Reducing Human Effort in Web Data Extraction

Jinsong Guo

St Hugh's College

Supervisor: *Professor Georg Gottlob*

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2017

Acknowledgements

It is my honor to get a D.Phil degree at Oxford University. I still remember the old days when I was at Jilin University and set my PC desktop background to a screenshot of Oxford's website to encourage myself. I regard this degree not only as a great achievement of the past 29 years, but more importantly, a force that will spur me in the future.

I wish to express my most sincere gratitude and appreciation to my supervisor Prof. Georg Gottlob, for his guidance, patience, and encouragement throughout my D.Phil study. His thoughtful kindness and charisma have inspired me profoundly. In addition, I was so lucky to witness Georg and the postdocs in his lab founding a new company, *Wrapidity*. I learned a lot from how the company seeks out customers, maintains public relations, and became acquired. As a student of Georg's, I aim to develop a similar career path.

I would also like to express my deepest thanks to Prof. Valter Crescenzi. Although he is not my official supervisor, he taught me a lot. My Skype app still has hundreds of pages of discussions we had over two years. I learned a vast amount from Valter about both data extraction techniques and research methodology.

My sincere thanks also go to my advisor Dr Tim Furche, an internationally well-known scientist and expert on web data extraction. It was delightful to have discussions with him, where he could always understand my ideas and point to the key immediately. His experience in both of engineering and research significantly impressed me.

It was so great to join a lab full of amazing people. It gives me immense pleasure to thank my other advisors, Dr Christian Schallhart, Dr Giovanni Grasso, and Dr Emanuel Sallinger. Your valuable suggestions always helped me when I had trouble with my research. I would also like to express my thanks to other members of the lab: Dr Giorgio Orsi, Dr Xiaonan Guo, Dr Cheng Wang, Dr Andrey Kravchenko, Dr Omer Gunes, Dr Stefano Ortona, Dr Ruslan Fayzrakhmanov, Julia Wiedmann, and Lianlong Wu.

My internship at QCRI has been a great journey. My sincere thanks also go to Prof. Ahmed K Elmagarmid, Dr Mourad Ouzzani, Dr Nan Tang, and Dr Saravanan Thirumuruganathan. Their profound knowledge of data cleaning helped me a great deal.

I am also very thankful to Prof. Thomas Lukasiewicz and Prof. Dan Olteanu, who have given me valuable suggestions at different stages of my research.

I am sincerely grateful to my parents, without whose love, affection, and encouragement this work would not have been possible. Without their guidance at some crucial stages of my life, I could never have gotten an offer from Oxford. I also want to thank all of my other family members. I hope that my grandma and uncle in heaven could feel happy about my D.Phil degree.

Last but not least, I wish to thank my girlfriend, Tiffany Wong, and all my lovely friends for making my stay at Oxford a pleasant and memorable one.

Publications

(1) Tim Furche, Jinsong Guo (corresponding author), Sebastian Maneth, Christian Schallhart: Robust and Noise Resistant Wrapper Induction. In *Proc. of SIGMOD*: pages 773-784, 2016

(2) (To be resubmitted) Jinsong Guo, Valter Crescenzi, Tim Furche, Giovanni Grasso, and Georg Gottlob: RED: Redundancy-Driven Data Extraction from Search Result Pages.

(3) (To be resubmitted) Jinsong Guo, Saravanan Thirumuruganathan, Ahmed Elmagarmid, Mourad Ouzzani, Nan Tang: Data Error Meter - Estimating the Number of Data Errors.

(4) (Under review) Zhe Li, Zhezhou Yu, Hongbo Li, Jinsong Guo (corresponding author), Zhanshan Li: Revisiting the Efficacy of Weak Consistencies: a Study of Forward Checking

(5) Hongbo Li,Yanchun Liang, Ning Zhang, Jinsong Guo, Dong Xu, Zhanshan Li: Improving degree-based variable ordering heuristics for solving constraint satisfaction problems. *Journal of Heuristics* 22(2): 125-145 (2016)

(6) Jinsong Guo, Hongbo Li, Zhanshan Li, Yonggang Zhang, Xianghua Jia: Efficient Singleton Consistency by Combining Forward Checking and Bound Consistency. *International Journal on Artificial Intelligence Tools* 23(4): 223-229 (2014)

(7) Hongbo Li, Haijiao Shen, Zhanshan Li, Jinsong Guo: Reducing consistency checks in generating corrective explanations for interactive constraint satisfaction. *Knowledge-Based Systems* 43: 103-111 (2013)

(8) Hongbo Li, Yanchun Liang, Jinsong Guo, Zhanshan Li: Making Simple Tabular Reduction Works on Negative Table Constraints. In *Proc. of AAAI*: 1629-1630, 2013

Abstract

The human effort in large-scale *web data extraction* significantly affects both the extraction flexibility and the economic cost. Our work aims to reduce the human effort required by web data extraction tasks in three specific scenarios.

(I) Data demand is unclear, and the user has to guide the wrapper induction by annotations. To maximally save the human effort in the annotation process, wrappers should be robust, i.e., immune to the webpage's change, to avoid the wrapper re-generation which requires a re-annotation process. Existing approaches primarily aim at generating accurate wrappers but barely generate robust wrappers. We prove that the XPATH wrapper induction problem is **NP**-hard, and propose an approximate solution estimating a set of top-k robust wrappers in polynomial time. Our method also meets one additional requirement that the induction process should be noise resistant, i.e., tolerate slightly erroneous examples.

(II) Data demand is clear, and the user's guide should be avoided, i.e.., the wrapper generation should be fully-unsupervised. Existing unsupervised methods purely relying on the repeated patterns of HTML structures/visual information are far from being practical. Partially supervised methods, such as the state-of-the-art system DIADEM, can work well for tasks involving only a small number of domains. However, the human effort in the annotator preparation process becomes a heavier burden when the domain number increases. We propose a new approach, called RED (abbreviation for "redundancy"), an automatic approach exploiting content redundancy between the result page and its corresponding detail pages. RED requires no annotation (thus requires no human effort) and its wrapper accuracy is significantly higher than that of previous unsupervised methods.

(III) Data quality is unknown, and the user's related decisions are blind. Without knowing the error types and the error number of each type in the extracted data, the extraction effort could be wasted on useless websites, and even worse, the human effort could be wasted on unnecessary or wrongly-targeted data cleaning process. Despite the importance of *error estimation*, no methods have addressed it sufficiently. We focus on two types of common errors in web data, namely duplicates and violations of integrity constraints. We propose a series of error estimation approaches by adapting, extending, and synthesizing some recent innovations in diverse areas such as active learning, classifier calibration, F-measure estimation, and interactive training.

Contents

1 Introduction **1**

1.1 The Web: an Abundant but Unfriendly Data Repository 1

1.2 Predicament due to Human Effort Required 4

1.2.1 Human Effort Required by Data Extraction 4

1.2.2 Human Effort Caused by a Lack of Awareness of Data Quality . . . 6

1.3 Our Solutions 7

1.3.1 Scenario I : Data Demand is Unclear 8

1.3.2 Scenario II : Data Demand is Clear 10

1.3.3 Scenario III : Data Quality is Unknown 11

1.4 Thesis Contributions 13

1.5 Thesis Organization 15

2 Preliminaries **16**

2.1 Wrapper Related Preliminaries 16

2.2 Error Estimation Related Preliminaries 17

3 Related Work **20**

3.1 (Partially) Supervised Wrapper Induction 20

3.2 Unsupervised Web Data Extraction 22

3.3 Error Estimation 24

4 Robust and Noise Resistant Wrapper Generator **26**

4.1 XPATH Fragment dsXPath 26

4.2 Problem Definition 28

4.3 dsXPath Induction Algorithm 30

4.4 Evaluation 36

4.4.1 Robustness 36

4.4.2 Expression Characteristics 42

4.4.3 Accuracy and Noise Resistance 43

5 Unsupervised Data Record Extraction from Result Pages **47**
5.1 Problem Description . . . 47
5.2 Overview of the Solution . . . 53
5.3 Extraction Rules Generation . . . 55
5.3.1 Template Analysis . . . 55
5.3.2 Rules Generation Algorithm . . . 57
5.3.3 Values Extraction . . . 59
5.4 Redundancy Seeking . . . 60
5.4.1 the Rules Alignment Problem . . . 61
5.4.2 Finding the Detail Links . . . 61
5.4.3 Soft Segmentation . . . 62
5.4.4 Redundancy Score . . . 64
5.5 Noise Removal . . . 66
5.5.1 Result Rules Template Validation . . . 67
5.5.2 Removing Redundant Noise . . . 69
5.6 Evaluation . . . 71
5.6.1 Evaluation Method . . . 72
5.6.2 Comparison with DIADEM . . . 73
5.6.2.1 Attribute Coverage . . . 74
5.6.2.2 Failures & RED's Limitations . . . 74
5.6.3 Multi-domain Evaluation . . . 74
5.6.3.1 Comparison with Other Approaches . . . 76
5.6.3.2 Robusteness to Parameter Setting . . . 77
5.6.3.3 Efficiency . . . 78

6 Data Error Meter **80**
6.1 Duplicate Estimation . . . 80
6.1.1 Proportion Estimation . . . 81
6.1.2 Distinct Value Estimation . . . 84
6.1.3 Duplicate Estimation via Classifier Evaluation . . . 85
6.2 Functional Dependency Violation Estimation . . . 88
6.3 Evaluation . . . 90
6.3.1 Experimental Setup . . . 90
6.3.2 Duplicate Estimation . . . 92
6.3.3 FD Violation Estimation . . . 94

7 Conclusion and Future Work **97**
7.1 Conclusion . . . 97
7.2 Future Directions . . . 98

Bibliography **99**

List of Figures

1.1 Elegant HTML code snippet from www.zoopla.co.uk. . . . 3
1.2 Examples of HTML code implicitly structuring data. . . . 3
1.3 Tree structure of an IMDB movie page . . . 9

4.1 Syntax of our XPATH Fragment . . . 27
4.2 Robustness of expressions for matching a single node . . . 39
4.3 Robustness of expressions for matching multiple nodes . . . 40
4.4 Nodetests/predicates of single-target queries . . . 44
4.5 Nodetests/predicates of multiple-target queries . . . 44
4.6 Result degradation with increased presence of noise . . . 45

5.1 Running example pages. . . . 48
5.2 A model describing the generation process over the running example. . . . 49
5.3 Running example result page. . . . 51
5.4 Running example detail pages. . . . 52
5.5 Extracted values. . . . 59
5.6 Aligned result rules. . . . 63
5.7 RED performance by varying the two key parameters. . . . 77

6.1 Budget vs Relative Error (Duplicate Estimation) . . . 92
6.2 Varying Duplicate to Non-Duplicate Ratio . . . 92
6.3 Varying Size of Candidate Set for Duplicate Estimation . . . 92
6.4 Budget vs Relative Error (FD Estimation) . . . 92
6.5 Error Injection Rate vs Budget . . . 94
6.6 #Erroneous FDs vs Budget . . . 94
6.7 #Erroneous FDs vs Cost per FD . . . 94
6.8 Error Injection Rate vs Cost per FD . . . 94

List of Tables

1.1 A comparison of the practical solutions with and without our methods. . . . 14

4.1 Matching single nodes, two typical queries and one difficult case 36
4.2 Matching multiple nodes, queries with sibling axes (top-ranked / human) . . 41

5.1 Correct result records. 50
5.2 Pair of rules with score less than 40% 67
5.3 Pair of rules after RED's noise removal step. 71
5.4 DIADEM's original target attributes (left); additional available attributes (right). 73
5.5 RED vs DIADEM Performance. 73
5.6 Results of RED on RED_DS. 75
5.7 #rules/pairs produced by every processing step. 78

6.1 Dataset Statistics - * refer to easy datasets while ‡ to challenging ones. . . . 91

Chapter 1

Introduction

1.1 The Web: an Abundant but Unfriendly Data Repository

The web, including the deep web, is a popular medium for publishing and acquiring information, and has become the largest data repository. For London alone, there were around 2,900[1] real estate agency websites in 2016, and the total number keeps increasing at a rate of one every 1.6 days. For the US alone, the number of online shopping sites with greater than $10,000 revenue is larger than 100,000[2]. Different from many other data acquisition means—such as manually collecting market data, which is labour-intensive, extracting user-related information from social apps that need to concern user privacy, or extracting information from documents that are usually unstructured—acquiring data in the web relies on the data's natural features. Web data are (i) available online and thus can be extracted by scalable programs, (ii) open to the public, and (iii) (semi-)structured.

There is a broad spectrum of applications based on web data. Firms adopting or providing business intelligence solutions may seek for relevant information from the web to support their analyses and decisions. For example, *McKinsey & Company*, which provides business intelligence solutions to its customers, extracts price-related data by using web data extraction tools developed by a subsidiary, *Lixto*, that it formerly acquired.[3] Governments may collect market data from the web to make an economic analysis such as a price inflation evaluation. Individuals may need to obtain some web content for a variety of tasks, such as building a catalogue of some entities of personal interest or creating a dataset as an academic research requirement.

Since web data extraction usually aims for data from a large number (ranging from hundreds to thousands) of websites, a desirable extraction process should only need a minimal amount of human effort. More specifically, it should require no user supervision when the user's data demand is already known, and should only ask for a once-for-all user's guide

[1] We estimated this number according to the research carried out by HouseSimple.com.

[2] This number was reported by http://www.marketresearch.com/MarketLine-v3883/ Online-Retail-United-States-7760207/ in 2013

[3] https://www.tnooz.com/article/mckinsey-ups-periscope-with-lixto-acquisition/

when the user's data demand is unclear. For the former case, i.e., the data's demand is clear, existing unsupervised methods [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], although attracting increasing attention, cannot achieve a high accuracy as supervised methods [12, 13, 14, 15, 16] do, because they purely rely on the analysis of HTML structures or visual information. Although some recent work such as DIADEM [17] has made significant progress on ontology-based approaches, relying on human experts in the annotator scripting process can become a burden when dealing with multiple domains. For the latter case, i.e., the data's demand is unclear, wrappers generated by existing approaches can barely survive webpage structure changes. Frequent webpage evolution may require the human effort for annotations to multiply (Annotator-based methods such as DIADEM may be an exception, as we will discuss in Section 1.2.1.) In addition, the quality of extracted data is usually unknown. Even if the extraction is highly accurate, web data themselves is sometimes dirty due to the errors in the sources [18, 19, 20, 21, 22]. We discuss these issues further in the following sections, but here we only illustrate the very basic challenges of extracting data from webpages to give readers an understanding of why the web is unfriendly to data extraction. Although we limit our discussion to the webpage-level extraction, we also remind users to be aware of other problems in the full site crawling process such as how to reach the deep web behind the forms. The studies on *automatic form filling* [23, 24] concern how to accurately locate the forms in a webpage and how to automatically fill them. Considering the limitations of accessing the data behind the web forms, a significant body of research [25, 26, 27] has been carried out on *querying the deep web* considering both the query plan generation aiming at getting the maximal answer and the efficiency of the execution of the query plan.

Web data are implicitly structured. Although web pages are convenient for a human to browse and acquire information from, the underlying HTML code based on which the extraction approaches work is complicated. Nested HTML nodes are interacting with the data but there is no universally applicable element in the HTML language for explicitly indicating the data.

Noticing the self-describing feature of HTML code, an intuitive solution could be utilizing HTML attributes of nodes to find data.[4] This approach works well when the HTML code has been written in an elegant way such that the HTML elements containing the data have semantically clear attributes. Figure 1.1 shows such an example; supposing the target data are the property address, a simple program locating the node with an attribute value of 'address' is sufficient to find the address detail.

However, HTML code of different websites is written in various styles according to different programmers' habits and application scenarios. The HTML language does not force the programmers to specify semantically clear and standardized values for element attributes, and elements without any attributes are grammatically legal. In fact, it is more common that

[4]We have frequently been asked why this solution is not working by engineers, so we'd like to discuss it in this thesis to make the challenge more clear to readers.

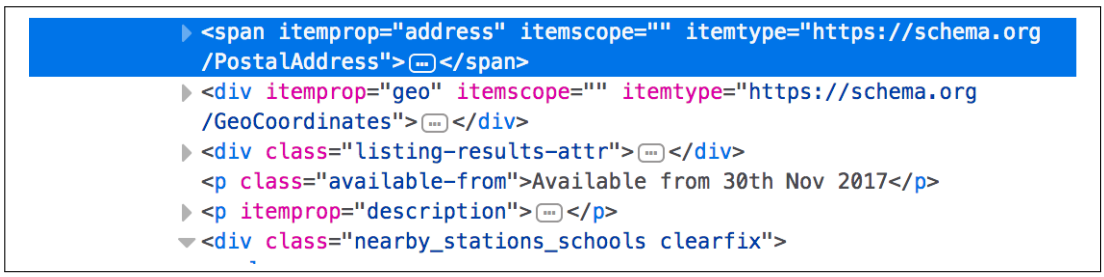
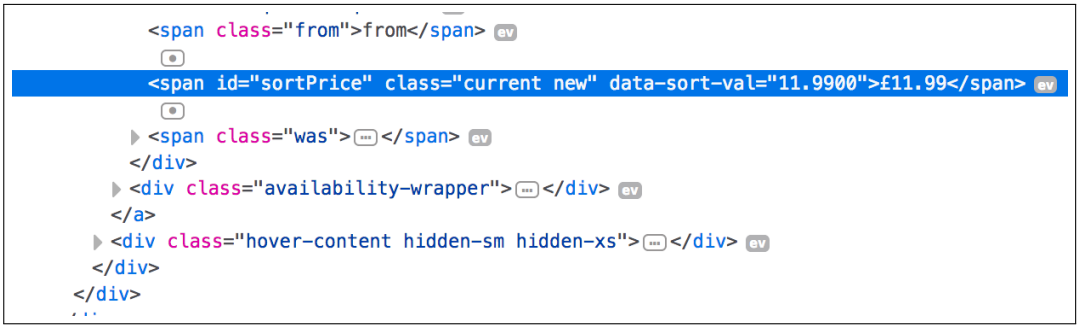
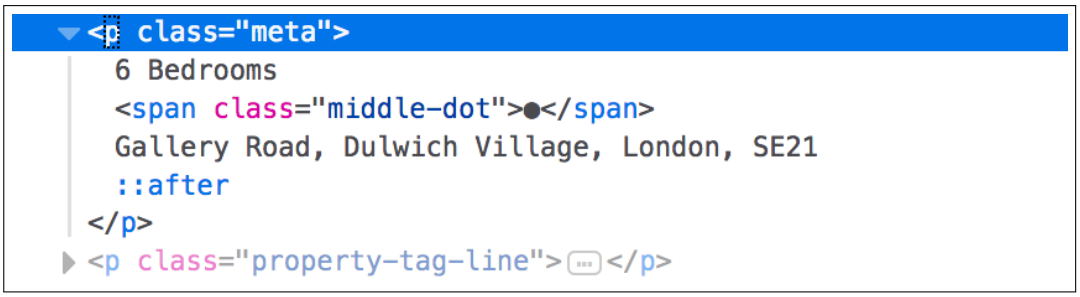


Figure 1.1: Elegant HTML code snippet from www.zoopla.co.uk.

(i) the HTML attributes are written using some customized words or phrases, such as a prefix or suffix combined with abbreviations. For example, in Figure 1.2a, the price value is wrapped in an element with `id` being 'sortPrice'. Although applying string similarity functions could improve the solution, it does not deserve further investigation because in many cases, (ii) there are no such semantically descriptive attributes at all, such as in Figure 1.2b.



(a) A snippet of HTML code of www.eflorist.co.uk.



(b) A snippet of HTML code of www.haart.co.uk.

Figure 1.2: Examples of HTML code implicitly structuring data.

Web wrapper quality is hard to guarantee. The target data identification process on one webpage should be executed once for all that its results (i.e., the location of nodes containing the data in the HTML document) should be reused by future extraction from the same page or other pages of the same template. Such results are scripted into web wrappers, i.e., programs extracting data from webpages. There exist various languages to write web wrappers, such as regular expressions [28], XPATH queries, and logic language [29]. In this thesis, we primarily focus on XPATH queries, which are widely used due to their simplicity for both understanding and scripting. Hereafter, when we mention web wrappers, we are referring to XPATH queries.

A wrapper is **(1)** *accurate* if it infers the full set of intended information from the given examples, **(2)** *robust* against structural changes of the webpage over time, so that users do not need to repeat the annotation process when a webpage changes, and **(3)** the induction process should be *resistant to noise* in the annotations, which allows users to save the time in finding a complete set of samples either manually or by scripting perfect annotators. Both the wrongly annotated nodes and missing annotations of target nodes are treated as noise. For example, if the target nodes are product names listed on a webpage, the noisy annotations may only consist of a subset of all the product names, or contain some menu items that are wrongly annotated. In both situations the induced wrappers should only select all the product names. Meeting these three requirements at the same time is tough and cannot be achieved by existing wrapper induction methods.

Web data are dirty. In addition, the data extracted from the web cannot be used directly because of the errors that are in the sources themselves [19, 20, 21, 22] or introduced by the extractors. Data errors primarily related to web data quality are duplicates [30, 31, 32, 33, 34, 35, 36], i.e., pairs of tuples pertaining to the same real-world entity, and violations of constraints/rules [37, 38, 39], i.e., the values that violate some pre-specified integrity constraints, such as functional dependencies (FDs) and denial constraints. We focus on duplicates and FD violations in this work. The classical "garbage in and garbage out" principle applies to any data analytical tasks including database queries, data mining pipelines, and machine learning processes. For example, extracting actual business value from the data is hard for organizations to make decisions that depend on data-driven processes when data are dirty. Unfortunately, in practice, it is simply impossible to make sure data are clean before performing data analytics [40], and the situation is even worse with the rapid increase in volume, variety, veracity, and velocity (a.k.a. the four V's of big data) of web data.

1.2 Predicament due to Human Effort Required

1.2.1 Human Effort Required by Data Extraction

All the web data extraction tasks fall into two application scenarios according to users' data demand. In Scenario I, the data demand is unknown. Such tasks usually concern webpages containing a broad category of data, such as a news webpage containing different articles on various topics. Users may be only interested in some particular information such as *Sports* news. In addition, the more generalized target "data" are sometimes not merely information but can be HTML elements with specific functions. For example, the DIADEM system [17] scripts how humans browse a site into OXPATH [41] expressions, which consist of XPATH expressions specifying different nodes such as the HTML `form`, the button, and the node containing the link to the next page, etc. In Scenario II, the data demand is clear by common

sense. Such tasks usually seek for data from the so-called *Deep Web* [42], or more specifically, the contents behind HTML forms. In response to queries, webpages can publish either one particular object or a list of objects. Usually, the attribute values of these objects are the target data. For example, in response to the query "houses in London" on a house rental agency website, the target data should be the attribute values of the listed properties.

Ultimate goal. The optimal solution for tasks in Scenario I should only require a minimal amount of efforts from users. More specifically, it should be capable of generating *accurate* and *robust* wrappers given a noisy set of samples. Users will only need to specify a partial set of samples or recur to annotators. Most importantly, robust wrappers will save the efforts of repeating the annotations once the webpage changes. The best solution for tasks in Scenario II should be accurate and fully unsupervised without any human effort such as annotations or domain-dependent annotator scripting.

Where are we? Now we discuss how well existing approaches can apply to these extraction tasks in different scenarios.

(*i*) **Supervised approaches.** In general, *wrapper induction* systems [12,13,14,15,16] take as input a bunch of example pages where the target data is annotated by human (more efficiently, through crowdsourcing [43]), and generate wrappers capable of extracting the target data from new pages. The underlying learning algorithms selectively generalize the properties of the annotated content representing either the structural or the visual features and infer extraction wrappers in a proper form such as XPATH expressions or automata. Most of these supervised approaches only care about the wrappers' accuracy while overlooking the importance of wrappers' robustness. The only work noting the wrapper robustness [44, 45, 46, 47] are not practical. The approaches in [44, 45] do not address attributes and attribute values at all, yet the choice of attributes and values is among the most crucial choices for robust wrappers. The methods in [46] and its extended work [47] assume that historical changes to webpages are available and that the future changes will follow the same pattern, which is not always true.

(*ii*) **Unsupervised approaches.** As big-data-driven applications have flourished in recent years, the importance of automating the wrapper induction process has been attracting increased attention. For tasks in Scenario II, existing unsupervised approaches [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] are primarily based on repeated structure analysis and visual features. The general principle of these methods is to reverse engineer the same template been used either for generating different result records in the same page or for generating different detail pages. Unfortunately, these methods share the same drawback that they are quite sensitive to any exceptions in the assumed repeated patterns such as some different template deliberately used to highlight some featured record.

(*iii*) **Partially supervised approaches.** Automatic annotations have been recently used for improving wrapper induction in terms of reducing the dependency on human specified an-

notations, and for improving repeated structure analysis in terms of increasing the accuracy. They can be used for tasks in both Scenarios I and II. The most prominent approaches in this category are [17, 48, 49, 50]. By using annotations, these systems achieve a reasonable trade-off between supervised and unsupervised methods, obtaining high accuracy while scaling at thousands of websites over a single domain. The required annotations usually rely on existing entity recognizers of a specific domain or the domain-specific ontology scripted by some ontology expert [17]. When dealing with tasks in Scenario I, these methods have the advantage that the wrapper failure will not cause the required human effort to multiply. They can adopt a similar method as DIADEM [17] does that a detection of wrapper failures directly triggers the regeneration of wrappers by reanalyzing the current webpages. Since the annotation process is done by automatic annotators, no extra human effort is required. Despite this advantage, the cost of webpage reanalysis and wrapper regeneration are still additionally caused by the lack of wrapper robustness. When dealing with tasks in Scenario II, the system DIADEM [17] generates accurate wrappers and can scale out to thousands of websites. However, when applied to a large number of domains, relying on human effort in the annotator scripting process for each domain becomes a burden. For example, it takes several days for an domain expert to adjust DIADEM for a new domain and the bootstrapping process for the very first domains requires even longer time (around one month, as claimed by the authors). □

In summary, for tasks in Scenario I, users can either specify annotations manually to adopt supervised approaches or script annotators to use partially supervised methods (for cases merely involving several domains). However, wrappers lacking robustness may cause an extra cost once they fail, especially the extra human effort when the manual annotation is adopted. Tasks in Scenario-II would require users to make choices depending on the number of domains involved. For the vertical extraction, i.e., the target data is from a single, or a small number of domains, the state-of-the-art system DIADEM can work quite well. For tasks over various domains, e.g. the task of collecting data of various products from the market to evaluate price inflation, current solutions have their drawbacks. Unsupervised methods cannot guarantee accurate results. (Partially) supervised methods require extra human effort to get the annotations.

1.2.2 Human Effort Caused by a Lack of Awareness of Data Quality

Human effort required by a complete data extraction pipeline producing clean data is also affected by the effectiveness of *error estimation*, i.e., estimating how dirty the extracted data are. Error estimation can help data extractors to detect which websites should be abandoned to avoid wasting effort on them and, more importantly, to decide whether the data cleaning process is necessary or not and to reasonably allocate human effort to it if so. Despite its importance, this problem has not received enough attention from either *web data extraction* or *data cleaning* community. Although there has been extensive study on *error detection* in the data cleaning literature [19, 20, 21, 22], the techniques that usually heavily relying on

human experts cannot directly work for the problem of error estimation. Actually, *error detection* and *error estimation* are two different problems. The latter only cares about the number of errors, but the former also aims to find out what the errors are.

During the extraction process, executors need to decide which websites should be excluded from future extraction according to already extracted data. Major concerns include avoiding a repetition of efforts and excluding the websites publishing false data (usually fake data). Estimating the number of duplicates among different websites can help executors to find out those whose data have been aggregated by others. For example, when extracting data from real estate websites, we should skip accessing those agencies who have submitted their data to some aggregator sites such as rightmove.co.uk. Without a reliable method for estimating duplicate numbers, unnecessary extraction work could be wasted. In addition, extraction executors should, in a timely manner, exclude the websites whose data does not obey any underlying logical rules, e.g., violating FDs. Otherwise, extraction-related efforts could also be wasted and, even worse, such data will confuse the future data analysis procedure.

Most importantly, data owners should decide whether extracted data should be cleaned before being used in the analytic processes. This decision is closely related to the type and number of errors, since different analytic algorithms can be sensitive to different error types, and their tolerances of noise level can differ. In addition, data owners need to devise a wise plan for allocating the data cleaning tasks. Current data cleaning techniques rely on human interactions (e.g. duplication detection methods usually require humans to specify both of true and false duplicates [30, 32]), FD rule discovery systems are also heavily based on user interactions [51]. Making optimal decisions about what types of errors should be cleaned can lead to a substantial difference in the human effort cost. In the worst case, where a wrong decision is made, all the human effort spent on the data cleaning process will only tell the user the data are clean. Knowing the number of different types of errors can guide decision optimization so that the human effort put into the cleaning can be reasonably saved and optimally allocated.

Surprisingly, despite the importance of error estimation, no work has addressed this problem sufficiently.

1.3 Our Solutions

In this section, we briefly introduce our solutions to web data extraction(-related) problems in three scenarios: (I) data demand is unclear, (II) data demand is clear by convention, and (III) data quality is unknown. For different web data extraction tasks, extraction programs have to deal with problems in either Scenario I or Scenario II according to the awareness of the user's data demand. The problem in Scenario III is a common problem that data owners will face with right after the data is extracted.

1.3.1 Scenario I : Data Demand is Unclear

In the scenario where users have to guide programs to the data the users actually want, the web data extraction problem is equal to the *wrapper induction* problem, i.e., the problem of inferring a wrapper from a given sample of annotated data that follows a certain template. Different from existing wrapper induction approaches, our method meets the three requirements of wrapper induction we discussed in Section 1.1 all at the same time. In other words, our wrapper induction method generates *accurate* and *robust* wrappers while being *resistant to noise* in annotations.

Our wrapper induction method considers samples that may be both noisy and minimal, allowing wrappers to be induced from annotations for a single page and from as few as a single sample. It achieves this primarily through the careful choice of an *expressive subset* of XPATH, that is sufficient to express most wrappers, but limited enough to force noisy annotations to be generalized into wrappers that select the likely intended items. The proposed method achieves remarkable robustness. It does so in a fashion that substantially differs from previous approaches—by optimizing *short, selective wrappers* that use HTML's *semantic mark-up* for selection where possible. Our approach favours **(1)** *short* expressions with a small number of steps over multi-step expressions. It also prefers **(2)** *selective* expressions that drill down quickly into a small portion of the HTML page over expressions that are unselective (though possibly shorter). Selective expressions are less likely to be affected by changes in unrelated parts of the document. The approach is also heavily biased towards exploiting hints about the **(3)** *semantic role* of elements in the template. In HTML, these are typically expressed in `id` or `class` attributes and often used for styling and scripting. HTML5's microdata adds further semantic attributes such as `itemprop`. Many templates also provide static labels, which are either visible or in form of `title` tooltips. These criteria are designed to mimic human-created robust XPATH expressions.

As an example, consider the following wrapper, extracting (`span`-elements of) directors from IMDB movie pages:

```
descendant::div[starts-with(.,"Director:")][1]/descendant::span
```

This XPATH query selects the first `div`-element with text value of the form 'Director: ...'. Starting from this `div`-element, the query selects all descendant `span`-elements. There is only one such span element, and this element contains precisely the director names of the movie. Thus, the above is an accurate wrapper for extracting director's names for the movies. However, this query is *not* robust against changes to the webpage: **(1)** Imagine more `span` elements (containing non-director information) are inserted under the correct div-element. The wrapper would wrongly select all of them. **(2)** Imagine that more `div`-elements (without director information) are inserted *before* the `div`-element containing the director information. The wrapper would select the wrong `div`-element and return its contained `span` elements, if any. Our approach does *not* attempt to build an accurate model of changes made to a specific

website or class of websites. Instead, we aim to model heuristics for building robust wrappers on any type of website.

What then, is a robust wrapper for this example?

```
descendant::div[starts-with(.,"Director:")]/descendant::span[@itemprop="name"]
```

This wrapper q_{director} is the most robust one in our setting. It follows the heuristic outlined above for short, selective wrappers that prefer semantic features in their predicates. It increases the selectiveness of the `span` step by using a predicate on a semantic attribute (`itemprop`) and shortens the `div` step by removing the positional predicate that is no longer needed (because of the `itemprop`-attribute). We often refer to the intermediate nodes selected by an XPATH expression as "anchors". Figure 1.3 shows a fragment of a tree structure of an IMDB movie page with "Martin Scorsese" as director; the red shaded nodes are anchors of q_{director}.

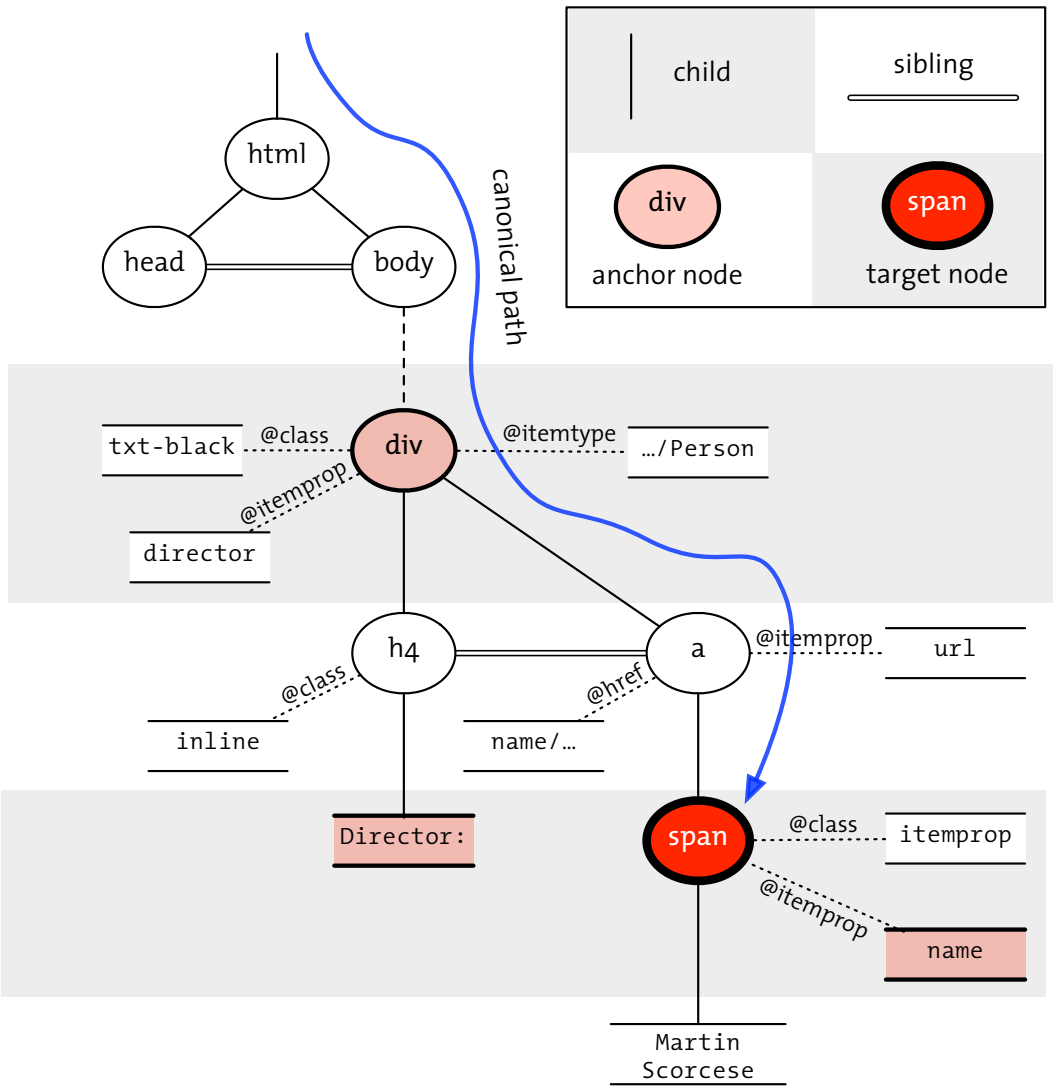


Figure 1.3: Tree structure of an IMDB movie page

To deal with noisy samples, we restrict to a particularly small XPATH fragment. This fragment is *not* selection-complete, i.e., there are annotated documents for which no query

in the fragment selects the annotated positions accurately. For instance, our fragment cannot select all but some particular elements of a list. Most notably, the fragment misses important features such as negation and union. Interestingly, while trying to keep the XPATH fragment as restricted and small as possible, we find that certain axes are crucially required in order to achieve full accuracy for all our samples. Besides the four base axis of XPATH (`child`, `parent`, `descendant`, and `ancestor`), we find that the two "sideways" axes, `following-sibling` and `preceding-sibling`, are essential for our wrappers. For instance, often we want to select all elements of a list appearing *after* a specific determining element.

