sets G and V , used to generate and validate rules. We will show that discovering negative rules is the dual problem of discovering positive rules, therefore we assume to be in the setting of generating positive rules. We will explain how to switch to the negative setting at the end of this section.

Given an input KB $\mathcal{K}$ and a predicate $rel \in \mathcal{K}$, we automatically build a generation set G and a validation set V as follows. G consists of positive examples for the target predicate rel . It is generated as all pairs of entities (x, y) such that $\langle x, rel, y \rangle \in \mathcal{K}$ – if the target predicate is `child`, it consists of all pairs of entities in a child relation. V instead consists of counter (negative) examples for the target predicate and it is slightly more complicated to generate, since the closed world assumption does no longer hold in KBs. Differently from classic database scenarios, we cannot assume that what is not stated in a KB is false (closed world assumption). Because of large incompleteness, everything that is not stated in a KB is *unknown* rather false. In order to generate negative examples, we make use of a popular technique for KBs: *Local-Closed World Assumption* (LCWA) [60, 84]. LCWA states that if a KB contains one or more object values for a given subject and predicate, then it contains all possible values (if a KB contains one or more children of Clint Eastwood, then it contains all possible children). This is definitely true for *functional* predicates (predicates such as `capital` where the subject can have at most one object value), while it might not hold for non-functional predicates (e.g., `child`). KBs contain many non-functional predicates, we therefore extend the definition of LCWA by considering the dual aspect: if a KB contains one or more subject values for a given object and predicate, then it contains all possible values.

To generate negative examples we then take the union of the two LCWA aspects: for a given predicate rel , a negative example is a pair (x, y) where either x is the subject of one or more triples $\langle x, rel, y' \rangle$ with $y \neq y'$, or y is the object of one or more triples $\langle x', rel, y \rangle$ with $x \neq x'$. As an example, if $rel = \text{child}$, a negative example is a pair (x, y) such that x has some children in the KB that are not y , or y is the child of someone that is not x . By considering the union of the two LCWA aspects we do not restrict predicates to be functional (such as in [84]).

The number of negative examples generated with the above technique could easily explode (for a target `child` predicate it is nearly the cartesian product of all the people having a child with all the people having a parent). In order to apply the same approach for positive and negative rule discovery, we require G and V to be of comparable sizes. We therefore introduce a further constraint, that significantly shrinks

the size of negative examples: given a pair of entities (x, y) , x must be connected to y via a predicate that is different from the target predicate. In other words, given a Kb $\mathcal{K}$ and a target predicate rel , (x, y) is a negative examples if $\langle x, rel', y \rangle \in \mathcal{K}$, with $rel' \neq rel$. This intuition exploits the LCWA for predicates rather than for entities. If a KB contains a relation between two entities x and y , then it contains all possible relations between x and y . If x and y are in a relation that is not the target predicate, then most likely x and y are not connected by the target predicate in the real world. This further restriction has multiple advantages: on the one hand it makes the size of V of the same order of magnitude of G (see Section 6.4), on the other hand it guarantees the existence of a path between x and y , for every $(x, y) \in V$. For positive examples, the existence of a path was already guaranteed since pairs in G are always connected by at least one predicate, the target one. We used three popular KBs in our experiments (Section 6.4) and we found out that our technique for generating V guarantees three main points in all of the real cases: first, V 's size is in the same order of magnitude of G ; second, the existence of a path between entities in V ensures the discovery of meaningful negative rules; third, we did not find a single case where a pair of entities connected via a different predicate than the target one was also connected via the target one.

Eventually V is generated with the intersection of the two constraints defined above. A negative example (x, y) for the target predicate **child** has the following characteristics: (i) x and y are not connected by a **child** predicate; (ii) either x has one or more children (different from y) or y has one or more parents (different from x); (iii) x and y are connected by a predicate that is different from **child**.

Example 20.

$\langle \text{Clint_Eastwood} \quad \text{child} \quad \text{Scott_Eastwood} \rangle.$
 $\langle \text{Clint_Eastwood} \quad \text{colleague} \quad \text{Quentin_Tarantino} \rangle.$
 $\langle \text{Clint_Eastwood} \quad \text{spouse} \quad \text{Maggie_Johnson} \rangle.$

From the above toy KB, the pair $(\text{Clint_Eastwood}, \text{Quentin_Tarantino})$ is a negative example for the target predicate **child** since Clint Eastwood has a child who is not Quentin Tarantino, Clint Eastwood and Quentin Tarantino are not connected via a child predicate and they are also connected via a different predicate (**colleague**).

Often KBs use same predicates to connect different semantic categories. We may find that a **child** predicate is used not only to connect two entities of type person, but also two companies to denote an ownership relation. In order to enhance the quality of the input examples and avoid cases of mixed unrelated semantic categories, we introduce the *type* restriction when generating G and V . The type restriction requires that for every example pair (x, y) belonging to either G or V , x is always of the same

type and y is always of the same type. All modern KBs include entity types (often through the `rdf:type` statement), therefore we make use of this information. For example, G and V for a target `child` predicate are two sets of pairs, where each pair consists in two entities of type `person`. The type information can be manually provided or, as we will see in Section 6.4.2, we can automatically compute it from the KB.

In the dual problem of negative rule discovery our approach remains unchanged, we just switch the role of G and V . The generation set becomes V (negative examples), while the validation set becomes G (positive examples). Now it should be more clear why we introduced the second constraint of LCWA on predicates. Since V becomes the generation set in the negative rules setting, V must be small in size and it must guarantee the existence of a path between pairs of entities in each example. Furthermore, we stress that our approach is independent on how G and V are generated: they could also be manually crafted by some domain experts, which would require additional manual effort. We will empirically show that our examples generation strategy leads to better quality rules than, e.g., those created with examples randomly selected from the KB.

The strategy for creating V aims at making this set as much symmetric to G as possible. Examples in V are counter examples for the target predicate when discovering positive rules, and this is guaranteed by the LCWA on the target predicate. When discovering negative rules instead, V must have similar properties of G , which translates to having examples that can lead to meaningful rules. This aspect is guaranteed by the constraint that examples in V must be connected via a different predicate than the target one. Such a constraint guarantees that examples in V are somehow related, i.e., they are not just two random entities that have nothing in common. Following these intuitions, V shows the same properties of G , and we can use the exact same algorithm for discovering both positive and negative rules by just switching the roles of G and V .

One of the downside of using a small generation set is the risk of discovering biased local rules, i.e., rules that hold only on G but not on the rest of the KB. Currently we are selecting all positive facts from the KB to construct G , and this has the clear advantage of being a fast and straightforward solution. However we are also investigating the problem of selecting *diverse* examples: examples that show different behaviours within the KB. In this work however we focus on how to discover qualitative rules given two sets of examples, and how these examples are generated is somehow orthogonal to the techniques we present here.

To the best of our knowledge, RuDiK is the first approach that is capable of discovering both positive and negative rules in KBs. Previous works have put their focus either on the positive setting [15, 38, 84], or on the negative one [75]. However, none of these techniques can be easily adapted to work in the dual scenario.

6.2.2 Literals and Constants

We defined our target language in Section 6.1.1 which, other than normal predicate atoms, includes literals comparison. The scope of literals comparison is to enrich the language with smarter comparisons among literal values other than equalities, such as greater than or less than. In order to discover such kind of atoms, the KB graph must contain edges that connect literal values with one (or more) symbol from $\{<, \leq, \neq, >, \geq\}$. As an example, Figure 6.1 should contain an edge ‘<’ from node “*March 31, 1930*” to node “*March 21, 1986*”. Unfortunately, the original KB does not contain this kind of information, and creating such comparisons among all literals in the KB is unfeasible.

Once again, the use of a generation set G is the key point to introduce literals comparison. Since we discover paths for a pair of entities from G in isolation, thus the size of a graph for a pair of entities is relatively small, we can afford to compare all literal values within a single example graph. This implies the creation of a quadratic number of edges w.r.t. the number of literals in the graph. We will show in the experimental section that within a single example graph the number of literals is usually relatively small, thus the quadratic comparison is affordable. Modern KBs include three types of literals: numbers, dates, and strings. Besides equality comparisons, we add ‘>’, ‘≥’, ‘<’, ‘≤’ relationships between numbers and dates, and $\neq$ between all literals. These new relationships are treated as normal atoms: $x \geq y$ is equivalent to $\text{rel}(x, y)$, where rel is equal to $\geq$. Once we added artificial edges for every input example graph, the discovery of literal comparisons is equivalent to discover normal predicate atoms.

Furthermore, we noticed that ‘ $\neq$ ’ relation could be useful for entities as well other than literals. Think about the following negative rule:

$$\text{bornIn}(a, x) \wedge x \neq b \Rightarrow \neg \text{president}(a, b)$$

The rule states that if a person a is born in a country that is different from b , then a cannot be president of b . The rule holds for most of the countries in the world. To consider inequalities among entities, we could add artificial edges among all pairs of entities in the graph. This strategy however, despite being inefficient for the high number of edges to add, would lead to many meaningless rules. We noticed that it is reasonable to compare two entities only when they are of the same type, e.g., they belong to the same conceptual category. As with the generation of G and V , we make use of `rdf:type` triples. We add an artificial inequality edge in the input example graph only between those pairs of entities of the same type. In the above rule it is reasonable to compare x and b because they are both countries.

As a last extension of our language, we also discover rules with constants (entities). For a given rule r , we promote a variable v in r to an entity e iff for every $(x, y) \in G$

covered by r , v is always instantiated with the same value e . Suppose that for the above negative rule for president, all examples in G are people born in the country “*U.S.A.*”. We can then promote variable b to “*U.S.A.*”, generating the rule:

$$\text{bornIn}(a, x) \wedge x \neq \text{U.S.A.} \Rightarrow \neg \text{president}(a, \text{U.S.A.})$$

6.3 A^* Greedy Algorithm

We introduce in RuDiK a greedy approach to solve the approximate version of the discovery problem (Section 6.1.4). The algorithm combines two phases: (i) it solves the set cover problem with a greedy strategy; (ii) it discovers new rules by navigating the graph in the A^* search fashion, allowing the pruning of unpromising paths.

6.3.1 Marginal Weight

In Section 6.1.3, we defined the weight associated with a set of rules R as follows:

$$w(R) = \alpha \cdot (1 - \frac{|C_R(G)|}{|G|}) + \beta \cdot (\frac{|C_R(V)|}{|U_R(V)|})$$

Our goal is to discover a set of rules that covers as many elements as possible in G , and as few elements as possible in V . We follow the intuitions behind the greedy algorithm for weighted set cover by defining a *marginal weight* for rules that are not yet included in the solution [45].

