Roma Tre University
Ph.D. in Computer Science and Automation
Cycle XXV

User-Assisted Synergic Crawling and Wrapping for Entity Discovery in Vertical Domains

Celine Badr

Advisor: Prof. Paolo Merialdo

PhD Coordinator: Prof. Stefano Panzieri

User-Assisted Synergic Crawling and Wrapping for Entity Discovery in Vertical Domains

A thesis presented by

Celine Badr

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in Computer Science and Automation

Roma Tre University

Department of Engineering

December 2013

COMMITTEE:
Prof. Paolo Merialdo

REVIEWERS:
Prof. Mirel Cosulschi
Prof. David Ruiz

To my loved ones

Contents

List of Tables

List of Figures

Roma Tre University

Abstract

User-Assisted Synergic Crawling and Wrapping for Entity Discovery in Vertical Domains

Celine Badr

Advisor:
Professor Paolo Merialdo
Computer Science and Automation

Large data-intensive web sites publish considerable quantities of information stored in their structured repositories. Data is usually rendered in numerous data-rich pages using templates or automatic scripts. This wealth of information is of wide interest to many applications and online services that do not have direct access to the structured data repositories. Therefore, there's a great need to locate such pages, accurately and efficiently extract data on them, and store it in a structured format more adapted to automatic processing than HTML.

In this context, we exploit intra- and inter-web site information redundancy to address the problem of locating relevant data-rich pages and inferring wrappers on them, while incurring a minimum user overhead. In the first part, we propose to model large data-intensive web sites, to crawl only the subset of pages pertaining to one vertical domain, and then build effective wrappers for attributes of interest on them, with minimum user effort. Our methodology for synergic specification and execution of crawlers and wrappers is supported by a working system devoted to non-expert users, built over an active-learning inference engine.

In the second part, we use the information gathered during inference on the training site, to automatically discover new similar sources on the same type of entities of the vertical domain, which can be useful to complement, enrich, or verify the collected data. Our proposed approach performs an automated search and filter operation by generating specific queries and analyzing the returned search engines results. It combines exploiting existing attributes, template, and page information with a semantic, syntactic, and structural evaluation of newly discovered pages to identify relevant semi-structured sources.

Both techniques are validated with extensive testing on a variety of sources from different vertical domains.

Chapter 1

Introduction

1.1 Web Information Retrieval

The World Wide Web is an enormous repository of documents of different kinds, mostly composed of textual content, with or without accompanying multimedia. This makes the Web a rich knowledge base where users seek and publish information. This information is usually organized into web pages that may display content in plain text, or semi-structured documents containing records, as shown in Figures 1.1a and 1.1b, respectively. Some pages may also consist of a mix of text and semi-structured data.

These types of content, however, are fitted for the user's cognitive abilities but are not well adapted to be processed by computers to answer complex queries, such as "list of restaurants in Paris that serve vegetarian dishes and have good reviews", for example, without forcing the user to read through tens of web pages in search for the needed information. With the growth of comparative shopping, aggregator sites, vertical search engines, and other domain-specific web activities, there is also a growing interest in both research and industry to automate accessing and processing of this huge wealth of information. These efforts aim to find solutions to automatically discover implicit structure and semantics on the pages and turn the content into machine-usable structured data, on which complex queries can then be run.

However, because a large proportion of publishing web sites are not interested in giving public access to their data or spending costly efforts for annotating their pages, approaches seeking to extract information with clear structure and semantics from web pages need to resolve the following tasks:

- Finding relevant web sources
- Locating and extracting information on the web sources
- Integrating and cleaning the extracted data

- Maintaining the process functional and the data up-to-date

These steps constitute the main focus of the field of *Web Information Retrieval*. They generally need to be performed automatically at Web scale, possibly with some human supervision or help from external resources. They are not strictly sequential, as extracting data from a web site can facilitate locating other relevant web sites in some cases, while maintenance is a recurrent task, for instance.