Our evaluation validates three claims: **(1)** The approach generates robust wrappers, that remain effective as long as the relevant data are present, often for hundreds of days. **(2)** The generated wrappers have comparable or higher robustness than human-created ones. **(3)** The approach is able to generate such wrappers even in the presence of a significant amount of noise in the annotations. We selected over 100 popular webpages from more than 50 different sites. We tracked the evolution of the pages over *six years* in the Internet Archive in 20-day intervals. For each of these over $> 10,000$ "page versions", we evaluated the human crafted wrapper, the wrapper produced by our system, and a simple baseline wrapper.

1.3.2 Scenario II : Data Demand is Clear

In the scenario where the user's data demand is clear, a fully unsupervised web data extraction approach should automatically identify the target data and generate highly accurate wrappers. In this thesis, we consider data extraction from result pages, a common case in this scenario.

Many websites in several application domains (e.g., those managed by real estate agencies, and local e-commerce shops) follow a simple publishing pattern to turn a fraction of their visitors into potential customers: they include simple web forms to let users gain access to their data by means of queries about a specific type of objects (e.g., house properties or products). A list of objects is returned in response to a query, in the form of one or several paginated *result pages*, each containing a list of *result records*. Every result record contains a summary of the key attributes about a single object so that the user can get a first impression of the resulting objects before deciding to dig further into detailed information. Usually, each result record also includes a link to another type of page that we call *detail pages*, which publishes a much more comprehensive set of attribute values about the very same object.

Many unsupervised approaches presented in the literature for data extraction and wrapper generation can be applied either over a collection of result pages, or over a collection of detail pages. However, to the best of our knowledge, no proposal has fully exploited the redundancy underlying this widespread publishing pattern to improve the level of automation of the data gathering process: the same objects are published both on the result and in the detail pages, and the latter also include a superset of the attributes published in the former type of pages.

Our approach, called RED (abbreviation of "redundancy"), exploits the publishing pattern of result and detail pages. It can be described as a pipeline of four main processing steps: (i) candidate rule generation; (ii) redundancy finding; (iii) rule validation; (iv) noise removal.

During the first step, RED generates the candidate extraction rules for both result pages and for detail pages. The extraction rule generation algorithm is *complete* both for result pages and for detail pages. In other words, it should generate at least one correct detail rule and one correct result rule for every attribute in the result pages.

In the *redundancy finding* step, RED selects only the pairs of *redundant* result and detail rules from the collection of pairs of rules received from the previous step. This step is not trivial, for several reasons. It requires aligning the possibly different number values extracted by the two rules (one detail rule and one result rule) and matching their values. There is also the underlying inherently difficult and errors-prone problem of segmenting the result page into separate records. RED adopts an innovative method that leverages intra-site redundancy between result and detail pages. Namely, it aligns the values extracted by the result query with those extracted by a detail query by analyzing the positions of the nodes extracted by the former wrt the detail link nodes, i.e., the nodes of the result record containing links to the corresponding detail pages. It turns out that given a result rule r, there exist at most two possible ways (denoted r^a and r^b) to interleave the nodes it extracts with the detail links, and that the wrong choice produces values that are unlikely to match by chance with values extracted by any (correct or incorrect) detail rule. Therefore, in this step we compare every result rule against every detail rule considering at most two possible inter-leavings of the values, and we select only pairs of rules extracting values that better match with those extracted by the detail rule.

The *validation* step discards result rules that appear to be ill-conceived based on an analysis of how the extracted values interleaves with the values extracted by other result rules.

The last step, *noise removal*, addresses the problem of selecting, among all the pairs of redundant rules produced by the previous step, those actually associated with a rule correctly extracting the redundant attributes values so as to discard any pair of redundant pairs involving noisy rules.

Wrt previous unsupervised methods, RED can achieve a significantly higher accuracy while automatically selecting only relevant attributes, a task which is out the scope of the traditional unsupervised approaches. Wrt previous supervised methods, it can scale to a large number of websites, while achieving a similar accuracy, and wrt DIADEM it can scale to several domains without requiring a human intervention for each domain.

1.3.3 Scenario III : Data Quality is Unknown

To benefit decision-making about (i) which websites should be excluded from extraction tasks, (ii) whether the post-process of cleaning the extracted data is necessary, and (iii) where to put in more efforts, we study the problem of *error estimation*. In particular, we focus on

two types of errors that are common in practice, namely duplicates and violations of integrity constraints.

Despite the importance of knowing the number of data errors for more targeted data cleaning, surprisingly, no work has tackled the practical setting where *only* the data are given. The data quality metric (DQM) [52] estimates the number of undetected errors, after a series of manual or algorithmic cleaning operations. However, our target is to guide data owners to the data that are worth cleaning and to the types of cleaning operations they should focus on. [53] proposed a sampling-based approach for estimating the number of duplicates by assuming the availability of a trained binary classifier. In contrast, our goal is to decide whether it is worth the effort to train a binary classifier. Furthermore, estimating the number of errors is different from detecting errors – the former should be cheaper and help decide what types of errors should be detected.

There are three possible ways to tackle the above problem.

(1) Finding complete rules. If we can find a comprehensive collection of business rules, integrity constraints and patterns, then it is fairly straightforward to both estimate and detect errors. However, for all practical purposes, such a data utopia never exists.

(2) Mining candidate (or approximate) rules. An alternative way is to approximately mine candidate rules on the dirty data themselves. Unfortunately, such methods are very sensitive to data errors. In practical scenarios, the number of candidate rules dramatically outnumbers the number of true rules [54], making it infeasible for domain experts to manually verify all of the candidate rules.

(3) Sampling-based methods. A naive approach is to work on a small sample, *i.e.,* asking the user to manually label correct/erroneous data in the sample, and then extrapolate the number of errors in the sample to all the data. Unfortunately, such naive approach will not work. Take duplicate detection, for example. Consider two datasets, R_A and R_B, with 10^6 tuples per dataset and there exists a 1-1 mapping between them. However, even in this benign scenario, the ratio of the number of duplicates (10^6) to all possible pairs $10^6 \times 10^6 = 10^{12}$ is exactly 10^6 to 1. This is known to be a severely imbalanced problem. In other words, if we sample a million tuple pairs, we would *expect* to see one duplicate pair. Even performing more sophisticated sampling such as stratified sampling would not alleviate the substantial amount. As we shall see in Section 6.1, even the direct use of relevant sophisticated estimators for distinct values, species size or graph components would require a large number of interactions with an expert to generate a decent estimate of the number of duplicates.

Challenges. Naturally, researchers will link error estimation to proportion estimation, a well-studied problem in statistics [55]. A key bottleneck in applying the aforementioned classical statistical techniques is that they are often very generic. For example, one can readily apply an identical uniform or stratified sampling-based approach to estimate the number of duplicates. While this generic nature is important, they often do not leverage a number of useful task-specific properties. Consider duplicates, for example. Most datasets exhibit a

monotonicity of precision property, wherein given two randomly chosen tuple pairs, the pair with a higher similarity score is more likely to be a duplicate. Adapting these properties into estimation procedures is often non-trivial - yet they provide substantial benefit by generating better estimates with fewer expert interactions. Hence, the essential challenge is: *What is the correct formulation for estimating various types of errors by combining both statistics and task-specific properties?*

We find solutions that gracefully connect statistics and task-specific properties. In particular, we propose to transform the error estimation for duplicates as proportion estimation of a classifier accuracy; and automatically generate and test candidate integrity constraints for estimating the number of violations. In addition, for each proposed approach, we only ask the expert simple questions about tuples, not the rules or constraints.

We first connect duplicate estimation to well-studied problems in statistics and describe their limitations. We then propose a two step approach based on classifier evaluation. The appoach first builds a classifier for duplicate detection using very little training data. It then calibrates the classifier and uses it to estimate its accuracy to generate an estimate of the number of duplicates. The additional calibration step is often required, as the classifiers are not 100% accurate even with a large amount of training data. The above quantitative approaches do not work for estimating the number of violations of integrity constraints. We thus propose a qualitative two-phase approach of candidate generation and testing adapted from a prior art [51] to estimate the number of FD violations. We conduct an extensive set of experiments that highlight the efficacy of our methods.

1.4 Thesis Contributions

Our work aims to reduce human effort in web data extraction pipelines. Table 1.1 presents a comparison of the practical solutions with and without our methods.

We also summarize the detailed technical contributions as follows.

Robust and noise resistant wrapper induction We present a novel approach for inducing wrappers under the combined requirements of robustness and noise resistance for tasks in Scenario I.

(1) We define a *directed* XPATH *with sideways checks* (for short, dsXPath). dsXPath is powerful enough to capture a large class of wrappers and is weak enough to support the generalization to correct queries from noisy samples.

(2) We define a natural framework for computing the *robustness* of a query. The essential feature of this framework is its composability: the robustness score of a composed query can be computed from the scores of its constituents.

(3) We show that computing the top-k most robust queries is intractable, hence there is no hope for an exact solution.

(4) We devise a recursive bottom-up scheme for estimating the top-k most robust dsXPath queries for a given set of samples.

Table 1.1: A comparison of the practical solutions with and without our methods.

Without our solutions	With our solutions
Scenario I : Data demand is unclear	
Wrappers are likely to fail when webpage changes, causing the human effort for annotations to multiply.	Wrappers are highly immune to the changes to webpages.
Users have to use different approaches according to different ways of annotations.	The same generator can deal with both perfect and noisy annotations.
Scenario II: Data demand is clear (Data extraction from result pages)	
For extractions tasks over a single or a small number of domains, users can adopt partially supervised approaches such as DIADEM.	When extracting data from websites with result/detail page redundancy, RED is a feasible solution that it avoids the human-effort intensive processes of scripting annotators for various domains, and achieves high accuracy.
The human effort effort required in the annotator preparation process becomes a burden when the domain number increases.	
Scenario III : Data quality is unknown	
Waste effort on websites with overlapping content and websites with false data.	Refine the websites included in the extraction tasks in a timely manner.
Blindly allocate the data cleaning tasks causing a waste of human effort.	Wisely decide whether to clean the data and where to put in more effort.

(5) We evaluate the robustness of wrappers generated by our approach using old versions of popular pages from the Internet Archive.

(6) We evaluate the noise resistance of our approach using both real-world and synthetic noises.

Unsupervised data record extraction from result pages We proposed a method called RED achieving a significantly higher accuracy than existing unsupervised methods while requiring no annotations, for a typical task in Scenario II, namely extracting data from result pages.

(1) To the best of our knowledge, RED is the first system fully exploiting the publishing pattern of result and detail pages to generate wrappers for attributes in result pages.

(2) RED avoids the problem of segmenting the result pages, which itself is tough, in a novel way.

(3) RED exploits the redundancy to pick target data and only keeps relevant attributes, which no existing unsupervised method can achieve.

(4) RED can flexibly scale to various domains without any tuning effort.

(5) We conduct a direct comparison with the state-of-the-art system DIADEM by running on sites from its datasets and conduct a further evaluation covering sites from 10 different domains.

Data error meter We propose the first solutions to estimating error numbers in datasets to guide data owners to make better decisions about data-quality-related issues in Scenario III.

(1) We introduce and define the problem of error estimation for two important types of errors, namely duplicates and integrity constraint violations.

(2) We propose a series of algorithms to estimate the numbers of these types of errors while minimizing the number of questions that need to be answered by an expert.

(3) We conduct an extensive set of experiments that highlight the efficacy of our methods.

1.5 Thesis Organization

The thesis is structured as follows. In Chapter 2, we introduce basic concepts and evaluation metrics. We then describe the features of existing approaches related to our work and discuss how they are different to our solutions in Chapter 3. Our solutions to problems in three different scenarios are respectively introduced in the three subsequent chapters. In Chapter 4, we present our solution to simultaneously tackling the wrapper robustness and noise resistance. First, we first introduce the XPATH fragment dsXPath, which is the core of our solution, in Section 4.1, and formally define the wrapper induction problem tailored to our setting of generating dsXPath queries in Section 4.2. Then we present our unified framework for generating a robust and accurate wrapper with noisy annotations in Section 4.3 and evaluate its performance in Section 4.4. Chapter 5 introduces our unsupervised system, RED, for extracting data from result pages. We first formalize the process of generating result pages and detail pages to give readers an impression of the underlying redundancy and define the problem in Section 5.1. Due to RED's complicated processes, we first summary the whole workflow in Section 5.2 and then introduce each step in separated sections (Section 5.3, Section 5.4, Section 5.5.1 and Section 5.5.2). An extensive evaluation of RED is presented in Section 5.6. In Chapter 6, Section 6.1 describes a series of algorithms for estimating the number of duplicates, Section 6.2 proposes an algorithm for estimating the number of violations of FDs, and Section 6.3 presents experimental findings. Finally, we conclude our solutions and discuss further work in Chapter 7.

Chapter 2

Preliminaries

2.1 Wrapper Related Preliminaries

We assume the reader to be familiar with the basics of XPATH 1.0 and use HTML and XML syntax freely. An (HTML or XML) *document* D is a text containing markup and attribute definitions. The latter two give rise to a tree structure, containing element nodes, attribute nodes, and text nodes. An example of a document's tree structure is shown in Figure 1.3. Element nodes are depicted as ellipses. The attributes of an element give rise to attribute nodes. These attribute nodes are depicted as boxes, which are connected by dotted lines to their element nodes. The dotted line is labeled by the name of the attribute (prepended by the `@`-symbol), while the box contains the attribute's value. For instance, the document fragment `<h4 @class="inline">Director:</h4>` gives rise to the circled `h4`-node in the lower left of the figure, and to the `inline`-labeled attribute box connected to it.

Wrappers. Let D be a document. XPATH queries are evaluated relative to a given node u of D. We denote by $q_D(u)$ the set of target nodes of q when evaluated relative to u in D, and simply write $q(u)$ if D is clear from the context. A *wrapper* is an XPATH expression q. These expression will be evaluated relative to the root of D, thus, $q_D(r)$ is the result set of the wrapper expression on D, where r is the root node of D. We denote $q_D(r)$ also by $q(D)$. *Wrapper induction* is an algorithm I that given a document D and a set of nodes V of D, returns a wrapper $q = I(D,V)$. We say a wrapper q "matches" a target node n from a context u if $n \in q_D(u)$, omitting u if $u = r$.

Canonical Path. Let D be a document and let u be an element or text node of D. The *canonical path of* u, denoted $\mathsf{canon}_D(u)$, is an XPATH query defined recursively as follows. If u is the root node then $\mathsf{canon}_D(u) = \texttt{/}$. Otherwise, let $p = \mathsf{canon}_D(v)$ where v is the (element) parent node of u, $k \geq 1$ such that u is the k-th child of v, and t is a node test for u (i.e., `text()` if u is a text node, or *name* if u is a *name*-labeled element node). Then, $\mathsf{canon}_D(u) = p/t[k]$.

For Figure 1.3, the canonical path of the director node on an IMDB director page (the red `span` element) is:

```
/html[1]/body[1]/ ... /div[4]/a[1]/span[1]
```

The dashed line in Figure 1.3 and the dots in the expression above refer to a part of the path that is not shown (consisting of eleven XPATH steps, seven of which are `div`s).

Let $\mathcal{D} = \langle D_1, \ldots, D_n \rangle$ be a sequence of documents all containing the node v of the same entity (typically a sequence of versions of a single page, e.g., snapshots of an IMDB director page). Then, we say that there are k **c-changes** in $\mathcal{D}$ w.r.t. v, if $k = |\{D_{i+1} : \{v\} \neq \mathsf{canon}_{D_i}(v)(D_{i+1}), i \in \mathbb{Z} \text{ and } i \in [1, n)\}|$. The number of c-changes is a measure for changes in $\mathcal{D}$ that affect the path from v to the document root. Such changes are far more likely to affect any query selecting v than changes in regions of the document far from v.

Robustness. Let q be a wrapper and let D and D' be documents. The query q is *robust* (*for D and D'*), if for each node v which is in $q(D)$ and there is a *c-change* w.r.t. v, there exists a bijection π between $q(D)$ and $q(D')$ so that $D/v = D'/\pi(v)$. Here, D/v denotes the (abstract, nodeId-free) subtree of D rooted at node v. Note that since $q(D)$ is a set (and not a sequence), our robustness definition is order independent.

Noise Resistance. Let D be a document and V, V' be sets of nodes of D such that V' is obtained from V by adding nodes and deleting a strict subset of V. A wrapper induction is *noise resistant* (*for V, V' and D*), if given D and V' it returns the same query q as for input D and V. A wrapper induction I is *k%-robust*, if for k% of all pairs (D, D') and sets of nodes V from D, the wrapper $I(D, V)$ is robust. A wrapper induction I is *k%-noise resistant*, if for k% of all documents D and sets of nodes V, V' of D, the wrapper $I(D, V)$ is noise resistant.

Precision, Recall, and F-Score. Imagine a set B approximating another set A. The elements in $B \cap A$ are called *true positives* and t^+ denotes their number $|B \cap A|$. The elements in $B - A$ are called *false positives* and we define $f^+ = |B - A|$. The elements in $A - B$ are called *false negatives* and we define $f^- = |A - B|$. The *precision* of B with respect to A is calculated as $\mathsf{prec}(A, B) = t^+/(t^+ + f^+)$. The *recall* of B with respect to A is $\mathsf{rec}(A, B) = t^+/(t^+ + f^-)$. The *F-score* $\mathsf{F}_\beta(A, B)$ measures the accuracy biased toward precision ($\beta < 1$) or recall ($\beta > 1$): $\mathsf{F}_\beta(A, B) = \frac{(1+\beta^2)\mathsf{prec}(A,B)\mathsf{rec}(A,B)}{\beta^2 \mathsf{prec}(A,B) + \mathsf{rec}(A,B)}$.

2.2 Error Estimation Related Preliminaries

Let $\mathbf{D}$ denote a relation with n tuples $\{t_1, t_2, \ldots, t_n\}$ and m attributes $\mathbf{R} = \{A_1, A_2, \ldots, A_m\}$. We write $t_i[A_j]$ as the value on attribute A_j of tuple t_i. Let $X \subseteq \mathbf{R}$ be a subset of attributes and $t[X]$ denotes the projection of tuple t on attributes X. A *data error* is an attribute value that is different from its ground truth value.

Data Error Types. Errors could be qualitative or quantitative: qualitative errors often arise in categorical attributes, while quantitative errors refer to the errors in quantitative attributes (*i.e.,* numeric values). The former are typically detected by integrity constraints or data

quality rules, and the latter are often found by statistical techniques. In this work, we focus on two types of qualitative errors that are most related to the web data: duplicates and violations of integrity constraints.

- **Duplicates** are the pairs of tuples pertaining to the same real-world entity.
- **Violations of constraints/rules** refer to the values that violate some pre-specified integrity constraints, *e.g.,* functional dependencies (FDs) and denial constraints. We focus on FDs in this work.

In the following, we will introduce notations and techniques that will be used in the rest of the thesis.

Entity Resolution (ER). Given all distinct tuple pairs (t_i, t_j) with $t_i \neq t_j$ from $\mathbf{D}$, the problem of *entity resolution* is to identify which pairs of tuples refer to the same real-world entity. The tuple pair (t_i, t_j) is said to be a *match* if they refer to the same real-world entity and a *non-match* otherwise.

Entity resolution is one of the major problems in data cleaning and diverse techniques have been investigated, such as declarative rules, machine learning based classifiers, and crowdsourcing. In this thesis, we primarily leverage the formulation of ER as a classification problem. The classifier is trained through a multitude of matched and mismatched tuple pairs - that are also referred to as positive and negative examples. Once trained, the classifier outputs *true* (resp. *false*) to indicate whether a tuple pair (t_i, t_j) match (resp. mismatch).

Similarity Vector. A typical input to the ML classifier for ER is a vector of similarity scores between aligned attributes. A similarity function $f(t_i[A_k], t_j[A_k])$ computes the similarity score in the real interval $[0, 1]$ with higher values for more likely duplicates. Popular similarity functions include Jaccard similarity, edit distance, and cosine similarity. Given a pair of tuples (t_i, t_j), we can compute the similarity vector as $sim_{\mathbf{R}}(t_i, t_j) \in \mathbb{R}^m$ by applying an appropriate similarity function for each attribute. We assume that the list of similarity functions for each attribute is provided to us by the domain expert. If they are not available, one can use tools such as Magellan [56] or Simmetrics [57] to identify them.

Blocking. Blocking is a key ER technique to avoid invoking the classifier on all $\binom{n}{2}$ tuple pairs to identify duplicates. Specifically, blocking identifies a group of tuples (dubbed blocks) such that the classifier needs to be invoked between pairs of tuples within that block. Tuples across different blocks can be considered as non-duplicates. There has been extensive work on performing effective blocking [58,59] which can dramatically reduce the search space.

Functional Dependencies (FDs). A functional dependency $X \rightarrow Y$ over attributes $X, Y \subseteq \mathbf{R}$ states that X functionally determines Y. X is the LHS (determinant) while Y is the RHS (dependent). An FD $X \rightarrow Y$ is said to be satisfied on $\mathbf{D}$ if $\forall t_i, t_j \in \mathbf{D}$, if $t_i[X] = t_j[X]$ then

$t_i[Y] = t_j[Y]$. The FD is said to be violated if there exists a pair of tuples with the same value for X but different values for Y.

In this work, we focus on FDs that are minimal (no subset of X determines Y) and normalized (Y is a single attribute). Also, an *approximate FD* (AFD) is violated by at most a small fraction (*i.e.,* under a predefined threshold) of the relation.

Metric for Data Errors. In this work, we focus on estimating the number of duplicates and FD violations. For duplicates, we seek to estimate the number of tuple pairs that are duplicates. For FDs, we consider the G_2 satisfaction metric [60] that measures the number of tuples that participate in some violation. Formally, $G_2(X \rightarrow Y) = |\{t | t \in \mathbf{D}, \exists t' \in \mathbf{D} : t[X] = t'[X], t[Y] \neq t'[Y]\}|$.

Types of Questions. We do not assume the availability of rules, constraints, or other information that could be directly used to estimate the data errors. Instead, we assume the availability of a human oracle who can answer simple questions reliably (our methods can be easily adapted for a noisy oracle). Specifically, we consider the following types of questions, which can be answered by cloud sourcing.

- **Duplicates**: Given a pair of tuples (t_i, t_j), do they refer to the same real-world entity?
- **FD Violations**: Given a pair of tuples (t_i, t_j) and a candidate FD $X \rightarrow Y$, we ask the question that, whether it is correct to consider values $t_i[X \cup Y]$ and $t_j[X \cup Y]$ together? For example, is that correct that one SSN number is associated with two different names in two tuples? Note that we do not ask the question whether $X \rightarrow Y$ is a valid FD, which is a much harder question.

Performance Measures. We consider two key metrics for measuring the efficacy of our algorithms.

- *Budget:* This measures the cost imposed on the expert in terms of the number of questions asked. Often, this is provided as an upper bound B on the number of questions that will be answered by the expert.
- *Relative Error:* Given an error type $\mathcal{E}$ (such as duplicates), let E be the exact number of errors while $\widetilde{E}$ be its corresponding estimate. The relative error of the estimate is measured as $|E - \widetilde{E}|/E$.

Chapter 3

Related Work

3.1 (Partially) Supervised Wrapper Induction

Wrapper induction is the process of inducing an extraction program, often one or more XPATH or similar expressions over the HTML tag tree, from a set of manually specified examples. Initially, wrapper induction approaches [12, 13] have been supervised where annotations are provided by humans (see [61]). The supervised setting has seen a flurry of industrial tools such as Lixto [12] or Mozenda, http://mozenda.com/. In fact, a simple incarnation of these inducers has become an essential tool for web developers, to inspect web sites and find relevant nodes. Firebug, as well as the developer tools provided with Chrome, Firefox, and Safari, allow to induce XPATH expressions from single samples. All of these tools assume a perfect input and *only* return expressions that cover all and only the given examples with no *noise*. Some of the more advanced systems [12, 13, 14, 15, 16], can infer a wrapper for a list of template items from examples of that list, but even in these cases positive noise is not considered. These methods usually use the user-specified examples as the "golden standard" for making selections of wrappers. For example, WIEN [13] treats the HTML document as a sequence of characters and learns wrappers by choosing the minimal prefix and suffix delimiting all the examples. WHISK [14] starts with an empty rule and extends it by adding terms according to its term selection metric. It repeats such process until the all the examples are covered by the generated rules, and performs a post-process of eliminating overfitting rules. STALKER [15] treats the document as a sequence of tokens and takes as input a set of sequences of tokens indicating the target data. It iteratively generates rules covering as many as possible uncovered examples and returns the disjunction of generated rules after all the examples are covered. The assumption that manually specified examples are accurate is not always true in practical scenarios, e.g., it is hard to guarantee the quality of examples provided by crowdsourcing.

That lack of robustness of earlier supervised approaches has been noted in [44, 45, 46, 47] and let to two methods that partially address that issue: **(1)** The first [44, 45] aims to find a more robust wrapper that selects the exact same data items as the given one. However, the approach is limited by the choice of wrapper language, a fairly narrow subset of XPATH. Most importantly, it does not address attributes and attribute values at all, yet the choice of

attributes and values is among the most crucial choices for robust wrappers (see Section 4.4). **(2)** The second is focused on learning probabilistic models of the evolution of web pages over time to rank XPATH expressions by their robustness [46]. This allows to tailor robustness to the specific change behavior of a website. However, the used XPATH fragment is strictly weaker than the one used in our approach (only `child` and `descendant` axis, at most one predicate per step, only equality predicates). Furthermore, our results demonstrate that expensive, site-specific training of change models is not required to achieve long term robustness. This work is extended to the notion of an optimal wrapper [47] with respect to a probabilistic and a adversarial (worst-case) change model.

Partially supervised methods, i.e., those methods using automatic annotators to replace human in the annotation process, have been recently proposed. Users are only required in the annotator preparation step. These systems achieve a reasonable trade-off between supervised and unsupervised methods, obtaining high accuracy while avoiding human effort for the annotation process. The approach in [50] first uses domain knowledge to annotate webpages and then apply (an extension of) Conditional Random Fields (CRFs), a machine-learned probabilistic model, to identify the data. The accuracy of this approach ranges from 60% to 80%, as reported in [50]. The state-of-the-art system, DIADEM, develops a novel domain ontology, called the DIADEM ontology, which not only includes the domain schema and its corresponding entity recognizers, but also the so-called *phenomenology* that describes each type's appearance features on the websites, e.g., some attributes only appear together. The additional constraints specified by the *phenomenology* provides extra clues for the repeated structure analysis and enhances the accuracy. Its data identification process is based on a sophisticated algorithm based on XPATH expressions capturing regular tree structures, and can well handle noise in the annotations. The approach in [49] provides interfaces to users to specify an intentional description of the target data, which is called *Structured Objet Description* (SOD). SOD is similar to DIADEM's ontology that it also regulates what entities the user wants to extract and how they appear on websites. However, wrt DIADEM, this approach is less tolerant to noisy annotations limiting its overall accuracy.

The advantage of partially supervised approaches based on automatic annotations is that they significantly save the human effort, as they work in a fully unsupervised manner once the annotator is ready. It is worth mentioning that, among these approaches, the state-of-the-art approach, DIADEM, achieves high accuracy while scaling at thousands of websites over a single domain. It has been extensively evaluated on two domains USED_CARS and REAL_ESTATE. However, to bootstrap the process, such approaches usually need entity recognizers to produce annotations for the attributed in given domain or an ontology-expert to manually tune the domain-specific ontology, thus limiting the ability to scale out over multiple and diverse domains. In addition, it is not realistic for normal users to prepare the annotators. Thus, partially supervised methods are more suitable for vertical data extraction tasks, usually from organizations who are interested in the data of some particular domain. Our method RED provides a feasible solution to extracting data from various domains.

For none of the approaches based on automatic annotations robustness of the induced wrapper is a primary concern, though [48] employs, among others, the induction method from [46]. Lacking of wrapper robustness seems not to be a problem for these methods. DIADEM [17] periodically monitors wrapper failures and directly triggers the regeneration of wrappers once a failure is found. Since the annotation process is done by automatic annotators, no extra human effort is required. The other systems based on automatic annotators can also adopt this method. However, robust wrappers are also favorable to these approaches since robust wrappers can largely save the wrapper regeneration cost. In many ways, the techniques for inducing a robust wrapper discussed in this thesis are orthogonal to these approaches. In most cases, our induction method could be integrated with these systems, thus yielding more robust wrappers, as shown in Section 4.4.

3.2 Unsupervised Web Data Extraction

Preexisting unsupervised approaches incorporates one or more features that broadly fall into three different strands in research. **(1)** Approaches based on repeated-pattern analysis [1, 2, 3, 4, 5], deals with pages generated from templates by searching for repeated structures either within the page or between different pages generated from the same template. **(2)** Approaches based on visual information [6, 7, 8], utilize visual information of webpages to improve the HTML document analysis. **(3)** Approaches based on content redundancy [9, 10, 11] , exploits either the intra-site or cross-site content redundancy to guide the data identification process.

Approaches based on repeated-pattern analysis. The most common unsupervised wrapper generation approaches are based on discovering repeated structures on pages presumably generated by a common template.

ROADRUNNER [1] is based on the webpage generative model that an HTML template is used to generate a set of web pages by taking different data from the underlying database. It iteratively improves the template which has been initialized from the first page of the input, until the template can generate all the example pages. Its assumptions that all the HTML tags are always from the HTML template and that the "grammar" of the HTML template is union free are not valid on many websites. To improve the drawback of ROADRUNNER, EXALG [2] assumes that HTML tags as part of the data, and it distinguishes the HTML template and the data by learning regular expressions from the large and frequently occurring equivalence classes of tokens (LFEQs). However, as pointed out by the authors, EXALG works well only when several assumptions all hold, such as a sufficiently large number of template tokens have unique roles and there are no regularities in the data making the data look like LFEQs. IEPAD [3] decodes the HTML tag sequence into a string and learns maximal repeated patterns by a PAT tree which are then left for users to pick the best one. It works well only when data is plain-structured but cannot handle the data records with a complex and nested structure. Following the work in IEPAD, DELA [4] uses a pattern extractor based on suffix trees to

discover continuously repeated patterns. An advantage of DELA is that it can extract the data-rich sections. However, it also often produces multiple rules requiring users to select, and it cannot handle the situations where repeated structures are not obvious due to optional tags. MDR is designed only for result-page segmentation. It treats a fixed combination of nodes as a generalize-node and uses edit distance between generalize-nodes to identify data regions. The problem of MDR is that it can be fooled by regular but noisy content on the page such as advertisements and it often over-segments the data area by recognizing optional nested structures as separate data records.

In contrast to those systems, RED combines a more sophisticated template analysis with the content redundancy to filter out the noise. Unlike MDR, RED does not directly solve the segmentation problem (actually it has not to). Instead, it uses details link to perform a (soft-)segmentation to identify the records thus reducing the chances to consider irrelevant structures. Similar to ROADRUNNER and EXALG, RED assumes detail pages mostly share the same templates. However, RED does not make assumptions on the features of repeated structures. The only assumption of RED is that it assumes the set of attributes published on a result page is the subset of the attribute set on detail pages, which in fact holds for most of the websites. RED borrows idea from ROADRUNNER and EXALG that its candidate rule generation step (see Section 5.3) also exploits the template discovery process. However, being aware of the inaccuracy of such process, RED only uses the found template nodes as pivots to build candidate rules.

Approaches based on visual information. Similar to our method RED and DIADEM both of which exploit additional information to improve the extraction accuracy, there are methods [6, 7, 8] using visual information to enhance the analysis of HTML document. VIPER [6] analyzes both visual similarities and the HTML tag structures to first find and rank candidate repetitive patterns, and then aligns the matching sub-sequences with global matching information. VIPER suffers from poor results for nested structured data. ViNTs [8] identifies the regularities of data values by utilizing the visual data value similarity (without HTML tags) and generates wrappers by combining the data value regularities with the HTML tag structure regularities. To weight the relevance of different extraction rules, both visual and non-visual features are exploited. The major problem of ViNTs is that it assumes data records are listed horizontally. DEPTA [7] is an improvement of MDR, although it is primary tag-structure based, it adopts the intuition that the gap within a record is typically smaller than that between records and builds a tag tree according to visual rendering information such as fonts of text to align attributes. DEPTA is noise-sensitive and incomplete. It needs to mine frequent repetitive patterns before the alignment step, however the mining process cannot deal with situations where some data is missing in the repetitive patterns. Its attribute alignment is partial because it can only align the values which can be assigned unique positions in the tag tree.