Definition 10. Given a set of rules R and a rule r such that $r \notin R$, the marginal weight of r w.r.t. R is defined as:

$$w_m(r) = w(R \cup \{r\}) - w(R) = -\alpha \cdot \frac{|C_r(G) \setminus C_R(G)|}{|G|} + \beta \cdot (\frac{|C_{R \cup \{r\}}(V)|}{|U_{R \cup \{r\}}(V)|} - \frac{|C_R(V)|}{|U_R(V)|})$$

The marginal weight quantifies the total weight increase that we would have by adding r to an existing set of rules R . In other words, the marginal weight indicates the contribution of r to R in terms of new elements covered in G and new elements uncovered covered in V . Due to the first negative part, $w_m(r) \in [-\alpha, +\beta]$. Since the set cover problem aims at minimising the total weight, we would never add a rule to the solution if its marginal weight is greater than or equal to 0. Algorithm 5 shows the straightforward greedy rules selection procedure. The algorithm takes as input the generation set G , the validation set V , and the universe of all possible rules R . The set of output rules R_{opt} is first assigned to an empty set. At each iteration, the algorithm picks from R the rule r with the minimum marginal weight, and it adds r to R_{opt} . r is then removed from R . The algorithm stops when one of the following termination

Algorithm 5: Greedy Rules Selection.

input : G – generation set
input : V – validation set
input : R – universe of rules
output: R_{opt} – greedy set cover solution

```
1  $R_{opt} \leftarrow \emptyset$ ;  
2  $r \leftarrow \underset{r \in R}{\operatorname{argmin}}(w_m(r))$ ;  
3 repeat  
4    $R_{opt} \leftarrow R_{opt} \cup \{r\}$ ;  
5    $R \leftarrow R \setminus \{r\}$ ;  
6    $r \leftarrow \underset{r \in R}{\operatorname{argmin}}(w_m(r))$ ;  
7 until  $R = \emptyset \vee C_{R_{opt}}(G) = G \vee w_m(r) \geq 0$ ;  
8 if  $C_{R_{opt}}(G) \neq G$  then  
9    $R_{opt} \leftarrow R_{opt} \cup \operatorname{singleInstanceRule}(G \setminus C_{R_{opt}}(G))$ ;  
10 return  $R_{opt}$ 
```

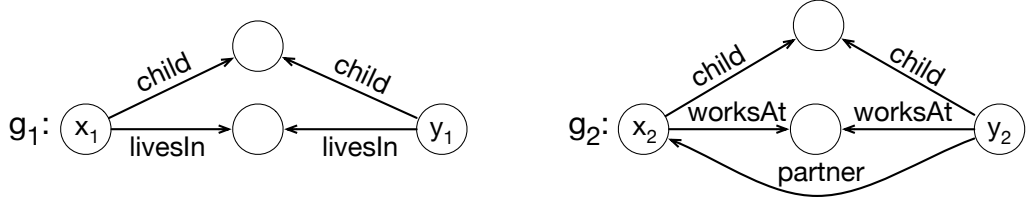
conditions are met: 1) R is empty – all the rules have been included in the solution; 2) R_{opt} covers all elements of G ; 3) the minimum marginal weight is greater than or equal to 0 – among the remaining rules in R , none of them has a negative marginal weight, hence the current solution is one of those with the minimum weight. If the second termination condition is not met, there may exist examples in G that are not covered by R_{opt} . In such a case the algorithm will augment R_{opt} with single-instance rules (rules that cover only one example), one for each element in G not covered by R_{opt} . This is necessary to let the approximate solution fully cover the input set G .

Since the coverage of a rule is always contained in its unbounded coverage, the marginal weight is greater than or equal to 0 whenever the rule does not cover new elements in G and does not unbound cover new elements in V . More specifically, a rule r has a negative marginal weight iff the sum of its new elements covered in G with its new elements unbounded covered in V is strictly greater than its new elements covered in V .

The greedy solution guarantees a $\log(k)$ approximation to the optimal solution [45], where k is the largest number of elements covered in G by a rule in R – k is at most $|G|$. If the output rules in the final solution cover disjoint sets of G , then the greedy solution is optimal.

6.3.2 A^* Graph Traversal

Algorithm 5 assumes that the universe of rules R has been generated, so that we can iteratively pick the rule with minimum marginal weight. In order to generate the universe of all possible rules, we need to traverse all valid paths from a node x to a node y , for every pair (x, y) in the generation set G . In this section we will analyse the following aspect: is it essential to generate all possible paths for every example in G ?



Example 21. Consider the scenario where we are mining positive rules for the target predicate **spouse**. The input generation set G includes two examples $g_1 = (x_1, y_1)$ and $g_2 = (x_2, y_2)$ retrieved from the KB, where the above figure shows the graph for the two examples. Assume for simplicity that all rules in the universe have the same coverage and unbounded coverage over the validation set V . One of the plausible rule is $r = \text{child}(x, v_0) \wedge \text{child}(y, v_0) \Rightarrow \text{spouse}(x, y)$, stating that entities x and y with a common child are married. Looking at the KB graph, r covers both g_1 and g_2 – in both g_1 and g_2 there exists a path that corresponds to r_{body} . Since all rules have the same coverage and unbounded coverage over V , if we include r in the solution before inspecting other potential rules, then there is no need to generate any other rule. In fact any other plausible rule will not cover new elements in G , therefore their marginal weights will be greater than or equal to 0. Hence any other rule will not be part of the solution. Thus the navigation (and creation) of edges **livesIn** in g_1 , **worksAt** in g_2 , and **partner** in g_2 becomes worthless.

Based on the above observation, we avoid the generation of the entire universe R , but rather we consider at each iteration the most promising path on the graph. The intuition is the same behind the A^* graph traversal algorithm [95]. Given an input weighted graph, A^* computes the smallest cost path from a starting node s to an ending node t . At each iteration, A^* maintains a queue of partial paths starting from s , and it expands one of these paths based on an *estimation* of the cost still to go to t . The path with the best estimation is expanded and added to the paths queue. The algorithm keeps iterating until one of the partial paths reaches t . RuDiK discovers rules with a similar technique. For each example $(x, y) \in G$, we start the navigation from x . We keep a queue of not valid rules (Section 6.1.1), and at each iteration we consider the rule with the minimum marginal weight, which corresponds to equivalent paths in the example graphs. We expand the rule by following edges on the graphs, and we add the new founded rules to the queue of not valid rules. Differently from A^* , we do not stop when a rule (path) reaches the node y . Whenever a rule becomes valid, we add the rule to the solution and we do not expand it any further. The algorithm keeps looking for plausible paths until one of the termination conditions of Algorithm 5 is met.

A crucial point in A^* is the definition of the estimation cost. In particular, if we want to guarantee the solution to be optimal, the estimation must be *admissible*, i.e., the estimation cost must be less than or equal to the actual cost. As an example,

when searching for the shortest route on a map, an admissible estimation might be the straight-line distance to the goal for every node, since that is physically the smallest possible distance between any two points. In our setting, given a rule that is not yet valid and needs to be expanded, we need to define an admissible estimation of the marginal weight.

Definition 11. Given a rule $r = A_1 \wedge A_2 \cdots A_n \Rightarrow B$, we say that a rule r' is an *expansion* of r iff r' has the form $A_1 \wedge A_2 \cdots A_n \wedge A_{n+1} \Rightarrow B$. In other words, r' is generated by adding a new atom to the body of r .

Given a rule r , expanding r means adding a new atom to the body of r . From the graph traversal point of view, expanding r means traversing one further edge on the path defined by r_{body} . In order to guarantee the optimality condition, the estimated marginal weight for a rule r that is not valid must be less than or equal to the actual weight of any valid rule that is generated by expanding r . Given a rule and some expansions of it we can derive the following:

Lemma 6. *Given a rule r and a set of pairs of entities E , then for each r' expansion of r , $C_{r'}(E) \subseteq C_r(E)$ and $U_{r'}(E) \subseteq U_r(E)$.*

The above Lemma states that the coverage and unbound coverage of a rule r' expansion of r are respectively contained in the coverage and unbound coverage of r . In fact expanding a rule means adding a new atom to the body of the rule, hence making the rule more selective. This is equivalent of adding an ‘AND’ condition to a SQL query q : the new result will obviously be a subset of the result obtained with q . Note that Lemma 6 is equivalent to the augmentation inference rule for functional dependencies [17].

We recall that the marginal weight of a rule r w.r.t. a solution R is defined as follow:

$$w_m(r) = -\alpha \cdot \frac{|C_r(G) \setminus C_R(G)|}{|G|} + \beta \cdot \left(\frac{|C_{R \cup \{r\}}(V)|}{|U_{R \cup \{r\}}(V)|} - \frac{|C_R(V)|}{|U_R(V)|} \right)$$

The only positive contribution to marginal weights is given by $|C_{R \cup \{r\}}(V)|$. $|C_{R \cup \{r\}}(V)|$ is equivalent to $|C_R(V)| + |C_r(V) \setminus C_R(V)|$, thus if we set $|C_r(V) \setminus C_R(V)| = 0$ for any r that is not valid, we guarantee an admissible estimation of the marginal weight. More simply, we estimate the coverage over the validation set to be 0 for any rule that can be further expanded (i.e., not valid), since expanding the rule may bring the coverage to 0.

Definition 12. Given a not valid rule r and a set of rules R , we define the *estimated marginal weight* of r as:

$$w_m^*(r) = \alpha \cdot \frac{|C_r(G) \setminus C_R(G)|}{|G|} + \beta \cdot \left(\frac{|C_R(V)|}{|U_{R \cup \{r\}}(V)|} - \frac{|C_R(V)|}{|U_R(V)|} \right)$$

where a rule is valid iff it respects the language biases defined in Section 6.1.1.

The estimated marginal weight for a valid rule is equal to the actual marginal weight since a valid rule will not be considered for expansion – it corresponds to reaching the ending node in the A^* algorithm. Given Lemma 6, we can easily see that $w_m^*(r) \leq w_m^*(r')$, for any r' expansion of r . Thus our marginal weight estimation is admissible.

RuDiK uses the concept of *frontier nodes* for a rule r ($N_f(r)$). Given a rule r , $N_f(r)$ contains the last visited nodes in the paths that correspond to r_{body} from every example graphs covered by r . As an example, given $r_{body} = \text{child}(x, v_0)$, $N_f(r)$ contains all the entities v_0 that are children of x , for every $(x, y) \in G$ – v_0 is the last visited node in the path. Expanding a rule r implies navigating a single edge from any frontier node. Algorithm 6 shows the modified set cover version that includes A^* -like rules generation. The set of frontier nodes is initialised with starting nodes x , for every $(x, y) \in G$ (Line 2). The algorithm maintains a queue of rules Q_r , from which it chooses at each iteration the rule with minimum approximated weight. The function **expandFrontiers** retrieves from the KB all nodes (along with edges) at distance 1 from frontier nodes and returns the set of all rules generated by this one hop expansion. Such expansions are computed with single-hop SPARQL queries. Q_r is therefore initialised with all rules of length 1 starting at x (Line 3). In the main loop, the algorithm checks if the current best rule r is valid or not. If r is valid, r is added to the output and it is not expanded (Line 8). If r is not valid, r is expanded iff the length of its body is less than $maxPathLen$ (Line 10). There is no point in expanding rules with body greater than or equal to $maxPathLen$ since such rules are not allowed in the output. The termination conditions and the last part of the algorithm are the same of Algorithm 5.

The simultaneous rules generation and selection of Algorithm 6 brings multiple benefits. First of all, we do not generate the entire graph for every example in G . Nodes and edges are generated *on demand*, whenever the algorithm requires their navigation (Line 12). If the initial part of a path is not promising according to its estimated weight, the rest of the path will never be materialised since there is no need to navigate it. Materialising the graph is an expensive task, since we need to query the disk in order to retrieve target nodes and edges. Rather than materialising the entire graph and then traversing it, we propose a solution that gradually materialises parts of the graph whenever they are needed for navigation (Lines 3 and 12). Secondly, the weight estimation could lead to pruning unpromising rules. If a rule does not cover new elements in G and does not unbound cover new elements in V , then its estimated marginal weight is 0 (Definition 12). A rule with 0 marginal weight is never picked as best rule to be expanded, and if it is the algorithm terminates (due to one of the termination conditions). This implies that a rule with 0 estimated marginal weight is pruned away from the search space, as we will never generate rules from its expansion.

The partial materialisation of the graph and the pruning of the search space have a significant impact on the amount of resources used by the algorithm. On the one

Algorithm 6: RuDiK Rules Discovery.

```
input :  $G$  – generation set
input :  $V$  – validation set
input :  $maxPathLen$  – maximum rule body length
output:  $R_{opt}$  – greedy set cover solution
1  $R_{opt} \leftarrow \emptyset$ ;
2  $N_f \leftarrow \{x | (x, y) \in G\}$ ;
3  $Q_r \leftarrow \text{expandFrontiers}(N_f)$ ; // SPARQL queries
4  $r \leftarrow \underset{r \in Q_r}{\text{argmin}}(w_m^*(r))$ ;
5 repeat
6    $Q_r \leftarrow Q_r \setminus \{r\}$ ;
7   if  $\text{isValid}(r)$  then
8      $R_{opt} \leftarrow R_{opt} \cup \{r\}$ ;
9   else
10    // rules expansion
11    if  $\text{length}(r_{body}) < maxPathLen$  then
12       $N_f \leftarrow \text{frontiers}(r)$ ;
13       $Q_r \leftarrow Q_r \cup \text{expandFrontiers}(N_f)$ ; // SPARQL queries
14     $r \leftarrow \underset{r \in Q_r}{\text{argmin}}(w_m^*(r))$ ;
15 until  $Q_r = \emptyset \vee C_{R_{opt}}(G) = G \vee w_m^*(r) \geq 0$ ;
16 if  $C_{R_{opt}}(G) \neq G$  then
17    $R_{opt} \leftarrow R_{opt} \cup \text{singleInstanceRule}(G \setminus C_{R_{opt}}(G))$ ;
18 return  $R_{opt}$ 
```

hand, we noticed the running time is halved in the worst case and sometimes it is up to ten times faster. On the other hand, we can also significantly decrease the amount of memory needed, since we load sub-portions of the graph. Consider the extreme case where there exists a rule r that covers all elements in G , unbound covers all elements in V and does not cover any element of V . In such a case, Algorithm 6 will output the optimal solution by just materialising paths on the example graphs that correspond to r_{body} with $\mathcal{O}(l \cdot |G|)$ SPARQL queries, where l is the length of r_{body} .