Approaches based on content redundancy. The approach in [9] and WEIR [10] both exploit cross-site redundancy that multiple sites contain pages for the same entity. The former exploits the redundancy in a way that it scans the webpages to find values that match attribute values in the pre-built seed database. It is actually partially supervised since it requires human effort to build the initial seed database. WEIR leverages the redundancy to prune the candidate rules that do not have redundancy. Both approaches work directly on detail pages, therefore they are not faced with the issue of record segmentation as in RED. Also, they rely on template consistency – just as RED does – but they both assume a different notion of content consistency: While RED requires redundancy in the presented objects between result and detail pages of same site, they require redundancy between different sites. The content consistency assumed by RED is inherent to today's web design patterns and occurs in many domains. Importantly, RED does not impose any assumption on the content distribution between different sites. Similarly to RED, the system [11] exploits the content redundancy between the result/detail pages but uses it differently. Their method majorly deals with segmenting result pages into records by using information contained in detail pages. However, unlike RED, they do not produce rules for extracting attribute values. They present two different techniques. The first one finds the record segmentation by computing a solution to the constraint-satisfaction problem (CSP) based on the model taking the relations between the result-page data and the detail-page data as constraints. This approach is sensitive to errors and inconsistencies in the sources. The second technique adopts a probabilistic inference approach and achieves better performance. In contrast to RED where similarity metrics are used when finding redundant content, they only consider those strings that perfectly match thus restricting its applicability to websites with variations.

3.3 Error Estimation

Data Cleaning. Data cleaning is a long-standing challenge which is exacerbated with the proliferation of big data and the emergence of a data-driven world. The overall process of data cleaning is given some dirty data, detect which parts of the data are dirty, *i.e.,* differ from the (unknown) ground truth, and possibly repair them. In general, data cleaning can be classified into four classes of solutions [40], namely rule-based methods, pattern-enforcement and transformation methods, quantitative methods for outliers and glitches, and entity resolution and record linkage methods. Please refer to [62] for additional details.

Entity Resolution. A good overview of ER can be found in surveys such as [63, 64]. Prior work on ER can be categorized as based on (a) declarative rules, (b) machine learning, and (c) expert or crowd based. Declarative rules such as DNF [30] specify rules for matching tuples that are easily interpretable but often requires a domain expert. Most of the ML approaches are variants of the classical Fellegi-Sunter model [31]. Popular approaches include

SVM [30], active learning [32], clustering [33]. ER using crowdsourcing has become popular [34, 35]. Lately, there have been also attempts at using distributed representation and deep learning to solve the ER problem [36]

Integrity Constraints. Several ICs, such as FDs, conditional FDs (CFDs), and denial constraints (DCs), have been proposed to capture different types of data errors that violate these ICs. In practice, in many applications, these ICs are provided by domain experts through expensive eyeballing exercise. There are various algorithms to automatically discover ICs. Many of them assume that there exists a clean yet representative sample of data, so the main issue is generally to improve the efficiency [37,38,39]. Moreover, there are algorithms to find approximate ICs when assuming that the data is not clean, such as discovering FDs [65, 66], CFDs [67], MDs [68], and DCs [69]. Usually, a very large number of approximate ICs might exist when we assume that the dataset is dirty, and approximate IC discovery algorithms are very sensitive to dirty data. Consequently, asking the user to select the relevant ones from a large number of candidate ICs is often unfeasible in practice.

Error Estimation. Two relevant and recent works are [52] and [53]. However, neither can be directly used. More specifically, [52] is interested in estimating the number of undetected errors after a series of manual or algorithmic cleaning operations have been performed. In contrast, we are interested in estimating the number of errors with limited interactions with the expert. [53] proposed a sampling-based approach for estimating the number of duplicates in a dataset. The authors assumed the availability of an algorithmic mechanism to estimate duplicates and are interested in getting a rough estimate of the number of duplicates identified by that algorithm. In contrast, we are interested in estimating the number of duplicates present in the dataset irrespective of a particular entity resolution algorithm. To see the contrast, if the classifier in [53] has a F-measure that is not sufficiently high, it might result in a significantly incorrect estimate. Further, the approach already assumes the availability of some mechanism such as a ML classifier to tell if a pair of tuples are duplicates. Training such a classifier would already require substantial amount of training data. Our objective is to estimate the error in a given dataset with minimal interaction with the expert.

Estimating Classifier Accuracy. There has been extensive work on evaluating classifier accuracy through diverse sampling based techniques such as importance sampling [70], stratified sampling [71,72] through auxiliary information such as classifier scores. However, most of these work focus only on precision and not on recall. This approach does not work ER as recall (and thereby F-measure) becomes a more important measure due to the unbalanced nature. Recently, [73] proposed a direct approach for evaluating the performance of a classifier for ER. Please refer to [73] and the references therein for additional details.

Chapter 4

Robust and Noise Resistant Wrapper Generator

In scenarios where the user's data demand is unclear, programs have to be bootstrapped with the help of users. To maximally reduce the user engagement, a good wrapper induction approach should allow users to specify only an incomplete set of samples or use annotators which are usually inaccurate, and avoid asking users to repeat the annotation processes after webpage changes. In other words, a good wrapper is expected to be **(1)** *accurate* which is a notion common to most approaches, **(2)** *robust* against structural changes of the webpage over time, and **(3)** the induction process should be *resistant to noise* in the annotations. To meet such requirements, we propose a a wrapper induction framework for generating accurate and robust wrappers under noisy annotations. Different from all existing methods, our wrapper induction framework satisfies all the three requirements at the same time. Key to our approach is a query language dsXPath that is powerful enough to permit accurate selection but limited enough to force noisy samples to be generalized into wrappers that select the likely intended items, and a score-based ranking schema to consider the robustness and accuracy at the same time.

4.1 XPATH Fragment dsXPath

In this section we define our XPath fragment, called *directed XPath with sideways checks* (for short, dsXPath). The idea of dsXPath is to be **(1)** *general enough* to cover all samples with 100% precision and recall and **(2)** *restricted enough* as to enforce noise resistant queries. In Section 4.2, we show how the careful choice of the language fragment together with a straightforward, but flexible scoring of expressions yields a noise resistant, yet accurate and robust wrapper induction.

Query Syntax The syntax of dsXPath queries is presented in Figure 4.1. A query consists of one or more steps. Each step consists of an axis, a nodetest, and an arbitrary number of predicates. The axis is any of XPATH's navigational axes (including the attribute axis), except the following and preceding axes. The nodetest is any of XPATH's nodetests. The

⟨*Query*⟩ ::= ⟨*Step*⟩ ('/' ⟨*Step*⟩)*

⟨*Step*⟩ ::= ⟨*Axis*⟩ '::' ⟨*Nodetest*⟩ ('[' ⟨*Predicate*⟩ ']')*

⟨*Axis*⟩ ::= **child** | **attribute** | **descendant** | **following-sibling**
| **parent** | **ancestor** | **preceding-sibling**

⟨*Nodetest*⟩ ::= '*' | 'node()' | 'text()' | ⟨*TagName*⟩

⟨*Predicate*⟩ ::= ⟨*Int*⟩ | 'last()' '-' ⟨*Int*⟩
| **attribute** '::' ⟨*AttrName*⟩
| ⟨*Function*⟩ '(' ⟨*Content*⟩ ',' ⟨*String*⟩ ')'

⟨*Function*⟩ ::= equals | contains | starts-with | ends-with

⟨*Content*⟩ ::= **attribute** '::' ⟨*AttrName*⟩ | normalize-space(.)

Figure 4.1: Syntax of our XPATH Fragment

nonterminals ⟨*TagName*⟩ and ⟨*AttrName*⟩ in Figure 4.1 refer to the strings allowed as element and attribute names. A predicate is either positional (i.e., consisting of a number or of `last()` minus a number), the test for the existence of an attribute, or one of four possible Boolean functions on strings. The second argument to a Boolean string-function is an arbitrary string of characters (indicated by the nonterminal ⟨*String*⟩). The first argument to a Boolean string-function is either an attribute selection (given by the attribute axis followed by a name) or a text access. The only text access that we consider is the fixed expression `normalize-space(.)`. This expression selects and normalizes (i.e., removes extra whitespace) the text-value of the current node. E.g., applied to the `div`-node in Figure 1.3 we obtain "Director: Martin Scorsese". For brevity, we write only `.` instead of `normalize-space(.)`.

We explain the semantics of XPATH using examples, see [74] for details. We implicitly assume a given document D. A query q is evaluated starting at a given node u of D. The result $q(u)$ is a set of nodes of D. Consider the following query q.

```
descendant::div[starts-with(.,"Director:")]/descendant::span[(@class="itemprop")]
```

The evaluation of q on the root of a document D goes step-wise through the XPath query: first all div-labeled descendants of the root node are selected, and of those, the ones with text-value starting with "Director:". Finally, we select from these nodes the span-labeled descendants with class-attribute equal to "itemprop" and return them as result.

The above query uses two anchor nodes to characterize the target `span`. An *anchor node of* q is a node of D that is selected during the evaluation of $q(r)$ on D. We do not give a formal definition but note that red-marked nodes in Figure 1.3 are the anchors of this query.

dsXPath Queries We now define *directed* XPATH *queries with sideways checks* (dsXPath Queries) as a restriction of the syntactical fragment given in Figure 4.1. We choose this fragment for two reasons, namely, to obtain a feasible search space and to foster generalization during induction to avoid overfitting of noisy query samples. Such queries are either *one-* or *two-directional*. Intuitively, a one-directional query does not change its up-/down direction, i.e., it strictly traverses the tree either top-down or bottom-up (but not

mixtures thereof). However, in a one-directional query we do additionally allow "sideways checks". A sideways check is a subquery consisting of the following- and preceding-sibling axes. A two-directional query is the concatenation of two one-directional queries. Given a query $q = a_1 :: t_1P_1/a_2 :: t_2P_2/\ldots/a_n :: t_nP_n$ with axes a_i, node tests t_i, and predicates P_i, we define $\mathsf{axes}(p) = a_1 \ldots a_{n-1}$ if $a_n =$ attribute and $\mathsf{axes}(p) = a_1 \ldots a_n$ otherwise. The query q is *one-directional,* if $\mathsf{axes}(q)$ matches one of the following regular expressions with `following-sibling`$^* \mid$ `preceding-sibling`$^* = \langle sideways \rangle$:

(1) $((\texttt{parent} \mid \texttt{ancestor})\ \langle sideways \rangle)^*$ or

(2) $((\texttt{child} \mid \texttt{descendant})\ \langle sideways \rangle)^*$.

Finally, for a given sequence of documents $\mathcal{D} = (D_1, \ldots, D_n)$, and a given dsXPath query q, we say that q is *plausible*, if all constants appearing in the predicates of q also appear in the considered document. Recall that the text-value of a document is the concatenation of all texts in the document. The query q is plausible, if it is generated by the EBNF in Figure 4.1 in such a way that **(1)** each string is either a substring of the text-value of a document in $\mathcal{D}$, or is a substring of an attribute value of a document in $\mathcal{D}$, and **(2)** each integer produced by the nonterminal $\langle Int \rangle$ is not larger than the number of nodes of any document in $\mathcal{D}$. From now on, we only consider plausible dsXPath queries.

4.2 Problem Definition

We first introduce the query model and types of induced queries. Then our ranking is defined and the query induction problem is noted to be intractable (**NP**-hard).

Data Demand Given a document D, the user's *data demand*, or the *target data*, denoted by T, is a set of nodes of D which are of interest to the user.

Query Model Given a document D, a *query sample* is a pair $\langle u, V \rangle$, where u is a node of D and V is a non-empty set of nodes of D. Our query induction induce takes a sequence of query samples $S = (\langle u_1, V_1 \rangle, \ldots, \langle u_n, V_n \rangle)$ over the documents $\mathcal{D} = (D_1, \ldots, D_n)$ (possibly containing duplicates) as input and returns a ranked set of *query instances* $Q = \mathsf{induce}(S)$. A query instance $q = \langle p, t^+, f^+, f^- \rangle$ consists of an XPath expression $p = \mathsf{query}(q)$ and the corresponding numbers of true positives, false positives, and false negatives, i.e., the numbers $t^+ = \Sigma_{i=1}^n |p(u_i) \cap V_i|$, $f^+ = \Sigma_{i=1}^n |p(u_i) - V_i|$, and $f^- = \Sigma_{i=1}^n |V_i - p(u_i)|$. Given q, we write $t^+(q)$, $f^+(q)$, and $f^-(q)$ for the individual values of q. We also write $q(u)$ in lieu of $p(u)$ with $p = \mathsf{query}(q)$.

Ideally, the input should be perfect, i.e., for each document D, V should equal to the target data T. In these cases, each XPath expression $p = \mathsf{query}(q)$ for $q \in \mathsf{induce}(S)$ should lead from u_i to V_i, i.e., $p(u_i) = V_i$ (evaluated in D_i) for all $1 \leq i \leq n$. Note however that a fully accurate expression p is not always possible or – in case of noisy input – not even desired.

Ranking In order to deal with noisy additional annotations, we choose $\beta = 0.5$ for the F-score to weight the precision more heavily. The rational is that, going the opposite way will favor wrappers also extracting the false positives. We rank the query instances q by maximizing the F-score $\mathsf{F}_{0.5}(q)$ before minimizing the robustness score $\text{score}(q)$. More precisely, we define the order $<$ with $q < q'$ iff (1) $\mathsf{F}_{0.5}(q) > \mathsf{F}_{0.5}(q')$ or (2) $\mathsf{F}_{0.5}(q) = \mathsf{F}_{0.5}(q')$ and $\text{score}(q) < \text{score}(q')$. The *robustness score* $\text{score}(q)$ is a positive number computed recursively from the structure of the query expression $\text{query}(q)$. The *smaller* the score, the more favorable the query. The robustness score of an XPATH expression is computed as the sum of the scores of its steps, because we favor shorter queries over longer ones; we call this *plus-composability*. The scores of steps are multiplied by a power of a *decay-factor* δ. This allows to favor cheaper steps towards the end (beginning), i.e., closer to (further from) the target node, by choosing $\delta \leq 1$ ($\delta \geq 1$). We define $\text{score}(a_1 :: t_1P_1/\cdots/a_n :: t_nP_n) = \sum_{i=1}^{n} \text{score}(a_i :: t_iP_i) \cdot \delta^{i-1}$. where for $i \in \{1,\ldots,n\}$, a_i is an axis, t_i is a node test, and P_i is a sequence of predicates.

For each axis a (such as `descendant` of `child`) we fix a constant score s_a. Similarly, for each node test t we fix a constant s_t. Note, that different tags can thus be scored differently, e.g., `div`'s differently from `script` tags, though in most cases a default value $c_{default}$ is used. The score for a single step $a :: tP$ is now computed as the sum of the scores for axis, node test, and predicates $P = p_1 \ldots p_m$: $\text{score}(a :: tp_1 \cdots p_m) = s_a + s_t + \sum_{j=1}^{m} \text{score}(p_j)$.

The score for a predicate p is computed as follows. If p is a positional predicate, i.e., of the form $p = [n]$ or $p = [\texttt{last()} - n]$ where n is a positive integer, then the score for p is $\text{score}(p) = c_{\text{pos}} \cdot n$, where c_{pos} is a fixed constant.

If p is of the form `[`f`(`**`attribute`**`::`a`, `w`)]` or `[`**`attribute`**`::`a`]` (in the latter case we set $\text{length}(w)$ and s_f to zero) where a is an attribute name, f is a function (such as "equals" or "contains"), and w is a string, then the score for p is the sum of a fixed score s_f, a *no-function-penalty* constant y, a fixed score s_a, and the product of the fixed length factor c_f and the length of the string w: $\text{score}(p) = s_f + y + s_a + c_f \cdot \text{length}(w)$, where y has a non-zero value only if $p =$`[`**`attribute`**`::`a`]`. For a predicate with text access of the form $p =$`[`f`(.,`w`)]` where f is a function (such as "equals" or "contains") and w is a string, we define $\text{score}(p) = s_f + s_{\text{text}} + c_f \cdot \text{length}(w)$, where s_{text} is a fixed score. If q contains no predicate, we add to $\text{score}(q)$ a *no-predicate-penalty*. The two penalty scores *no-function-penalty* and *no-predicate-penalty* bias the scoring towards queries with predicates that are likely more selective.

Query Induction Problem We define the query induction problem and prove that it is intractable. The query induction problem is the optimization problem of finding a selection of the best ranked query instances among the plausible directed expressions in dsXPath, where optimality is in terms of the ranking given in the previous section. For a natural number K, a top-K set is a set of ranked query instances with the highest achievable scores; there might be several such top-K sets achieving the same scores.

Problem 4.2.1 (XPATH Query Induction Problem)**.** *Given a sequence* $S = (\langle u_1, V_1 \rangle, \ldots, \langle u_n, V_n \rangle)$ *of query samples over implicit documents* $\mathcal{D} = (D_i, \ldots, D_n)$ *and a constant K, the* query induction problem *is to find a top-K set of plausible query instances.*

Theorem 4.2.1. *The query induction problem is* **NP***-hard, even with query samples of the form* $\langle u, \{v\} \rangle$*, requiring an optimal query instance leading from u to v.*

Proof. We reduce MINIMUM SETCOVER to the problem of inducing an accurate XPATH expression with minimal robustness score. The input to MINIMUM SETCOVER is a (finite) universe U and a family $S = \{S_1, \ldots, S_m\}$ of subsets $S_i \subseteq U$, together with a constant K. The output of MINIMUM SETCOVER is "yes" if there exists a cover $C \subseteq S$ with $|C| \leq K$ such that $U = \bigcup_{S_i \in C} S_i$, and is "no" if no such cover exists.

Reducing instances. We construct a hard instance of the query induction problem. Our sample consists of a single tree T. This tree has a root node r with $|U|+1$ children, one child for each $u \in U$ (such a child is denoted u), plus one extra child v. For every $1 \leq i \leq m$ and $u \in U$, the child u has an attribute S_i if and only if $u \notin S_i$. Finally, the extra child v has all attributes $S_1, \ldots, S_m$. In this sample only the node v is annotated. For the robustness scoring we define $s_{\text{child}} = s_* = 1$ and for each S_i define the score $s_{attribute::S_i} = 1$. For all other axes, tag names, and predicates we set the score to $s_{\text{child}} + s_* + |U| + 1$. These values guarantee that optimal query instances that represent solutions to the set cover problem are of the form `/*[attribute::`S_{i_1}`]...[attribute::`S_{i_k}`]`.

It is now suffices to run query induction to obtain the top-1 set $\{\langle p, t^+, f^+, f^- \rangle\}$ of plausible query instances. If p is not of the form above, or if $f^+ \neq 0$ or $f^- \neq 0$, then there is no solution to the set cover problem. Otherwise, it follows from the definition of attributes that p is of the form `/*[attribute::`S_{i_1}`]...[attribute::`S_{i_k}`]`, and $C = \{S_{i_1}, \ldots, S_{i_k}\}$ is a cover of S. This is the case, because v contains all attributes $S_i \in S$ and is thus included in the result set. On the other hand, for all $u \in U$, since u is not included in the result set (by $f^+ = 0$), there exists an $S_i \in C$ with $u \in S_i$ such that node u does not feature attribute S_i. Hence, p matches v from the root node uniquely if and only if C is a cover. Since q has the lowest robustness score it follows that p is the shortest query expression of the form that matches v uniquely. This implies that the set cover problem has a solution if and only if $k < K$. □

4.3 dsXPath Induction Algorithm

In this section our query induction algorithms are explained, first for single-target query samples, and then for multiple-target query samples. We always work in the context of a given number K, and estimate the sequences of "K-best" query instances (ordered according to our ranking). The output of our algorithm is a set of approximately "K-best" query instances, since finding the exact "K-best" query instances has been proven intractable. The resulting algorithm has polynomial runtime since it executes a polynomial number of XPATH queries of polynomial size over the analyzed document. We define

the *set B of base axes* as $B := \{$**child**, **parent**, **following-sibling**, **preceding-sibling** $\}$, and define **child**.transitive = **descendant** and **parent**.transitive = **ancestor**. Further, we set α.transitive $= \alpha$ for $\alpha \in \{$**following-sibling**, **preceding-sibling**$\}$. For an axis β, we say that node v is β-*reachable* from u if and only if $v \in q(u)$ for the query $q = \beta :: *$.

Single-Target Query Samples Let D be a document and u a node of D. Here we consider the restricted case of one target node, i.e., $V = \{v\}$ where v is in D. Our algorithms are constructed in such a way that v is guaranteed to be reachable from u via the transitive axis of one of our base axes. The most interesting case of our induction algorithm, is for the case that v is a descendant of u, i.e., for the **child** base axis. The cases for the other base axes follow from this case in a straightforward way. Let us thus discuss now the case that v is a descendant of u. We denote the sequence of nodes on the path from u to v by $\mathsf{spine}(u,v)$, and refer to the nodes in $\mathsf{spine}(u,v)$ as *possible anchors*. To compute the best-K query instances leading from u to v, we incrementally compute the best-K instances $best(n)$ for each node n in $\mathsf{spine}(u,v)$. Initially, we set $best(v)[1] = \langle \varepsilon, 1, 0, 0 \rangle$ and $best(v)[k] = \langle \bot, 0, 0, 0 \rangle$ for $2 \leq k \leq K$, The "empty query" ε occurs only as intermediate result; When applied to $best(v)$, it selects only the node v itself. By $\bot$ we denote the "fail query" which always returns $\emptyset$. We now recursively compute for each node $n \in \mathsf{spine}(v,u) - \{v\}$, the superset $cand(n) \supseteq best(n)$ of *candidate instances*, by using the already computed best-K sets $best(t)$ for nodes t preceding n in the sequence $\mathsf{spine}(v,u)$. From this superset, we assign the best-K instances to $best(n)$ and proceed along the spine until we have computed $best(u)$, which is returned as result. The superset $cand(n)$ is computed as $cand(n) = \bigcup_{t \in \mathsf{spine}(v,n) \setminus \{n\}} \mathsf{stepPattern}(n,t,\mathsf{axis},K) \times best(t)$ as shown in Algorithm 2. Here, axis is the unique base axis in B such that t is axis.transitive reachable from n, i.e., axis =**child** for the case discussed here. The function $\mathsf{stepPattern}(n,t,\mathsf{axis},K)$ computes the best-K query instances that match t from n along axis (possibly involving "sideways detours" but no intermediate anchors), and $\times$ yields the concatenated instances of the cross-product of the two instance sets. We next explain function $\mathsf{stepPattern}(n,t,\mathsf{axis},K)$ performing induction along the spine.

Spine Step Induction The function $\mathsf{stepPattern}(n,t,\mathsf{axis},K)$ shown in Algorithm 1 generates the available spine patterns to match spine node t while moving along axis (or its transitive closure) starting at context node u; only the case of the child-axis is shown. Note that in the return specification of the algorithm, axis.reverse refers to the reverse of an axis, i.e., **child**.reverse = **parent** (and vice versa), **descendant**.reverse = **ancestor** (and vice versa), and **following-sibling**.reverse = **preceding-sibling** (and vice versa). This algorithm works as follows. We first compute for axis = child recursively through inducePath (Lines 2–5). Note, child is the *unique* axis for which we ever produce sideways checks in our algorithms! This explains that in the recursive case (for the child axis), we iterate over all siblings s of t (Line 2), compute the node patterns for s (Line 3) and the best-K paths from s to t recursively

(Line 4). Note that each such path consists of exactly one XPATH step that uses either the `following-sibling` or `preceding-sibling` axis (depending on the position of s with respect to t). Then we add all possible concatenations between node patterns and paths (Line 5). Thus, each query in P starts with a node test. As all recursive calls to inducePath generate a path along the sibling axes, the subsequent calls to stepPattern will be handled non-recursively (because of Line 1).

Algorithm 1: Step Induction stepPattern(n,t,axis,K)

input : node t is axis.transitive-reachable from node n
returns : path set P such that $\forall p \in P$ and all nodes x
if x is axis.reverse.transitive-reachable from t,
then $t \in p(x)$

```
1 if axis = child then
2     for s ∈ siblings(t) do
3         P' := nodePattern(s);
4         P'' := inducePath(s, {t}, K);
5         P := P ∪ {p'/p'' | p' ∈ P' and p'' ∈ P''};
6 else P := nodePattern(t);
7 R := {axis.transitive :: p | p ∈ P};
8 if t ∈ axis(n) then R := R ∪ {axis :: p | p ∈ P};
9 return rescore(n, {t}, R);
```

Let us discuss the function nodePattern. Given a node u, this function iteratively computes a set of possible node tests followed by at most two predicates. In a first step, a node test for the node u is generated. This starts with the most general such test `node()` followed by the test of the tag name of node u. Then one predicate with an attribute (or text-value) equality-comparison is produced. The resulting queries, with axis prepended are tested on node n. If u is *not* uniquely matched, then an additional positional predicate is concatenated. Thus, each pattern returned by nodePattern contains at most two predicates: one attribute (or text) comparison (possibly) followed by a positional predicate. The selection of attribute names and text functions follows our definition of scoring for such query constructs (see Section 4.2). As an example if the node u is labeled `div`, then the first few node patterns that are returned by the call nodePattern(u) are of the form:

```
node()                          div
div[@id='x']                    div[@class='y']
div[contains(.,'z')]
```

Note that x,y,z are constrained to be single strings that appear in the input document as follows: either as single words (space-separated and/or bordered) or as the full text-value of a node.

The node tests P at hand, we complete these patterns by prepending axis.transitive to all patterns in P (Line 7). If t is reachable with a single step along axis, we prepend axis to

all patterns in P (Line 8). At last, the algorithm returns the resulting patterns R, scored as expressions matching only t from context node u (Line 9).

As an example, consider the sample $\langle u, \{v\}\rangle$ where u is the body-node and v the em-node in this document:

```
<body>
  <div class="content">
    <div id="main">
      <em class="highlight">The Target</em>
</div></div></body>
```

This means that we have four nodes on the spine from u to v: the body-node, the two div-nodes, and the em-node. We now generate stepPatterns, starting at the lower div-node (with id-attribute), and matching the em-node. E.g. these expressions are generated:

```
descendant::em
child::em
child::node()[class="highlight"]
```

Next, similar patterns are generated for expressions matching the em-node, starting at the upper div-node, and then starting at the body-node. In the next step we generate stepPatterns for the lower div-node; first, starting from the div-node above it, e.g., the expressions `descendant::div` and `descendant::div[@id="main"]`. Next, patterns are generated that match the lower div-node, starting at the body-node, e.g., the expression `descendant::div[@id="main"]`. Note however that the expression `descendant::div` it *not* generated at this step, because it matches both div-nodes, and hence is not accurate (strictly speaking, the expression may be generated, but receives a very low score). To update the best patterns (from u to v), we now also generate combined patterns such as patterns such as

```
descendant::div[@id="main"]/child::em
```

Finally, stepPatterns for the upper div-node are generated, starting from the body-node, and are combined with the previous expressions, so that for instance the expression

```
descendant::div[@class="content"]/child::div[@id="main"]/child::em
```

is generated. The precise ranking of these expressions depends on the parameter; see Section 4.4.2 for typical parameter choices.

Multiple-Target Query Samples We now generalize to multipe-target query samples, i.e., samples $\langle u, V\rangle$ with $|V| > 1$. Thus, inducePath takes u, V, and K as arguments, along with pre-initialized tables *best* and *tar*, respectively for the best-K instances and the relevant targets to be matched (explained below). The *best* table is initialized as before to $\langle \bot, 0, 0, 0\rangle$ for all relevant entries, except for the first entry in every target node $v \in V$ which is set to $\langle \varepsilon, 1, 0, 0\rangle$. The table *tar* contains for each relevant node n the axis-reachable targets in V, where axis is the direction of the one-directional dsXPath reaching from u to V Both tables are handed in as arguments to enable the reuse of this algorithm in more general settings beyond the base case. For dealing with $|V| > 1$, the algorithm iterates over all targets $v \in V$

Algorithm 2: Axis Path Induction
inducePath$(u, V, K, \text{axis}, best, tar)$

input : node u and node set $V \neq \emptyset$,
$\forall v \in V : v$ is axis.transitive-reachable from u,
$K > 0$ best-K bound,
$best$ initial table with best-K paths (wrt. score),
tar table of reachable target nodes
returns : query instances q with $q(u) \approx V$, ranked by score

```
1  for v ∈ V do
2      for t ∈ spine(v,u) − {u} do
3          for n ∈ spine(u,t) − {t} do
4              V' := tar(n);
5              for 1 ≤ k ≤ K and p ∈ stepPattern(n,t,axis) do
6                  p' := p/best(t)[k];    M := p'(n);
7                  q := ⟨p', |M ∩ V'|, |M − V'|, |V' − M|⟩;
8                  if q < best(n)[K] then
9                      insert q into best(n);
10 return best(u);
```

(Line 1) while maintaining a single best-K table $best$ throughout all iterations. In each iteration v, inducePath induces the best-K instances which match v from u, possibly along with other targets from V. However, to produce accurate results, inducePath evaluates the accuracy of the induced instances against V (or subsets of V in for intermediate instances). Hence, $best(u)$ contains upon loop termination the so-far best expressions matching V and is returned as result (Line 10). The sign $\approx$ in the specification of the algorithm means that the query instances select *some* nodes of the sample. Note that $q < best(n)[K]$ in Line 8 of means that the ranking of q is strictly smaller than the ranking of the K-th query instance in the table $best(n)$.

For each v, inducePath computes for all nodes $n \in \text{spine}(u,v) - \{v\}$ the best instances to match v and evaluates them against all reachable targets $tar(n) = V \cap \text{axis.transitive}(n)$, given tar is initialized accordingly. To enable a dynamic programming approach, we need to ensure that the entries in $best$ are already computed when the algorithm reads these entries. Thus, we first iterate (Line 2) over all possible anchors $t \in \text{spine}(v,u) - \{u\}$ and in an inner loop (Line 3) over all nodes n for which t could serve as anchor, i.e., $n \in \text{spine}(u,t) - \{t\}$. In each inner iteration, we enumerate all instances in $\text{stepPattern}(n,t,\text{axis},K) \times best(t)$ (Line 5) but only add those to $best(n)$ which beat the K-best know instance (Lines 8–9). This way, $best(t)$ is not altered anymore, once the algorithm reads $best(t)$ to generate candidates for $best(n)$. The remaining lines in Algorithm 2 deal with the accuracy evaluation of the generated instances: we obtain the relevant targets V' and wrap each generated XPATH expression p' into an instance q with the true positives, false positives and negatives evaluated against V'.

Algorithm 3: Path Induction $Q = \mathsf{induce}(S, K)$

input : samples $S = \{\langle u_1, V_1\rangle, \ldots, \langle u_n, V_n\rangle\}$ such that
$V_i \neq \emptyset$ for all $1 \leq i \leq n$, $K > 0$ as best-K bound
returns : query instances Q with $q(u_i) \approx V_i$
for all $q \in Q$ and $1 \leq i \leq n$

1 **for** $\langle u_i, V_i\rangle \in S$ **do**
2 **if** $\exists a \in B\ \forall v \in V_i :$ v *is* axis.transitive-*reachable from* u_i **then**
3 $Q_i := \mathsf{inducePath}(u_i, V_i, K, a, \mathsf{init}(u_i, V_i, K))$;
4 **else**
5 $l_i := \mathsf{lca}(V_i)$;
6 **if** $\nexists a \in B :$ l_i *is* a.transitive-*reachable from* u_i **then**
7 $l_i := \mathsf{lca}(V_i \cup \{u_i\})$;
8 $\langle best, \_\rangle := \mathsf{init}(u_i, \{l_i\}, K)$;
9 $a :=$ unique axis in B so that $\forall v \in V_i$: v is a.transitive-reachable from l_i
10 $best(l_i) := \mathsf{inducePath}(l_i, V_i, K, a, \mathsf{init}(l_i, V_i, K))$;
11 **for** $n \in \mathsf{spine}(u, l_i) - \{l_i\}$ **do** $tar(n) := V_i$;
12 $a :=$ unique axis in B so that l_i is a.transitive-reachable from u_i
13 $Q_i := \mathsf{inducePath}(u_i, \{l_i\}, K, a, best, tar)$;
14 **return** $\mathsf{aggregate}(\bigcup_{i=1}^{n} Q_i)$;

Two-Directional Paths and Multiple Samples Our general induction algorithm is shown in Algorithm 3. It uses inducePath from Algorithm 2 as subprocedure, and generalizes our previous considerations to two-directional paths and multiple samples. Two-directional paths are needed if a match node $v \in V$ is not a descendant (and not an ancestor) of the context node u.

(1) *Samples requiring two-directional paths* (Lines 5–13). A two-directional path from u_i to V_i is required, if not all nodes in V are reachable via the same base axis. Then induce searches for the closest node l_i such that u_i and V_i are both reachable via one-directional paths from l_i. This node l_i is the least common ancestor of either V_i or $V_i \cup \{u_i\}$ (Lines 5–7). Once l_i is fixed, induce first computes the best-K tail instances from l_i to V_i. For computing the final result, the algorithm induces instances leading from u_i to l_i, but with an deviating initialization: First, we take standard initialization for *best* (Line 8) but *initialize* $best(l_i)$ *with the tail instances* from l_i to V_i (Line 10). Then, the overall induced expressions lead from u_i to l_i and continue with one of the expressions in $best(l_i)$ to match the nodes in V_i. Second, we set the *targets for computing the accuracy* of the obtained instances to V_i, i.e., we set $tar(n) = V_i$ for all $n \in \mathsf{spine}(u_i, l_i) - \{l_i\}$, since we want to match V_i from each node between u_i and l_i.

(2) *Multiple Samples.* We deal with multiple samples by first computing a best-K set Q_i for each sample $\langle u_i, V_i\rangle$ individually, and second choosing from these instances the ones which perform best on all samples (Line 14).

Site	Induced / Human XPATH queries	valid days	c-changes
S1	`descendant::div[@id="console"]/descendant::p`	400	4
	`descendant::div[starts-with(.,"Top")]/descendant::p`	400	4
S2	`descendant::h3[@class="f-quote"]`	382	16
	`descendant::div[@id="channel0"]/child::h3`	382	16
S3	`descendant::img[@class="adv"][1]` (rank=1)	300	2
	`descendant::div[@class="contentSmLeft"]` `/descendant::img[@class="adv"]` (rank=3)	456	2
	`descendant::div[@class="contentSmLeft"]` `/descendant::img[contains(@class,"adv")]` (rank=5)	1999	34
	`descendant::img[ancestor::div[1][@class="contentSmLeft"]]`	1999	34

S1 = www.foxnews.com, **S2** = espn.go.com, **S3** = www.wellsfargo.com

Table 4.1: Matching single nodes, two typical queries and one difficult case

4.4 Evaluation

Our experiments show that our wrapper induction produces accurate and robust XPATH expressions even from noisy query samples. We first compare our system with two state-of-the-art XPath wrapper induction system [10, 46]. We first evaluate the robustness of our induction system and look at change frequencies occurring on the involved pages (Section 4.4.1). Finally, we point out some characteristics of the induced expressions (Section 4.4.2) and analyze the noise resistance of the wrapper induction (Section 4.4.3).

Running time of the wrapper induction is generally in the same order of magnitude as page retrieval and loading. For a single node expression, it ranges from a few milliseconds to several seconds, with a median of 1.4 seconds and more than 60% of the cases running faster than page retrieval and loading.

4.4.1 Robustness

State-of-the-Art Comparison For robust wrapper induction, [46] presents one of the seminal approaches and includes experiments on IMDB pages. As the system is not available, we rather replicate their experiments as faithfully as possible: as in [46], we take 15 snapshots of director names on IMDB movie pages taken at 2 months intervals. If within such a 2 month interval, no new snapshot is available, we extend the interval until one becomes available, in order to ensure that we use 15 distinct snapshots. We use their success ratio measure as the percentage of snapshots at time t where the induced wrapper still works on the immediately following snapshot at time $t+1$. We tried to approximate the likely time period used in their paper by considering the following three snapshot periods: 2004–2006, 2005–2007, and 2006–2008. Our approach achieves a success ratio of 100%, 86%, 86%, respectively. This compares to 86% reported in their paper [46], where the precise time period is not mentioned. We also compare with a more recent, automatic wrapper induction system [10] (WEIR). WEIR is based on the observation that different sources for a certain type of data often have some overlapping and is able to induce wrappers automatically in

many cases. The trade-off is that WEIR requires multiple pages that follow the same template as input for each source, and each induced XPath expression matches at most one node per page. The induced expressions are generally of two types: absolute expressions that are similar to canonical paths, but starting from the closest anchestor of the target node with a unique ID. The second type of expressions are relative to a close-by "template" node, i.e., a node with a text content that is likely fixed (such as "Director"). To tailor to the limitations of WEIR, we select hotel pages from Tripadvisor with the same template. We let WEIR induce expressions from 10 such pages from 2012, and average over 5 such sets with differing pages and/or target nodes. WEIR exploits the availability of 10 pages to guess which nodes contain static content (such as "Country") and which contain variable values. WEIR's induction returns an unranked set of on average 30 expressions. We compare the robustness of these expressions over the time period from 2012 to 2016 against our system. For our system, we provide only a *single* page with the same target node as for WEIR. The top-10 expressions returned by our system survive on average 67% of the time period, compared with 32% on average for the 10 expressions returned by WEIR. If we compare the most robust expression returned by each system, ours survive for 93% of the period compared to 56% for WEIR. If we only consider the top ranked expression for our case, we survive 92%—our ranking very closely models robustness—of the period. Also note, that our top-ranked and best expressions are robust for the entire period in well over 80% of the cases, compared with around 15% for WEIR.

We evaluate the robustness of the generated expressions in Figures 4.2 and 4.3 respectively for single and multiple target nodes. For both cases, we assembled a data set of 50 test tasks respectively, each specifying a URL u together with a manually crafted XPATH expression q leading to a single node in the first set or multiple nodes in the second set (between 3 and 59 nodes with an average of 10). The sites in the dataset come from over 20 different verticals, such as "Movies", "News", and "Travel", and include some of the most visited websites on the web. The data sets select nodes from a range of different node types arising in the construction of site wrappers: Such wrappers do not only select a single type of data on a page but navigate to the data via forms and links to extract multiple objects which are dispersed over multiple pages. To build such a wrapper, one needs to induce expressions matching form elements, menu entries, next links, and data attributes. Our data sets are assembled to reflect this variety of use cases.

To check the robustness of induced expressions, we obtain from the Internet Archive a sequence of snapshots taken from u. On the first snapshot we induce an expression q' to match the same nodes as q does. The induction is restricted to expressions which do not refer to textual data contents, i.e., texts that are not part of the template but belong to changing data, such as article titles. Such textual content would be too volatile and would cause the induced expression to break quickly.

Once induced, we evaluate each such expression q' on subsequent snapshots of u taken at 20 day intervals until **(1)** q' does not match the same nodes as q, **(2)** q breaks itself, or **(3)** we

reach a given end date. We check whether q is still correctly working by evaluating a pre-specified predicate on the nodes matched of q. For example, such a predicate might check that the matched nodes belong to a certain class or start with certain string. When these predicates did not match anymore, we manually verified that q did break; if q was still valid, we improved the corresponding predicates. If the Internet Archive does not contain a required snapshot, we search for the closest existing snapshot as replacement. Our evaluations start 01-01-2008 and end 12-31-2013.

We compare the robustness of induced wrappers with canonical wrappers that consist of absolute XPATH from root node to target nodes, and human wrappers carefully designed by experienced experts. Note that many human wrappers are *not* in dsXPath using axes such as `following`, see query **S3** in Table 4.2. When designing human wrappers, we aimed at making them as robust as possible, in some cases even rewriting a wrapper that broke by manually inspecting the snapshot.

Results for matching a single node Figure 4.2 shows a density distribution comparing the robustness of induced, canonical, and human wrappers. Intuitively, the higher the curve is towards the right, the more robust wrappers it contains. From the generated 53 expressions, 6 break within 100 days, however, most of these expressions are not robustly wrappable, since in 5 of these cases the corresponding manually crafted expressions fail as well. 27 expressions remain valid between 100 and 400 days, and 20 expressions remain valid for more than 400 days. Thus, discounting the 5 expressions which have no robust wrapper, we only fail in a single case, i.e., our wrapper induction is over 98%-robust.

Table 4.1 shows the induced expressions for three cases taken from our dataset (following the same format as Table 4.2). For sites **S1** and **S2**, we show the top-ranked induced expression in the first line and the manually written expression in the second line; for **S3** we show three induced expression and the manual one. In addition to the number of days the expression remained valid, we also show the number of changes in the canonical path leading to the target node during this period, as a very rough indicator for the amount of changes occurring on the page. **S1** achieves maximal robustness, since the top video is replaced by list of videos and thus it is unclear what the intended selection should be. **S2** also achieves maximal robustness, as the targeted quotation disappears from the site. **S3** is a hard case; our ranking does not work perfectly, since in the top-ranked expression the positional predicate just outruns the expressions involving a second step which is necessary for more robustness. On the other hand, the 5th induced expression would already deliver a very good result – after 1999 days, the involved attributes obtain new names which cannot be anticipated at all.

The **break reasons** for the wrappers fall into five groups:

(a) The first group contains wrapper pairs (induced/human) where both wrappers work over the *full test period* (2008–13), i.e., these are the most robust wrappers. From the 51 wrappers, four fall into this group. For instance, the IMDB search field is selected by this induced wrapper `descendant::input[@name="q"]` and, as the reader may verify, this wrapper still works today.

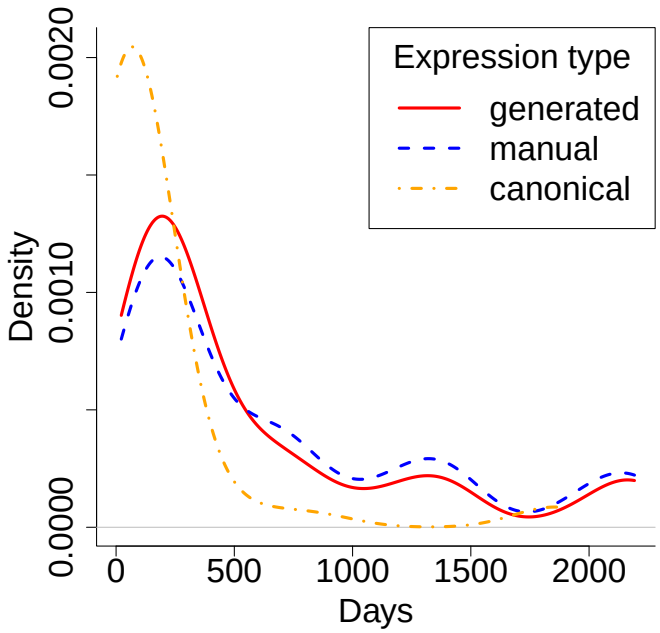


Figure 4.2: Robustness of expressions for matching a single node

(b) For 7 wrappers, both human and induced wrapper fail at the *same* time, yet the target information is still present after the break date. This is typically the case when the markup has changed considerably, causing the wrappers to break. In fact, these changes typically coincide with a site-wide visual redesign. Preeminent for these 7 queries are changes in the values of important attributes: e.g., class attributes change from `"hp-content-block"` to `"homepage-content-block"`, or from `"headline20"` to `"headline16"`, or the id-attribute changes from `"searchInputArea"` to `"searchArea"`. We believe that these cases could be dealt with by incorporating more sophisticated text queries. We leave this issue for future work.

(c) For another 7 wrappers the induced wrapper works *longer*. This may seem surprising, but coincidentally our heuristics chose here an expression that proofed more robust. For example, the induced wrapper (on salesforce.com) $q =$ `descendant::input[@type="text"][last()]` works during the entire period, while the human query `descendant::*[@id="search_box_hm"]`/q breaks after 1.5 years.

(d) Conversely, there are 2 cases where the induced query *breaks before*han the human one: first, the induced query `descendant::div[(@id="cnnT1Col")]/descendant::h1` breaks after 241 days, while the human query `descendant::*[@class="cnnT1Txt"]/descendant::h1` breaks only after 681 days. Coincidentally, in this case the `class` attribute proves to be more robust than the `id` attribute. In the last case, the human query `/descendant::img[@id="jobs"]/ancestor::a[1]` lasts for 1201 days, while the induced query `descendant::a[@href="http://www.jobs.nih.gov/"]` breaks after 1170 days, when a second node with the same `href` value is added to the page.

(e) 7 wrappers break because of *erroneous archive snapshots* which are either empty or structurally broken.

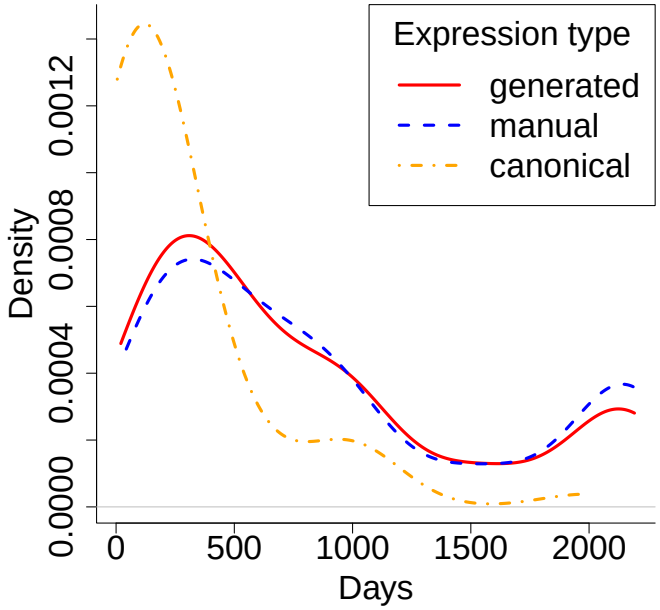


Figure 4.3: Robustness of expressions for matching multiple nodes

(f) 24 wrappers break when the *relevant targets have been removed* from their respective pages; thus, no wrapper could survive such a change. Strictly speaking, these wrappers belong to Group **(a)**, as they did not break in the maximally possible (but not full) time range. Here we categorize them as an independent group to illustrate the fact that *no wrappers* could still work.

Results for matching multiple nodes The results in Figure 4.3 for the expressions matching multiple nodes show similar but slightly better results. For example, only 2 out of 50 expressions are less than 100 days valid, with an average of 815 days and a median of 568 days. It is interesting to observe that in both datasets, after the initial density peaks at 250 (single node) and 350 days (multiple nodes), the density falls until reaching a minimum at 1500 days. Apparently, the cases before 1500 belong to sites which do change significantly eventually, while the cases beyond belong to sites which are never updated significantly at all.

Table 4.2 shows four of our multi-target wrappers. For sites **S1–S3** the top-ranked induced expression is shown in the first line, and the manually written expression in a line below; for **S3** we additionally first show the 49th-ranked induced expression. We show the number of target nodes, the number of days the expression remained valid, and the number of c-changes. The latter is a rough indicator of the amount of changes to the document that directly affect the path from the document root to the target nodes.

S1 matches a list of "channels" which are moved to another page after 219 days. Both the human and generated expression achieves perfect robustness. Note that the induced expression does not rely on textual content but on attributes only. **S2** does not break during the full observation period. The induced expression prefers the shorter string but is otherwise

Site	Inferred / Human XPATH queries		#res	valid days	c-changes
S1	`descendant::a[contains(@class,"hpCH2")]` `/preceding-sibling::a[contains(@class,"hpCH")]`		22	219	1
	`descendant::div[contains(.,"Channels")]` `/descendant::a[@class="hpCH"]`			219	1
S2	`descendant::tr[contains(.,"News")]` `/following-sibling::tr`		7	2056	2
	`descendant::tr[contains(.,"News and` `Latest Reviews")]/following-sibling::tr`			2056	2
S3	`descendant::div[(@class="tvgrid")]` `/following-sibling::node()/descendant::li`	(rank=1)		339	12
	`descendant::p/following-sibling::node()` `/descendant::li`	(rank=49)	8	622	13
	`descendant::p[contains(., "Hit")]` `/following::ul[1]/descendant::li`			700	13

S1 = www.about.com, **S2** = www.mobiletechreview.com, **S3** = imdb.com

Table 4.2: Matching multiple nodes, queries with sibling axes (top-ranked / human)

identical to the manual one. **S3** shows a good but suboptimal result: the `class`-attribute of our top-ranked expression is removed at day 339; in contrast, the manual expression (relying on textual contents) only breaks at day 622. Interestingly, our rank-49 induced expression (using no attributes or text) holds even for 700 days.

We checked by hand the **break reasons** of the wrappers. This time, no induced wrapper ever worked longer than the human made, so we have only four groups: **(a)** 6 wrapper pairs work over the full period, selecting general lists such as "latest news" items. **(b)** 10 wrapper pairs break at the same time. **(c)** There are no cases where the induced wrapper works longer than the human one. **(d)** 3 induced wrappers break before the human one. For example, the induced top-ranked wrapper `descendant::div[@class="widePanel"]/descendant::a` breaks after 817 days, while the human wrapper is robust for the entire 5 year period. However, the 30th ranked induced expression is fully robust too: it first selects the text `"offers:"` from a header and then continues with `/following-sibling::node()/descendant::a`. Another example is **S3** from Table 4.2, where the human selects a node containing the text `"Hit"`, while the induced query uses a `class` attribute. **(e)** 10 wrapper pairs break due to issues with the internet archive. **(f)** 21 wrapper paris break due to diminishing targets. Often, the removal of the list information co-occurs with a complete visual redesign of the respective site.

Both examples of induced wrappers that fail before the human ones suggest that selection through the text-value of nodes gives more robust wrapper than through attributes. This is confirmed by analyzing the 10 cases where induced and human wrappers break simultaneously: often the only possible robust approach is based on selecting keywords within the target list. We believe that our approach can induce robust queries for these cases; however, as we mostly target volatile data entries, our configuration is biased to deal with (relatively) dynamic item lists.

Change Rate Here our discussions will be for both, single- and multiple-target wrappers. We use a very rough change measure: only if the *canonical paths* to the target nodes change, do we count this as one change. Thus, if such a "c-change" occurs, then we increase the change counter, induce a new path, and continue in the same way. Recall that the canonical path consists only of element names and position numbers. In this way, we obtain *very low* change frequencies, the largest being 25 (obviously, many more things change on the pages, even considering attribute values on the canonical path gives completely different numbers). Since the canonical path is our yardstick for a simple wrapper, we find c-changes an appropriate measure. For single-target wrappers, 11 wrappers survive more than 5 c-changes; 16 wrappers survive exactly 1 c-change (that being the largest group). The average is at 4.1 c-changes. For multiple-target wrapper, 13 wrappers survive more than 5 c-changes, but the highest number is only 19 (versus 25 for single). Interestingly, the average is again 4.1 c-changes. We only checked two sites (IMDB and espn) by hand to see what kind of canonical path changes occur. The most frequent here, were positional changes on div nodes. For instance, in a typical path (from IMDB) `html[1]/.../layer[1]/div[1]/div[3]/div[4]/form[1]/input[1]` the `div[4]` becomes a `div[3]` in a new version. The second-frequent changes are inserted or removed `div` nodes on the canonical path (i.e., changing the length of the path).

4.4.2 Expression Characteristics

Parameter Choices As explained in Section 4.2, the typical expression characteristics are chosen as suggested by our experience, as well as previous literature on wrapper induction, e.g., that **`descendant`** is preferred over **`child`**. These are further discussed below. The most important other parameter of our approach is the *decay factor*, which favors anchors further away from the targets. Our decay factor is $\delta = 2.5$, as was determined as optimal through a sequence of experience varying delta between 0.5 and 5.

For single- and for multiple-target wrapper induction we use the *same* ranking parameters. We use *no* specific scores for different tags, but merely $c_{\text{node()}} = c_* = 1$ and $c_{default} = 10$. Our positional factor is 20, *no-function-penalty* is 15, and *no-predicate-penalty* is 1000. All other parameters are as follows:

Axes		Attributes		Functions	
descendant	1	id	1	equals	1
attribute	1	type	1	position	1
foll.-sibling	1	title	1	contains	5
child	10	class	5	starts-with	5
parent	10	for	10	norm.-space	5
ancestor	20	name	50	last	20
prec.-sibling	25	*default*	1000	string	100

As an example, for **`descendant`**`::img[@class="adv"][1]` we compute as score $c_{\text{descendant}} + c_{default} = 1 + 10 + 1 = 11$ plus the two predicates. The first one gives $s_{\text{class}} = 5$ plus $1 \cdot 3$, and `[1]` gives 21. The final score equals 40.

Analysis Starting with the 53 expressions generated to match a single node, we see that 34 of these expressions have 1 step, 19 expressions have 2 steps, amounting to 72 total XPATH steps. All of these steps follow the `descendant` axis. Figure 4.4 gives some details on the characteristics of these expressions: On the left the figures shows that 26 of the 72 steps check for `div` elements, followed by `input`, and `a` elements. Of these 72 nodetests, 62 are refined by at least one predicate, and 10 are refined by two predicates, yielding again coincidentally 72 predicates. As shown in the right of Figure 4.4, these predicates check in 26 cases for an `id` attribute, in 18 cases for an `class` attribute, and in 17 cases for a position. In contrast, textual contents are only checked in 2 cases. This data suggest the following pattern: If the expression contains two steps, the first step identifies the region containing the targets by typically matching a `div`, `input`, or `a` node, predicated with an `id` or `class` attribute or position. The second step matches the target node, referring to a wide range of different tag names and often checking for its position within the context.

The expressions to match multiple nodes have quite different characteristics than the ones for single nodes. Fist of all, only 9 out of 50 examples use a single step, 34 employ 2 steps, and 7 need 3 steps, yielding a total of 98 steps. These steps employ a bigger variety of axes, although 88 still rely the `descendant` axis; 7 steps use `following-sibling`, 2 step use `preceding-sibling`, and 1 step `child`. The use of the sibling axes is at first surprising, as most useful node lists are grouped below a common ancestor. However, to identify the correct subset of siblings belonging to our target list to avoid, e.g., adverts at the beginning or end, robustly matching lists require sibling anchors. For instance, query **S3** in Table 4.2 selects the first ul-sibling that follows the h3-node, and similarly, query **S4** (at rank 49) select all list items under following siblings of a p-node. The nodetests, shown on the left of Figure 4.5, also check in 33 cases for the `div` tag. However, the next nodetests check for unordered lists and table entries with `ul` and `td` as tags containing or neighboring the target nodes. Finally, the 27 steps checking for list items with the `li` occur mostly in the second and third step to match the sequence of target nodes. The right of Figure 4.5 shows the the distribution of the 62 predicates occurring in the induced expressions. These predicates mostly check for `id` and `class` attributes. However in contrast to the single-target cases, the expressions for multiple targets do not check for any other attribute; aside these, only 7 expressions check in the first step for textual contents, and only 5 expressions check in their second step for positions.

4.4.3 Accuracy and Noise Resistance

Real-Life Noise In this experiment, we verify that indeed our approach is able to deal with realistic noise as produced by a real-life entity recognizer. We use the Stanford NER [75] to annotate 10 pages randomly sampled from a set of product listing websites (cf. [17]). We only select pages that contain at least one list of entities supported by the Stanford NER, here one of date, person, location, organisation, money. The size of the lists vary between 8 and 77 elements. For each of these pages, we run the Stanford NER and map its annotations to DOM nodes that serve as target nodes for our approach. The resulting input contains

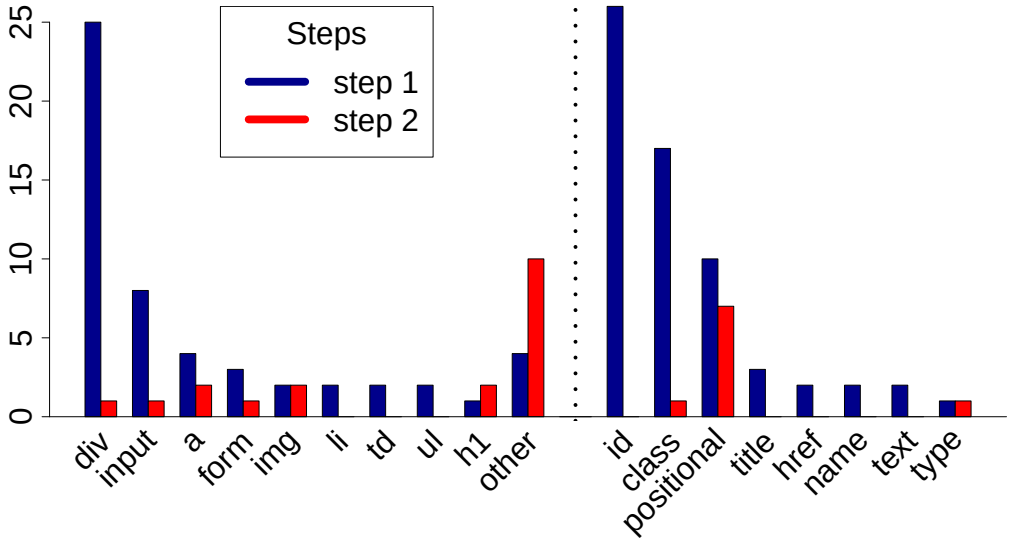


Figure 4.4: Nodetests/predicates of single-target queries

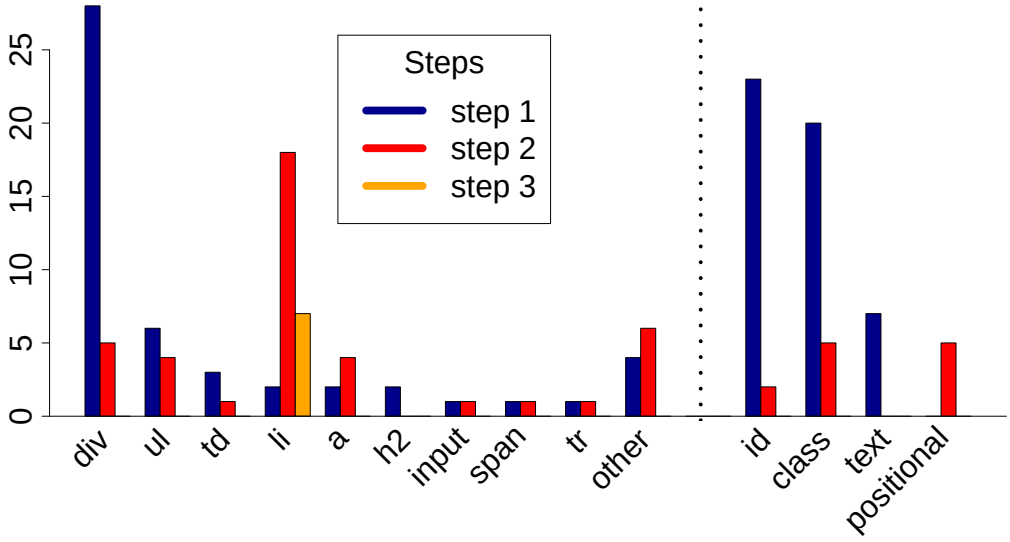


Figure 4.5: Nodetests/predicates of multiple-target queries

on average 32% negative and 28% positive noise, though both vary widely: $0-67\%$ for negative and $0-145\%$ for positive noise, with each page having at least some noise.

Our approach deals surprisingly well with these noisy samples: in 80% of the cases, our top-ranked expression identifies the correct intended set of nodes, ranging from dates, book authors, artists, and event locations to prices. There are two cases where it fails. **(1)** For "organisations" on newyorknewyork.com, the NER returns 145% positive noise, yet misses 55% of the sought-for target nodes. The returned expression is overgeneralized to a significantly larger set of nodes. **(2)** For "persons" on waterstones.com, the NER returns 50% positive, and 28% negative noise. The target nodes are authors of books returned by a search. Unfortunately, most of the positive noise is structural, annotating a list of author names in a sidebar list used to refine the query. The generated expression selects that list rather than the intended one. Both of these cases could be addressed by limiting the induction to the main content area, obtained by removing boilerplate content [76]. However, this is out of the

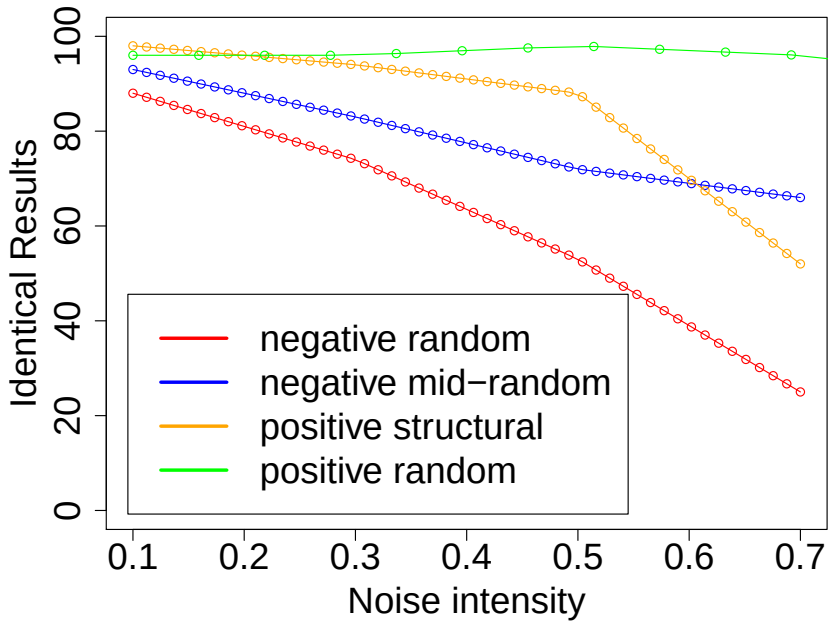


Figure 4.6: Result degradation with increased presence of noise

scope of this thesis. It is worth pointing out, that even if we eliminate the two cases where the system fails, noise remains significant (30% negative and 11% positive on average), yet in all of these cases our method compensate flawlessly for that noise.

Synthetic noise Here we evaluate noise resistance by adding or removing a percentage of the target nodes of a sample. We evaluate different four types of noise: **(N1)** *negative random noise* **(N2)** *negative mid-random noise* where the first and last nodes (in document order) of a target set are not removed, **(N3)** *positive structured noise* where we add random nodes chosen from a node set which is structurally related (via an XPATH expression) to the target nodes. Finally, we consider **(N4)** *positive random noise* where we add random leaf nodes to the target set. We use two datasets: The first one for negative noise has 100 query samples matching between 3 and 59 nodes, with a median of 6 and an average 9.43 nodes. The second one for positive noise features 50 query samples matching between 2 and 100 nodes, with a median of 20 and an average of 34.92.

Figure 4.6 summarizes our results for all noise types. We show for the noise intensities of 10, 30, 50, and 70% the percentage of cases where the induction with and without noise delivered identical results. Thus, we evaluate the noise resistance in the most aggressive way. Typically, with negative noise, around 10% of the cases deliver with noise a result which is at least within the top 50 results induced without noise. With positive noise, the fraction of such expressions is negligible. Our system deals well with **(N1)** negative random noise at 10 and 30% intensity in achieving *identical* results in respectively 88% and 74% of the cases, thus showing that our approach is 88%- and 74%-noise resistant. However, at 50 and 70% intensity, 38 and 70% of the cases deliver non-identical results. Seemingly

unsatisfying, this relatively low noise resistance is caused by removed head and tail nodes – which are critical to determine the start and end of a sequence of list entries. Therefore, we consider **(N2)** negative mid-random noise where we keep the very first and last node (in pre-order) and choose the noisily removed nodes only from the remaining nodes in between. In this case, our system returns identical results in 93, 83, 72, and 66% of the cases for the analyzed intensities. Our system deals exceptionally well with positive noise. For **(N3)** structural positive noise until 50% intensity, our system achieves in at least 90% of the cases an identical expression with and without noise. At 70% intensity however, this percentage drops to 54%. For **(N4)** we see almost no effect on the results, e.g., at 70% noise intensity, our system delivers the same result as without noise in 96% of the cases. Even at a noise intensity of 300%, we obtain identical results in 84% of the cases.

Chapter 5

Unsupervised Data Record Extraction from Result Pages

In the scenario where the user's data demand is clear, data extraction should be completely unsupervised. In this chapter, we present our solution to a typical case in such scenarios, i.e., the problem of extracting data from result pages. We propose a new method, called RED, that exploits the publishing pattern of result/detail pages, and instanced in Figure 5.1, for the pages of a fictional real estate website that will be used as a running example throughout this chapter: the contents published on result pages and detail pages usually overlap, i.e., the object published in some result record is also published in a corresponding detail page, and the attributes published in the result record are also published in its corresponding detail page. Existing unsupervised approaches overlooked this indication of data and purely rely on HTML document analysis. RED combines HTML structure analysis and such content redundancy between result pages and detail pages to achieve a significantly higher accuracy.

5.1 Problem Description

We first describe the *result/detail publishing pattern*, and then we formulate the problem of recovering the values of the attributes that are redundantly published in two given collections of detail/result pages from a website obeying to that pattern.

The result/detail publishing pattern is depicted in Figure 5.2. It can be described by assuming that a website publishes data coming from an underlying abstract relation with attributes $\mathcal{A}$ and containing all the available information about a set of objects $\mathcal{O}$. In our running example the abstract relation contains several tuples, one per every property whose information are published, and attributes such as Price, Address, Beds and so on.

Result pages and detail pages are generated by several *script*s that are orchestrated for retrieving data from the abstract relation and then to publish them as HTML pages. During the publishing process, they operate a sequence of transforming operations: we assume that a first selection operation (σ) produces the set of objects published in a single result page as encoded by the *result page script* λ. Every produced result page contains a set of *result*

Search results:

- **Featured!** **1.5k** pcm; **House** *at* **W2U 0DF London**; 3 **Beds**
- **1.5k** pcm; **Flat** *at* **W3U 1AF London**; two **Beds**
- **!Hot!** **Studio** *at* **W3U 0AF London**; 2 **Beds**
- **2.05k** pcm; **House** *at* **E15 2CD Oxford**; 3 **Beds**
- **1.2k** pcm; **Ensuite** *at* **E15 2XD Oxford**; 1 **Beds**
- **Single Room** *at* **E18 2XD Oxford**; 2 **Beds**

(a) A result page.

Featured!

Property type: House
Location: **W2U 0DF London**
Listing Price: **1,500** pcm
A 3 bedroom house located in Worcestershire...

(b) Detail page 1.

Property

Property type: Flat
Location: **W3U 1AF London**
Listing Price: **1,500** pcm
A 2 bedroom flat...

(c) Detail page 2.

!Hot!

Studio

Property type: Studio
Location: **W3U 0AF London**
A 2 bedroom studio with a nice view...

(d) Detail page 3.

Figure 5.1: Running example pages.

records, each produced by the *result record script* λ^r and containing data for one the published object. The contents of result and detail pages are strictly related each other: indeed, every record contains a link to a detail page that, similarly, is produced by applying a distinct script, the *detail script* λ^d, on the same object.

Although the detail and result record scripts works on the same set of objects, according to the model depicted in Figure 5.2, we assume that they end up publishing different sets of attributes. The detail pages publish all the attributes in the abstract relation, whereas the result pages usually publish only a subset of its attributes (as selected by a projection operation π).

We now introduce the problem of recovering the values of the attributes which are published in the result pages. Actually, we can only hope to recover their serial encoding as string values in the pages, either from the result records, or from the detail pages, as located by a suitable *extraction rule* (or simply rule). A rule can locate one or several string values from a single page, or it can produce a distinct special value *nil*. For our purposes, extraction rules can be expressed by using XPath expressions taken from a simple fragment (as detailed in Section 5.3) of XPath query language over XML/HTML documents. According to the specification of the XPath language [77], the result of evaluating one XPath expression on a page can be interpreted in several ways, but we just focus on the *node-set* and *string-value* interpretations. An XPath expression evaluated on a page produces a node-set, that is, a set of nodes in the DOM tree representation of the page [78]. A sample DOM tree for the result page of our running example is depicted in Figure 5.3a. The string-value interpretation of a text node is trivially its text value.

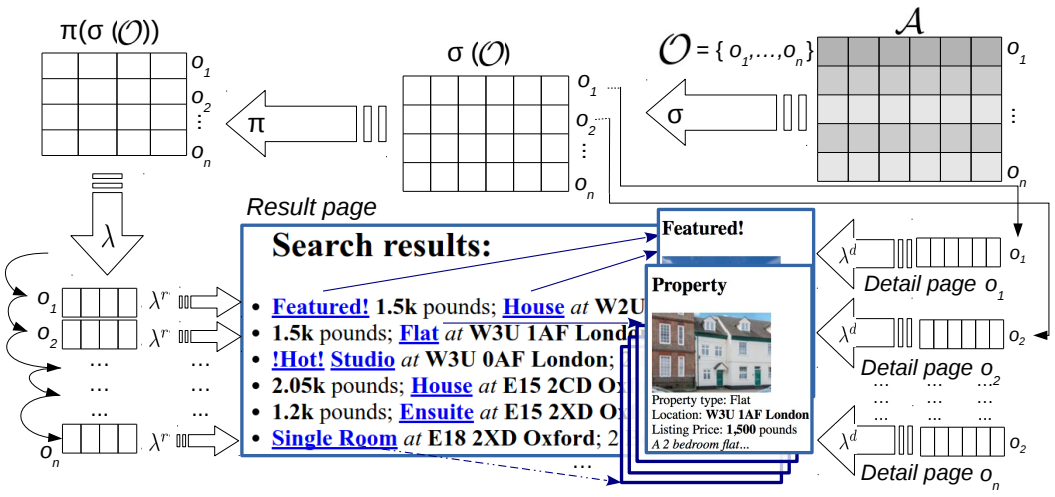


Figure 5.2: A model describing the generation process over the running example.

We distinguish two types of extraction rules accordingly to the type of pages they are meant to work on.