6.4 Experiments

We carried out an extensive experimental evaluation of our rule discovery approach. We grouped the evaluation into 4 main sub-categories: (i) a first set of experiments aims at demonstrating the quality of our output, both for negative and positive rules; (ii) a second set of experiments compares our method with state-of-the-art systems; (iii) in the third set of experiments we outline the applicability of rule discovery by enhancing machine learning algorithms; (iv) in the last set of experiments we discuss internal system settings and some KB properties.

Settings. We evaluated our approach over several popular KBs. For each KB, we downloaded the most up-to-date core facts and loaded them into our SPARQL query

engine. We experimented several SPARQL engines, including Jena ARQ,⁵ OWLIM Lite,⁶ and RDF-3x.⁷ We also implemented a naïve relational database solution with PostgreSQL. Eventually we opted for OpenLink Virtuoso,⁸ as it was the fastest among all the solutions. Virtuoso took on average 20 minutes to load a medium size KB (i.e., 10 GB) into its store, and around 100 milliseconds to execute a single hop query. All experiments are run on a iMac desktop with an Intel quad-core i5 at 2.80GHz with 16 GB RAM. We run Virtuoso server with its SPARQL query endpoint on the same machine, optimised for a 8 GB available RAM.

Our method needs the two input parameters α and β of Equation 6.1. α measures the importance of the coverage over the generation set, while β measures the coverage over the validation set. In other words, a high α privileges recall over precision, while a high β gives more importance to precision. We can afford a high α and a low β when the input KB is accurate and complete. We will show that KBs contain many errors and missing information, therefore we set $\alpha = 0.3$ and $\beta = 0.7$. Increasing α and decreasing β means a higher number of output rules with a lower accuracy.

We also set the *maxPathLen* parameter to 3. This number represents the maximum number of atoms that we can have in the body of a rule. We will show in Section 6.4.4 that increasing this parameter does not bring any benefits, as body rules longer than 3 atoms start to be very complicated and not insightful.

All the choices of the above parameters have been empirically validated through extensive experiments that are accessible in the online technical report <http://bit.ly/2bDZ09F>.

Evaluation Metrics. RuDiK discovers rules for a given target predicate. For each KB, we chose 5 representative predicates as follows: we first ordered predicates according to descending popularity (i.e., number of triples having that predicate), and then we picked the top 3 predicates for which we knew there existed at least one meaningful rule, and other 2 top predicates for which we did not know whether some meaningful rules existed. We repeated the procedure for each input KB, and for positive and negative rules. Despite working one predicate at time, RuDiK can also discover rules for the entire KB by listing all predicates in the KB, and discover rules for each of them. We will show in Section 6.4.2 how this can be achieved.

Positive rules are very useful to enrich the KB by discovering new facts. Since we are generating new data, we cannot evaluate the output over the existing data. Inspired by [84], we proceed as follows: we run the algorithm over the KB, and for each output rule we generate all new predictions that are not already in the KB (we execute the body of the rule against the KB and remove all those pairs that are already connected by the target predicate in the KB). As an example, for the rule $\text{spouse}(b, a) \Rightarrow \text{spouse}(a, b)$,

⁵<https://jena.apache.org/documentation/query/>

⁶<https://confluence.ontotext.com/display/OWLIMv54/OWLIM-Lite+Installation>

⁷<https://code.google.com/archive/p/rdf3x/>

⁸<http://virtuoso.openlinksw.com/>

we retrieve all pairs (b, a) such that b is **spouse** with a but a is not **spouse** with b in the KB. If the rule is universally correct (like the previous example), we mark all the new predictions as true. If the rule is unknown, we randomly sampled 30 new predictions and manually checked them against the Web. The *precision* of the rule is then computed as the ratio of true predictions out of true and false predictions.

Negative rules are slightly more complicated to evaluate. In fact, despite that KBs are usually incomplete, a large percentage of the data not stated in a KB is false – a very small fraction of the cartesian product of all the people in a KB will be actually married. Therefore negative rules will always discover many correct negative facts. However, negative rules are a great means to discover errors in the KB, and we leverage this aspect to evaluate them. For each discovered rule, we retrieve from the KB pairs of entities for which the body of the rule can be instantiated over the KB and that are also connected by the target predicate. As an example, for the rule $\text{child}(a, b) \Rightarrow \neg \text{spouse}(a, b)$, we retrieve all those pairs (a, b) such that b is **child** of a and a is **spouse** with b . We call these generated pairs of entities *potential errors*. Similarly to positive rules, whenever a rule is universally correct we mark all its potential errors as true, whereas if the rule is unknown we manually check 30 sampled potential errors. The final precision of a rule is computed as actual errors divided by potential errors. Furthermore, a potential error is evaluated as actual error whenever a single atom in the rule is incorrect, no matter if the atom is in the body or the head of the rule. For example, for the negative rule $\text{child}(a, b) \Rightarrow \neg \text{spouse}(a, b)$, a specific instantiation (a, b) is an actual error if either b is not an actual **child** of a , or b is not an actual **spouse** of a . As we will explain further below, negative rules cannot point exactly where the error is, but they give a hint that something is wrong and needs to be further inspected.

The full suite of test results, together with all the KBs, induced rules, and annotated gold standard examples and rules is available online at <http://bit.ly/2csR0sc>.

6.4.1 Rule Discovery Accuracy

The first set of experiments aims at evaluating the accuracy of discovered rules over the 3 most popular and widely used KBs: DBPEDIA [29], YAGO [139], and WIKI-DATA [145]. For each KB we downloaded the most recent version and selected core facts, facts about people, geolocations and transitive **rdf:type** facts. WIKIDATA provides only the entire dump, therefore we just eliminated from it no-english literal values. Table 6.1 shows the characteristics of the 3 KBs.

As the figure shows, the size of the KB is relevant. Loading the entire KB into main memory is not feasible unless we have high memory availability [38, 75], or we reduce the KB by eliminating facts such as **rdf:type** or literals [84]. We propose an approach that is disk-based where only a small portion of the KB is loaded into main memory, such that we can discover rules on any size KB with limited resources.

Table 6.1: Dataset characteristics.

<i>KB</i>	<i>Version</i>	<i>Size</i>	<i>#Triples</i>	<i>#Predicates</i>
DBPEDIA	3.7	10.056GB	68,364,605	1,424
YAGO	3.0.2	7.82GB	88,360,244	74
WIKIDATA	20160229	12.32GB	272,129,814	4,108

6.4.1.1 Positive Rule Discovery

We first evaluate the precision of positive rules for the top 5 predicates on the 3 KBs. The number of new induced facts varies significantly from rule to rule – a rule with literals comparison will produce a very high number of facts. In order to avoid the precision to be dominated by such rules, we first compute the precision for each rule as explained above, and then we average values over all induced rules. Table 6.2 reports precision values, along with average running time over all predicates.

Our first observation is that the more accurate is the KB, the better is the quality of induced rules. WIKIDATA contains very few errors, since it is manually curated and every triple is manually checked by different individuals before being inserted. DBPEDIA and YAGO instead are automatically generated by extracting information from the Web, hence their quality is significantly lower. Discovering perfect positive rules is a hard task, mostly because there is no guarantee of the existence of valid negative examples. A striking example in this direction is one of the rule induced for **founder** in DBPEDIA. Our approach discovers that if a person is born in the same place where a company is founded, then the person is the founder of the company. The rule is obviously wrong, as there are many people who are born in the same place of a company and have not founded the company. However this rule has a very high coverage over the generation set (many companies’ founders founded their company in their birth place), and a very low coverage over the validation set – indeed among the cartesian product of all the people and companies, a very small fraction includes people and companies born and founded in the same place. Despite such hard cases, our approach is always capable of producing correct rules for those predicates for which we knew there existed some valid rules. Cases like **academicAdvisor**, **child**, and **spouse** have a precision above 95% in all of the KBs, and final precision values are brought down by few predicates where meaningful rules arguably do not exist at all.

The running time is influenced by different factors. First of all the size of the KB has obviously a huge impact, as RuDiK is slower in WIKIDATA which is the biggest KB.

Table 6.2: Positive Rules Accuracy.

<i>KB</i>	<i>Avg. Running Time</i>	<i>Precision</i>
DBPEDIA	34min, 56sec	63.99%
YAGO	59min, 25sec	62.86%
WIKIDATA	2h, 21min, 34sec	73.33%

Not only the number of triples is relevant, but also the different number of predicates. In fact the more predicates we have in the KB, the more alternative paths we observe when traversing the graph, hence a bigger search space. The second relevant aspect is the target predicate involved. We noticed that some kind of entities have a huge number of outgoing and incoming edges (“*United States*” in WIKIDATA is connected to more than 600K entities). When the generation set includes such types of entities, the navigation of the graph is slower as we need to traverse a high number of edges. This is what happens in YAGO, where the most popular predicates are `isLeaderOf` and `exports`. Eventually the *maxPathLen* parameter also has a big say in the final running time. The longer the rule, the bigger is the search space. We will show in the next Section how we can be much faster if we set to 2 atoms the maximum length of the rule. In the next chapter we will discuss some future directions to significantly cut down the running time based on the above observations.

6.4.1.2 Negative Rule Discovery

Negative rules are very useful to discover inconsistencies in the KB. We evaluated negative rules as the percentage of correct errors discovered for the top 5 predicates in each KB. Table 6.3 shows, for each KB, the total number of potential erroneous triples discovered with negative rules, whereas the precision is computed as the percentage of actual errors among potential errors.

Negative rules generally have better accuracy than positive ones. This is mostly due to the completeness of the validation set: for negative rules the validation set is the universe of all possible counter examples stated in the KB, whereas for positive rules the validation set is just a small fraction of it. Therefore negative rules are usually better validated than positive ones. WIKIDATA shows lower numbers just because it does not contain as many errors as DBPEDIA and YAGO: even though discovered rules are almost correct, the percentage of actual errors identified is lower in WIKIDATA. For example, we identify the same rule that two people with same gender cannot be married both in YAGO and WIKIDATA. Such a rule retrieves errors in YAGO with 94% accuracy, while the accuracy in WIKIDATA for the same rule is 57%. YAGO is the KB with the highest number of errors. As an example, there are 9,057 cases in the online YAGO where a child is born before her parent. We cannot point exactly where the error is – there could be an error in one of the birth dates or an error in the parent relation –

Table 6.3: Negative Rules Accuracy.

<i>KB</i>	<i>Avg. Run Time</i>	<i># Potential Errors</i>	<i>Precision</i>
DBPEDIA	19min, 40sec	499	92.38%
YAGO	10min, 40sec	2,237	90.61%
WIKIDATA	1h, 5min, 38sec	1,776	73.99%

but we can affirm that at least one of these values is wrong and send them to human evaluators to spot the inconsistency.

Differently from positive rules, literals play a vital role in discovering negative rules. In fact in many cases correct negative rules rely on temporal aspects in which something cannot happen before/after something else. Temporal information are usually expressed through dates, years, or other primitive types that are represented as literal values in KBs.

As a last observation, discovering negative rules is usually faster than discovering positive rules. This is mostly due to the time we spend executing validation coverage queries. Whenever we discover a rule that respects our language bias (Section 6.1.1), we execute the body of the rule against the KB with a SPARQL query to compute its coverage over the validation set. These queries are faster for negative rules since the validation set is composed of all the entities connected by the target predicate, whereas in the positive case the validation set corresponds to counter examples as described in Section 6.2.1, which are usually more complex to evaluate for standard SPARQL engines.