We call *detail rule* any extraction rule that if applied on a detail page, produces at most a string value or *nil*. We prefer to use d to denote a detail extraction rule, and we use $d(p)$ to denote the value it extracts from detail page p. Namely, $d(p)$ is either the XPath *string-value* obtained by applying the rule d on detail pages p, or it is *nil* if the XPath expression returns an empty node-set.

Correct/Noisy detail rule We call *correct* a detail rule for an attribute A such that it extracts the encoding of the value of the attribute A from every detail page, or it just extracts *nil* if that page does not provide a value for that attribute. We call *noisy* (or *incorrect*) any rule that is not a correct extraction rule: it may mix values of different attributes from page to page, or it might just include values which are not encoding of any attribute but are instead part of the underlying HTML template. Among not correct rules, we distinguish the special case of an *incomplete* rule for an attribute A, i.e., a rule extracting only correct values of attribute A but for some pages, on which it wrongly extracts a *nil*. Thus, an alternative view of an incomplete rule that motivates a different name, is as partially correct rule, at least at the level of the set of values it can extract.

Similarly we call *result rule* any extraction rule meant to work on result pages and we prefer to denote it by r. A result rule applied on a result page produce zero, one, or several string values. Our XPath based result rules map each node in the node-set returned into a distinct value, by taking the *string-value* of the node. It is crucial to note that *nil* values are not produced at all (even in the case that such a node-set is empty).

Correct/Noisy result rule Unfortunately, the correctness of result rules is trickier to define because differently from a detail rule, a result rule can extract multiple values from a single result page, not at most one.

Price	Type	Address	Beds
1.5k	House	W2U 0DF London	3
1.5k	Flat	W3U 1AF London	two
nil	Studio	W3U 0AF London	2
2.05k	House	E15 2CD Oxford	3
1.2k	Ensuite	E15 2XD Oxford	1
nil	Single Room	E18 2XD Oxford	2

Table 5.1: Correct result records.

Example 5.1.1. *Consider the result page shown in Figure 5.1a; its DOM tree is represented in Figure 5.3a.*

A result rule extracting some Price *values is* r_4 = //span[@class="price"]/*/text()[1] *which is one of the result rules shown in Figure 5.3c. Unfortunately, neither the rule extracts all price values, nor there exist a single XPath expression able to extract all the* six *price values including the nil values.*

The root of the issue is that data from result pages are organized according to a nested structure (every result page contains a *list* of records, each containing a tuple) that is intrinsically more complex than that of detail pages (each detail page contains a *single* tuple): Extraction rules based on one XPath expression can easily deal with the detail page case, whereas the "nested" structure of result page does not couple well with any single-step extraction process based on a single XPath expression. A two steps process would be much more appropriate for result pages: first, every result page should segmented by finding the result records, and then from every record at most one value should be extracted. Unfortunately the segmentation of generic result pages into result records is a quite complex problem by itself, it has already motivated dedicated approaches [11,79], and any self-contained solution is at risk of introducing noise in the extraction process. Our approach does not consider any two step extraction process by leveraging the redundancy with detail rules as described in the Section 5.2.

Therefore, a *correct* result rule for attribute A is a result rule such that if applied on any result page p, it produces exactly the set of (encodings of) non-*nil* values of attribute A in that page. The concept of *noisy* result rules can be easily adapted from that of noisy detail rules, while by *incomplete* result rule for an attribute A we mean a noisy result rule extracting a strict subset of the correct values of A from result page p.

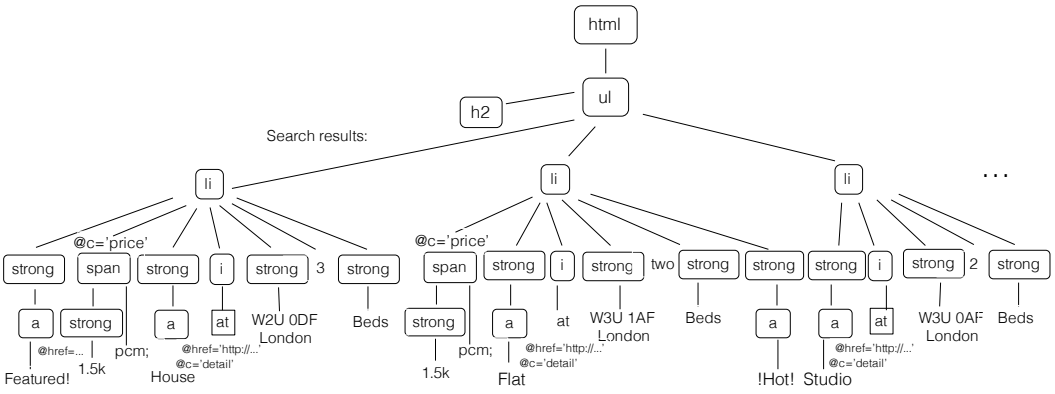
The problem that we solve can then be formulated as follows.

Problem 5.1.1 (Finding Result Attributes)**.** *Given a set of result pages P_r and the corresponding set of detail pages P_d over the same set of objects, find the correct values (including nil) for every attribute A of the abstract relation published in the result pages.*

Our proposed technique, RED, tries to solve this problem by producing as output a bunch of result rules, one for every attribute A, together with an additional complete extraction result rule, called *link rule*: every output result rule r_A is suitably annotated (either r_A^a or

r_A^b) to determine how to use the link rule to inject the missing *nil* values between the values extracted by r_A.

Designing a completely unsupervised solution to this problem is challenging for several reasons: first, the result pages and the detail pages follow a different template and are organized according to different organization of the data; second, there are irregularities in the templates, such as those due to the presence of optional attributes with *nil* values that are often encoded just by skipping whole fragments of the template in the corresponding result record and detail page; finally, several attributes may assume quite similar values and can be enconded, in the HTML source, close to each other, to the extent that is not simple to recognize which value correspond to which attribute.



(a) A portion of the DOM tree of the result page.

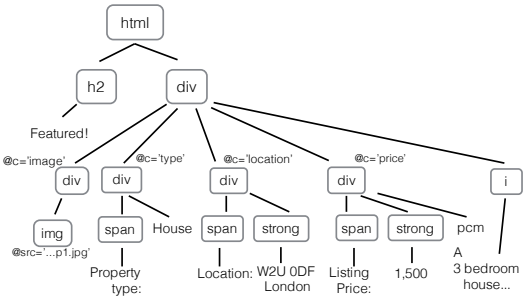
class	*nodes*	*occ. vector*
ε_0^r	{LI, I, at, A[@class = "detail"], Beds}	[6]
ε_1^r	{HTML, UL, H2, Search results:, ...}	[1]
ε_2^r	{SPAN[@class = "price"]/STRONG, pcm;}	[4]
ε_3^r	{STRONG}	[12]
ε_4^r	{House, A, 1.5k, W3U, E15, 2XD}	[2]
ε_5^r	{Oxford, London}	[3]

(b) Template analysis of result pages.

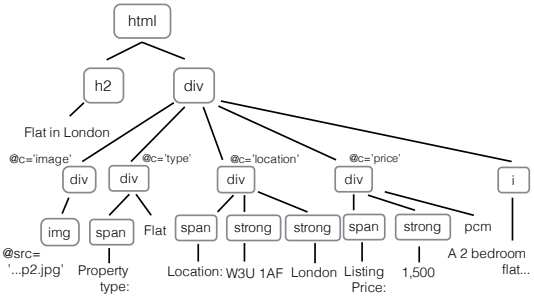
r_0 : //text()[contains(.,"pcm;")]/../*[1]/text()[1]
r_1 : //li/*[1]/*[1]/text()[1]
r_2 : //i/ps::*[1]/*[1]/text()[1]
r_3 : //span[@class="price"]/../*[1]/*[1]/text()[1]
r_4 : //span[@class="price"]/*/text()[1]
r_5 : //li/*[4]/text()[1]
r_6 : //i[contains(.,"at")]/fs::*[1]/text()[1]
r_7 : //i[contains(.,"at")]/../*[4]/text()[1]
r_8 : //span[@class="price"]/fs::*[3]/text()[1]
r_9 : //strong[contains(., "Beds")]/ps::text()[1]
l : //a[@class="detail"]

(c) Examples of result candidate rules.

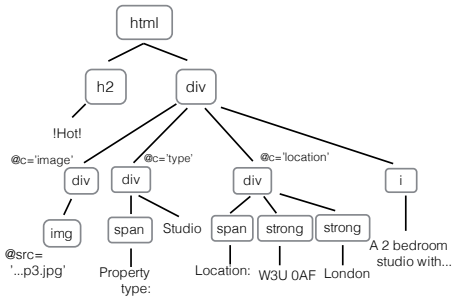
Figure 5.3: Running example result page.



(a) DOM tree of the detail page 1.



(b) DOM tree of the detail page 2.



(c) DOM tree of the detail page 3.

class	*nodes*	*occ. vector*
ε_1^d	{DIV[@class = "price"], DIV[@class = "price"]/SPAN, DIV[@class = "price"]/STRONG, Listing Price:,pcm;}	$[1,1,0,1,1,0]$
ε_0^d	{Location:,Property type:, *...all other tags*}	$[1,1,1,1,1,1]$
ε_2^d	{House}	$[1,0,0,1,0,0]$
ε_3^d	{1,500}	$[1,1,0,0,0,0]$
ε_4^d	{London}	$[1,1,1,0,0,0]$
ε_5^d	{Oxford}	$[0,0,0,1,1,1]$
...	...	...

(d) Template analysis of detail pages.

d_0 : //span[contains(.,"Listing Price:")]/../*[1]/text()
d_1 : //span[contains(.,"Listing Price:")]/../ps::*/*[1]/text()
d_2 : //h2/text()
d_3 : //span[contains(.,"Listing Price:")]/../fs::*[1]/text()
d_4 : //span[contains(.,"Location:")]/fs::*[1]/text()
d_5 : //span[contains(.,"Listing Price:")]/ps::*[1]/text()[1]
d_6 : //i/text()
d_7 : //span[contains(.,"Location:")]/fs::*[2]/text()

(e) Detail candidate rules.

Figure 5.4: Running example detail pages.

5.2 Overview of the Solution

We overview RED, our algorithm for solving the *Finding Result Rules problem* by explaining how it works over the running example. RED takes as input a set of result pages P_r and the corresponding set of detail pages P_d, and produces as output a collection of result extraction rules. For instance, consider the result page shown in Figure 5.1a and the detail pages shown in Figures 5.1b, 5.1c, and 5.1d, for our running example. RED is capable of generating result rules extracting the attribute values of *six* result records on the result page, as shown in the table of correct records reported in Table 5.1.

RED can be described as a pipeline of four main processing steps: (i) candidate rules generation; (ii) redundancy finding; (iii) rules validation; (iv) noise removal.

During the first step, RED generates the candidate extration rules for result pages and for detail pages. The extraction rules generation algorithm needs to be *complete* both for result pages and for detail pages to solve the finding result rules problem, that is to say, it should generate at least one correct detail rule and one correct result rule for every attribute in the result pages. On the one hand, this can be easily accomplished by generating extraction rules taken from an expressive enough XPath fragment [43]. On the other hand, by using a too expressive XPath fragment, two important drawbacks might arise: first, it becomes increasingly likely that many additional useless noisy rules are generated, thus complicating the next processing steps that should separate the good redundancy (related to correct extraction rules) from the weak one (involving noisy rules); second, the efficiency could deteriorate since it clearly depends on the number of rules that are processed. Figure 5.3c and Figure 5.4e respectively show (a subset of) the result and detail rules generated by our algorithm on the running example.

In the *redundancy finding* step, RED aims at selecting only the pairs of *redundant* result/detail rules from the collection of pairs of rules received from the previous step. By redundant rules we mean those extracting the same values for the same object. This step is not trivial, for several reasons: it requires to align the possibly different number values extracted by the two rules, respectively one detail rule and one result rule, then matching their values, and finally, there is the underlying inherently difficult and errors-prone problem of segmenting the result page into separate records.

RED adopts an innovative method that leverages intra-site redundancy between result and detail pages: it aligns the values extracted by the result rule with those extracted by a detail rule by analyzing the positions of the nodes extracted by the former wrt the detail link nodes, i.e., the nodes of the result record containing links to the corresponding detail pages. It turns out that given a result rule r, there exist at most two possible ways (denoted r^a and r^b) to interleave the nodes it extracts with the detail links, and that the wrong choice produce values that are unlikely to match by chance with values extracted by any (correct or incorrect) detail rule. Therefore, in this step we compare every result rule against every

Algorithm 4: RED

Input: Sets of result/detail pages P_r/P_d associated with the same objects.
Output: The detail links result rule l.
Output: Set of annotated result extraction rules $\{r^x : x \in \{a,b\},\ r \text{ is a result rule}\}$;

```
1  begin
2      R_r ← GENERATERULES(P_r);
3      R_d ← GENERATERULES(P_d);
4      l ← FINDDETAILLINKS(P_r, P_d);
5      R'_r ← {ALIGN(r, l, P_r), r ∈ R_r};
6      Π ← { (r', d) ∈ R'_r × R_d : REDUNDANT(r', d) };
7      Π ← VALIDATION(l, Π, P_r);
8      Π ← REMOVEREDUNDANTNOISE(Π);
9      return l and {r^d : (r, d) ∈ Π};
10 end
```

detail rule considering at most two possible interleavings of the values, and select only pairs of rules extracting values that better match with those extracted by the detail rule.

Table 5.2 reports for our running example all the pairs of rules ordered by a similarity score between rules that we define in Section 5.4.4 and scoring less than 40% (zero-score means perfect redundancy).

The *validation* step discard result rules that appear to be ill-conceived according to an analysis of how the extracted values interleaves with the values extracted by other result rules.

The last step, *noise removal*, deals with the problem of selecting, among all the pairs of redundant rules produced by the previous step, those actually associated with rule correctly extracting the redundant attributes values so as to discard any pair of redundant pairs involving noisy rules. Several filtering techniques are used: first, rule pairs are grouped together by checking whether they end up extracting at least one common value occurrences; then only the best pairs, one per cluster, are saved. Eventually, to break ties on the remaining pairs, we also count the number of *nil* values extracted by the detail rules, preferring those with fewer *nil* values to privilege correct over incomplete rules. Table 5.3 (at pag. 67) shows the pairs of rules for our running example after the filtering operated by this step: all the result rules in this table are correct and extract the values shown in Table 5.1.

Finally, RED outputs the result extraction rules of the redundant pairs, together with the *detail link result rule* (or shortly *link rule*), i.e., a result rule extracting one node containing a link to the corresponding detail page from every result record. Every result rule r is annotated either r^a or r^b to specify whether the missing *nil* values[1] has to be inserted *after* the detail link nodes or *before them*. In this way, the downloading of detail pages can be saved, which is desirable for large-scale extraction tasks or any other task where bandwidth consumption is one of the considered cost factors.

[1] Remind that result rules cannot extract *nil* values.

Algorithm in Listing 4 shows an high-level description of RED: the technique adopted by the candidate rules generation step (lines 2-3) are described in Section 5.3 together with the inference of the detail links result rule l (line 4).

The extraction rules are then processed to find redundant pairs of (result and detail) rules (lines 5-6), as described in Section 5.4. RED improves its precision by means of a validation technique (lines 7-8) over redundant pair of rules described in Section 5.5.

5.3 Extraction Rules Generation

We introduce an extraction rule generation algorithm specifically designed for our setting. Namely, we designed an algorithm to fulfill the following requirements: *universality* – it aims at generating extraction rules both for result pages and for detail pages, even if these two classes of pages obey to different HTML template; *completeness* – it has to generate at least one correct result rule and one detail rule for every redundant attribute; *controllable expressiveness/noisiness* – the expressivity of the XPath fragment considered should be configurable to generate as many rules as need for the fragment to be complete (see previous requirement), but at the same time it should not generate too many noisy rules because that might complicate the following processing steps by injecting undesired incidental redundancy.

RED data extraction is based on a new rules generation algorithm that fulfill all these conflicting requirements, both on result and on detail pages. It is a single-parameter algorithm that can be easily configured to get a working trade-off between the expressivity and the completeness of the generated class of extraction rules in any practical scenario. The rules generation algorithm includes these steps: (i) template analysis; (ii) rules generation; (iii) values extraction, as detailed in the following.

5.3.1 Template Analysis

Given a set of pages sharing a common layout, the *template analysis* step aims at classifying every node in the DOM tree representation of the input pages as either *template* node or as *value* node.

By value node we mean a text node, containing the value, e.g., '`W20 0DF`' of an attribute, e.g., PostCode, as encoded by a script of our generative model after querying an underlying database. Conversely, by template node we mean a node that is part of the template and therefore it has been introduced in the final HTML source code directly by the generating script, without querying an underlying database, such as the text nodes with content of "at" in Figure 5.3a. Value nodes are associated with a specific object of which they represent an attribute value; on the contrary, the template nodes are not directly associated with any published objects, but rather with their HTML formatting/structure as shared by all the pages generated by the script.

In practice, the vast majority of element nodes belonging to the DOM tree of pages from a templated website are template nodes, whereas text nodes are most likely value nodes. Unfortunately, this is not always the case, and there are, for example, textual labels, e.g., 'Price:', that are part of template rather than values. There are (quite rare) element nodes that should be considered as value nodes, as well: for example, a long textual description could contain formatted text by making use of <U>, <B> element nodes that should be considered as value nodes rather than as template nodes.[2]

The precise analysis and inference of a formal description of the HTML template underlying a given collection of pages remains a complex problem, considering that in the most general setting, the data could be arranged according to a schema of arbitrary complexity, including tuple constructs, optional attributes, and arbitrarily nested lists.

Fortunately this problem has been already extensively studied in the literature [80] and for our own purposes, we just need to solve the much simpler problem of classifying every node as either template or value node without inferring a precise description of the HTML template and its structure, which is essentially given by hypothesis in the result/detail publishing pattern considered. We adopt an algorithm inspired by [2] but specifically tailored to the publishing pattern of our interest.

Given a set of pages P (they are either the set of input result pages P_r or the set of input detail pages P_d), a preprocessing step builds their DOM representation. Then, every node of the DOM tree is associated with an *occurrence-vector*.[3]

An occurrence-vector is a vector f of $|P|$ integers indexed by the pages in P, where $f(p)$ reports how many occurrences of *equivalent* DOM nodes are present in page p. Two nodes are considered *equivalent* iff two conditions hold: (i) they are either leaf nodes with the same textual value, or they are element nodes associated with the same element name;[4] (e.g., <TD>, <LI>); (ii) their respective parent nodes are equivalent, or they both have not a parent node.

As observed in [2], by considering a sufficiently large number of pages, the nodes that occur in large and frequently occurring classes are the scaffold of the underlying template: based on the occurrence-vectors, we can group the nodes into equivalence classes composed of nodes associated to the same occurrence-vector.

Namely, we consider as template nodes all those occurring in the equivalence class whose occurrence-vector f is such that $f(p) \leq |\mathcal{O}(p)|$, $\forall p \in P$, where $|\mathcal{O}(p)|$ is the number of objects which have been published in the set of pages P, an information which is given as input in our problem formulation (it suffices to count the number of detail pages corresponding to

[2]The formatting tags are most likely stored in the value of the corresponding tuple in the underlying database rather than being "hard coded" in the publishing script.

[3]Actually our implementation adopts a basic tokenization algorithm to split text node into several word-level tokens and perform the template analysis at this fine-grained level of granularity. For the sake of simplicity we blur this detail in the following.

[4]Actually our prototype can be configured to also consider the attribute names and values. Our default configuration take into account all the attribute names but bgcolor, and also considers class attribute values. We also ignore the part of the DOM tree rooted in elements with certain names, e.g., <METADATA> and <SCRIPT>.

each result page). We call *root* equivalence class that having an occurrence vector $f(\cdot)$ such that $f(p) = |\mathcal{O}(p)|, \forall p \in P$.

We classify as template nodes only the nodes of equivalence classes occurring in every pages at most as many times as the number of occurrences of the root equivalence class, i.e., at most once for detail pages, and at most as many times as the number of result records for result pages. All the other textual nodes not occurring in the inferred equivalence classes are considered value nodes. We set a minimum number of required occurrences (support) and of nodes (size) of the equivalence classes to avoid to consider too many classes, which becomes more and more noisy as their support and size decrease. [5]

Example 5.3.1. *Our running example includes a collection of six detail pages and one result page, the latter shown in Figure 5.1a. For the sake of brevity we just show 3 detail pages in Figures 5.1b-5.1d. The other detail pages not shown here can be considered as having characteristics perfectly in line with these pages. The equivalence classes for the collection of six detail pages in the running example can be deduced by counting the nodes in the DOM tree of the pages shown in Figures 5.4a-5.4c. Part of these are shown in Figure 5.4d. As for the result page, the DOM tree is shown in Figure 5.3a and all the equivalence classes having at least two occurrences in the page are shown in Figure 5.3b.*

Notice that all the occurrences of nodes from the root equivalence classes, i.e., ε_0^r and ε_0^d, are correctly considered as template nodes. Similarly, nodes from ε_1^r (each with just an occurrence: less than 20% of 6 occurrences required) and ε_3^r (it has more than 6 occurrences) are correctly not considered as template nodes. However, nodes from ε_4^r, and ε_j^d, $j \geq 2$, would be erroneously considered as such (they are rather node values that by chance occur more than once).

5.3.2 Rules Generation Algorithm

The output of the first phase of the extraction rule generation algorithm consists of two sets of nodes for every page: the set of template nodes (either elements or texts) and the set of value nodes (only texts). For every generated rule, one node from the former set will be chosen to play the role of *pivot*, so as to find in the pages one node from the latter set, playing the role of *value* to be extracted.

A pivot node is a template node whose occurrences can be easily and univocally located in every page by a suitable XPath expression based on that pivot node itself. An XPath based extraction rule is then generated to match a path reaching a target value node in the DOM tree starting from an occurrence of the pivot node.

The next step, *enumeration of tree-paths of bounded length*, takes every node from the set of template nodes as a candidate pivot, and tries to enumerate all the paths over the DOM trees that lead to one nodes in the set of values. In order to contain the number of tree-paths

[5] We consider only equivalence classes whose total number of occurrences is at least 20% of the total number of input detail pages, but never fewer than 2. We consider only classes including at least 3 tokens.

enumerated, several precautions are put in place. First, we limit the path length to be at most a natural number δ and prefer the shortest paths leading to a value node: we discard paths crossing other nodes beside the pivot classified as template nodes because that means a shorter path can be generated by pivoting elsewhere. The rational is that template nodes far away from the values are progressively less likely to generate reliable XPath expressions: within increasingly bigger slices of the pages' DOM tree, it is more and more likely that irregularities may occur, due for instance due to the presence of optional attributes.[6] Second, we limit the analysis to a subset of the input assuming it is a sufficiently unbiased sample of all the available pages within a common HTML template (we use at most 3 pages during the experimental evaluation).

The final step of the extraction rules generation algorithm, *XPath expressions generation*, processes every tree-path produced by the previous step, and produces one XPath expression per tree-path by appending two subexpressions: (i) an expression that just matches the pivot node; (ii) a sequence of XPath steps leading from the pivot node to the last value node. Each of these latter XPath step is associated with a node in the tree-path and move along one of four XPath axes parent, child, following-sibling, preceding-sibling.[7] Every step (along the last three axes) is also followed by a position predicate.

As for the initial requirements, we found out that to attain *completeness*, tree-paths in *every* direction might be needed, while using only parent and child axes [81] may not be sufficient, as illustrated by the following example.

Example 5.3.2. *Consider rule r_6 shown in Figure 5.3c: it uses the* following-sibling *axis to move from the pivot node* `at` *to the* PostCode *value. Rule r_6 is obtained from the tree-path:*

`at` $\rightsquigarrow$ `<strong>` $\rightsquigarrow$ `W0U 1LE`

by juxtaposing the XPath expression //i[contains(.,'at')] *locating the pivot node, and the XPath expression* following-sibling::*[@class='postcode'][1]/text()[1].

However, any rule using only parent *and* child *axes does not work: it leads to an XPath expression starting by* //i[contains(.,'at')]/.. *as for rule r_7. Then, among the* `<li>` *children, it cannot be distinguished which child contain the postcode, in every result record. For example r_7 can not extracts the postcode from the featured property in the first result record of Figure 5.1a and corresponding to detail page* 1 *shown in Figure 5.1b.*

This algorithm is *universal*, i.e., once the template and value nodes have been provided, the enumeration of tree-path work in the same identical way both on the result pages and on the detail pages, thus producing a set of result rules $\mathcal{R}_r$ and a set of detail rules $\mathcal{R}_d$. By simply increasing δ, the *expressivess can be controlled* and raised up to the point to achieve

[6] In the implementation not every node in the tree-path contributes to its length: The node occurring in long chains of one-child element nodes are weighted zero because oftern they are used for "accumulating" presentation properties of a leaf node rather than for structuring the document.

[7] For the sake of brevity, in the reported XPath expressions we abbreviate the last two axes with fs and ps, respectively.

r_0: ⟨1.5k, 1.5k, 2.05k, 1.2k⟩
r_1: ⟨Featured!, 1.5k, !Hot!, 2.05k, 1.2k, Single Room⟩
r_2: ⟨House, Flat, Studio, House, Ensuite, Single Room⟩
r_3: ⟨Featured!, 1.5k, 2.05k, 1.2k⟩
r_4: ⟨1.5k, 1.5k, 2.05k, 1.2k⟩
r_5: ⟨at, W3U 1AF London, W3U 0AF London, E15 2CD Oxford, E15 2XD Oxford, Beds⟩
r_6: ⟨W2U 0DF London, W3U 1AF London, W3U 0AF London, E15 2CD Oxford, E15 2XD Oxford, E18 2XD Oxford⟩
r_7: ⟨at, W3U 1AF London, W3U 0AF London, E15 2CD Oxford, E15 2XD Oxford, Beds⟩
r_8: ⟨W2U 0DF London, W3U 1AF London, E15 2CD Oxford, E15 2XD Oxford⟩
r_9: ⟨3, two, 2, 3, 1, 2⟩

(a) Result rules.

d_0: ⟨1,500, 1,500, *nil*, 2,050, 1,200, *nil*⟩
d_1: ⟨House, Flat, Studio, House, Ensuite, Single Room⟩
d_2: ⟨Featured!, Property, !Hot!, Property, Property, Property⟩
d_3: ⟨"", "", A studio with..., "", "", A single room...⟩
d_4: ⟨W2U 0DF London, W3U 1AF, W3U 0AF, E15 2CD Oxford, E15 2XD, E18 2XD⟩
d_5: ⟨W2U 0DF London, W3U 1AF, *nil*, E15 2CD Oxford, E15 2XD, *nil*⟩
d_6: ⟨A 3 bedroom house located in..., A 2 bedroom flat..., ⋯⟩
d_7: ⟨*nil*, London, London, *nil*, Oxford, Oxford⟩

(b) Detail rules.

Figure 5.5: Extracted values.

completeness. We have empirically observed that it is already achieved when $\delta = 4$ for the vast majority of hundreds attributes used in our experimental evaluation.

5.3.3 Values Extraction

To get the extracted values, our XPath-based extraction rules are evalutated against the input collection of pages. Every detail rule d must produce at most one value from a single detail page p: RED gets the extracted value $d(p)$ by taking the string-value of the node-set obtained by evaluating its associated XPath expression, or just *nil* if the expression matches an empty node-set [77].

Example 5.3.3. *Figure 5.5b lists the values extracted by the detail rules generated starting from the sample pages whose DOM trees are shown in Figures 5.4a-5.4c and then applied over* 6 *detail pages of our running example. Each detail rule extracts exactly* 6 *(possibly nil) values, one per each detail page.*

Differently from detail rules, a result rule r is supposed to extract zero, one, or several string values from a single result page p. RED evaluates a result rule by composing an ordered sequence of strings, denoted $r(p)$, obtained by mapping each node in the node-set evaluation of its XPath expression on p, into its correspondent string-value. We denote the ordered sequence of extracted values from a result rule as follow by $r(p) = \langle v_0, v_1, \ldots \rangle$ if v_0, v_1, ... are the extracted string values.

Example 5.3.4. *Figure 5.5a lists the values extracted by the result rules shown in Figure 5.3c from the result page whose DOM tree is shown in Figure 5.3a.*

It is crucial to notice that *nil* values are not produced at all. If the resulting node-set is empty, the empty sequence is produced.

Example 5.3.5. *Comparing the values extracted in Figure 5.5b by detail rules, with those extracted in Figure 5.5a by result rules, one may notice that* d_0 *extracts a nil element from the third detail page, while none result rule can extract nil values. Result rule* r_6 *extracts* 6 *values,* r_1 *only* 3.

Since in modern rich web pages there might be many template nodes in the area surrounding a target value, our rules generation algorithm that generates for every target node all the rules pivoting on a not-too-far template node, might end up generating a large number of candidate rules. Therefore RED adopts additional filtering heuristics to detect and discard rules that are very unlikely to be correct. For instance, RED removes rules which extract all identical values, e.g., a rule always extracting '`Add to Basket`' is clearly incorrect. To preserve the completeness of the rule generation algorithm, rules extracting template nodes (for example, r_5 in Figure 5.3c extracts `at`), are not discarded even if they are potentially wrong. Indeed, false positives can also be removed by following processing steps, as discussed in Section 5.5), while the template analysis is a noisy process itself, and therefore it could lead to discard correct rules.

Finally RED groups the extraction rules by the extracted values over all input pages: different extraction rules producing the same values on every page end up being indistinguishable each other for our purposes. Only one rule is saved to represent that group.

Example 5.3.6. *The result rules* r_0 *and* r_4 *shown in Figure 5.3c extract identical values from the result page shown in Figure 5.1a,* RED *keeps only one of them for further consideration. Same also holds for result rules* r_5 *and* r_7.

5.4 Redundancy Seeking

RED analyzes the redundancy between all pairs of automatically generated result rules and detail rules for sifting out correct rules. For two rules to be considered *redundant*, they have to extract from the result pages, and from the correspondent set of detail pages, respectively, the encoding of the same value from the underlying abstract relation.

The key intuition is that by analyzing this form of redundancy, correct rules can be separated from noisy ones. Indeed, the result pages are organized according to a different HTML template wrt the detail pages, so it is highly likely that a noisy result rule extracts different values wrt any correct and noisy detail rule, and vice-versa. Conversely, correct result rules are highly likely to be redundant with correct detail rules, because they collect data about the same set of objects as published by the same source in different but related pages.

This section is organized as follows. First, we introduce the *rules alignment* problem for comparing the values extracted by a result and a detail rule. Then, we describe the *soft segmentation* technique for analyzing the result records of the result pages for without

actually finding their boundaries. Finally, we explain how this technique fits in the frame of our redundancy-based techniques for the result/detail pages publishing pattern, and we introduce a score function specifically designed for measuring the redundancy in this context.

5.4.1 the Rules Alignment Problem

A standard distance function between the extracted values cannot be directly adopted to measure the redundancy between a result and a detail rule because the two kinds of rules produce a different number of values, when applied respectively to a set of detail pages, and to the corresponding set of result pages. A detail rule always extracts the same number of (possibly *nil*) values as the number of considered detail pages, while a result rule extracts at most as many values as the total number of records in the considered set of result pages, or even strictly less, as a result rule never extracts, by definition, any *nil* value.

Also, correct result rules should be properly aligned with a supposedly redundant detail rule: given two values, one extracted by a result rule, and the other by a detail rule, they both should refer to the same object for a meaningful comparison, even if the first value comes from a result record, and the second value from the corresponding detail page.

Solving the latter two issues corresponds to finding a correct alignment between the extracted values, and to insert among the values extracted by a result rule an appropriate number of missing *nil*s.

5.4.2 Finding the Detail Links

A preliminary problem that is necessary to solve as a prerequisite of the soft-segmentation technique introduced in the next section is that of finding the *detail links* (or shortly, *link*s), between a result record and the corresponding detail page. These are inherently part of the publishing pattern targeted by RED.

We assume that there exist at least one occurrence of a link in every result record, but we do not exclude the existence of several other copies, including optional ones.

We locate and choose exactly one link occurrence per record, with the additional requirement that every selected occurrence corresponds to the same template node, as illustrated by the following example.

Example 5.4.1. *The first property listed in the result page shown in Figure 5.1a for our running example is a featured property exhibiting an additional link in its result record. The links can be extracted from the DOM tree of the result page shown in Figure 5.3a as those corresponding to every last* `<A>` *element node containing a link, or as the* `<A[@c='detail']>` *node occurrences for the result pages root equivalence class* ε_0^r*. In Figure 5.3c, l denotes the result rule extracting these link occurrences.*

While searching the link occurrences, RED leverages the knowledge of the detail pages, whose url addresses are assumed given as input, and adopts a technique based on the template analysis already executed by the rules generation algorithm. RED leverages the root

equivalence class of the result pages for selecting the best rule extracting the detail links, denoted l and named *link rule*. It validates the link rule by checking that it actually extracts from every result page the same number of occurrences as the number of detail pages, and if not, it falls back to another heuristics: it finds all the element nodes containing detail links for every result page, then it removes duplicates by saving only the last of several consecutive occurrences leading to the same detail page.

5.4.3 Soft Segmentation

We do not tackle directly the problem of finding the result record boundaries (still unknown). Instead, we build on the link rule l [8] extracting exactly one link occurrence in *every* result record span. We also assume that all the occurrences of links extracted by the link rule are equivalent (even if, as it is often the case) several copies of the same detail link occur in every record span, and there might even be optional occurrences.

By analyzing the relative position of the links wrt to values extracted by any other result rule, a slightly uncertain situation arises: it is unknown where each link occurrence is positioned within the span of its result record.

Therefore, once a total ordering for the DOM tree nodes of the input result pages P_r has been fixed, either the nodes extracted by a result rule always occur *before* the link nodes or they always occur *after* them. It worth also noticing once again that since *nil* values are not extracted at all by result rules, a result rule could end up extracting from a page a number of nodes strictly lower than those located by l: So it might be well the case that between two consecutive links, none value extracted by the result rule occurs.

Given the result rule r, we call (result) rule *aligment* one out of at most two possibile solutions to this kind of uncertainty. We use r^a (resp., r^b) to denote the alignment in which all nodes extracted by r are considered occuring *after* (resp., *before*) the corresponding detail link extracted by l in every result record. Given an alignment, say r^b (respectively, r^a), the missing (*nil*) values not already extracted by r can be easily injected after (resp., before) any link extracted by l.

Example 5.4.2. *The link rule* $l =$ //a[@class="detail"] *in Figure 5.3c extracts all the links from the result page in our running example,* p_r*, and let* l_i *denote the detail link node to the i-th detail page. Consider the rule* r_8 *extracting* 4 *nodes associated with these string values:* $r_8(p_r) = \langle$W2U 0DF London, W3U 1AF London, E15 2CD Oxford, E15 2XD Oxford$\rangle$.