The table below shows some interesting cases, for both correct and incorrect rules. The example rules show the full power of our language, including literals, smart literals comparisons, entities inequalities and constants. As previously mentioned, the extension including literals comparisons and entities inequalities gives a significant boost in accuracy for negative rules, while it is rarely used in discovering positive rules. The full set of discovered rules can be inspected online at <http://bit.ly/2csR0sc>.

<i>Rule</i>	<i>KB</i>	<i>Precision</i>
<code>notableWork(b, a) ⇒ creator(a, b)</code>	WIKIDATA	100%
<code>nationality(a, v₀) ∧ nationality(b, v₀) ∧ notableStudent(b, a) ⇒ academicAdvisor(a, b)</code>	DBPEDIA	100%
<code>hasChild(v₀, b) ∧ isMarriedTo(a, v₀) ⇒ hasChild(a, b)</code>	YAGO	50%
<code>activeYearsEndDate(a, v₀) ∧ activeYearsStartDate(b, v₀) ⇒ successor(a, b)</code>	DBPEDIA	0%
<code>inception(a, v₀) ∧ dateOfBirth(b, v₁) ∧ v₁ > v₀ ⇒ ¬founder(a, b)</code>	WIKIDATA	100%
<code>country(a, England) ∧ country(b, United_Kingdom) ∧ England ≠ United_Kingdom ⇒ ¬county(a, b)</code>	DBPEDIA	100%
<code>actedIn(a, b) ⇒ ¬wroteMusicFor(a, b)</code>	YAGO	40%
<code>birthYear(a, v₀) ∧ birthDate(b, v₁) ∧ v₀ < v₁ ⇒ ¬academicAdvisor(a, b)</code>	DBPEDIA	28.6%

Table 6.4: Interesting Output Rules.

6.4.2 Comparative Evaluation

We compared the performance of our rule discovery method against AMIE [84], the state-of-the-art system in discovery Horn Rules from KBs.

AMIE is a rule discovery system designed to discover positive rules. It first loads the entire KB into memory, and then discovers positive rules for every predicate in the KB. AMIE lists all the predicates in the KB and inserts each of them as head of the rule. Once the head is filled, the system tries to expand the rule by pivoting on one of the variables of the current predicate and looks for predicates sharing the same variable with high coverage in the KB. The coverage of a rule is penalised with the partial closed world assumption, where the set of negative examples for a given pair (x, y) and a target predicate p is composed of all those pairs where x is connected through p to an entity different from y . Differently from us, AMIE outputs all possible rules that exceeds a given threshold and ranks them according to their coverage function.

Given the in-memory implementation, AMIE cannot handle large KBs. We tried to run it on the KBs of Table 6.1, but the system goes quickly out of memory. Thus we downloaded and used the modified versions of YAGO and DBPEDIA used in their experiments.⁹ These versions consist in the core facts of the KB, without literals and `rdf:type` facts. Table 6.5 summarises the characteristic of this dataset.

Table 6.5: AMIE Dataset characteristics.

<i>KB</i>	<i>Size</i>	<i>#Triples</i>	<i>#Predicates</i>	<i>#rdf:type</i>
DBPEDIA	551M	7M	10,342	22.2M
YAGO	48M	948.3K	38	77.9M

Removing literals and `rdf:type` triples drastically reduce the size of the KB (Table 6.1). Since our approach needs type information (both for the generation of G and V and for the discovery of entities inequality atoms), we run AMIE on its original dataset, while we run our algorithm on the same dataset plus `rdf:type` triples. The last column of Table 6.5 shows how many triples we added to the original AMIE dataset.

Positive Rules. We first compared RuDiK against AMIE on its natural setting: positive rules discovery. AMIE takes as input an entire KB, and discovers rules for every predicate in the KB. We adapted our system to simulate AMIE as follows: we first list all the predicates in the KB, and for each predicate that connects a *subject* to an *object* we computed the most common type for both subject and object. The most common type is computed as the most popular `rdf:type` that is not super class of any other most popular type.

After computing type domain and co-domain for each predicate, we run our approach sequentially on every predicate. Furthermore we set the *maxPathLen* parameter to 2, since this is the default setting for AMIE.

AMIE outputs a huge amount of rules along with their scores – 75 output rules in YAGO, and 6090 in DBPEDIA. We followed their experiments setting and picked the first 30 best rules according to their score. We then picked the rules produced by our approach on the head predicate of the 30 best rules output of AMIE. RuDiK

⁹<http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/amie/>

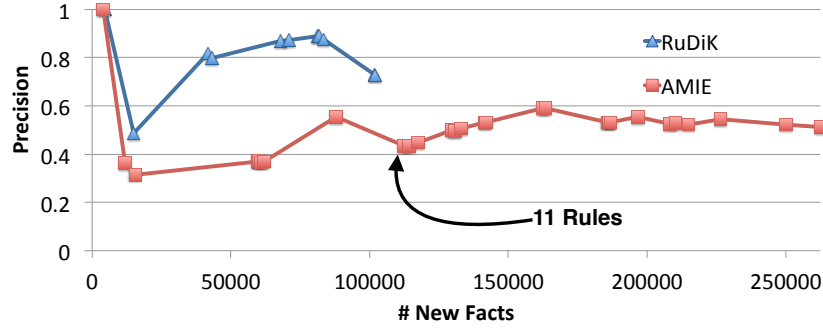


Figure 6.2: Predictions Accuracy on YAGO

is more conservative and produces much less rules than **AMIE**. We noticed that for every predicate **AMIE** always discovers more than one rule, while there are several predicates where the output of our algorithm is empty since none of the plausible rules has enough support. As a consequence, for instance, **RuDiK** outputs just 11 rules for 8 target predicates on the entire YAGO— for the remaining predicates **RuDiK** does not find any rule with enough support.

The precision of each rule is computed as described above with a minor modification: whenever a rule is unknown with its new predictions, we first check the existence of the new induced facts in a newer version of the KB. This is possible because the **AMIE** dataset does not contain the most up to date versions. If a new fact does not appear neither in a newer version nor on the Web, it is evaluated as false. We implemented this small modification to follow the evaluation technique used in [84].

Figure 6.2 plots the total number of new unique predictions (x-axis) versus the aggregated precision (y-axis) on YAGO. The n -th point from the left represents the total number of predictions and the total precision of these predictions, computed over the first n rules (sorted according to **AMIE**’s score). **AMIE** produces many more predictions (262K vs 102K), but with a significant lower accuracy. This is due to the high number of rules in output of **AMIE**, but also to the way these rules are ranked. In fact if we limit the output of **AMIE** to the best 11 rules (same output of our approach), the final accuracy is still 29% below our approach, with just 10K more predictions. **AMIE** outputs many good rules that are preceded by meaningless ones in the ranking, and it is not clear how to set a proper k in order to get the best top k rules. **RuDiK** instead understands that in some cases meaningful rules do not exist, and it outputs something only when it has a strong confidence. This results in a lower number of predictions with a very high accuracy – precision is above 85% before the last rule, with more than 80K predictions. Moreover, our approach can also simulate **AMIE** if we are more interested in recall. By modifying the α and β parameters, we can obtain a higher number of predictions in output, at the expense of accuracy.

Figure 6.3 shows the same evaluation on DBPEDIA. DBPEDIA has a richer set of relations, therefore also **RuDiK** is capable of producing 30 rules in output. Despite the same number of rules, once again our approach leads to a lower number of predictions

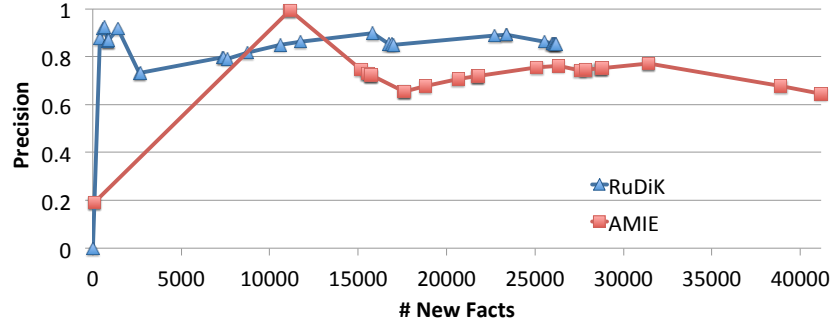


Figure 6.3: Predictions Accuracy on DBPEDIA

(26K vs 41K) with a significant higher accuracy (85% vs 74%). The only point where AMIE outperforms our approach is when we consider the top 3 rules: the third rule discovered by AMIE is indeed a universally true rule that produces more than 11K correct predictions.

Negative Rules. As a second set of comparative experiments we used AMIE to discover negative rules. AMIE is not designed to work in this setting, and can discover rules only for predicates explicitly stated in the KB. Therefore we proceeded as follows: we sampled the top 5 most popular predicates in each KB and we created for each predicate a set of negative examples as explained in Section 6.2.1. For each negative example we added a new fact to the KB connecting the two entities with the *negation* of the predicate. For example, we added a `notSpouse` predicate connecting each pair of people who are not married according to our negative examples generation technique. We then run AMIE on these new created predicates. The evaluation of negative rules is then carried out as explained before: we generate potential errors in the KB, and we manually evaluated the precision of such errors. Table 6.6 shows the results on the two KBs. RuDiK outperforms AMIE in both cases of almost 20%. This is because for the negation of a predicate, we use the actual predicate as counter examples. AMIE instead is not aware that the actual predicate provides counter examples for the negation of it, hence it is much less precise. In fact the output of AMIE consists in a high number of rules for each predicate (often more than 30), and in many cases AMIE produces same rules for both positive and negative scenarios. As an example, AMIE outputs that if a country a exports a good b , then a imports b and a does not import b .

Despite clearly outperforming AMIE, numbers look significantly lower than the ones showed in Section 6.4.1. This is because we are using the AMIE modified KBs which do not contain literals. As explained earlier, literals play a vital role when discover-

Table 6.6: Negative Rules vs AMIE.

<i>KB</i>	AMIE		RuDiK	
	<i># Errors</i>	<i>Precision</i>	<i># Errors</i>	<i>Precision</i>
DBPEDIA	457	38.85%	148	57.76%
YAGO	633	48.81%	550	68.73%

Table 6.7: Run Time vs AMIE.

<i>KB</i>	<i>#Triple</i>	<i>#Predicates</i>	<i>AMIE</i>	<i>RuDiK</i>	<i>Types Time</i>
YAGO 2	948.3K	20	30s	18m,15s	12s
YAGO 2s	4.1M	26 (38)	> 8h	47m,10s	11s
DBPEDIA 2.0	7M	904 (10342)	> 10h	7h,12m	77s
DBPEDIA 3.8	11M	237 (649)	> 15h	8h,10m	37s
WIKIDATA	8.4M	118 (430)	> 25h	8h,2m	11s
YAGO 3*	88.3M	72	-	2h,35m	128s

ing negative rules, both in terms of total errors discovered and in terms of precision. Excluding literals is a big disadvantage, and we will emphasise this aspect in detail in Section 6.4.4.

Running Time. We report here the running time of **AMIE** compared to our approach. Note that numbers are different from [84], where **AMIE** was run on a 48 GB RAM server. **AMIE** could finish the computation only on YAGO 2, while for the other KBs it got stuck without outputting any more rules. When this happened, we stopped the computation after we did not see any new rule in output for more than 2 hours.

Table 6.7 reports the running time on different KBs. The first five KBs are the **AMIE** modified versions, while YAGO 3* is complete YAGO, including literals and `rdf:type`. The third column shows the number of predicates for which **AMIE** was able to produce at least one rule. In some cases **AMIE** got stuck without producing any rule for some predicates, hence we report the total number of predicates in brackets. For a fair comparison we run our algorithm only on those predicates for which **AMIE** could produce at least one rule. The fourth and fifth columns report the total running time of the two approaches. Despite being disk-based, **RuDiK** can successfully complete the task faster than **AMIE** in all cases, except for YAGO 2. This is because of the very small size of the KB, which can easily fits in main memory. However, when we deal with real KBs (YAGO 3*), **AMIE** is not even capable of loading the KB due to out of memory errors. Eventually the last column reports the total time needed to compute `rdf:type` information for each predicate in the KB. This time is negligible w.r.t. the total running time. The running time justifies our disk-based strategy: **RuDiK** can successfully discover rules for any size KB on any machine.

Other Systems. We found other available systems to discover rules in KBs. [15] discovers new facts at instance level, hence it is less generic than our approach. On **AMIE** YAGO 2 KB they can discover 2K new facts with a precision lower than 70%. The best rule we discover on YAGO 2 already produces more than 4K facts with a 100% precision. [38] implements **AMIE** algorithm with a focus on scalability. They do not introduce any novelty in the algorithmic part, but a clever way of splitting the KB into multiple cluster nodes so that the computation can be run in parallel. The output is the same as **AMIE**. Eventually we did not compare with classic Inductive

Logic Programming systems [58, 119], as these are already significantly outperformed by AMIE both in accuracy and running time.

6.4.3 Machine Learning Application

The main goal of this set of experiments is to prove the applicability of our approach in helping Machine Learning algorithms to provide meaningful training examples. We chose DeepDive [135], a Machine Learning approach to incrementally construct KBs. DeepDive extracts entities and relations from text articles via distant supervision. The key idea behind distant supervision is to use an external source of information (e.g., a KB) to provide training examples for a supervised algorithm. For example, the main showcase in DeepDive extracts mentions of married couples from text documents. In such a scenario DeepDive uses as a first step DBPEDIA in order to label some pairs of entities as *true* positive (those pairs of married couples that can be found in DBPEDIA). These labelled examples are then used to construct a factor graph, similar to Markov Logic, that will predict labels on the rest of candidates. Unfortunately, a KB can only provide positive examples. Hence in DeepDive the burden of creating negative examples is left to the user through manual rules definition.

In this set of experiments we will use our negative rules on DBPEDIA to generate negative examples, and we will compare the output of DeepDive trained with different set of negative examples. We used DeepDive showcase example,¹⁰ where the goal is to extract mentions of married people from text articles. DeepDive already provides some negative rules to generate negative examples (e.g., if two people appear in a sentence connected by the words *brother* or *sister* then they are not married). We therefore compare the output of DeepDive using our generated negative examples and the ones generated with DeepDive rules.

Figure 6.4 shows DeepDive accuracy plot run on 1K input documents. The accuracy plot shows the ratio of correct positive predictions over positive and negative predications (y-axis), for each probability output value of DeepDive (x-axis). The dotted blue line represents the ideal situation, where the system finds a high number of evidence positive predictions for higher probability output values – when the output probability is 0 there should not be positive predictions. The plot is computed over a test set, while the system is trained over a separated training set. The figure shows 4 lines other than the ideal ones. RuDiK is the output of DeepDive using our approach to generate negative examples. OnlyPos uses only positive examples from DBPEDIA, Manual uses positive examples from DBPEDIA and manually defined rules to generate negative examples, while ManualSampl uses only a sample of the manually generated negative examples in size equal to positive examples. The first observation is that OnlyPos and Manual do not provide valid training, as the former has only positive

¹⁰<http://deepdive.stanford.edu/>

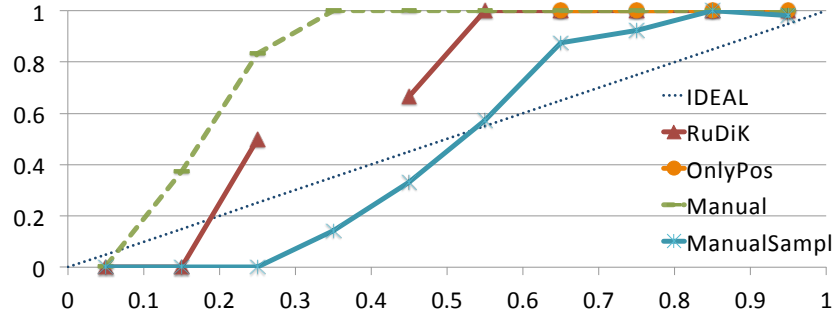


Figure 6.4: DeepDive Application 1K articles

examples and labels everything as true, while the latter has many more negative examples than positive and labels everything as false. **ManualSampl** is the clear winner, while our approach suffers mostly the absence of data: over the input 1K articles, we could find only 20 positive and 15 negative examples from DBPEDIA. The lack of enough evidence in the training data also explains the missing points for **RuDiK** in the chart, where there are not predictions in the probability range 25-45%.

If we extend the input to 1M articles, things change drastically (Figure 6.5). All the three approaches except **OnlyPos** can successfully drive **DeepDive** in the training, with the examples provided with **RuDiK** leading to a slightly better result. This is because of the quality of the negative examples: our negative rules generate representative examples that can help **DeepDive** in understanding discriminatory features between positive and negative labels. The output of **ManualSampl** and **RuDiK** are very similar, meaning that we can use our approach to simulate user behaviour and provide negative examples. With manually defined rules the number of generated examples is significantly higher (23K vs 5K generated with **RuDiK**), however the results are very similar since a small number of significant examples is enough to provide complete evidence for the training. This confirms the main finding of this experiment: as long as we have an external source of information with a decent coverage over the input articles, users do not need to worry about providing rules to generate negative examples.

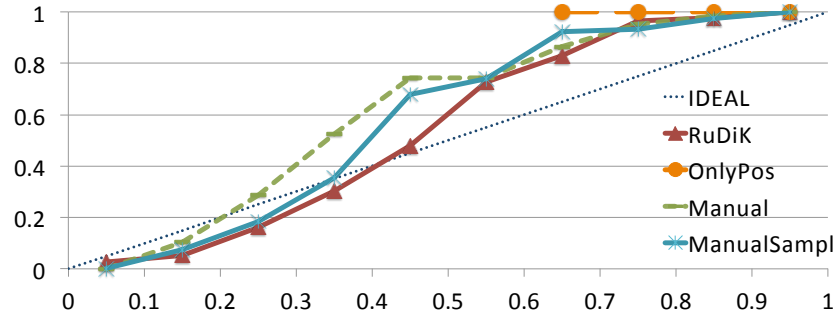


Figure 6.5: DeepDive Application 1M articles

6.4.4 Internal Evaluation

In this last set of experiments we measured the impact of RuDiK relevant features in order to quantify the benefit of three main aspects in rules discovery. We run RuDiK on DBPEDIA with different settings than the standard ones. We report results on the same top-5 predicates of Section 6.4.1 for both positive and negative rules.

<i>Type</i>	With Literals		Without Literals	
	<i>Run Time</i>	<i>Precision</i>	<i>Run Time</i>	<i>Precision</i>
Positive Rules	~35min	63.99%	~54min	60.49%
Negative Rules	~20min	92.38% (499)	~25min	84.85% (235)

Table 6.8: Rules Accuracy without Literals on DBPEDIA.

Effect of Literals. Since previous KB rules discovery approaches exclude literals from the mining [15, 38, 84], we wanted to quantify the impact of having literal rules. Thus we run RuDiK excluding all literal values. Table 6.8 reports the output precision with and without literals. Including literal values in the mining has a considerable impact on final accuracy, both for positive and negative rules. The effect is particularly evident for negative rules, where excluding literals involves finding less than half potential errors (numbers in brackets) with a lower precision. **founder** is the most evident example: RuDiK discovers 79 potential errors with a 95% precision with literal rules, while there are not output errors for rules without literals.

Surprisingly, including literals reduces also the running time. This is due to the pruning effect of the A^* algorithm: if we include literals the algorithm can find rather soon valid literal rules, which causes the pruning of several paths on the graph. If we excluded literals instead these paths cannot be pruned and needs to be inspected by the algorithm, which entails a bigger search space.

Rules Length Impact. The *maxPathLen* parameter fixes the maximum number of atoms allowed in the body of a rule. Low values for *maxPathLen* may exclude from the search space meaningful rules, while high values may exponentially increase the search space and consequently the running time. Table 6.9 reports accuracy values and run times for different *maxPathLen* settings. If we set *maxPathLen* = 2 we observe a significant improvement in running time, at the expense of losing several meaningful rules. In particular we loose all the rules that involve literals comparison, as these require a minimum of three atoms in the body. This is particularly evident for negative rules, where we spot a fewer number of errors (in round brackets) with

Table 6.9: *maxPathLen* Parameter Impact on DBPEDIA.

<i>Type</i>	<i>maxPathLen</i> = 2		<i>maxPathLen</i> = 3		<i>maxPathLen</i> = 4	
	<i>Run Time</i>	<i>Precision</i>	<i>Run Time</i>	<i>Precision</i>	<i>Run Time</i>	<i>Precision</i>
Positive	~ 3min	49.17%	~35min	63.99%	> 24h	66%
Negative	~ 56sec	90% (131)	~20min	92.38% (499)	> 24h	92.71% (302)

a lower precision. At the other side of the spectrum, with $maxPathLen = 4$ the search space explodes and RuDiK was not capable of finishing the computation within 24 hours for each predicate. We therefore report the accuracy of rules discovered in 24 hours of computation. The results are comparable to those computed with $maxPathLen = 3$, provided the absence of rules that might have been discovered after 24 hours. Furthermore, we did not notice any new discovered correct rule with body length equal to 4. Rules with length 4 are usually very complex to understand, and when executed over the KB they often return an empty result (i.e., they are useless both for discovering additional information and for identifying inconsistencies). Here a couple of examples of output rules with length 4:

$$\begin{aligned} & \text{birthPlace}(a, v_0) \wedge \text{areaTotal}(v_0, v_1) \wedge \text{viafId}(b, v_2) \wedge v_1 < v_2 \Rightarrow \neg \text{academicAdvisor}(a, b) \\ & \text{foundingYear}(a, v_2) \wedge \text{birthPlace}(b, v_0) \wedge \text{foundingYear}(v_1, v_2) \wedge \text{foundationPlace}(v_1, v_0) \Rightarrow \text{founder}(a, b) \end{aligned}$$

We therefore decided that $maxPathLen = 3$ is a good compromise between a reasonable running time and a good output accuracy.

Negative Examples Generation. A key point in RuDiK is the generation of negative examples with a modified version of the LCWA (Section 6.2.1). We therefore evaluate two different strategies to generate negative examples. Given a target predicate p from a KB $\mathcal{K}$, we call t_a the most common type of entities that are subject of p , and t_b the most common type for the object. As an example, if $p = \text{founder}$, $t_a = \text{Company}$ and $t_b = \text{Person}$. We define k as the number of triples having p as predicate in $\mathcal{K}$ – k is the cardinality of the positive examples set. The first alternative negative examples generation strategy is *Random*: we randomly select k pairs (x, y) from the cartesian product of all entities x of type t_a and all entities y of type t_b , such that the triple $\langle x, p, y \rangle \notin \mathcal{K}$. The second generation strategy, named *LCWA_Random*, leverages on the LCWA. In *LCWA_Random* we randomly pick k pairs (x, y) from the cartesian product of all entities x of type t_a and all entities y of type t_b , such that: (i) $\langle x, p, y' \rangle \in \mathcal{K}$, with $y' \neq y$; (ii) $\langle x', p, y \rangle \in \mathcal{K}$, with $x' \neq x$; (iii) $\langle x, p, y \rangle \notin \mathcal{K}$. *LCWA_Random* is equivalent to RuDiK generation strategy, minus the constriction that x and y must be connected by a predicate different from p . The constraint of having k examples aims at reducing the size of the cartesian product that can easily explode otherwise.

Table 6.10 reports the accuracy of discovering negative rules with the three different generation strategies (RuDiK is our modified LCWA strategy). *Random* and

Table 6.10: Effect of Negative Examples Generation Strategy on DBPEDIA.