It turns out that an in-order traversal of the DOM tree in Figure 5.3a finds links and values extracted by r_8 *according to the following total order:*

$$\langle l_1, \text{W2U 0DF London}, l_2, \text{W3U 1AF London}, l_3, l_4, \text{E15 2CD Oxford}, l_5, \text{E15 2XD Oxford}, l_6 \rangle.$$

[8] We actually leverage the extracted nodes containing the detail link rather than the extraction rule itself. For the sake of presentation, in the following we blur this aspect and refer to both cases by just mentioning the extraction rule l as it was always available.

Two distinct alignments into a sequence of exactly 12 *elements (*6 *values plus* 6 *detail links) are possibile. Either the attribute value occurs before every link:*

$$r_8^b\colon \langle nil, l_1, \text{W2U 0DF London}, l_2, \text{W3U 1AF London}, l_3, nil, l_4, \text{E15 2CD Oxford}, l_5, \text{E15 2XD Oxford}, l_6\rangle$$

or, it always occurs after every link:

$$r_8^a\colon \langle l_1, \text{W2U 0DF London}, l_2, \text{W3U 1AF London}, l_3, nil, l_4, \text{E15 2CD Oxford}, l_5, \text{E15 2XD Oxford}, l_6, nil\rangle.$$

Therefore:

$$r_8^b(p_r) = \langle nil, \text{W2U 0DF London, W3U 1AF London}, nil, \text{E15 2CD Oxford, E15 2XD Oxford}\rangle\ ;$$

$$r_8^a(p_r) = \langle \text{W2U 0DF London, W3U 1AF London}, nil, \text{E15 2CD Oxford, E15 2XD Oxford}, nil\rangle.$$

RED enforces each result rule to obey to one out of two admissible alignments patterns: either every detail link is followed by exactly one node extracted by r (if any, *nil* otherwise), or vice-versa. The noisy result rules not following any of these patterns are already eliminated during this process and namely, those extracting multiple values between two contiguous links.

Example 5.4.3. *Figure 5.6 shows the alignments for the result rules of in Figure 5.3c wrt link rule l shown in the same figure (we simply use r to denote $r^a = r^b$). The extracted values can be found in Figure 5.5. Notice how the result rule r_1 has been dropped because it extracts several nodes (about* Price *and* PropertyType*) in between two consecutive links.*

r_0: ⟨1.5k, 1.5k, *nil*, 2.05k, 1.2k, *nil*⟩
r_2: ⟨House, Flat, Studio, House, Ensuite, Single Room⟩
r_3: ⟨Featured!, 1.5k, *nil*, 2.05k, 1.2k, *nil*⟩
r_5: ⟨at, W2U 0DA, W3G 0AF, WC4H 2DC, E15 X0A, Beds⟩
r_6: ⟨W2U 0DF London, W3U 1AF London, W3U 0AF Cambridge, E15 2CD Oxford, E15 2XD Oxford, E18 2XD Oxford⟩
r_8^b: ⟨*nil*, W2U 0DF London, W3U 1AF London, *nil*, E15 2CD Oxford, E15 2XD Oxford⟩
r_8^a: ⟨W2U 0DF London, W3U 1AF London, *nil*, E15 2CD Oxford, E15 2XD Oxford, *nil*⟩
r_9: ⟨3, two, 2, 3, 1, 2⟩

Figure 5.6: Aligned result rules.

It worth noticing that in absence of missing *nil* values a result rule has a single possible alignment.

Listing 5 presents a pseudo-code description of our soft-segmentation technique for returning a set of at most two possible alignments r^a, and r^b for a given result rule r wrt to a detail link result rule l in a set of result pages P_r received as input.

It starts by orderly collecting the links and values node occurrences within a sequence s (line 8). Then it checks whether the sequence need to be expanded to a sequence of exactly

Listing 5: ALIGN(r, l, P_r) – Find result rule aligments.

Input: A result rule r
Input: The detail link result rule l
Input: A set of result pages P_r
Output: A set of at most 2 possible alignments, r^a and r^b, of r wrt l in all pages of P_r

```
1  begin
2      A ← {r^a, r^b} ; /* set of possible alignments */
3      foreach p ∈ P_r do
4          if (A = ∅) then
5              break; /* no alignment found */
6          end
7          n ← |l(p)| ; /* number of detail links */
           /* orderly accumulate the occurrences of nodes extracted either by l or r from
              p */
8          s ← TOTALORDER(l, r, p) ;
           /* are nil values missing? then injects nils after/before on every two
              contiguous links */
9          if (|s| < 2 · n) then
10             s^b ← FILLNILGAPS^b(s) ;
11             s^a ← FILLNILGAPS^a(s) ;
12         end
           /* a sequence s follows the link after value (resp., value after link) pattern if it
              contains exactly n links, each preceded (followed) by either one nil or one
              value */
13         if not VALUEAFTERLINKPATTERN(s^a, n) then
14             remove r^a from A;
15         if not LINKAFTERVALUEPATTERN(s^b, n) then
16             remove r^b from A;
17     end
18     return A;
19 end
```

$2 \cdot n$ elements, where n is the number of objects in the page, by inserting *nil* value before or after the links (lines 9-12).

If after this operation the sequence does not follow one of the two allowed alignment patterns, i.e., every link must be followed (or preceded) by exactly one value or *nil*, the alignment is dropped (lines 13-16).

Also, if it turns out that a result rule has only one possible alignment, it is marked as correctly aligned wrt to the link rule, an information that later will be used during the validation step.

5.4.4 Redundancy Score

In order to measure the redundancy between a result rule and a detail rule, we introduce a normalized *redundancy score* function, denoted red$(\cdot)$; it assumes real values in the interval

$[0,1]$: the lower the score, the closer and redundant are considered its arguments. It is defined by means of a pairwise comparison between values coming from one aligned result rule and one detail rule, which both can be seen as producing ordered sequences of strings of the same length. Indeed, the values coming from an alignment r^x of the result rule r (with $x \in \{a,b\}$) and the detail rule d applied, respectively, over a set of result pages and the corresponding set of detail pages, can be easily compared by means of a standard (normalized) distance function for strings, because given any of the result page p_r, $r^x(p_r)$ extracts an ordered sequence of strings of the same length as the sequence obtained by orderly applying d over the corresponding set of detail pages, i.e., the corresponding detail pages linked by the result records.

We consider two strings perfectly redundant when one is a substring of the other, and we consider *nil* values perfectly *not* redundant with any other non-*nil* value. For all other cases, i.e., pair of non-*nil* values, we use a standard *string* distance such as the *Jensen-Shannon* distance (*JS*) [82]. Namely, the redundancy score of a pair of values (v_1, v_2) is computed as:

$$\text{score}(v_1,v_2) = \begin{cases} 1, & \text{if } v_1 \neq v_2 \text{ and } (v_1 = nil \text{ or } v_2 = nil); \\ 0, & \text{if } v_1 \text{ is substring of } v_2 \text{ or viceversa;} \\ JS(v_1,v_2), & \text{otherwise.} \end{cases} \tag{5.1}$$

The above definition for the score function is rather relaxed, to the point that a few false positives might arise due to two incidentally similar substrings. The rationale behind this choice is that false positive can be filtered later, during the steps considering the pairwise redundancy between many pairs of aligned objects, while at this stage it is more crucial not to compromise the recall of the redundancy seeking process rather than its precision.

Given a rule pair (r,d) composed of one result and one detail rule, and a set of objects $\mathcal{O}$, the redundancy score of the pair is defined as the average pairwise score between their values, as follows:

$$\text{red}(r,d) = \sum_{o \in \mathcal{O}} \frac{\text{score}(r^d(o), d(o))}{|\mathcal{O}|} \tag{5.2}$$

where r^d is either r^a or r^b; it denotes one of the two possible valid alignments of the result rule r with respect to the link rule l; $d(o)$ and $r^d(o)$ denote respectively the string value extracted for object o by the detail rule d, and the string value corrisponding to the same object extracted by the aligned result rule r^d.

r^d is the best alignment (that with the lowest score) chosen by means of an analysis of its redundancy wrt the detail rule d, as follows (we prefer r^a to break ties):

$$r^d = \text{argmin}_{r^x \in \text{ALIGN}(r,l,P_r)}[\text{red}(r^x,d)],$$

where ALIGN is the soft-segmentation procedure discussed in Section 5.4.3 and P_r is the set of result pages considered.

Example 5.4.4. *Let r_0 be the result rule as shown in Figure 5.5a but adopting for the prices the same formatting adopted for the prices in the detail pages of the running example. The detail link result rule l is described in Example 5.4.1. The two alternative alignment for r_0 are:*

$$r_0^a = \langle \text{1,500}, \text{1,500}, nil, \text{2,050}, \text{1,200}, nil \rangle$$
$$r_0^b = \langle nil, \text{1,500}, \text{1,500}, nil, \text{2,050}, \text{1,200} \rangle$$

Given the detail rule d_0 shown in Figure 5.5b, it turns out that $\text{red}(r_0, d_0) = \min(\text{red}(r_0^a, d_0), \text{red}(r_0^b, d_0)) = 0$ *since r_0^a extracts the same values as d_0.*

5.5 Noise Removal

Given a redundancy score threshold ρ, we might consider as *redundant* all the pairs of rules having a redundancy score (as defined in Eq. 5.2) not greater than ρ.

Unfortunately, many of these pairs might still contain noisy rules: On one hand, there could be an incorrect alignment of the values, due to the presence of too many similar values in result records close each other; on the other hand, sometimes even noisy values happen to be incidentally similar, especially when the range of the possible values for the involved attributes is quite limited.[9]

Therefore, finding a correct value of the threshold is not trivial: by choosing a too small value redundant pairs of correct rules could be regarded as not redundant due to small discrepancies in the format of the values or even actual differences in the reported values.

Conversely, by taking a too large value, pairs involving noisy rules could be considered as redundant and eventually, incorrectly classified as correct. To further complicate the matter, consider that the best threshold value, i.e., that separating pairs composed of correct rules from all other noisy pairs, changes from attribute to attribute, as illustrated by the following example.

Example 5.5.1. *Table 5.2 reports the pairs scoring less than $\rho = 40\%$ ordered by their redundancy score. Each pair is composed of an aligned result rule from Figure 5.6, and a detail rule from 5.5b of our running example. The Attributes column reports a note on the attributes whose values are involved in each pair, and whether the rules of the pair are noisy or incomplete, or mis-aligned.*

For example, d_6 reports the attribute Description *containing a relatively long textual description of the property: it also reports, as a substring, the type of the property; the adopted definition of the* $\text{RED}(\cdot)$ *score regards it as perfectly redundant with the* Type *in r_2.*

Observe that the first three pairs involve correct result rules, but also the last one with score 39% (due to the slightly different format adopted for encoding the Price *in the result records wrt detail pages, e.g., 1.5k vs 1.500).*

Therefore, to conclude on the running example, the correct result rules cannot be trivially separated by the incorrect ones from any value of the threshold.

[9] To give an example, several attributes in the real estate domain, such as Rooms, Beds, and Baths, all happen to assume values from zero to a small positive integer.

pair π	red(π)	*Attributes*
(r_2, d_1)	0	Type
(r_6, d_4)	0	Address
(r_2, d_6)	0	Type vs Description
(r_8^b, d_7)	0	mis-alig. Address vs incom. City
(r_8^a, d_5)	0	incom. Address vs incom. Address
(r_9, d_6)	0.167	Beds vs Description
(r_5, d_4)	0.333	noisy Address vs Address
(r_0, d_0)	0.397	Price

Table 5.2: Pair of rules with score less than 40%

RED processes the redundant pairs by ascending score order to remove the *noise* redundancy, i.e., pairs of rules involving at least a noisy rule. Several techniques are put in place: first, the *result rule template validation* analyzes how well the result rules fit in the HTML template of the result pages (see 5.5.1); second, *noise redundancy removal* leverages the redundancy score to select for each attribute, only the best and most complete pairs of rules, i.e., those having lowest score and extracting less *nil*s (see 5.5.2).

The surviving redundant pairs are considered correct, i.e., each composed of an aligned result rule, and a correct detail rule.

5.5.1 Result Rules Template Validation

RED validates every result rule r from a redundant pair (r^d, d) and filters out those involving invalid alignments of the result rule.

The technique is based on the observation that result rules correctly aligned are already available from previous processing steps. Beside the link rule, which has been already used somehow to validate all other result rules and it is assumed to be correct, several other result rules may happen to extracted only non-*nil* values, and therefore they have been associated with a single alignment wrt to the link rule. RED leverages these already validated rules to validate others, as described in the algorithm shown in Listing 7.

We say that a result rule r is *valid* wrt another reference result rule g assumed valid and extracting only non-*nil* values if the relative order of their values is the same within every result record of every result page, coherently with the abstract model described in Section 5.1. The relative order is set by the underlying HTML template, but cannot be changed from result record to result record.

For example, on a result page from a real estate agency advertising properties for sale, if the PostCode of a property is published before its Price in a record, then it precedes the price in every other result record.

The ISVALID$(\cdot)$ function shown in Listing 6 is based on a trivial generalization of ALIGN$(\cdot)$ function (Listing 5) working with any result rule extracting only non-*nil* values,

Listing 6: ISVALID(g, r, P_r) – Redundant result rules validation

Input: A reference result rule g (assumed valid) extracting only non-*nil* values
Input: A result rule r
Input: A set of result pages P_r
Output: Whether r is a valid or not

```
1 begin
2 |   return (|ALIGN(r, g, P_r)| ≥ 1);
3 end
```

not only with the link rule. The validation function checks that at least one alignment of the result rule is possible, and therefore the occurrences of the values of r and g must be correctly interleaved in the pages: occurrences of r values are either always preceding or always following g values occurrences in every result record.

For the ISVALID$(\cdot)$ function to produce correct results, it needs a correct result rule g to start from, as shown by the following example.

Example 5.5.2. RED *detects the wrongly aligned rule* r_8^a *from Figure 5.3c by validating it against already validated rules. Rules* r_0 *and* r_9 *are correctly aligned because each of them has only one alignment. They can be used as reference to check whether* r_8^a *(as shown in Figure 5.5a), is a valid aligned rule or not. The result page in Figure 5.3a is such that the actual sequence of the extracted nodes is as follows:*

⟨1.5k, W2U 0DF London, 1.5k, W3U 1AF London, 2.05k, E15 2CD Oxford, 1.2k, E15 2XD Oxford⟩.

r_8^a *is validated by* r_0 *because their value occurrences are perfectly interleaved.*

Suppose then to validate r_8^a *against* r_9*, with the actual sequence of node occurrences as follows:*

⟨W2U 0DF London, 3, W3U 1AF London, one, 2, E15 2CD Oxford, 3, E15 2XD Oxford, 1, 2⟩

that shows several consecutive values from the same rule: the extracted values are not perfectly interleaved. So r_8^a *is not valid wrt* r_9 *and the pairs involving it in Example 5.5.1 are not considered anymore.*

Valid result rules can be used as a reference to check other result rules. By repeatedly removing the invalid rules and saving the valid ones, the process can be iterated. Eventually, all the result rules are validated against all others. Listing 7 reports an algorithm that uses the link rules l as a starting point of the iterations (line 2). Then it quadratically processes all the result rules saving only those found valid wrt every other rule already considered valid (lines 3-14).

Listing 7: VALIDATION(l, Π, P_r) — Validation of redundant result rules

```
   Input: The link rule l which is assumed valid
   Input: Redundant pairs of rules Π = {(r,d) : red(r,d) ≤ ε}
   Input: A set of result pages P_r
   Output: A subset of Π with only valid result rules
1  begin
       /* a set of valid result rules                                  */
2      R^v ← {l} ;
3      Π' ← ∅ ;
4      foreach (r,d) ∈ Π do
5          foreach g ∈ R^v do
6              if (not IsValid(g, r, P_r)) then
7                  remove (r,d) from Π' ;
8                  mark r as invalid ;
9          end
10         if [not (r marked as invalid) and
11          (r does not extract nil values)] then
12             add r to R^v ;
13             add (r,d) to Π' ;
14     end
15     return Π' ;
16 end
```

5.5.2 Removing Redundant Noise

Once all the result rules have been validated, the surviving redundant pairs include correct pairs together with noisy-redundant pairs that might fall into one of three categories: (i) a noisy result rule and a correct detail rule; (ii) a noisy result rule and a noisy detail rule; (iii) an incomplete result rule and a detail rule.

The key assumption to remove the noisy pairs in the first two categories is that redundant and correct pairs have always a smaller redundancy score than pairs containing at least one noisy rule. The assumption is exploited in the algorithm REMOVEREDUNDANTNOISE shown in Listing 8 and dealing with weak redundancy among already validated result rules (lines 2-9).

It starts by ordering the pairs by redundancy score (line 3), then it groups the pairs according to the nodes extracted by the result rule: two pairs will be grouped together (line 7) iff their result rules end up extracting at least one common node from the DOM tree of any of the result pages. Since at most one of the result rules is correct, only the best pair of each group is kept (line 8 and line 9).

Example 5.5.3. *Consider again two pairs involving d_4 in Example 5.5.1, i.e., (r_6, d_4) and (r_5, d_4). The first step (lines 2-9) removes the latter pair since r_6 extract nodes that overlap with those extracted by r_5, and that pair has the worst redundancy score.*

Listing 8: REMOVEREDUNDANTNOISE — Removing redundant noise

Input: A set of redundant rule pairs $\Pi = \{(r,d) : \text{red}(r,d) < \varepsilon\}$.
Output: A set of correct rule pairs.

```
1  begin
       /* Group pairs by overlap of nodes extracted by the result rule */
2      Π' ← ∅ ;
3      order pairs in Π by increasing redundancy score;
4      foreach (r,d) ∈ Π do
5          remove (r,d) from Π ;
6          foreach (r',d') ∈ Π do
7              if OVERLAP(r,r') then
8                  remove (r',d') from Π ;
           /* For each class saves only the best pair. */
9          add (r,d) to Π' ;
       /* Among redundant but incomplete result rules. */
       /* prefer pairs whose detail rule extracts less nils. */
10     Π'' ← ∅ ;
11     order Π' by increasing n. of nil in the detail rules;
12     foreach (r,d) ∈ Π' do
13         remove (r,d) from Π' ;
14         foreach (r',d') ∈ Π' do
15             if OVERLAP(r,r') then
16                 remove (r',d') from Π' ;
17         add (r,d) to Π'' ;
18     return Π'';
```

The REMOVEREDUNDANTNOISE procedure targets pairs of the first two categories of weak redundancy, but unfortunately, it does not remove pairs involving non overlapping incomplete rules, i.e., those extracting at least one *nil* value instead of the correct value. It turns out that the pairwise comparison of values can easily lead to a very small, or even zero, redundancy score.

The lines 10-17 of Listing 8 prefer pairs of rules extracting fewer *nil* values: at this stage, all the noisy pairs have already been removed so that we can directly remove the result rules extracting a larger number of *nil*s. Notice that with the exception of the initial ordering (line 11 vs line 3), there are two almost identical processing loops (lines 10-17 and lines 2-9). However, they cannot be merged because the latter is based on the assumption that it received as input only pair in which the noisy rules have already been removed.

Example 5.5.4. *Consider again the pairs listed in Example 5.5.1. Both pair* (r_6, d_4) *and* (r_8^b, d_5) *are* 0*-scored.*

The first step of REMOVEREDUNDANTNOISE *(lines 2-9) does not group them together because their result rules differs for two elements that are missing in* r_8 *wrt* r_6*. Both pairs*

pair π	red(π)	note
(r_2, d_1)	0	Type
(r_6, d_4)	0	Address
(r_2, d_6)	0	Type vs Description
(r_9, d_8)	0.2	Beds
(r_0, d_0)	0.36	Price

Table 5.3: Pair of rules after RED's noise removal step.

reach the second loop (lines 10-17). The (r_8^b, d_5) *pairs is removed because it contains the greatest number of nil elements.*

After the two steps conclude, the surving pairs should only contain correct result rules, i.e., r_0, r_2, and r_6 which are the output of the RED algorithm in Listing 4.

Example 5.5.5. *Table 5.3 reports the pairs left after all the steps of the noise removal process, ordered by the redundancy score. The pairs* (r_2, d_1) *and* (r_2, d_6) *are both* 0*-scored and the correct rule* r_2 *for* Type *is included in the output for its redundancy with the* Description.

5.6 Evaluation

We conducted several experiments on real-world websites to evaluate the performance of our approach. We first tested RED on the dataset used by DIADEM, an ontology-based data extraction system and achieving both high recall and high precision [17] in the cited dataset covering many different websites from just two domains. We then applied it to another dataset including websites of a variety of domains, so targeting the main weak-point of the former system: it needs a manually crafted ontology for every domain, while we show that RED can achiveve a stable extraction performances over many domains, even without any domain dependent tuning. Please note that our evaluation is limited to those sites with result/detail page redundancy.

As for comparing RED with other known approaches in the literature, we also considered another unsupervised system DEPTA, and a commercial system IMPORT.IO. Indeed, to the best of our knowledge, the unsupervised system with the most similar setting to ours is DEPTA [7]. Unfortunately, it is a quite old system that suffers from modern websites. Other available systems either merely solve the problem of result-page segmentation or learn from detail pages a regular expression that models the template but without solving the problem of selecting the relevant attributes. As for DIADEM, it is rather large project containing several components, including one identifying and aligning attributes from result pages. We comparing against this component, that we refer to simply as DIADEM hereafter.

5.6.1 Evaluation Method

Datasets We evaluate RED on two datasets, named DIA_DS and RED_DS, including 130 websites in total. DIA_DS consists of 100 websites randomly picked from DIADEM's dataset (half from REAL_ESTATE and half from USED_CARS domain). RED_DS consists of 30 site from 10 domains as listed in Table 5.6, obtained by randomly picking 3 sites from the *Alexda Top* 50 *Global Sites*, but excluding from the selection any website from metasearch portails, whose detail pages are usually obtained by merging pages from different, unrelated, websites.Each test-case is related to a single website and it consists of one result page and the corresponding set of detail pages. All the records listed in the result pages are obtained by following the default sorting criteria from the website.

Metrics For each result page in our datasets, we manually crafted the correct rules as golden standard, based on which we computed the number of true positives (tp), false negatives (fn), and false positives (fp), and the precision (P) as $P = \frac{tp}{tp+fp}$, recall (R) as $R = \frac{tp}{tp+fn}$. Also, we computed their harmonic mean, the F measure, as $F = \frac{2 \cdot P \cdot R}{P+R}$.

These metrics have been computed both at rules level and at level of extracted values. At rule level, we consider a rule correct only if it extracts a set of values perfectly matching with that extracted by the golden rule, without any missing value: any rule extracting just a few noisy values or missing a single value, is considered as it was completely wrong. This is a quite strict metric that end up counting incomplete rules as completely wrong even if they extract a large portion of the correct values. Therefore, we further calculate all the metrics at value level by simply counting the number of correct/noisy extracted values.

Target Attributes In order to build the golden dataset, we selected the target attributes that should be extracted from every website. We aim to extract all the attributes of the published objects, with some specific treatments worth mentioning. We only extract attributes with distinct values in different records while overlooking those having constant values over all records. For example, the Country of all the hotels from a booking website working only with UK hotels. However, we also include optional attributes assuming both non-*nil* values and *nil* values. For example, in real estate websites, we target the attribute Furnishing because properties have this attribute assuming either the value `Furnished` or *nil*. On the contrary, we excluded attributes not strictly related to the main entity of interest in a domain: for example attributes about the number of website visitors, or the number of views for each object. These nodes are always considered out of RED's set of target attributes.

While comparing RED with DIADEM, to be fair to DIADEM, we further split the DIA_DS dataset into two datasets composed of the same pages but with different target attributes: in the first dataset, we limit the target attributes to those identifiable by the ontology of DIADEM (attribute names as detailed in the second column of Table 5.4); the second dataset

also includes all the attributes that are available in the result pages of the dataset even if not directly targeted by DIADEM's ontology.

Domain	Diadem Attributes	Additional Attributes
REAL_ESTATE	Price, Location, PostCode, PropertyType, Status, Furnishing, Beds, Baths, Receptions	Agency, QuoteReference, Phone, FloorArea, PublishDate, FloorPlan, AvailableDate, *etc.*
USED_CARS	Price, Location, PostCode, Model, Make, Transmission, Colour, BodyType, FuelType, Age, EngineSize, DoorNumber, Mileage	Registration, Reg, Dealer, MPG, DepositAmount, *etc.*

Table 5.4: DIADEM's original target attributes (left); additional available attributes (right).

5.6.2 Comparison with DIADEM

In this section, we present the results of the experimental evaluation of RED on DIA_DS. We illustrate that: (i) RED can achieve a performance as accurate as DIADEM although it works in a fully-unsupervised manner; and (ii) RED has an advantage since it is capable of identifying all the attributes without the skilled human intervention required by DIADEM to craft its input ontology. Also, we illustrate RED limitations and scope of application by detailing some of the reasons that prevented RED from achieving a perfect F-measure.

Dataset		DIA_DS	DIA_DS	DIA_DS
System		DIADEM	RED	RED
Target Attributes		ONTOLOGY	ONTOLOGY	ALL
REAL_ESTATE	P_r	0.94	0.91	0.92
	R_r	0.90	0.90	0.90
	F_r	0.92	0.91	0.91
	P_v	0.98	0.98	0.98
	R_v	0.92	0.94	0.93
	F_v	0.95	0.96	0.96
USED_CARS	P_r	0.97	0.96	0.97
	R_r	0.96	0.96	0.96
	F_r	0.96	0.96	0.97
	P_v	0.97	0.96	0.97
	R_v	0.94	0.98	0.97
	F_v	0.95	0.97	0.97

Table 5.5: RED vs DIADEM Performance.

Table 5.5 presents $P/R/F$ results of RED and DIADEM systems over the two datasets considered, DIA_DS and RED_DS, both at the rules level and at the values level (denoted by using the r and v as subscripts, respectively) first considering only the attributes targeted by DIADEM's ontology, and then including all available attributes.

RED's performance is as good as DIADEM (0.01 lower F_r at rules level and 0.01 higher F_n at values level). Also, among all the $50+50$ websites of the two considered domains, RED

performed perfectly $33+40$ times while DIADEM did so for $37+39$ times. Although at rule level RED is slightly less precise than DIADEM (by 0.03), it achieves the same value level precision as DIADEM with a better recall. It worth noticing that in many practical scenarios this copes well with approaches (such as those dealing with data cleaning [18]) which can deal with false positives to improve precision.

5.6.2.1 Attribute Coverage

Unlike DIADEM, which is limited to its ontology scope, RED can extract all the redundant attributes published in the result pages without human supervision. As can also be seen in Table 5.5, RED has steady performance, even when evaluated over all the attributes. In detail, RED generated 26 more correct rules than DIADEM does (204) over all the websites of the dataset.

5.6.2.2 Failures & RED's Limitations

In all failure cases, RED and DIADEM failed for different reasons, but for a single case (on davidtompkins.co.uk, both DIADEM and RED failed to generate a complete rule for Status due to an irregularity in the structure of HTML template around the status value in some property.) We report the reasons of these failures considering two main categories of errors.

Noises DIADEM leverages its ontology to automatically annotate target pages. Wrong extracted values from DIADEM system are mainly due to misleading annotations. In our experiments, there are 5 sites where it suffers of this problem, e.g., top-lettings.co.uk. DIADEM extracts the template nodes `Available:` always occurring before the attribute AvailableDate and wrongly consider it as a value of the attribute PropertyStatus. In contrast, RED's noisy rules are usually incomplete, that is, they extract only a fraction of the correct values.

Misses Unlike DIADEM, which misses attributes due to the failure of the underlying annotation process, RED misses attributes for two main reasons. (i) *No-redundancy:* Some attributes in the result pages are not redundant at all; for example Description on davidtompkins.co.uk. (ii) *Optional constant:* sometimes RED misregards the optional attributes with constant values by considering its frequently occurring values as template nodes rather than as values to extract; for example, on cotswoldlettings.co.uk, the status of the properties is optional and is always `SOLD`.

Overall, we found only 9 no redundant result attributes in our datasets, and just 5 attributes were missed for the latter reason.

5.6.3 Multi-domain Evaluation

We then applied RED on RED_DS, a dataset containing 10 domains with 3 randomly selected sites for each domain. The results are listed in Table 5.6, where n denotes the number

Table 5.6: Results of RED on RED_DS.

Domain	*Site*	n	tp_r	fn_r	fp_r	P_r	R_r	F_r	tp_v	fn_v	fp_v	P_v	R_v	F_v
Coffee	counterculturecoffee	12	4	0	0	1	1	1	41	0	0	1	1	1
	ferriscoffee	20	2	0	0	1	1	1	40	0	0	1	1	1
	hasbean	12	4	0	0	1	1	1	48	0	0	1	1	1
Concerts	justshows	50	6	1	0	1	0.83	0.91	278	50	0	1	0.82	0.90
	concert-diary	25	6	1	0	1	0.83	0.91	150	25	0	1	0.83	0.91
	songkick	50	4	0	0	1	1	1	199	0	0	1	1	1
Florist	fnp	60	4	1	0	1	0.75	0.86	240	60	0	1	0.75	0.86
	inbloomflorist	12	6	1	1	0.83	0.83	0.83	64	12	4	1	0.88	0.93
	beneva	11	6	1	0	1	0.83	0.91	66	11	0	1	0.83	0.91
Jewellry	goldpalace	12	4	0	0	1	1	1	48	0	0	1	1	1
	mynamenecklace	40	3	0	0	1	1	1	120	0	0	1	1	1
	rubylane	30	3	0	0	1	1	1	90	0	0	1	1	1
Jobs	snagajob	16	3	0	0	1	1	1	47	0	0	1	1	1
	ncfjobs	20	5	2	4	0.43	0.60	0.50	92	20	15	1	0.90	0.95
	usajobs	10	6	1	0	1	0.83	0.91	60	10	0	1	0.83	0.91
Cinema	megaplextheatres	5	8	0	0	1	1	1	40	0	0	1	1	1
	showplaceicon	14	2	0	0	1	1	1	27	0	0	1	1	1
	alletheatresinc	18	2	0	0	1	1	1	36	0	0	1	1	1
Books	booktopia	23	10	0	1	0.91	1	0.95	163	0	2	0.99	1	0.99
	bookdespositoty	30	8	0	0	1	1	1	227	0	0	1	1	1
	blackwell	48	7	0	0	1	1	1	295	0	0	1	1	1
Camera	abesofmaine	30	6	2	0	1	0.67	0.80	180	60	0	1	0.67	0.80
	precision-camera	20	3	0	0	1	1	1	34	0	0	1	1	1
	photovillage	10	3	0	0	1	1	1	30	0	0	1	1	1
Light	eaglelight	14	4	0	0	1	1	1	56	0	0	1	1	1
	thelightingsuperstore	25	2	0	0	1	1	1	50	0	0	1	1	1
	brandlightingusa	20	4	0	0	1	1	1	80	0	0	1	1	1
Cigars	famous-smoke	60	8	1	1	0.88	0.88	0.88	439	15	20	0.95	0.97	0.96
	cigars-of-cuba	9	9	0	1	0.90	1	0.95	71	0	6	0.92	1	0.96
	vipcigars	18	3	0	0	1	1	1	54	0	0	1	1	1
10 domains	*30 sites*	*724*	*145*	*11*	*8*	*0.94*	*0.92*	*0.93*	*3365*	*248*	*28*	*0.99*	*0.93*	*0.96*

of records contained in the result page, and P, R, F stand for the precision, recall and F-measure, with the subscript of either r or v to distinguish whether these metrics have been computed at rules level or at values level. We also detail the number of correctly extracted (tp) values/rules, the number of misses (fn), the number of noises (fp).

The RED_DS dataset contains a variety of websites with different characteristics: first, it covers a wide range of result record numbers ranging from 5 (on megaplextheatres) to 60 (on famous-smoke), as well as a diverse number of correct target attributes ranging from 2 (on ferriscoffee) to 10 (on booktopia) which is also reflected by the diverse number of correct values ranging from 27 (showplaceicon) to 439 (on famous-smoke). The considered domains also differ for the number of attributes in each result record. For example, on average, in the BOOK domain a result record contains more attributes than for the COFFEE domain.

Also, both quite old-fashioned sites such as fnp and modern sites such as justshows, are included in the dataset.

Overall, RED attained a high performance at both rules level (0.93 of F-measure) and values level (0.96 of F-measure). At the rule level, it achieved a well balanced compromise between precision and recall, while at the values level, it achieves a much higher precision (0.99).

Among 8 noisy rules in RED's results, only 3 of them are pure noises while all the other

noisy rules are incomplete rules, i.e., partially correct rules. On ncfjobs RED seems to perform badly as there are 4 noisy rules; however, 3 of the noisy rules are incomplete rules of the Description while the other one is the incomplete rule for the JobTitle. Since RED rules generation algorithm could not generate the correct rules for these two attributes, 4 incomplete rules were not removed during the noise-removal step. For 7 out of 9 sites where RED failed to output a complete set of rules, only one rule missed, e.g., famous-smoke.

5.6.3.1 Comparison with Other Approaches

As a comparison with other approaches documented in the literature, we ran DEPTA [83] on the dataset RED_DS. This system is also capable of extracting records from result pages and does not require multiple sample pages. Although the algorithm was proposed in 2006, we compared with its runnable version implemented in 2012. Since it blindly outputs multiple tables without telling the user which one contains the target data, we manually picked the table containing the published records and evaluated its precision, recall and F-measure at the *column* level (somehow this is comparable to the rule level in evaluation of RED).

We first evaluated DEPTA's performance of record identification which is its first step. It turned out that DEPTA could only output a correctly-aligned table on 14 out of 30 sites considered. As a contrast, RED never failed to mix attributes from different records, which well illustrates that our soft segmentation method is not only innovative but also benefiting the performance. Among these 14 sites, DEPTA achieved precision of 0.70 and recall of 0.93. It failed quite often to distinguish target data and HTML template nodes, such as on abesofmaine.com it output a column of values "`Price:`".

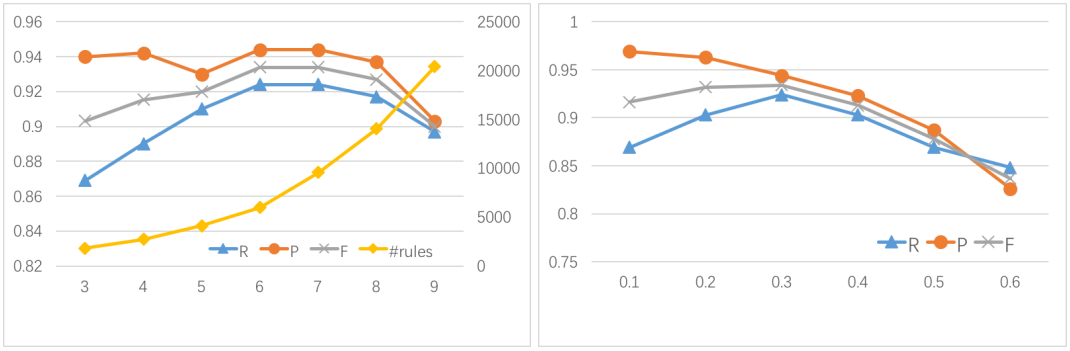
We also compare RED with IMPORT.IO[10] a recent industrial tool for extracting data from result pages. As for the F-measure, both its precision (0.83) and recall (0.75) on RED_DS is not comparable to RED's. Even if after discarding those sites (goldpalace, photovillage and eaglelight) on which IMPORT.IO reported an error and those sites (thelightingsuperstore and cigars-of-cuba) on which IMPORT.IO worked extremely bad that it could not even locate the data region, its precision (0.84) and recall (0.88) are still much lower than RED's.