Strategy	# Potential Errors	Precision
<i>Random</i>	247	95.95%
<i>LCWA_Random</i>	263	95.82%
RuDiK	499	92.38%

LCWA_Random strategies show very similar behaviours, with a slightly better precision than *RuDiK*. This is because whenever we randomly pick examples from the cartesian product of subject and object, the likelihood of picking entities from a different time period is very high, and negative rules pivoting on time constraints are usually correct. As an example, a correct rule for the target predicate `child` is $\text{birthDate}(a, v_0) \wedge \text{birthDate}(b, v_1) \wedge v_0 > v_1 \Rightarrow \neg \text{child}(a, b)$, stating that whenever a person a is born after a person b , then b cannot be child of a . The likelihood of randomly picking some pairs of persons (x, y) where x is born after y is very high, hence such kind of rules will often be discovered. However these are the *only* kind of rules that random generation strategies are able to find, as shown from the smaller number of errors that they can identify. Instead, the constriction of forcing x and y to be connected by a different predicate generates negative examples with heterogeneous properties that lead to different results. Rules such as $\text{parent}(a, b) \Rightarrow \neg \text{spouse}(a, b)$ are very unlikely to be generated with random strategies, since the likelihood of picking two people that are in a parent relation is very low. The generation strategy we adopt in *RuDiK* allows the discovery of more types of rules, and not only rules involving time constraints. This has multiple benefits: on the one hand, we discover more rules, which entails discovering a higher number of errors (Table 6.10); on the other hand, different rule types lead to high quality negative examples, that can be used in several applications such as training Machine Learning algorithms (see Section 6.4.3).

6.5 Related Work

Our work is inspired by Inductive Logic Programming, but uses techniques from dependencies discovery in relational databases, KBs construction, and mining graph patterns. We review a few of the most relevant works below.

Relational Database Constraints. A significant body of work has addressed the problem of discovering constraints over relational database sources. Functional Dependencies (FDs), a formalism to express relations and dependencies among attributes, has been studied in constraints discovery literature for more than 20 years [17], with a recent focus on performance [16]. FDs can be grouped into two strands: the schema-level approaches [97] (similar to our positive rules discovery), and the instance-driven approaches [153]. More recently, Conditional FDs extend standard FDs by enforcing patterns of semantically related constants [71]. In the context of inconsistencies discovery for relational data, Denial Constrains (DCs) are the current state-of-the-art techniques [41]. DCs are a universally quantified first order logic formalism to express constraints over relational data, and they are directly related to our negative output rules. Efficient DCs algorithms have been proposed for data cleaning and consistent query answering [26, 42].

Despite being directly related to our output and expressing a richer language, FDs and DCs cannot be applied to RDF databases for three main reasons: *(i)* the schema-less nature of RDF data and the closed world assumption which no longer holds on RDF KBs; *(ii)* FDs and DCs techniques rely on the assumption that data is either clean or has a negligible amount of errors; *(iii)* scalability issues on large RDF dataset: applying relational database techniques on RDF KBs would imply to materialise all possible predicate combinations into relational tables. Some FDs approaches focus on dataset with erroneous data [14, 103], however they are still inapplicable to RDF data due to scalability problems.

KBs Rule Mining. Recently, the focus for constraints and rule discovery is moving towards RDF databases. The closest works to ours are AMIE [84] and OP algorithm [38], which discovers positive Horn Rules from RDF KBs with same language biases. They both use the same discovery algorithm: it first loads the entire KB into memory and then, working one predicate at time, tries to expand rules by connecting a predicate to others that share common variables. Rules are ranked according to a confidence measure that leverages on KBs partial closed world assumption. Our graph generation and navigation technique is similar to their approach, however our examples generation allows us to discover rules on just a small fraction of the KB. This is beneficial not only from a scalability point of view (see Section 6.4.2), but also from the language perspective: neither of the two approaches can afford smart literals comparisons. We have not tested our running time against [38] because we could not find an implementation of it, however it requires a powerful cluster of several machines to split the KB into multiple nodes. We showed in the experimental section how RuDiK outperforms AMIE both in final accuracy and running time.

[15] is an instance-level approach that discovers new facts for specific entities rather than generic variables. We showed in the experimental section that RuDiK is capable of generating more facts with a better precision with just a single rule. [75] is a modern system to discover Conditional Denial Constraints (CDCs) from RDF Data. Differently from other systems, it includes literals in its language. CDCs can be directly mapped to our negative rules, however there is not a general correlation between CDCs and positive rules. Another major difference of our setting is the hardware: our disk-based approach is designed to handle large KBs with limited memory resources, while [75] works on a distributed environment with a total of 832 GB RAM memory.

To the best of our knowledge, RuDiK is the first approach that is generic enough to use the same algorithm to discover both positive and negative rules in RDF KBs.

Inductive Logic Programming. Inductive Logic Programming (ILP) is a sub-field of Machine Learning and Logic Programming which investigates the inductive construction of first-order Horn Rules from examples and background knowledge, usually expressed through logic formalisms [120]. RuDiK can be seen as an ILP system

where the KB is the background knowledge, and the generation and validation sets correspond to positive and negative training examples.

WARMR is an ILP system that discovers frequent patterns (expressed through **DATALOG** queries) that succeed with respect to a sufficient number of examples [58]. When translated to databases, such patterns correspond to conjunctive queries. **ALEPH**¹¹ is an available ILP system that is based on Prolog Inverse Entailment [119]. **ALEPH** works iteratively, by selecting examples from the background knowledge. It first constructs the most specific clause that entails the example selected (*bottom clause*), and then searches for some subset of the literals in the bottom clause that has the *best score* in order to define more general rules. The system allows the user to choose among several scoring functions. ILP systems such as **WARMR** and **ALEPH** are designed to work with a closed world assumption, and always require the definition of positive and negative examples. **AMIE** clearly outperforms these two systems [84], showing evident scalability issues when dealing with medium-size KBs. Moreover, one of the main limitations of classic ILP systems is the assumption of having high-quality, errors-free training examples. We showed how this assumption does no longer hold on incomplete and erroneous KBs.

Sherlock [133] is an interesting ILP system that learns first-order Horn Rules from Web text. One of the key advantage of **Sherlock** is being unsupervised: it does not require negative training examples. It uses statistical significance and statistical relevance in order to discover rules that exceed a given threshold. Differently from our setting, **Sherlock** is specifically designed to learn Horn Rules from open domain facts that are extracted from the Web. An interesting future direction is to adapt **RuDiK** to discover rules from relations extracted from free text rather than on a well-defined KBs.

ILP systems take inspiration from Association Rule Mining [18], where given a database of costumer transactions the goal is to discover all frequent itemsets, i.e., all combinations of items that are found together with some minimum confidence. A well known example in a supermarket database could state that 90% of transactions that purchase bread and butter also purchase milk. As previously mentioned, adapting such a relational database setting to KBs would require the materialisation of all possible predicate combinations into relational tables.

Eventually, most of the ILP and KBs rule discovery systems rank rules according to a support value, and output only those rules that exceed a given threshold. We showed in Section 6.4.2 that properly setting such thresholds is not trivial, as often good rules are preceded in rank by meaningless ones. **RuDiK** does not use any threshold and outputs rules only when coverages over generation and validation sets are considered acceptable.

¹¹<http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/aleph>

Relation to other areas. Our examples generation strategy leverages on the Local Closed World Assumption (LCWA). When dealing with incomplete KBs, the LCWA is a popular technique that replaces the canonical Closed World Assumption of standard relational databases. The LCWA has been used in Google Knowledge Vault to estimate the quality of extracted triples [62, 60], AMIE [84] uses the LCWA to penalise discovered rules, and sometimes the LCWA is used to evaluate the quality of a target KB [63]. We see our examples generation strategy as complementary to our approach. It is possible to run RuDiK with any input examples, no matter how such examples have been generated.

Our graph-based rule discovery approach is close in spirit to mining graph patterns [66, 162], where given a (big) input graph the goal is to discover the most frequent patterns (subgraphs) according to some scoring functions. Our setting presents a key difference: we primarily look at edge labels and we are not interested in node labels, since nodes are mapped to variables when translating subgraphs to Horn Rules. Moreover, given the portion of the graph between two entities x and y , we are interested in discovering *all* possible subgraphs between x and y , which makes the problem easily solvable with BFS-like techniques.

Another interesting setting is the one of SpiderMine [160]. SpiderMine looks for the top- K largest patterns in a graph, where each node in the pattern is at most at distance r from a head vertex u . In our setting we do not have a single head vertex, but rather many vertexes (starting nodes) from which we begin the path computation. Furthermore our goal is not to find graph patterns (subgraphs), but we are simply interested in all possible paths between a pair of vertexes.

Fan et. al. [74] laid the theoretical foundations of Functional Dependencies on Graphs (GFDs). They also propose parallel algorithms for GFDs computation and evaluate the accuracy on YAGO and DBPEDIA. Despite existing a natural correlation between FDs and Horn Rules, the language they propose covers only a portion of our negative rules to detect inconsistencies in graph databases. Moreover their language does not include smart literals comparisons, shown to be vital when detecting errors in KBs.

6.6 Main Takeaways

In the last part of this thesis we studied the problem of discovering positive and negative rules over RDF KBs. We showed that despite being largely incomplete and erroneous, KBs carry a big amount of hidden information that can be extremely beneficial to discover. We presented a system that discovers such hidden information by inducing rules that, when instantiated over the KB, can both enrich the KB with implicit data and also discover erroneous facts and contradictions. We showed how the classic database closed world assumption does not longer hold on KBs, and how we can overcome this

limitation by exploiting the LCWA to compute positive and negative examples for a given predicate. We implemented these techniques in **RuDiK**, a fast and scalable disk-based solution for rules discovery that does not rely on the assumption that the KB has to fit in main memory. **RuDiK** overcomes both in quality and running time all competitor systems. Eventually we gave several example applications of rules discovery, including the improvement of Machine Learning algorithms by providing meaningful training examples.

Chapter 7

Conclusion

This thesis work focuses on the automated discovery of repairing rules for web information extraction: rules that can repair erroneous and misaligned data extracted from the Web, adjusting the extraction process where possible such that future executions will produce clean data. We inspected the problem from three different angles with the following fundamental results:

- (1) An extensible framework for knowledge-aware integration of entity extractors that considers each entity extractor as a “black-box” and does not assume any prior knowledge about their performance or competence w.r.t. a certain set of entities. We implemented a fully unsupervised system, namely **ROSeAnn**, to integrate and reconcile semantic annotations on top of semantic entity extractors. We showed that current entity extractors suffer from serious accuracy deficiency (Chapter 2), and that we can benefit from the integration of several systems both in terms of a larger vocabulary and in terms of a significant boost in accuracy (Chapter 3).
- (2) Though many works have addressed the problem of extracting and repairing data extracted from the Web, most of these works leverage on instance-level, cross-site redundancy that seriously limits their scalability (Chapter 4). We proposed **WADaR**, an unsupervised approach that replaces redundancy with the ensemble of entity recognisers encoded in **ROSeAnn** (Chapter 5). Such entity recognisers are used to discover hidden attribute structures in web extracted relations and construct relation-wide repair rules that can adjust both the extracted relation and the extraction program where possible. **WADaR** can improve the quality of web extracted data by 15%-60%, being both independent of the involved extraction system and of the target vertical domain. Moreover, we showed how inducing most likely attribute sequences cope well with annotators noise and ambiguity, resulting in high quality repairs even when the recall of the input annotators is below 20%.
- (3) Despite alleviating most of the restrictions due to redundancy-based methods, implementing and maintaining annotators is still a tedious work that often requires

domain experts. Therefore in the last part of the thesis we dropped the assumption of relying on semantic annotators and we exploited entity-level relationships extracted from the Web in order to discover hidden and conflicting pieces of information. We presented RuDiK, a scalable system for the discovery of declarative rules over knowledge-bases (Chapter 6). RuDiK output is not limited to rules that rely on “positive” relationships between entities, but it discovers also “negative” rules, i.e., patterns that lead to contradictions in the data. While the former class is fundamental to infer new relationships in the KB, the latter class is crucial for other tasks, such as error detection in data cleaning. We introduced a language with a higher expressive power w.r.t. competitive systems and encoded the problem to allow the discovery of approximate rules that are robust to errors and incompleteness in the KB.

With the advent of big data a major issue in web information extraction concerns scalability. We showed how the presented techniques that encode cross-site redundancy with an ensemble of entity recognisers scale well with the number of input sources, and how the identification of representative examples allows fast and scalable discovery of positive and negative rules in KBs, no matter what the size of the KB is. In this thesis we presented three innovative techniques that are not only capable of clearly outperforming state-of-the-art systems both in terms of precision and recall, but that are also significantly faster and less resource demanding than any other available competitor.

7.1 Future Directions

Obviously, there are still numerous open questions that we did not address in this thesis and that we would like to further investigate in the near future. We briefly highlight some interesting future works.

- (1) **Semi-Unsupervised Annotation Aggregation.** We outlined in ROSeAnn an aggregation technique that does not require any per-annotator/per-concept supervision. However, we briefly showed how supervised techniques can deliver better performances when training data is available. An interesting open direction is how to combine these methods, and more specifically, how to make the system aware of which concepts could benefit from supervision and which can be easily integrated without. ROSeAnn currently produces an aggregate annotation without producing a measure of confidence for each annotation. We are currently investigating a modification in the output such that concepts that have several annotators supporting them would be associated with a strong signal of correctness.
- (2) **Automated Ontology Creation.** Another interesting direction in ROSeAnn is how to automatically adjust the merged ontology whenever concept classes are added/removed from annotator vocabularies. At the moment our system can only

report when a new class is not recognised in the ontology, leaving to the user the burden of manually updating the ontology. We could leverage on similar known annotations occurring on the same text span to automatically infer the semantic meaning of newly discovered classes.

- (3) **More NLP Capabilities.** Entity recognition is the most-widely supported function among online annotators, however some of them offer more NLP capabilities such as *entity disambiguation* or *relation extraction*. Such additional functions could be well coupled with the techniques showed both in WADaR and RuDiK. However, our algorithm would need to be adapted to more complex reasoning than simply subsumption and disjointness, since the ontology may have richer constraints (e.g., part-of, foreign keys) that would break the polynomial assumption proved in Theorem 1.
- (4) **Bootstrap Annotators.** The most likely sequences we identify in WADaR are capable of dropping wrong annotations, as well as inferring annotation concepts on unannotated span texts. We could use such repairs to incrementally learn entity recognisers, both in terms of precision (elimination of wrong annotations), and in terms of recall (discovery of new attribute values through regex generalisation). We showed how our approach delivers high quality repairs even when annotators' recall is below 20%. We could hence manually create an annotator that identifies just few correct instances and expand its capabilities from there.
- (5) **Semantic XPath.** The repair rules output of WADaR has the same expressive power of regular expressions. We can easily identify subcontent based on the syntax, such that these expressions can be directly encoded in XPath wrappers. Oftentimes, we noticed a *semantic* regularity in the extracted relations rather than a syntactic regularity. A very promising future direction would be to extend XPath with entity recognition capabilities, to identify for example that COUNTY is usually after CITY and before POSTCODE in a given extracted attribute. Such an extension on the one hand would increase the expressive power of the language, on the other hand it would also increase the complexity of the language and of the evaluation of these semantic expressions.
- (6) **Multivalued Attributes.** Currently WADaR is designed to work only with single-value attributes, and it does not handle situations where an attribute can have multiple values for a single entity (e.g., actors of a movie). We are planning to extend our approach to also be robust to multivalued attributes, in particular to allow sequences having multiple nodes with the same attribute label. This essentially translates to allowing multiple text spans labelled with the same attribute value within the same attribute content.

- (7) **RuDiK Optimisation Techniques.** The size of the example generation set has a relevant impact on rule discovery algorithm in RuDiK. We are currently investigating smart sampling techniques to reduce the size of the generation set even further, as we noticed that just a few examples are enough for the discovery of meaningful rules. We believe that we can significantly reduce the search space by sampling examples based on measures such as diversity and popularity [70]. Furthermore, we noticed that navigating similar paths within the same example graph is often redundant. For example, “*United States*” is connected to more than 600K entities in WIKIDATA via the predicate `livesIn`. It is not necessary to navigate all of the 600K edges, as most of them identify equivalent paths that translate to equivalent rules.
- (8) **Temporal and Local Rules.** Another interesting future direction is to discover more expressive rules that can exploit temporal and local information through smarter analysis of literals [14]. For instance, “if two people are not born in the same century, then they cannot be married” is an example rule that requires non-trivial analysis of temporal information. Furthermore, we discovered rules that are not universally valid but become valid within a certain period or within a certain location. The rule “two people with same gender cannot be married” is not universally valid, however it was valid until few decades ago or it is still valid in countries like Russia.
- (9) **Combine Positive and Negative Rules.** We approach the discovery of positive and negative rules in isolation. A current ongoing work aims at combining them. In fact if a positive and a negative rule for the same target predicate both identify shared instances over the KB, then one of them must be wrong. We could use such signals to drive the search and prune those paths covered by contradictory rules.

Bibliography

- [1] MUC7. <http://www ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2001T02>.
- [2] Reuters. <http://about.reuters.com/researchandstandards/corpus/index.asp>.
- [3] **AlchemyAPI**. <http://www.alchemyapi.com>.
- [4] **CiceroLite**. <http://www.languagecomputer.com/products/text-annotation/cicerolite.html>.
- [5] **LingPipe**. <http://alias-i.com/lingpipe/>.
- [6] **Lupedia**. <http://www.old.ontotext.com/lupedia>.
- [7] **OpenCalais**. <http://www.opencalais.com>.
- [8] **ROSeAnn**. <http://diadem.cs.ox.ac.uk/roseann>.
- [9] **Saplo**. <http://saplo.com/api>.
- [10] **WADaR**. <http://diadem.cs.ox.ac.uk/wadar>.
- [11] **Wikimeta**. <http://www.w3.org/2001/sw/wiki/Wikimeta>.
- [12] **Yahoo Analysis API**. <https://developer.yahoo.com/contentanalysis/>.
- [13] **Zemanta**. <http://www.zemanta.com>.
- [14] Ziawasch Abedjan, Cuneyt G Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. Temporal rules discovery for web data cleaning. *Proceedings of the VLDB Endowment*, 9(4):336–347, 2015.
- [15] Ziawasch Abedjan and Felix Naumann. Amending rdf entities with new facts. In *European Semantic Web Conference*, pages 131–143. Springer, 2014.
- [16] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. Dfd: Efficient functional dependency discovery. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 949–958. ACM, 2014.
- [17] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.
- [18] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.
- [19] Arnon Amir, Marco Berg, Shih-Fu Chang, Winston Hsu, Giridharan Iyengar, Ching-Yung Lin, Milind Naphade, Apostol Natsev, Chalapathy Neti, Harriet Nock, et al. Ibm research trecvid-2003 video retrieval system. *NIST TRECVID-2003*, 2003.

- [20] Javed A Aslam and Mark Montague. Models for metasearch. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 276–284. ACM, 2001.
- [21] Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction for the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.
- [22] Roberto Basili, Alessandro Moschitti, Maria Teresa Pazienza, and Fabio Mossimo Zanzotto. Personalizing web publishing via information extraction. *IEEE Intelligent systems*, (1):62–70, 2003.
- [23] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *VLDB*, volume 1, pages 119–128, 2001.
- [24] Oliver Bender, Franz Josef Och, and Hermann Ney. Maximum entropy models for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 148–151. Association for Computational Linguistics, 2003.
- [25] Michael Benedikt and Christoph Koch. Xpath leashed. *ACM Computing Surveys (CSUR)*, 41(1):3, 2009.
- [26] Leopoldo Bertossi. Database repairing and consistent query answering. *Synthesis Lectures on Data Management*, 3(5):1–121, 2011.
- [27] Leopoldo Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4):407–434, 2008.
- [28] Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and dtlds. *ACM Transactions on Database Systems (TODS)*, 35(2):11, 2010.
- [29] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web*, 7(3):154–165, 2009.
- [30] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 143–154. ACM, 2005.
- [31] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.
- [32] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and integration of partially overlapping web sources. *Proceedings of the VLDB Endowment*, 6(10):805–816, 2013.
- [33] Paul Buitelaar, Philipp Cimiano, Stefania Racioppa, and Melanie Siegel. Ontology-based information extraction with soba. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2006.

- [34] Michael J Cafarella, Alon Halevy, and Jayant Madhavan. Structured data on the web. *Communications of the ACM*, 54(2):72–79, 2011.
- [35] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.
- [36] Rafael Carreira, Sónia Carneiro, Rui Pereira, Miguel Rocha, Isabel Rocha, Eugénio C Ferreira, and Anália Lourenço. Semantic annotation of biological concepts interplaying microbial cellular responses. *BMC bioinformatics*, 12(1):460, 2011.
- [37] Luying Chen, Stefano Ortona, Giorgio Orsi, and Michael Benedikt. Aggregating semantic annotators. *Proceedings of the VLDB Endowment*, 6(13):1486–1497, 2013.
- [38] Yang Chen, Sean Louis Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. Ontological pathfinding: Mining first-order knowledge from large knowledge bases. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, pages 835–845. ACM, 2016.
- [39] Boris Chidlovskii, Bruno Roustant, and Marc Brette. Documentum eci self-repairing wrappers: Performance analysis. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 708–717. ACM, 2006.
- [40] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. Tegra: Table extraction by global record alignment. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1713–1728. ACM, 2015.
- [41] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.
- [42] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 458–469. IEEE, 2013.
- [43] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1247–1261. ACM, 2015.
- [44] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. Context-aware wrapping: synchronized data extraction. In *Proceedings of the 33rd international conference on Very large data bases*, pages 699–710. VLDB Endowment, 2007.
- [45] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.
- [46] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. Towards the self-annotating web. In *Proceedings of the 13th international conference on World Wide Web*, pages 462–471. ACM, 2004.
- [47] Fabio Ciravegna and Alberto Lavelli. Learningpinocchio: Adaptive information extraction for real world applications. *Natural Language Engineering*, 10(02):145–165, 2004.
- [48] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd international conference on Very large data bases*, pages 315–326. VLDB Endowment, 2007.

- [49] Eli Cortez, Altigran S da Silva, Marcos André Gonçalves, and Edleno S de Moura. Ondux: on-demand unsupervised learning for information extraction. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 807–818. ACM, 2010.
- [50] Eli Cortez, Daniel Oliveira, Altigran S da Silva, Edleno S de Moura, and Alberto HF Laender. Joint unsupervised structure discovery and information extraction. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 541–552. ACM, 2011.
- [51] Valter Crescenzi and Giansalvatore Mecca. Automatic information extraction from large websites. *Journal of the ACM (JACM)*, 51(5):731–779, 2004.
- [52] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. A framework for learning web wrappers from the crowd. In *Proceedings of the 22nd international conference on World Wide Web*, pages 261–272. ACM, 2013.
- [53] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. 2011.
- [54] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 541–552. ACM, 2013.
- [55] Nilesh Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 335–348. ACM, 2009.
- [56] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230, 2011.
- [57] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment*, 5(7):680–691, 2012.
- [58] Luc Dehaspe and Hannu Toivonen. Discovery of frequent datalog patterns. *Data Mining and knowledge discovery*, 3(1):7–36, 1999.
- [59] Omkar Deshpande, Digvijay S Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Building, maintaining, and using knowledge bases: a report from the trenches. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1209–1220. ACM, 2013.
- [60] Xin Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmman, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.
- [61] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Integrating conflicting data: the role of source dependence. *Proceedings of the VLDB Endowment*, 2(1):550–561, 2009.