IMPORT.IO's errors resemble those of DIADEM and RED: template textual nodes from the area surrounding the target attribute values are confused with the correct target values. For example, it extracts some buttons as data, such as `Add to basket` on blackwell.com. Also, it merges several attributes whose values correspond in the DOM tree to sibling nodes by outputting the juxtaposition of all the values, somehow under-segmenting the attributes. For example, on counterculturecoffee.com it outputs Price and SubscribePrice merged as a single attribute. RED has its advantage that it never combines text nodes together but instead follows the way how the text nodes are separated in the original DOM, so as to benefit users to the maximum degree. It is of practical worth that combining several attribute values together is always much easier than splitting a long text into different attribute values.

[10] We got the tool from https://www.import.io in March, 2017

5.6.3.2 Robusteness to Parameter Setting

RED depends on two key parameters. δ is the maximum allowed pivot-to-value distance used during the generation of the extraction rules. It affects the number of generated extraction rules. ρ is a minimum threshold on the redundancy score to consider a pair of result vs detail rules as redundant during the search for redundant pairs. It affects the number of pairs that will be processed during the searching of the correct rules. Albeit RED takes into account other configuration parameters,[11] they are excluded by this analysis because much less directly related to the core of our approach, and their default values are hardly significantly improvable on large datasets covering several domains.



(a) Max pivot-to-value distance δ (b) Max redundancy score ρ

Figure 5.7: RED performance by varying the two key parameters.

Figure 5.7a plots the performance of RED on the RED_DS dataset over several δ values. While the precision is only slightly affected, the recall is significantly related to the maximum pivot-to-value distance. The larger the number of generated rules, the higher is the probability of generating correct rules. In our experiments, δ at 6 already attained the largest number of sites for which the considered XPath fragment is complete. However, by generating too many additional rules, RED ends up introducing too many noisy rules, and, by chance, it happens that a few noisy result and detail rules incidentally extract similar values. Eventually, RED considers that noisy pair better than a pair involving the actual correct rules (or an incomplete version of it), and the weak-redundancy removal step discards the latter, so causing a precision loss. As can be seen in the plot of Figure 5.7a, this effect is clearly visible when δ is raised to 9: both the precision and recall has a sudden decrease.

The other key parameter is the maximum redundancy score, ρ, to consider a pair as redundant. It significantly affects the weak redundancy removal step whose goal becomes harder and harder as additionaly noisy pairs are injected in the RED's processing pipeline. As shown in Figure 5.7b,[12] the precision starts with maximum values, and then it gradually decreases as ρ becomes larger and larger. Although the noise removal process can still manage to remove most of the noises when ρ is at 0.4, for larger values of the parameter,

[11] Such as that used during the template analysis step described in Section 5.3.1 for classifying nodes of a DOM tree as either template or value node according to the number of their occurrences

[12] In this experiment we set $\delta = 6$.

the greater number of noisy pairs makes the weak redundancy removal less effective, to the point that it again causes a loss of precision.

On the other hand, too small values for ρ, meaning a too strict criteria for considering a pair of rules as redundant, makes RED wrongly remove some correct but not perfectly redundant pair, as is evidenced by the plot in Figure 5.7b at $\rho < 0.3$.

For RED experimental evaluation, we measured RED performance by using the best empirically known values for the two parameters δ and ρ, that is, $\rho = 0.3$ and $\delta = 6$.

5.6.3.3 Efficiency

RED is based on a pipeline of processing operations working on pages, extraction rules, and pairs of extraction rules. Its efficiency stricly depends on the number of extraction rules generated, since in the vast majority of practical settings, a few result pages (or even a single result page, as in our experimental evaluation) and a few dozens of detail pages are enough for getting unbiased results. By considering the number of input pages a constant, RED efficiency can be conveniently analyzed in term of the number of rules generated during the initial rule generation step.

Table 5.7 shows the average numbers of extraction of rules (pairs of rules) *produced after* every processing step composing RED's pipeling both for detail pages and for result pages. For producing these numbers, the system has been run only on the RED_DS dataset, that covers a much wider range of domains than DIA_DS dataset.

Output	***Processing Step***	***Result***	***Detail***
rules	*generation*	1259.17	4912.10
rules	*alignment*	1086.90	—
rules	*duplicates removal*	205.50	289.03
pairs	*redundant*	88.87	
pairs	*validation*	88.30	
pairs	*weak redundancy removal*	20.20	
rules	*final*	4.73	

Table 5.7: #rules/pairs produced by every processing step.

At the beginning, RED's rules *generation* step create quite a large number of extraction rules as the generation algorithm has been up-front designed for achieving the completeness of the class of extraction rules generated, which is an important factor for achieving the best possible recall in the final outcome of the processing pipeline.

The number of detail rules were far larger ($\simeq 3.5\times$) than the number of result rules, as expected: each detail page, by definition, contains a lot of values to extract about a single object, whereas one result page summarizing attribute values for several objects contains many values that can be extracted by a single result rule.

The *alignment* step runs over every result rule: it removes more than 13% of the result rules, i.e., all the generated rules that cannot be aligned in any meaningful way wrt the link rule.

The *duplicate removal* step groups the rules by the set of extracted nodes and save only one rule per every cluster of rules extracting the same values. The number of candidate rules and pairs processed is dramatically reduced by this step: on average 86.5% of the result rules and 95% of detail rules are discarded. On average, about 46485 pairs of rules need to be scored, but only 88.87 are potentially redundant, i.e., with redundancy score lower than ρ.

The next step, *validation*, did not remove many vectors on average but still it is crucial to solve a few difficult and specific cases: for instance, for the site usajobs it removed 7 pairs, well above the average number of pairs it removes over all the websites of the considered dataset. This can be explained by saying that the usefulness of the validation step is strongly affected by the peculiarity of the considered website, as it filters result rules mainly when the HTML template is weakly structured and ambiguous.

Finally, *weak removal redundancy* is vital for the final precision, as evidenced by a roughly 3.5× decrease of the surviving pairs in the final output.

Although RED is meant to work as an offline tool, it is still able to achieve an average running time of 72.6 seconds[13] processing in less than 30 seconds half of the sites in our dataset. Most of the RED running time is spent on executing the generated extraction rules, an easy task to distribute on several parallel executors.

[13]Our experiments were conducted on a 64-bit Ubuntu system with Intel Core i7 CPU @ 3.40GHZ x 8

Chapter 6

Data Error Meter[1]

To help data owners evaluate the data quality, in this chapter, we propose a series of new algorithms for an important but not well addressed problem: estimating the number of errors of a given dataset. In particular, we present solutions for two types of errors that commonly arise in web data: duplicates and violations of integrity constraints. We first connect duplicate estimation to well studied problems in statistics and describe their limitations (Sections 6.1.1 and 6.1.2). We then propose a two step approach based on classifier evaluation (Section 6.1.3). It first builds a classifier for duplicate detection using very little training data. It then calibrates the classifier and uses it to estimate its accuracy to generate an estimate of the number of duplicates. The additional calibration step is often required as the classifiers are not 100% accurate even with a large amount of training data. The above quantitative approaches do not work for estimating the number of violations of integrity constraints. We thus propose a qualitative two phase approach of candidate generation and testing adapted from a prior art [51] to estimate the number of FD violations (Section 6.2).

Problem 6.0.1 (Error Estimation Problem)**.** *Given a dirty dataset D, error type $\mathcal{E}$ and budget B, estimate the number of errors of type $\mathcal{E}$ while minimizing the relative error.*

6.1 Duplicate Estimation

In this section, we investigate the duplicate estimation problem and propose a series of algorithms to this end. We are given $\mathbf{C}$, the set of candidate tuple pairs with $|\mathbf{C}| = N$. $\mathbf{C}$ could be either the set of all $\binom{n}{2}$ tuple pairs or the output of a conservative blocking algorithm that ensures that almost all of the duplicates are still in $\mathbf{C}$. Our objective for the duplicate estimation problem is to reliably estimate the number of duplicate tuple pairs in $\mathbf{C}$ by asking at most B questions to the expert. We begin our discussion by connecting duplicate estimation to well studied problems in statistics, namely proportion estimation (Section 6.1.1) and distinct value estimation (Section 6.1.2). Unfortunately, directly applying those intuitive techniques does not provide satisfactory results. We thus propose a novel approach that

[1] This part of work was majorly done during my internship at QCRI.

combines ideas from the traditional statistical estimation problem with properties specific to duplicate estimation (Section 6.1.3).

6.1.1 Proportion Estimation

Our initial approach is based on the observation that duplicate estimation can be readily formulated as a proportion estimation problem that is well studied in Statistics [55]. We are given a population of N items and our objective is to estimate the proportion p of items that has a certain characteristic. A canonical example is to estimate the fraction of people who voted for a particular political party. In our case, the population is the set of tuple pairs in $\mathbf{C}$ and we are interested in the number of tuple pairs that have the characteristic that they denote the same real-world entity. We can accurately and trivially compute this fraction by asking each and every item from the population. When this is not feasible, one often resorts to sampling whereby one collects a sample of items through a pre-specified sampling strategy, computes the fraction of items in the sample that has the desired characteristic (called *sample proportion*) and then estimates the *population proportion* from it. Informally, our objective is to design a technique that produces an estimate of the proportion, $\widetilde{p}$, that is as close to the correct value p, as possible. Formally, the bias of an estimator is the difference between the *expected value* of the estimator and the true value of the population proportion. An estimator with a zero bias is said to be *unbiased*. The variance of an estimator measures how far the estimates are from the expected value. Ideally, we prefer an unbiased estimator with small variance.

Simple Random Sampling. A natural solution for proportion estimation is to randomly select B tuple pairs from $\mathbf{C}$ and ask for the labels from the domain expert. We denote this process as simple random sampling. If b out of B sampled tuple pairs are labeled as duplicate, then the sample proportion is computed as $\widetilde{p} = \frac{b}{B}$. Sampling theory states that the sample proportion is an *unbiased* estimator of the population proportion. Furthermore, we can state that $p \in [\widetilde{p} - \varepsilon, \widetilde{p} + \varepsilon]$ with confidence α where $\varepsilon = Z_{1-\alpha/2}\sqrt{\frac{\widetilde{p}(1-\widetilde{p})}{B}\frac{N-B}{N-1}}$ with $Z_{1-\alpha/2}$ is the $1-\alpha/2$ percentile of the standard Normal distribution [55, 84].

Simple random sampling is often effective when p is reasonably high. Unfortunately, the entity resolution problem has an extreme imbalance between the number of duplicates and non-duplicates. Recall that even under the benign scenario where every tuple has a corresponding duplicate, the ratio of duplicates to all possible tuple pairs is $O(n)$. Typically, the proportion of duplicates in $\mathbf{C}$ is very small and often $p \approx \frac{1}{n}$ (and often closer to $\frac{1}{n^2}$). In other words, we would expect to sample $O(n)$ tuple pairs before we expect to see a duplicate. With $B \ll n$, one may not even have a duplicate in the B sample tuple pairs. Furthermore, even though the estimator is unbiased, it has a large variance [55, 84].

Proposal Distribution-based Sampling. In addition to high query cost and large variance, simple random sampling suffers from two subtle issues. First, each of the sample is picked with exactly the same probability ($\frac{1}{N}$) and thereby has the same weight. Second, it does

not take into account certain ER specific properties such as monotonicity of precision [85] and the phenomenon of "messy middle" – thereby many of the samples provide redundant information. According to the monotonicity of precision assumption, a pair with a higher similarity will be a duplicate with higher probability than another pair with lower similarity. While this assumption may not always hold well in real-world data, it is often a good rule-of-thumb that could be exploited to improve sampling. Second, the "messy middle" phenomenon states that the tuple pairs with very high or very low similarity are often duplicates and non-duplicates respectively. Hence, one might get more information by sampling the tuple pairs in the messy middle - whose similarity is neither very high nor very low.

A natural approach to incorporate these insights is to sample the tuple pairs in $\mathbf{C}$ according to a proposal distribution $q(\cdot)$. Based on the discussion above, $q(x)$ will have a lower (but non-negative) weight towards tuple pairs with very high or very low similarity while other tuple pairs might have a higher likelihood of being selected. Let $\{x_1, x_2, \ldots, x_B\}$ be the samples picked by this approach with x_i being picked with a probability $\propto q(x_i)$. Then, the population proportion can be estimated as

$$\widetilde{p} = \frac{1}{\sum_{i=1}^{B} \frac{1}{q(x_i)}} \sum_{j=1}^{B} \frac{\mathbb{I}_j}{q(x_j)} \tag{6.1}$$

where $\mathbb{I}_j$ is an indicator function that returns 1 if the tuple pair x_j is a duplicate and 0 otherwise. If the proposal distribution $q(\cdot)$ is "well-aligned" with the true underlying distribution of duplicates, then this approach will provide a good estimate. In practice, designing an effective $q(\cdot)$ is very challenging and bad approximations can provide a substantially worse estimate than even simple random sampling. While the direct application of proposal distribution is not promising currently, we will leverage this crucial insight in Section 6.1.3 for designing a better duplicate estimator.

Stratified Sampling. One of the key issues of simple random sampling is that the information that one obtains from a tuple pair does not "transfer" to another tuple pair. Consider an extreme case where the candidate set $\mathbf{C}$ is partitioned into K classes $\{P_1, P_2, \ldots, P_K\}$. Also suppose that we know that the partitions are "homogeneous" (or pure) [86]. In other words, all the tuple pairs in partition P_i are either duplicates or non-duplicates. We can readily see that one can accurately estimate the number of duplicates by asking the expert to label one pair from each partition, totaling K questions. This thought experiment gives us the idea that if we can somehow partition $\mathbf{C}$ into a number of nearly homogeneous partitions, one can generate a better estimate of the number of duplicates with comparatively lesser query cost.

The key idea behind stratified sampling [55, 84] is to partition $\mathbf{C}$ into K partitions $\{P_1, P_2, \ldots, P_K\}$ with size $N_1, N_2, \ldots, N_K$ such that $\cup_{i=1}^{K} P_i = \mathbf{C}$ and $\forall i, j$ with $i \neq j$, $P_i \cap P_j = \emptyset$. Given a budget B, we split it into $B_1, \ldots, B_K$ such that B_i tuple pairs are selected uniformly at random from partition P_i. Let b_i be the number of duplicates identified by asking B_i questions in partition P_i with $\widetilde{p}_i = \frac{b_i}{B_i}$. Using these notations, an unbiased estimate for the population

proportion is:

$$\widetilde{p} = \sum_{i=1}^{K} \frac{N_i}{N} \sum_{j=1}^{b_i} \frac{\mathbb{I}_j}{B_i} \tag{6.2}$$

The unbiased estimate of variance of the stratified sampling is [55, 84, 86]:

$$Var(\widetilde{p}) = \sum_{i=1}^{K} \left(\frac{N_i}{N}\right)^2 \frac{\sigma_i^2}{B_i} = \sum_{i=1}^{K} \left(\frac{N_i}{N}\right)^2 \frac{\widetilde{p}_i \times (1-\widetilde{p}_i)}{(B_i - 1)} \tag{6.3}$$

In order to apply stratified sampling, we need to address two key questions: (1) how to stratify the data and (2) how to allocate the budget across different strata?

Stratification Strategy. In practice, identifying a set of partitions such that each strata is nearly homogeneous is quite tricky. Note that the "purity" of the partitions is specified based on whether (most of) the tuple pairs in that partition are either duplicates or non-duplicates. However, this information is not available and is the objective of the problem of entity resolution! One can alleviate this chicken-and-egg problem by *approximating* the notion of homogeneity with another metric. As mentioned above, the similarity function that computes a scalar similarity score is often a good substitute. Alternatively, if we are only provided with similarity vectors $\phi_{ij} \in \mathbb{R}^m$, then one can use simplistic similarity scores such as by assigning equal weight to all components of $sim(t_i, t_j) = \frac{1}{k}\sum_{k=1}^{m} \phi_{ij}[k]$. Then, we can partition the tuple pairs based on the similarity scores such that the variance within each strata is minimized. One can also use other classical approaches such as equi-size, *i.e.,* allocate an equal number of tuple pairs to each strata, and equi-width, *i.e.,* partition the similarity score into each strata. As an example, if $N = 2000$ and $K = 10$, then under equi-size strategy each partition will have 200 tuple pairs such that those with least score will be in P_1 while those with highest score will be in P_K. In equi-width strategy, one partitions the similarity space into $[0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0]$ and all the tuple pairs whose similarity score falls into the interval is assigned to it.

Budget Allocation Strategy. We next discuss an effective strategy to allocate the budget B to various strata such that P_i receives a budget B_i. There exist a number of traditional strategies such as (a) equal allocation where each strata receives the same number of budget with $B_i = \frac{B}{K}$ and (b) proportional allocation wherein larger partitions receive a larger budget with $B_i = \frac{N_i}{N} \times B$. There is also a known *optimal allocation* strategy that tries to minimize the variance of the estimation. Specifically, this approach takes into account both the size of the stratum and the variance of the data points within the stratum. While intuitive, this approach is not applicable in our scenario as we do not know the underlying duplicate characteristics. We now propose an iterative budget allocation strategy [86] that is adapted for duplicate estimation. Specifically, recall the messy-middle problem that we discussed above. If during the process of interacting with the expert, we come to realize that a particular stratum is relatively homogeneous (such as strata for very high or very low similarity), then we could allocate lesser labels to it in the future. We achieve this by leveraging the unbiased estimators for the proportion and variance from Equations 6.2 and 6.3. Algorithm 9 shows the pseudocode for our approach.

Listing 9: Iterative Stratified Sampling

1: **Input:** B, B_{init}, B_{step} : Total budget, exploration budget and per-iteration budget
2: Randomly sample and label B_{init} tuple pairs from each strata
3: Estimate $\widetilde{p}_k$ and $Var(\widetilde{p})$ from Equations 6.2 and 6.3
4: **while** budget is not exhausted **do**
5: Allocate B_{step} among strata based on variance estimation
6: Select samples based on the allocation and get labels
7: Update estimate of $\widetilde{p}_k$ and $Var(\widetilde{p})$ from Equations 6.2 and 6.3

6.1.2 Distinct Value Estimation

In this subsection, we briefly discuss - for the sake of completeness - two classical estimation problems that are seemingly related to error estimation. However, directly applying these estimators does not provide good results due to the properties specific to duplicate estimation.

Distinct Value Estimation Problem (DVE). Both the statistical and database communities have extensively studied the problem of estimating the number of distinct values of an attribute in a relation through sampling [52, 87, 88, 89]. This has a number of practical applications such as selectivity estimation and query optimization in databases. DVE is a more general problem than duplicate estimation and can be parameterized to estimate different metrics.

Species Estimation Problem (SE). Another problem that is seemingly related to duplicate estimation is that of species size estimation [52, 90, 91]. Informally, we obtain a sample S of B observations drawn from a finite population $\mathbf{C}$ through sampling with replacement. Suppose that we observed d distinct "species" (i.e., duplicates) from the sample. The species estimation problem seeks to estimate the number of distinct species in $\mathbf{C}$ using d and S. SE is an extremely well studied problem due to its practical applications in ecology and other natural fields.

Hardness of DVE and SE. Both DVE and SE are extremely challenging estimation problems. While there are estimators that provide good estimates for known distributions, they often have poor estimation quality for skewed distributions such as in our problem. In fact, [88] showed the following powerful negative result: There exists a worst case input for any strategy (possibly randomized and even adaptive) that examines at most B values such that it would incur an error of at least $\sqrt{\frac{n-B}{2B}\ln\frac{1}{\gamma}}$ with a probability of at least γ for every $\gamma > e^{-B}$.

Applying Estimators for DVE and SE for Error Estimation. Most of the estimators for DVE and SE work by computing the frequency distributions f_i on S. Specifically, f_1 is the number of singleton species that occurred exactly once in S, f_2 are those that occurred exactly twice, and so on. Hence, the problem of species estimation can be boiled down to estimating f_0 (the number of unseen species) given various f_i. We evaluated a number of well known estimators for both DVE and SE. However, we found in our empirical evaluation that they did not provide good results consistently. The key reason is that both DVE and

SE rely heavily on the parameters f_i to generate good estimates. However, in our problem setting, every duplicate is a distinct value. In other words, every distinct value (or species) either occurs once or very many times (as all non-duplicates belong to the same species). This reduces the amount of information available to the estimator.

Inapplicability of DVE and SE Estimators. It might seem that one can fix this issue by a different semantics for distinct values such as by measuring the number of real-world entities. As an example, if (t_i, t_j) and (t_j, t_k) are categorized as duplicates, then our current method considers them as two distinct values. Alternatively, one could consider these tuple pairs as one distinct value as all three tuples refer to the same entity. This leads to a natural semantics of f_i - the number of real-world entities that are represented by exactly i tuples in the sample. However, this "fix" results in a dramatic blowup in query cost. As an example, given a tuple pair (t_i, t_j), it is not sufficient to identify that they are duplicates. There might exist another tuple pair (t_k, t_l) that is not only duplicates but refer to the same real-world entity as (t_i, t_j). The only way to identify this is to compare all possible tuple combinations. Hence, given a budget of B, one can compare at most $\sqrt{B}$ tuples.

Another serious issue with these estimators is their impact on skew. Entity resolution is a famously unbalanced problem and straightforward adaptations of estimators for DVE and SE often produce bad results. We also considered a number of estimators that incorporate skew correction - such as Chao92 [52] - without much improvement. In summary, the estimators for DVE and SE are not suitable for error estimation due to the combination of poor performance under skew and high query cost. Adapting the rich work on DVE and SE for error estimation is an interesting open problem.

6.1.3 Duplicate Estimation via Classifier Evaluation

Our previous approaches sought to formulate our problem to well studied estimation problems of proportion, distinct values and species size. Informally, each of those approaches identifies the number of duplicate values in the sample and comes up with various statistical estimators to estimate the number of duplicates in the entire population. Despite the availability of rich theory behind these approaches, they suffer from the fact that ER specific properties are not exploited at all.

A Novel Two Step Approach. We propose a novel two step approach to estimate the number of duplicates with high accuracy. We observe that Machine Learning (ML) based techniques are extensively used for ER and often have state-of-the-art accuracy. If one could train a highly accurate ML model, then one can simply run it on $\mathbf{C}$ (or a representative sample of it) and generate a good estimate of the number of duplicates. However, this seemingly simple approach, does not work well in practice due to two reasons. First, ML models often require a lot of training samples that often exceed the budget B. Second, for many real-world datasets the state-of-the-art ML methods might have less than stellar performance. As an example, two ML methods trained on Products dataset in [35] had F-measure of 40.5

and 69.5 even though they used a training data set size of 3,205 and 36,076 tuple pairs respectively. We address this performance issue by allocating a portion of our budget to evaluating the performance of the classifier. If one generates a reasonably accurate estimate of the precision, recall and F-measure of a classifier, one can re-weight the estimation of the classifier. For example, if we estimated that the classifier has a recall of 0.5, then a simplistic correction would be to double its estimate of duplicates! Algorithm 10 provides the high level steps of our overall approach.

Listing 10: Duplicate Estimation as Classifier Evaluation

1: Partition budget B into B_{train} for training and B_{eval} for evaluation
2: Use B_{train} to train a ML classifier $\mathcal{H}$
3: Use B_{eval} to evaluate $\mathcal{H}$
4: d = #Duplicates identified by $\mathcal{H}$ in $\mathbf{C}$.
5: $\widetilde{d}$ = Re-weighted estimate of d based on classifier accuracy

In order to generate a good estimate using our approach, we need to effectively solve two non-trivial problems: (i) how to train a reasonably accurate ML classifier with a limited budget? and (ii) how to estimate the effectiveness of the classifier with reasonable accuracy under a limited budget?

Let us first consider the problem of training an effective classifier with a limited budget. We evaluated a wide variety of relevant techniques and describe two most promising approaches. Intuitively, simplistic selection of training set is not very effective. For example, a random selection strategy chooses B tuple pairs from $\mathbf{C}$ to be labeled and then uses its for training a classifier. As we discussed previously for proportion estimation, this approach ignores ER specific properties such as monotonicity of precision and messy middle phenomenon.

Active Learning-based Approaches. A more promising strategy is active learning [92], which is based on the observation that one can train an accurate classifier with comparatively less training data if the classifier is allowed to choose the training data. This is often achieved by asking an oracle to label a carefully selected set of tuples. The classical active learning seeks to minimize $0-1$ loss by reducing the number of both false positive and negatives. Not surprisingly, the unbalanced nature of ER makes this approach less effective. The solution is to propose a ER-aware active sampling strategy [85, 93] that seeks to maximize recall under a precision constraint wherein precision has to exceed a given threshold. We leverage an adaptation of algorithm *MinExpError* that was proposed in [94]. This algorithm combines the current classifier accuracy with label uncertainty by using sophisticated techniques from bootstrap theory. Informally, this algorithm seeks labels for data items that, when labeled differently from the current prediction of the classifier, would have a huge impact on the classifier's future predictions and thereby its accuracy. Empirically, this approach requires substantially less training data than when using the algorithm CVHull from [93].

Active Learning with Clustering. We next adapt an approach originally proposed in [95] that exploits the cluster structure of the data for active learning. Recall from our earlier discussion that sampling from nearly homogeneous clusters dramatically simplifies the estimation process. However, the key issue in Section 6.1.1 is that the similarity vector is only a naive approximation of strata homogeneity. We sidestep that obstacle through active learning by training a ML model interactively and using its preliminary predictions as a proxy for homogeneity. We begin with the similarity vectors for all tuple pairs in $\mathbf{C}$ and recursively split them into smaller and smaller subsets until these subsets are identified to be homogeneous. For each cluster, we identify the most informative tuple pairs to label and use the response to estimate the homogeneity of the cluster. If the homogeneity is high enough, all the tuple pairs in that cluster are assigned as either positive or negative training examples. If not, the cluster is partitioned and the process is repeated till the budget is exhausted or all clusters are sufficiently homogeneous.

Classifier Evaluation. Once a ML classifier $\mathcal{H}$ has been trained, a preliminary estimate of the number of duplicates can be obtained by running $\mathcal{H}$ on $\mathbf{C}$. If the classifier is very accurate, this estimate is also quite accurate. If not, the estimate has to be appropriately weighted to generate a better estimate. This can be achieved through the emerging area of classifier evaluation - that estimates the performance of a classifier on a test dataset without labels through sampling.

Estimators for Classifier Performance. A classifier for ER is often measured through three measures: precision, recall and F-measure. Our objective is to evaluate these measures by asking at most B_{eval} questions to the domain expert. Not surprisingly, the most obvious estimators also happen to be unbiased estimators [35, 73]. Let S be the set of sample tuple pairs that are used for labeling with $|S| = B_{eval}$. Let n_{tp} be the set of tuple pairs that were predicted as duplicate by $\mathcal{H}$ and validated by the expert. Let n_{fp} and n_{fn} be the set of tuple pairs that were predicted as duplicate (resp. not duplicate) by $\mathcal{H}$ but turned out to be non-duplicate (resp. duplicate). Note that in Equation 6.4, F-measure is a weighted combination of precision and recall. When $\alpha = 0$, resp. $\alpha = 1$, it boils down to precision, resp. recall. When $\alpha = 0.5$, it boils down the canonical definition of F-measure which is defined as the harmonic mean of precision and recall.

$$\begin{aligned} \widetilde{Precision}(\mathcal{H}) &= \frac{n_{tp}}{n_{tp}+n_{fp}} \\ \widetilde{Recall}(\mathcal{H}) &= \frac{n_{tp}}{n_{tp}+n_{fn}} \\ \widetilde{F-measure}_{\alpha}(\mathcal{H}) &= \frac{n_{tp}}{\alpha(n_{tp}+n_{fp})+(1-\alpha)(n_{tp}+n_{fn})} \end{aligned} \tag{6.4}$$

Approaches for Classifier Evaluation. Our objective is to estimate the accuracy of $\mathcal{H}$ on $\mathbf{C}$ by asking as few questions as possible. One can readily notice that classifier accuracy estimation boils down to the proportion estimation problem! For each tuple pair in $\mathbf{C}$, we

have a prediction from $\mathcal{H}$ as to whether it is a duplicate. We now need to estimate the proportion of $\mathbf{C}$ where the prediction of $\mathcal{H}$ matches the true nature of the tuple pair (duplicate or non-duplicate). While the approaches proposed in Section 6.1.1 are ineffective for *duplicate* estimation, they are very effective for *classifier accuracy* estimation. Informally, this is due to two reasons: (a) while the proportion of duplicate is very small (such as 0.01), even an adequate ML classifier has an accuracy that is often in excess of 0.5, and (b) for each tuple pair in $\mathbf{C}$, the classifier can provide the probability (according to itself) that it is likely to be a duplicate. This can be used to identify strata with minimal variance or design a sophisticated proposal distribution. There also a number of effective heuristics to handle the unbalanced nature through under- or over-sampling. For example, we could ensure that the set of labels that we ask to the expert has a near-equal mixture of duplicates and non-duplicates based on the classifier scores. This simple approach was shown to be effective in [35].

6.2 Functional Dependency Violation Estimation

Functional dependencies (FDs) are one of the most important form of integrity constraints used for data cleaning. In this section, we describe an effective interactive algorithm to estimate the number of FD detectable errors. The proposed approach can be adapted to address other variants of FDs such as conditional functional dependencies (CFDs) [96].

Feasibility of Sampling based Approaches. One might think that we could leverage some of the sampling- and ML-based techniques proposed in Sections 6.1 for estimating FD errors. Unfortunately, this is not the case. We can show by simple combinatorial arguments that sampling based approaches do not work [51]. To wit, there are $m \times (2^{m-1} - 1)$ candidate FDs each of which can hold in the relation. There are $2^{m-1} - 1$ possible FDs with a given attribute A_i in the RHS or containing A_i as part of the determinant. Furthermore, if one randomly samples a pair of tuples, it is unlikely that they violate any FD at all! Similarly, ML-based approaches also do not work as they often rely on statistical approaches while FD discovery is often a logical problem. While there has been some work on formulating FD discovery as a logical induction problem [97], they do not scale for large datasets [98, 99]. Hence, we need a new approach to estimate FD errors.

Interaction with the Expert. There are two possible ways to interact with the expert – cell-based and FD-based. We focus on the cell-based questions where the expert is shown a pair of tuples (t_i, t_j) that differ in some carefully chosen attributes. We ask if the difference in value constitutes a violation. As an example, we might show two tuples that have the same value for attribute A_i but different value for A_j. The expert might then confirm that the discrepancy is indeed erroneous. However, note that we do not ask the expert if $A_i \rightarrow A_j$ a valid FD. This is often a substantially harder question as the expert might be able to identify some discrepancies but might not be able to fully specify the perfect FD. Another issue is that the number of approximate rules often outnumbers the number of valid rules [54]. Furthermore,

we do consider the FD-based questions as a baseline and our proposed approach was able to outperform it. We describe additional details in Section 6.3.

Our Approach: Candidate Generation and Testing. We leverage a candidate generation and testing based approach that was first proposed in [51]. Intuitively, we generate a candidate set of FDs that are likely to be true and interactively validate these FDs with the expert by asking carefully chosen cell-based questions. While our candidate generation approach is similar to [51], our testing step is significantly different as it needs to be adapted for error estimation rather than error detection as in [51].

Candidate Generation. Our first step is to generate a set of candidate FDs that could be violated by the relation. Consider a valid FD $A \rightarrow C$ that holds in the "clean" version of **D**. However, due to the errors in **D**, this FD does not hold. Specifically, there exists at least a pair of tuples that have the same value for A but different values for C. If we run a traditional exact FD discovery algorithm, it will not identify this FD. However, even if $A \rightarrow C$ does not hold in **D**, a specialization of it - say $AB \rightarrow C$ (or $ABD \rightarrow C$ etc) has to hold in **D**. This provides a natural mechanism to generate a set of candidate FDs. We use a traditional exact FD discovery algorithm to get the list of exact FDs and systematically generalize these FDs by removing say one and two attributes. For example, we obtain $AB \rightarrow C$ and the candidate set now includes $\{AB \rightarrow C, A \rightarrow C, B \rightarrow C\}$. Alternatively, one can run an approximate FD discovery algorithm that identifies all FDs with violations less than some small threshold such as 5% or 10%. However, identifying this threshold is often challenging. Setting it too low might miss some true FDs while setting it to a very high value results in a deluge of false FDs.

Candidate Testing. Once the set of candidate FDs is obtained, we could use it to identify all possible candidate violations. These are the set of tuples that have the same set of LHS for some FD but with a different RHS. Due to the way the candidate FDs were generated, there might be a number of false FDs and thereby false violations. Our objective now is to issue a set of carefully crafted questions to the expert such that the size of true violations could be estimated in a fast manner. [51] proposed a bipartite graph based formulation where the nodes in one partition correspond to FDs while the nodes in other partition correspond to violations. An edge exists between a node for a FD and a node for a violation if that FD identified the violation. One can then greedily validate violations identified by multiple FDs with the expert.

While this approach is effective, it is possible to improve it dramatically as our objective is estimation rather than detection. Specifically, we use multiple sophisticated approaches such as Armstrong's axioms to rapidly prune incorrect FDs. For example, consider the pair of FDs: $f_1 : A \rightarrow C$ and $f_2 : AB \rightarrow C$. If one knows that f_1 is true, then we know that f_2 is also true. Contrapositively, if we know that f_2 is a false FD, then f_1 cannot be a true FD. Similarly, one could use other Armstrong inference rules to perform even more sophisticated pruning. The above observation also gives us a new way to select which violation to ask the

expert about. Unlike [51] which asked any violation, we choose a violation such that it is identified by f_1 but not f_2. Answering this often provides much more information about both the FDs than picking an arbitrary violation. One could even identify a violation that exists in the difference between multiple pairs of FDs for additional information.