- [62] Xin Luna Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *Proceedings of the VLDB Endowment*, 7(10):881–892, 2014.
- [63] Xin Luna Dong, Evgeniy Gabrilovich, Kevin Murphy, Van Dang, Wilko Horn, Camillo Lugaresi, Shaohua Sun, and Wei Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *Proceedings of the VLDB Endowment*, 8(9):938–949, 2015.
- [64] Deborah Duong, Jim Venuto, Ben Goertzel, Ryan Richardson, Shawn Bohner, and Edward Fox. Support vector machines to weight voters in a voting system of entity extractors. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 1226–1230. IEEE, 2006.
- [65] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078–1089, 2009.
- [66] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7):517–528, 2014.
- [67] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Web-scale information extraction in knowitall. In *Proceedings of the 13th international conference on World Wide Web*, pages 100–110. ACM, 2004.
- [68] Jérôme Euzenat. Semantic precision and recall for ontology alignment evaluation. In *IJCAI*, pages 348–353, 2007.
- [69] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Cleaning inconsistencies in information extraction via prioritized repairs. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 164–175. ACM, 2014.
- [70] Georgios Fakas, Zhi Cai, and Nikos Mamoulis. Diverse and proportional size-l object summaries for keyword search. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 363–375. ACM, 2015.
- [71] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering*, 23(5):683–698, 2011.
- [72] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyan Yu. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment*, 3(1-2):173–184, 2010.
- [73] Wenfei Fan, Shuai Ma, Nan Tang, and Wenyan Yu. Interaction between record matching and data repairing. *Journal of Data and Information Quality (JDIQ)*, 4(4):16, 2014.
- [74] Wenfei Fan, Yinghui Wu, and Jingbo Xu. Functional dependencies for graphs. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, pages 1843–1857. ACM, 2016.
- [75] Mina H. Farid, Alexandra Roatis, Ihab F. Ilyas, Hella-Franziska Hoffmann, and Xu Chu. Clams: Bringing quality to data lakes. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, pages 2089–2092. ACM, 2016.
- [76] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

- [77] David Ferrucci and Adam Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.
- [78] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.
- [79] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Querying and repairing inconsistent numerical databases. *ACM Transactions on Database Systems (TODS)*, 35(2):14, 2010.
- [80] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 168–171. Association for Computational Linguistics, 2003.
- [81] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.
- [82] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Diadem: Thousands of websites to a single database. *Proceedings of the VLDB Endowment*, 7(14):1845–1856, 2014.
- [83] Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, and Andrew Sellers. Oxpath: A language for scalable data extraction, automation, and crawling on the deep web. *The VLDB Journal*, 22(1):47–72, 2013.
- [84] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24(6):707–730, 2015.
- [85] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Ajax: an extensible data cleaning tool. In *ACM Sigmod Record*, volume 29, page 590. ACM, 2000.
- [86] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. Corroborating information from disagreeing views. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 131–140. ACM, 2010.
- [87] Paul Suganthan GC, Chong Sun, Haojun Zhang, Frank Yang, Narasimhan Rampalli, Shishir Prasad, Esteban Arcaute, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, et al. Why big data industrial systems need rules and what we can do about it. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 265–276. ACM, 2015.
- [88] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. The llunatic data-cleaning framework. *Proceedings of the VLDB Endowment*, 6(9):625–636, 2013.
- [89] Anna Lisa Gentile, Ziqi Zhang, Isabelle Augenstein, and Fabio Ciravegna. Unsupervised wrapper induction using linked data. In *Proceedings of the seventh international conference on Knowledge capture*, pages 41–48. ACM, 2013.
- [90] Andrew V Goldberg and Robert E Tarjan. Efficient maximum flow algorithms. *Communications of the ACM*, 57(8):82–89, 2014.

- [91] Davide Grossi and Gabriella Pigozzi. Introduction to judgment aggregation. In *Lectures on Logic and Computation*, pages 160–209. Springer, 2012.
- [92] Frank E Grubbs. An introduction to probability theory and its applications. *Technometrics*, 9(2):342–342, 1967.
- [93] Dan Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of mathematical biology*, 55(1):141–154, 1993.
- [94] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 775–784. ACM, 2011.
- [95] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [96] Stephan Hartmann, Gabriella Pigozzi, and Jan Sprenger. Reliable methods of judgement aggregation. *Journal of Logic and Computation*, 20(2):603–617, 2010.
- [97] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.
- [98] Sham Kakade, Yee Whye Teh, and Sam T Roweis. An alternate objective function for markovian fields. In *ICML*, pages 275–282, 2002.
- [99] Nanda Kambhatla. Minority vote: at-least-n voting improves recall for extracting relations. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 460–466. Association for Computational Linguistics, 2006.
- [100] Anitha Kannan, Inmar E Givoni, Rakesh Agrawal, and Ariel Fuxman. Matching unstructured product offers to structured product specifications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 404–412. ACM, 2011.
- [101] Vangelis Karkaletsis, Constantine D Spyropoulos, Claire Grover, M Pazienza, Jose Coch, and Dimitris Souflis. A platform for cross-lingual, domain and user adaptive web information extraction. In *ECAL*, volume 16, page 725, 2004.
- [102] Atanas Kiryakov, Borislav Popov, Ivan Terziev, Dimitar Manov, and Damyan Ognyanoff. Semantic annotation, indexing, and retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):49–79, 2004.
- [103] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.
- [104] Judice LY Koh, Mong Li Lee, Wynne Hsu, and Kai Tak Lam. Correlation-based detection of attribute outliers. In *Advances in Databases: Concepts, Systems and Applications*, pages 164–175. Springer, 2007.
- [105] Zornitsa Kozareva, Óscar Ferrández, Andrés Montoyo, Rafael Muñoz, Armando Suárez, and Jaime Gómez. Combining data-driven systems for improving named entity recognition. *Data & Knowledge Engineering*, 61(3):449–466, 2007.
- [106] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

- [107] Kristina Lerman, Steven Minton, and Craig A Knoblock. Wrapper maintenance: A machine learning approach. *J. Artif. Intell. Res.(JAIR)*, 18:149–181, 2003.
- [108] Liping Ma and John Shepherd. Information extraction using two-phase pattern discovery. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 534–535. ACM, 2004.
- [109] Ashwin Machanavajjhala, Arun Shankar Iyer, Philip Bohannon, and Srujana Merugu. Collective extraction from heterogeneous web lists. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 445–454. ACM, 2011.
- [110] Alexander Maedche, Günter Neumann, and Steffen Staab. Bootstrapping an ontology-based information extraction system. In *Intelligent exploration of the web*, pages 345–359. Springer, 2003.
- [111] Imran R Mansuri and Sunita Sarawagi. Integrating unstructured data into relational databases. In *Data Engineering, 2006. ICDE’06. Proceedings of the 22nd International Conference on*, pages 29–29. IEEE, 2006.
- [112] MarketLine. Online retail in the united states, 2013. On-line at <http://www.marketresearch.com/MarketLine-v3883/Online-Retail-United-States-7760207/>.
- [113] Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. Maximum entropy markov models for information extraction and segmentation. In *ICML*, volume 17, pages 591–598, 2000.
- [114] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8. ACM, 2011.
- [115] Xiaofeng Meng, Dongdong Hu, and Chen Li. Schema-guided wrapper maintenance for web-data extraction. In *Proceedings of the 5th ACM international workshop on Web information and data management*, pages 1–8. ACM, 2003.
- [116] Eirinaios Michelakis, Rajasekar Krishnamurthy, Peter J Haas, and Shivakumar Vaithyanathan. Uncertainty management in rule-based information extraction systems. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 101–114. ACM, 2009.
- [117] Matthew Michelson and Craig A Knoblock. Creating relational data from unstructured and ungrammatical data sources. *Journal of Artificial Intelligence Research*, pages 543–590, 2008.
- [118] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, pages 777–782, 2013.
- [119] Stephen Muggleton. Inverse entailment and prolog. *New generation computing*, 13(3-4):245–286, 1995.
- [120] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.
- [121] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, 2007.

- [122] Aditya Parameswaran, Nilesch Dalvi, Hector Garcia-Molina, and Rajeev Rastogi. Optimal schemes for robust web extraction. In *Proceedings of the VLDB Conference*, volume 4. VLDB Endowment, 2011.
- [123] Aditya Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: it’s okay to ask questions. *Proceedings of the VLDB Endowment*, 4(5):267–278, 2011.
- [124] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
- [125] Vijayshankar Raman and Joseph M Hellerstein. Potter’s wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.
- [126] Juan Raposo, Alberto Pan, Manuel Álvarez, and Justo Hidalgo. Automatically maintaining wrappers for semi-structured web sources. *Data & Knowledge Engineering*, 61(2):331–358, 2007.
- [127] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.
- [128] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.
- [129] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.
- [130] Giuseppe Rizzo and Raphaël Troncy. Nerd: a framework for unifying named entity recognition and disambiguation extraction tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–76. Association for Computational Linguistics, 2012.
- [131] Riccardo Rosati. On the complexity of dealing with inconsistency in description logic ontologies. *Trans (R)*, 1:C2, 2011.
- [132] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008.
- [133] Stefan Schoenmackers, Oren Etzioni, Daniel S Weld, and Jesse Davis. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1088–1098. Association for Computational Linguistics, 2010.
- [134] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceedings of the 10th ACM workshop on Web information and data management*, pages 9–16. ACM, 2008.
- [135] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment*, 8(11):1310–1321, 2015.
- [136] Luo Si, Tapas Kanungo, and Xiangji Huang. Boosting performance of bio-entity recognition by combining results from multiple systems. In *Proceedings of the 5th international workshop on Bioinformatics*, pages 76–83. ACM, 2005.

- [137] Shaoxu Song, Hong Cheng, Jeffrey Xu Yu, and Lei Chen. Repairing vertex labels under neighborhood constraints. *Proceedings of the VLDB Endowment*, 7(11):987–998, 2014.
- [138] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble learning for named entity recognition. In *The Semantic Web–ISWC 2014*, pages 519–534. Springer, 2014.
- [139] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706. ACM, 2007.
- [140] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.
- [141] Fabian M Suchanek, Mauro Sozio, and Gerhard Weikum. Sofie: a self-organizing framework for information extraction. In *Proceedings of the 18th international conference on World wide web*, pages 631–640. ACM, 2009.
- [142] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–41. ACM, 2002.
- [143] Marieke Van Erp, Giuseppe Rizzo, and Raphaël Troncy. Learning with the web: Spotting named entities on the intersection of nerd and machine learning. In *# MSM*, pages 27–30. Citeseer, 2013.
- [144] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, 4(9):528–538, 2011.
- [145] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.
- [146] Daisy Zhe Wang, Michael J Franklin, Minos Garofalakis, Joseph M Hellerstein, and Michael L Wick. Hybrid in-database inference for declarative information extraction. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 517–528. ACM, 2011.
- [147] Haochang Wang and Tiejun Zhao. Identifying named entities in biomedical text based on stacked generalization. In *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pages 160–164. IEEE, 2008.
- [148] Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 457–468. ACM, 2014.
- [149] Michael Wick, Aron Culotta, and Andrew McCallum. Learning field compatibilities to extract database records from unstructured text. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 603–611. Association for Computational Linguistics, 2006.
- [150] Michael L Wick, Khashayar Rohanimanesh, Karl Schultz, and Andrew McCallum. A unified approach for schema matching, coreference and canonicalization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 722–730. ACM, 2008.

- [151] Tak-Lam Wong, Wai Lam, and Tik-Shun Wong. An unsupervised framework for extracting and normalizing product attributes from multiple web sites. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 35–42. ACM, 2008.
- [152] Dekai Wu, Grace Ngai, and Marine Carpuat. A stacked, voted, stacked model for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 200–203. Association for Computational Linguistics, 2003.
- [153] Catharine Wyss, Chris Giannella, and Edward Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 101–110. Springer, 2001.
- [154] Mohamed Yakout, Laure Berti-Équille, and Ahmed K Elmagarmid. Don’t be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 553–564. ACM, 2013.
- [155] Yanhong Zhai and Bing Liu. Extracting web data using instance-based learning. In *Web Information Systems Engineering-WISE 2005*, pages 318–331. Springer, 2005.
- [156] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, pages 76–85. ACM, 2005.
- [157] Chen Jason Zhang, Lei Chen, Yongxin Tong, and Zheng Liu. Cleaning uncertain data with a noisy crowd. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 6–17. IEEE, 2015.
- [158] Chang Zhao, Jalal Mahmud, and IV Ramakrishnan. Exploiting structured reference data for unsupervised text segmentation with conditional random fields. In *SDM*, pages 420–431. SIAM, 2008.
- [159] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th international conference on World Wide Web*, pages 66–75. ACM, 2005.
- [160] Feida Zhu, Qiang Qu, David Lo, Xifeng Yan, Jiawei Han, and Philip S Yu. Mining top-k large structural patterns in a massive network. *Proceedings of the VLDB Endowment*, 4(11):807–818, 2011.
- [161] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, 2004.
- [162] Lei Zou, Lei Chen, and M Tamer Özsu. Distance-join: Pattern match query in a large graph database. *Proceedings of the VLDB Endowment*, 2(1):886–897, 2009.