To achieve this, we propose a bipartite formulation wherein the nodes in one partition corresponds to a pair of FDs such as f_1 and f_2 that differ in one attribute. The nodes in the other partition corresponds to violations that are detected by one FD but not the other. We store the difference in the number violation with each FD-pair node (i.e. $|Violation(f_1) \setminus Violation(f_2)|$. Each violation node gets a weight that is the sum of all its FD-pair violation sizes. In other words, if a given violation is present in the set difference of multiple pairs of FDs, it will get a higher weight. Our algorithm proceeds as follows. We employ a greedy approach that asks FD pairs with the largest difference in their respective violations. For the chosen FD pair (FD_i, FD_j), we ask the violation with the highest weight. If the expert says that the violation is not a true violation, we can prune all FDs that identified this violation and their generalizations. For example, if the violation identified by f_2 is a false violation, then both f_2 and f_1 are false FDs that can be pruned. We repeat this process till budget is exhausted. Finally, we return the size of all the violations of the remaining candidate FDs as our estimate. Algorithm 11 shows the pseudocode for our approach.

Listing 11: Estimate FD Errors

```
1: Generate candidate set of FDs
2: Construct the bipartite graph and compute the weight of nodes in both partitions
3: while budget is not exhausted do
4:     Identify FD pair (FD_i, FD_j) with largest difference in violation size
5:     Identify the violation in Violation(FD_j) \ Violation(FD_i) with largest weight
6:     if the expert validated it as false violation then
7:         Prune the false positive FDs
8: Return the number of violations for the remaining candidate FDs
```

6.3 Evaluation

6.3.1 Experimental Setup

Hardware and Platform: All our experiments were performed on a quad-core 2.2 GHz machine with 16 GB of RAM. The algorithms were implemented in Python and Java. We used Metanome [100] data profiling tool for discovery of functional dependencies and Scikit-Learn [101] library for building ML models. We used Magellan [56] for performing feature engineering and blocking for ER tasks.

Datasets for Duplicate Estimation. We conducted extensive experiments over 7 different datasets each of which are popular benchmark datasets that are extensively evaluated by prior ER work using both ML and non-ML based approaches. Table 6.1 provides some

statistics of these datasets. The number of tuples in a source varies from about 500 to more than 2.5 million while the Cartesian product varies between 15,000 to more than 4.5 trillion. They also exhibit a wide variety of data characteristics such as ratio of duplicates to non duplicates. Some of these datasets are "easy" for ER while others are "challenging". On easy datasets, ML based approaches often achieve a F-score of 0.9. The challenging datasets often have unstructured attributes such as product description and traditional ML methods have a F-score between 0.6 and 0.7.

Dataset	#Tuples	#Duplicates	#Attr
DBLP-ACM (Pub-DA)* [102]	(2,616 - 2,294)	2,224	4
DBLP-Scholar (Pub-DS)* [102]	(2,616 - 64,263)	5,347	4
DBLP-Citeseer (Pub-DC)* [103]	(1,823,978 - 2,512,927)	558,787	4
Abt-Buy (Prod-AB)‡ [102]	(1,081 - 1,092)	1,097	4
Amazon-Google (Prod-AG)‡ [102]	(1,363 - 3,226)	1,300	5
Walmart-Amazon (Prod-WA)‡ [103]	(2,554 - 22,074)	1,154	17
Fodors-Zagat (Rest-FZ)* [104]	(533 - 331)	112	7

Table 6.1: Dataset Statistics - * refer to easy datasets while ‡ to challenging ones.

Datasets for Functional Dependency Estimation. We conducted our experiments over one synthetic and one real-world datasets. We use the Tax generator from [105] to generate a synthetic dataset with 100K tuples with taxpayer information such as name, gender, address, salary along with tax rates and exemptions. The Hospital dataset consists of health-care provider information from USA Medicare scheme with 115K tuples. The attributes include provider/hospital name, addresses, and type of services provided. The numbers of exact FDs on these datasets are 364 and 83 respectively.

Error Generation for FD Estimation. We used BART [106], a state-of-the-art system for benchmarking data-cleaning algorithms for generating FD violations. Given a set of FDs, BART can introduce detectable errors in a customizable manner. For Hospital and Tax datasets, we generated two types of dirty datasets. In the *uniform* error model, each of the FD generates a similar number of violations. Under the *systematic* error model, the number of violations generated per FD is skewed. We generated multiple dirty datasets where the fraction of FD violations varied from 5% to 25%. We also varied the number of FDs involved in generating violations from 20% all the way to 100%. For example, 20% of the FDs could generate all the FD violations that cover 25% of the entire dataset. Since Tax has 364 FDs, this will mean that around 37 randomly chosen FDs generated all the errors. Under the uniform model, each of those 37 FDs generate errors equally while under systematic model, approximately 20% of them (16 FDs) account for 80% of the errors. For each combination, we generated 10 different dirty datasets using BART.

Workflow. We assume that each interaction with the expert counts for 1 unit of budget. Hence, asking whether a pair of tuples is a duplicate or whether a tuple is an outlier or

whether two tuples have a FD violation all cost exactly the same. As mentioned in Section 2.2, we assumed the availability of similarity functions for each attribute that is used for ER. For ER, we used Magellan to automatically identify appropriate similarity functions for each attribute and a conservative blocking function that ensures all duplicates are in the candidate set. Once the initial setup is done, we interactively ask appropriately chosen questions to the expert and based on the results update our internal model.

Algorithms. As mentioned in Section 6.1, directly using estimators for population proportion or distinct values does not lead to an acceptable performance. Hence, we primarily concern ourselves with the active learning based algorithms followed by classifier evaluation as described in Section 6.1.3. For FDs, we implemented the algorithm described in Algorithm 11.

Performance Measures. We evaluate the performance of our algorithms through two factors: (a) the cost imposed on the expert based on the different types of questions asked and (b) the relative error of the estimate defined as $\frac{|actual - estimate|}{actual}$. A smaller relative error is preferable.

6.3.2 Duplicate Estimation

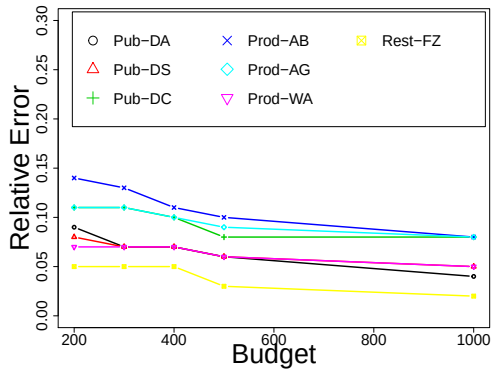


Figure 6.1: Budget vs Relative Error (Duplicate Estimation)

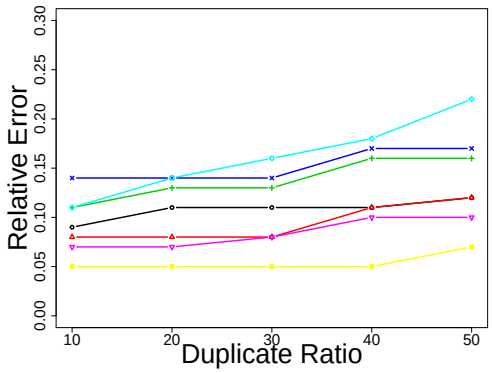


Figure 6.2: Varying Duplicate to Non-Duplicate Ratio

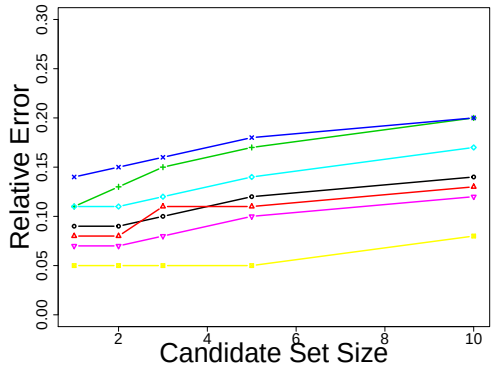


Figure 6.3: Varying Size of Candidate Set for Duplicate Estimation

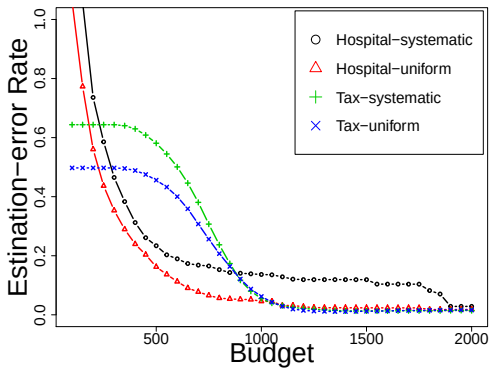


Figure 6.4: Budget vs Relative Error (FD Estimation)

In this subsection, we evaluate our algorithms for duplicate estimation. As mentioned before, we do not evaluate the baseline algorithms that use estimators for population proportion and distinct values. They simply were not competitive with the algorithms proposed in our thesis even when given 5-10x budget compared to our classifier evaluation based algorithms.

Budget vs Relative Error. In our first set of experiments, we vary the budget provided to our algorithm and measure the relative error of our estimation. Figure 6.1 shows the results of the experiment. We allocated 75% of the budget for training a classifier and 25% for classifier accuracy estimation. Note that this includes validation of both duplicates and non-duplicates. As expected, our algorithms start providing estimates with a relative error of less than 10% even with as little as 200 questions. This behavior is consistent with both easy and challenging datasets - albeit for different internal reasons. For easy datasets, one can actually achieve reasonably accurate classifier using active learning with a small budget and can achieve competitive relative error. For challenging datasets, we might only achieve middling performance (say F-measure around 0.5) in the given budget. However, since our classifier evaluation component has correctly evaluated this performance, it "corrects" the estimate appropriately thereby improving the relative error. Our performance improves with increasing budget but at a slower rate and often plateaus around 5% after a budget of 500 questions. Note that in order to improve performance of our algorithm, we need to improve either (or ideally both) the ML classifier and the classifier evaluation. The former is challenging due to the inherent complexity of the ER problem that often requires substantially large number of training examples to generalize well - especially on the datasets that we marked as challenging. The latter is challenging as the classifier itself might have a hidden "bias" (systematic mistakes for certain class of tuples) and estimating it is trickier through purely sampling based approaches if the affected tuples are small in number.

Impact of Duplicates to Non Duplicates Ratio. One of the key challenges in ER is the unbalanced nature of the dataset. Hence we evaluated how our approach is impacted by the ratio of duplicates to non duplicates. Recall that this factor has an especially deleterious impact on the proportion estimation based algorithms. Figure 6.2 shows the result of this experiment. For each dataset, we systematically removed 10%, 20%, ..., of the duplicates and ran our algorithm. Interestingly, the impact of removing duplicates is surprisingly mild. This is due to two reasons. First, our active learning based approach chooses appropriate pairs of tuples for labeling as it is less affected by the imbalance of the dataset. Second, our classifier evaluation based approach is still effective as the impact of reduced duplicates have a proportionally smaller impact on the classifier accuracy. For example, the classifier might have had 70% accuracy on the original dataset while it might have 60% accuracy on the version with only 50% duplicates remaining. However, the proportion is still large enough to be effectively estimated by our approach.

Impact of Size of Candidate Set. Another key dimension in the efficiency of our algorithm is the size of the candidate set that contains (almost) all duplicates. For our experiments,

we used a conservative blocking function using Magellan to reduce the candidate set size. In this experiment, we vary the size of the candidate set to 2x, 3x, ..., 10x. For some of the smaller datasets, we also evaluated our algorithm on the candidate set that contained all possible $\binom{n}{2}$ tuple pairs as the candidate set. Figure 6.3 shows the result of the experiments. As expected, the size of the candidate set has at best a mild impact on the accuracy. The rationale is similar to that of the previous experiments. Due to the use of active learning followed by classifier accuracy estimation, we are less affected by these issues. However, this does have a substantial impact on the running time of the algorithm. Recall that a key of our active learning based approach is to identify the tuple pairs that could have most impact on future classifier accuracy. Estimating it becomes very time consuming when the size of the candidate set increases.

6.3.3 FD Violation Estimation

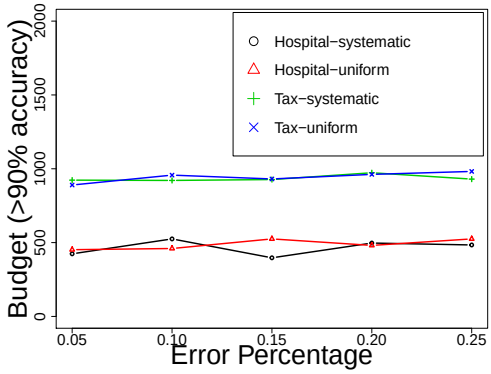


Figure 6.5: Error Injection Rate vs Budget

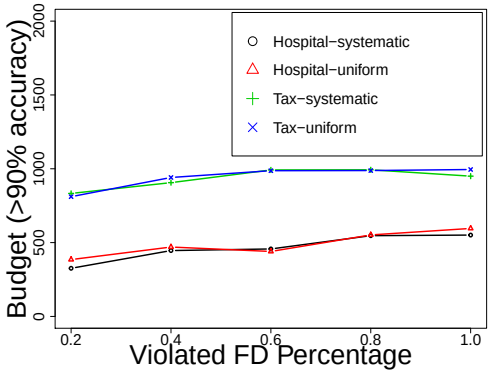


Figure 6.6: #Erroneous FDs vs Budget

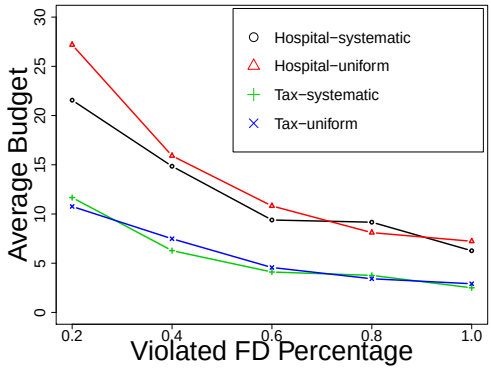


Figure 6.7: #Erroneous FDs vs Cost per FD

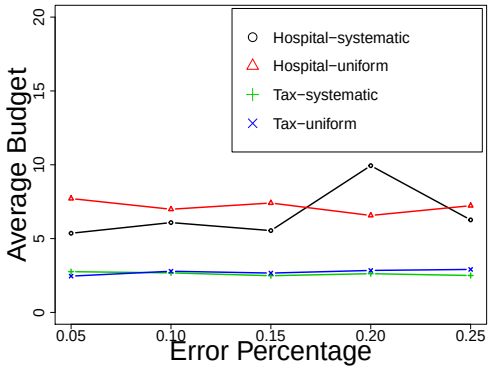


Figure 6.8: Error Injection Rate vs Cost per FD

In this subsection, we evaluate our algorithms for estimating FD violations. Recall that we generated multiple datasets by varying the injection rate and the number of erroneous FDs. Unless otherwise stated, the experimental results are obtained by averaging the results across all these variants.

Budget vs Relative Error. In our first set of experiments, we evaluate the impact of budget on relative error. Figure 6.4 shows the result of the experiments. As expected, the relative error decreases with increasing budget. Note that the reduction is especially steep at the beginning which indicates that our algorithms chooses questions with high utility in reducing relative error early on. In this experiment, we considered a pessimistic setting where the set of candidate FDs is especially large. Given a set of exact FDs that hold on the dirty dataset, we consider all generalizations of these FDs such that they violate at most 10% of the relation as a potential erroneous FD and add it to the candidate set. This is an extremely conservative approach to generate the candidate set and not surprisingly, the number of false positive FDs outnumbers the number of true erroneous FDs. If the domain expert could give us some additional information such as no FD can violate more than x% of the dataset - even if they do not know how dirty their entire relation is - this might result in a dramatic reduction in query cost. For example, considering only FDs with at most 5% violation dramatically reduces the query cost with only minimal impact on the relative error. Even under this adverse setting, our algorithms could achieve a relative error of 0.1 within 1000 interactions with the expert.

Error Injection Rate vs Budget. We next consider the impact of the error injection rate (or the "dirtiness" of the data) on our algorithm. Figure 6.5 shows the budget that is required to achieve a relative error of 0.1. Not surprisingly, the dirtiness of the data does not have a major impact on the query cost. One requires a similar number of questions for estimating the dirtiness of the data whether 5% of the data contains some FD violation or 25% of it does. This is due to the fact that our estimation algorithm uses a two phased approach of candidate generation and testing. In other words, the major factor that impacts the budget is simply the size of the candidate set rather than the number of erroneous FDs that generate violations.

#Erroneous FDs vs Budget. We next evaluate the impact of the number of FDs that were used to generate violations in BART. Figure 6.6 shows the result of this experiment where we measure the number of expert interactions needed to achieve a relative error of 0.1. As expected, the query cost increases linearly with the number of erroneous FDs. Once again, this is due to the candidate generation and testing based approach that we adopted in our thesis. This is also corroborated by the fact that Hospital dataset, which had a much lesser number of FDs than the Tax dataset, also has reduced query cost to achieve the same relative error.

Cost Per Erroneous FD. Our next set of experiments evaluates the average cost required to achieve a fixed relative error. Figure 6.7 shows the result where we fixed the error injection rate to 0.25. One can see that the average number of expert interactions required per erroneous FD decreases with increasing number of erroneous FDs as our algorithm could infer and prune a number of these FDs. Figure 6.8 shows the result of our experiments where we fixed the fraction of violated FDs to 10%. We can see that the cost per erroneous FD remains flat when the database dirtiness is varied. This once again confirms the fact that our

algorithmic approach is impacted more by the number of erroneous FD than by the fraction of the dataset that is in violation.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have proposed solutions to reducing the human effort required by web data extraction tasks in three different scenarios:

(I) Data demand is unclear. To save human effort—which is inevitable in figuring out the user's data demand in this Scenario—we have presented a wrapper induction algorithm that combines the ability to deal with noisy samples with a robustness that allows generated wrappers to survive, often until the selected data are no longer present on the page. The generated wrappers are specified in dsXPath which is a fragment of XPATH that is **(1)** *general enough* to cover all samples with 100% precision and recall and **(2)** *restricted enough* as to enforce noise resistant queries. With our method, users can either manually specify or use annotators to make their data demand clear, and they do not need to worry much about the noise in the annotations. Most importantly, generated wrappers are not only accurate, but are also immune to webpages' changes for hundreds of days. Such robust wrappers will save users' effort for re-annotating the pages.

(II) Data demand is clear. We have considered a common problem in this scenario, namely data extraction from result pages. We have proposed a fully-unsupervised method, called RED, that exploits the publishing pattern of result and detail pages. Namely, the contents published on result pages and detail pages usually overlap, i.e., the object published in some result record is also published on its corresponding detail page, and the attributes published in the result record are also published in its corresponding detail page. Wrt previous unsupervised methods, our method can achieve a significantly higher accuracy while automatically selecting only relevant attributes. Wrt previous supervised methods, our method can scale to a large number of websites while achieving similar accuracy. And wrt DIADEM, it can scale to several domains without requiring any human intervention for each of them. Human effort is greatly saved when users extract data from various domains on a large scale.

(III) Data quality is unknown. We have studied an important but not well-addressed problem: estimating the number of errors in a given dataset. In particular, we have focused on two types of errors that commonly arise in practice: duplicates and violations of functional dependencies. We have proposed a series of error estimation approaches by adapting,

extending, and synthesizing a number of recent innovations in diverse areas such as active learning, classifier calibration, F-measure estimation, and interactive training and evaluation. As verified by the extensive experimental study, our proposals can achieve very good estimations using a relatively small budget, compared with the budget needed to actually detect all errors. Our solutions will enable users to eliminate worthless sites from extraction tasks in a timely manner and guide data owners about what types of errors they should clean and how much effort they should put into it.

7.2 Future Directions

Robust and noise-resistant wrapper Induction. Our experiments on robust and noise resistant wrapper induction point to a number of further directions we can take to improve our methods: **(1)** Extending the method to deal with *multi-node wrappers*—where not only a single item or list of items, but multiple related items are to be extracted—is a natural step forward. Our method is already designed to allow the induction not only of absolute, but also of relative expressions and thus should be a great fit for a multi-node wrapper, as in [17]. **(2)** *Learning an effective scoring* for different types of node types, textual values, and axes from a given corpus of websites. This is particularly relevant for distinguishing different text fragments used to identify the same node and for best reflecting the specifics of a scenario. **(3)** Entity extractors, as used in information extraction or to generate automatic annotations in, e.g., [48], can provide information about the type of entity (e.g., "director") of a piece of text. Using such types in a wrapper to identify the containing node may yield more robust wrappers. **(4)** No matter how sophisticated the wrapper language or scoring, without constraints on the shape of future versions of a page, the robustness of a single wrapper is always limited. Therefore, we can now *induce multiple wrappers* that use a variety of independent means to select a target node.

Unsupervised web data extraction from detail pages. There are other extraction tasks where data demand is clear, such as extracting data from detail pages. Current unsupervised approaches [2, 80, 107] take a set of detail pages with the same template and infer their target data. As we discussed, these methods are highly sensitive to noise in repeated HTML structures. Although our RED system currently works for result pages, it is also possible to exploit the redundancy among result and detail pages to guide the data extraction from detail pages. However, this is a more difficult problem, since only a subset of data on detail pages have redundancy. We will try to utilize both of the repeated HTML structures and content redundancy.

Error Estimation Our work on error estimation is merely our first attempt at this problem. Our furture work on this topic will be to **(1)** *study the problem of estimating the number of other error types,* such as outliers and other forms of integrity constraint violations, and

(2) *reduce user interactions.* Our solutions still require human interactions, although at a minimal number. We will further investigate how to estimate errors without any human interactions.

Apply Web Data Extraction techniques to Web Data Cleaning. In *Data Cleaning* process, the difficulty of *Error Detection* is primarily caused by the toughness of learning correct rules. It is sometimes a "chicken-and-egg" problem that correct rules required for detecting noises have to be learned from a cleaned dataset. For example, to detect FD violations in a dataset, we have to know exactly the correct FD rules, which can only be learned from a clean dataset. To solve such problems, current solutions such as [51] first "guess" a set of possible rules and refine them through interactions with users. This problem, when narrowed down to the scope of web data, could have other solutions. We noticed that the data quality of small and not well-known websites is usually bad while those prominent websites usually publish relatively clean data. Rules learned from the data extracted from those good sources can be used to clean data from other sites. However, the practical situation might be somehow different because even such good sources also contain noises. Our solutions will need to reconcile conflicts among rules learned from several relatively reliable sources.

Incorporate Duplicate Estimation into Entity Resolution Pipelines. In some sense, our Duplicate Estimation method is a partial solution to the Entity Resolution problem. The trained and evaluated classifier for the Duplicate Estimation is already capable of detecting a subset of the duplicates. To increase its precision and recall, we can use those classifier based entity resolution methods to further train it. Moreover, based on the training examples we have already adopted and the estimated accuracy of the classifier in the Duplicate Estimation process, we can get a sense of what training examples are more preferable for the further training. Thus, in the future, we will consider how to properly incorporate our Duplicate Estimation method into existing Entity Resolution pipelines.

Bibliography

[1] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *Proc. of VLDB*, pages 109–118, 2001.

[2] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proc. of SIGMOD*, pages 337–348, 2003.

[3] Chia-Hui Chang and Shao-Chen Lui. IEPAD: information extraction based on pattern discovery. In *Proc. of WWW*, pages 681–688, 2001.

[4] Jiying Wang and Fred H. Lochovsky. Data extraction and label assignment for web databases. In *Proc. of WWW*, pages 187–196, 2003.

[5] Bing Liu, Robert Grossman, and Yanhong Zhai. Mining data records in web pages. In *Proc. of SIGKDD*, pages 601–606, 2003.

[6] Kai Simon and Georg Lausen. ViPER: Augmenting Automatic Information Extraction with visual Perceptions. In *Proc. of CIKM*, pages 381–388, 2005.

[7] Yanhong Zhai and Bing Liu. Web Data Extraction Based on Partial Tree Alignment. In *Proc. of WWW*, pages 76–85, 2005.

[8] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. Fully Automatic Wrapper Generation For Search Engines. In *Proc. of WWW*, pages 66–75, 2005.

[9] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, and Ashwin Tengli. Exploiting content redundancy for web information extraction. In *Proc. of WWW*, pages 1105–1106, 2010.

[10] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and integration of partially overlapping web sources. *PVLDB*, 6(10):805–816, 2013.

[11] Kristina Lerman, Lise Getoor, Steven Minton, and Craig A. Knoblock. Using the structure of web sites for automatic segmentation of tables. In *Proc. of SIGMOD*, pages 119–130. ACM, 2004.

[12] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visula Web Information Extraction with Lixto. In *Proc. of VLDB*, page 119–128, 2001.

[13] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper Induction for Information Extraction. In *Proc. of IJCAI*, 1997.

[14] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine learning*, 34(1-3):233–272, 1999.

[15] Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In *Proc. of AGENTS*, pages 190–197. ACM, 1999.

[16] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information systems*, 23(8):521–538, 1998.

[17] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. DIADEM: thousands of websites to a single database. *PVLDB*, 7(14):1845–1856, 2014.

[18] Ziawasch Abedjan, Cuneyt G Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. Temporal rules discovery for web data cleaning. *PVLDB*, 9(4):336–347, 2015.

[19] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. Corroborating information from disagreeing views. In *Proc. of WSDM*, pages 131–140, 2010.

[20] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? *PVLDB*, 6(2):97–108, 2012.

[21] Jeff Pasternack and Dan Roth. Making better informed trust decisions with generalized fact-finding. In *Proc. of IJCAI*, pages 2324–2329, 2011.

[22] Bo Zhao, Benjamin IP Rubinstein, Jim Gemmell, and Jiawei Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

[23] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *PVLDB*, 1(2):1241–1252, 2008.

[24] Ali Mesbah, Arie van Deursen, and Stefan Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Trans. Web*, 6(1):3:1–3:30, March 2012.

[25] Manuel Álvarez, Juan Raposo, Alberto Pan, Fidel Cacheda, Fernando Bellas, and Víctor Carneiro. Crawling the content hidden behind web forms. In *Proc. of ICCSA*, pages 322–333, 2007.

[26] Andrea Calì and Davide Martinenghi. Querying the deep web. In *Proc. of EDBT*, pages 724–727, 2010.

[27] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *PVLDB*, 1(2):1241–1252, 2008.

[28] Arnaud Sahuguet and Fabien Azavant. Wysiwyg web wrapper factory (w4f). In *Proc. of WWW*, 1999.

[29] Georg Gottlob and Christoph Koch. Logic-based web information extraction. *ACM SIGMOD Record*, 33(2):87–94, 2004.

[30] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. of SIGKDD*, pages 39–48, 2003.

[31] Ivan Fellegi and Alan Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64 (328), 1969.

[32] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proc. of KDD*, pages 269–278, 2002.

[33] William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. of SIGKDD*, pages 475–480, 2002.

[34] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[35] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *Proc. of SIGMOD*, pages 601–612, 2014.

[36] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. Deeper–deep entity resolution. *arXiv preprint arXiv:1710.00597*, 2017.

[37] Sebastian Kruse, Anja Jentzsch, Thorsten Papenbrock, Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. RDFind: Scalable Conditional Inclusion Dependency Discovery in RDF Datasets. In *Proc. of SIGMOD*, pages 953–967, 2016.

[38] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. Divide & conquer-based inclusion dependency discovery. *PVLDB*, 8(7):774–785, 2015.

[39] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. Scalable discovery of unique column combinations. *PVLDB*, 7(4), 2013.

[40] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[41] 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.

[42] Nigel Hamilton. The mechanics of a deep net metasearch engine. In *WWW (Posters)*, 2003.

[43] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. Crowdsourcing large scale wrapper inference. *Distributed and Parallel Databases*, 33(1):95–122, 2015.

[44] Bettina Fazzinga, Sergio Flesca, and Andrea Tagarelli. Learning robust web wrappers. In *Proc. of DEXA*, pages 736–745, 2005.

[45] Bettina Fazzinga, Sergio Flesca, and Andrea Tagarelli. Schema-based web wrapping. *Knowl. Inf. Syst.*, 26(1):127–173, 2011.

[46] Nilesh N. Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proc. of SIGMOD*, pages 335–348, 2009.

[47] Aditya G. Parameswaran, Nilesh N. Dalvi, Hector Garcia-Molina, and Rajeev Rastogi. Optimal schemes for robust web extraction. *PVLDB*, 4(11):980–991, 2011.

[48] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. Automatic wrappers for large scale web extraction. *PVLDB*, 4(4):219–230, 2011.

[49] Nora Derouiche, Bogdan Cautis, and Talel Abdessalem. Automatic extraction of structured web data with domain knowledge. In *Proc. of ICDE*, pages 726–737, 2012.

[50] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proc. of WIDM*, pages 9–16, 2008.

[51] Saravanan Thirumuruganathan, Laure Berti-Equille, Mourad Ouzzani, Jorge-Arnulfo Quiane-Ruiz, and Nan Tang. Uguide: User-guided discovery of fd-detectable errors. In *Proc. of SIGMOD*, pages 1385–1397, 2017.

[52] Yeounoh Chung, Sanjay Krishnan, and Tim Kraska. A data quality metric (dqm). *PVLDB*, 10(10), 2017.

[53] Arvid Heise, Gjergji Kasneci, and Felix Naumann. Estimating the number and sizes of fuzzy-duplicate clusters. In *Proc. of CIKM*, pages 959–968, 2014.

[54] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[55] William G Cochran. *Sampling techniques*. John Wiley & Sons, 2007.

[56] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[57] Simmetrics: A java library of similarity and distance metrics. https://github.com/Simmetrics/simmetrics.

[58] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27, 2003.

[59] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537–1555, 2012.

[60] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129–149, 1995.

[61] Sergio Flesca, Giuseppe Manco, Elio Masciari, Eugenio Rende, and Andrea Tagarelli. Web wrapper induction: A brief survey. *AI Commun.*, 17(2):57–61, 2004.

[62] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proc. of SIGMOD*, pages 2201–2206, 2016.

[63] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[64] Felix Naumann and Melanie Herschel. An introduction to duplicate detection. *Synthesis Lectures on Data Management*, 2(1):1–87, 2010.

[65] Fei Chiang and Renée J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

[66] 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.

[67] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *TKDE*, 23(5):683–698, 2011.

[68] Shaoxu Song and Lei Chen. Discovering matching dependencies. In *CIKM*, pages 1421–1424. ACM, 2009.

[69] Xu Chu, Ihab F Ilyas, Paolo Papotti, and Yin Ye. RuleMiner: Data quality rules discovery. In *Proc. of ICDE*, pages 1222–1225. IEEE, 2014.

[70] Christoph Sawade, Niels Landwehr, and Tobias Scheffer. Active estimation of f-measures. In *Proc. of NIPS*, pages 2083–2091, 2010.

[71] Paul N Bennett and Vitor R Carvalho. Online stratified sampling: evaluating classifiers at web-scale. In *Proc. of CIKM*, pages 1581–1584. ACM, 2010.

[72] Gregory Druck and Andrew McCallum. Toward interactive training and evaluation. In *Proc. of CIKM*, pages 947–956, 2011.

[73] Neil G Marchant and Benjamin IP Rubinstein. In search of an entity resolution oasis: Optimal asymptotic sequential importance sampling. *PVLDB*, 10(5), 2017.

[74] Georg Gottlob, Christoph Koch, Reinhard Pichler, and Luc Segoufin. The complexity of XPath query evaluation and XML typing. *Journal of the ACM (JACM)*, 52(2):284–335, 2005.

[75] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proc. of ACL*, pages 363–370, 2005.

[76] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proc. of WSDM*, pages 441–450, 2010.

[77] Steven DeRose and James Clark. XML path language (XPath) version 1.0. W3C recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[78] Arnaud Le Hors, Philippe Le Hégaret, et al. Document object model (dom) level 3 core specification. W3C technical reports, W3C, November 2004. https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/.

[79] Tim Furche, Georg Gottlob, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. AMBER: Automatic Supervision for Multi-Attribute Extraction. *CoRR*, arXiv:1210.5984, 2012.

[80] Valter Crescenzi and Paolo Merialdo. Wrapper inference for ambiguous web pages. *Applied Artificial Intelligence*, 22(1&2):21–52, 2008.

[81] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Wrapper generation for overlapping web sources. In *Proc. of Web Intelligence*, pages 32–35, 2011.

[82] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proc. of IIWeb*, pages 73–78, 2003.

[83] Yanhong Zhai and Bing Liu. Structured Data Extraction from the Web Based on Partial Tree Alignment. *TKDE*, 18(12):1614–1628, 2006.

[84] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[85] Arvind Arasu, Michaela Götz, and Raghav Kaushik. On active learning of record matching packages. In *Proc. of SIGMOD*, pages 783–794, 2010.

[86] Anurag Kumar and Bhiksha Raj. Classifier risk estimation under limited labeling resources. *arXiv preprint arXiv:1607.02665*, 2016.

[87] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. of VLDB*, pages 311–322, 1995.

[88] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Towards estimation error guarantees for distinct values. In *Proc. of SIGMOD*, pages 268–279, 2000.

[89] Rajeev Motwani and Sergei Vassilvitskii. Distinct values estimators for power law distributions. In *Proc. of ANALCO*, pages 230–237, 2006.

[90] Irving J Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

[91] Anne Chao. Species estimation and applications. *Encyclopedia of statistical sciences*, 2005.

[92] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[93] Kedar Bellare, Suresh Iyengar, Aditya G Parameswaran, and Vibhor Rastogi. Active sampling for entity matching. In *Proc. of SIGKDD*, pages 1131–1139, 2012.

[94] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. Scaling up crowd-sourcing to very large datasets: a case for active learning. *PVLDB*, 8(2):125–136, 2014.

[95] Peter Christen, Dinusha Vatsalan, and Qing Wang. Efficient entity resolution with adaptive and interactive training data selection. In *Proc. of ICDM*, pages 727–732, 2015.

[96] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2):6, 2008.

[97] Peter A Flach and Iztok Savnik. Database dependency discovery: a machine learning approach. *AI communications*, 12(3):139–160, 1999.

[98] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. DFD: Efficient functional dependency discovery. In *Proc. of CIKM*, pages 949–958, 2014.

[99] Catharine Wyss, Chris Giannella, and Edward Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *Proc. of DaWaK*, pages 101–110, 2001.

[100] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. Data profiling with Metanome. *PVLDB*, 8(12):1860–1863, 2015.

[101] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[102] Benchmark datasets for entity resolution. `https://dbs.uni-leipzig.de/en/research/projects/object\_matching/fever/benchmark_datasets_for_entity_resolution`.

[103] Magellan data repository. https://sites.google.com/site/anhaidgroup/useful-stuff/data.

[104] Duplicate detection, record linkage, and identity uncertainty: Datasets. http://www.cs.utexas.edu/users/ml/riddle/data.html.

[105] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proc. of ICDE*, pages 746–755, 2007.

[106] Patricia C Arocena, Boris Glavic, Giansalvatore Mecca, Renée J Miller, Paolo Papotti, and Donatello Santoro. Messing up with bart: error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2):36–47, 2015.

[107] Mohammed Kayed and Chia-Hui Chang. FiVaTech: Page-Level Web Data Extraction from Template Pages. *TKDE*, 22(2):249–263, 2010.