1.2 Large Data-Intensive Web Sites

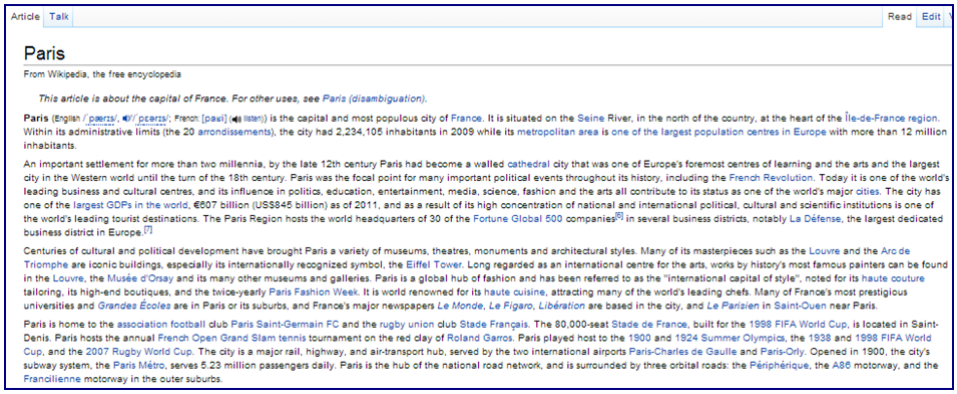
Because web content is heterogeneous in nature, information retrieval techniques that have been developed to deal with text content are generally different from those targeting information that has some level of structure.

The former combine statistical machine learning with natural language processing techniques, such as Part-of-Speech tagging for words grammatical role identification, and semantic role labeling for logical role identification, and apply rules and constraints for more accurate extraction results.

On the other hand, semi-structured information retrieval is concerned with pages that generally display data in attribute-value format. The data represents records originating from an underlying structured or semi-structured data repositories (such as databases or XML files) belonging to a specific domain. This data is commonly rendered on the page with a dynamic encoding into a common template that has some layout structure. Each page displaying one record is then referred to as detail page or entity page, as it describes a particular instance of a given conceptual entity in that domain. The objective of information retrieval from semi-structured content is then to rediscover the implicit domain schema encoded in the web pages and extract the content from each entity page to populate back the records' attributes. This is done using template estimation techniques and expressive extraction rules.

In particular, web sites listing hundreds or thousands of detail pages for a given domain entity are of special interest for information extraction because of the richness of data that they offer. They are qualified as *data-intensive*. The level of structural redundancy they present constitutes an advantage for template discovery and developing effective extraction rules. Moreover, since many such large data-intensive sites exists on the web, cross-site content redundancy is another factor that can also be exploited to support extraction techniques.

Since both information retrieval from text and that from semi-structured sources are very wide areas of study, in this work we focus on specific aspects of information retrieval from large data-intensive web sites.



Article Talk Read Edit

Paris

From Wikipedia, the free encyclopedia

This article is about the capital of France. For other uses, see Paris (disambiguation).

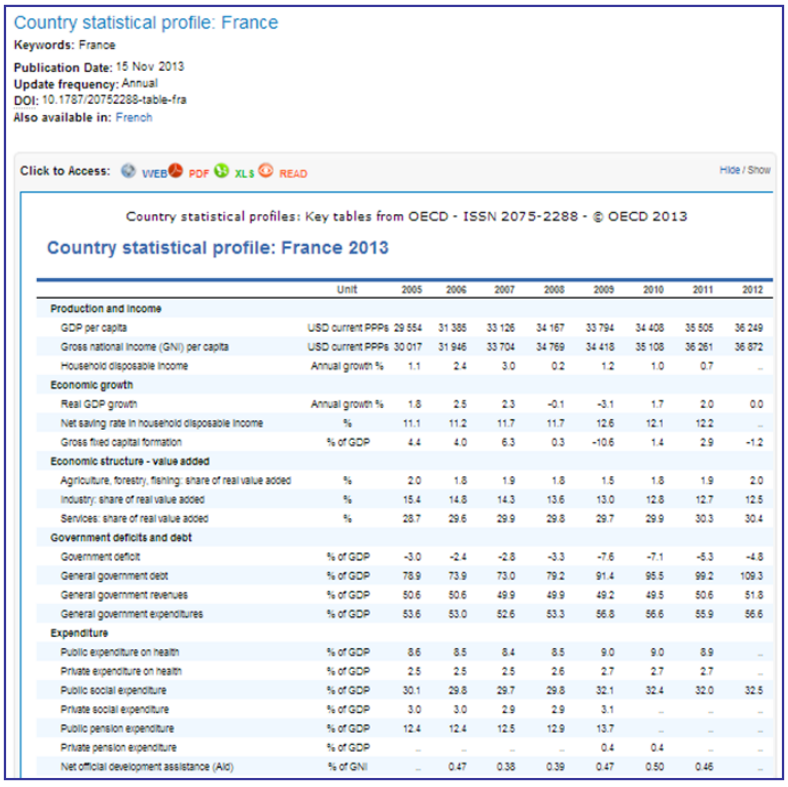
Paris (English /ˈpærɪs/, French: [paʁi] (listen)) is the capital and most populous city of France. It is situated on the Seine River, in the north of the country, at the heart of the Île-de-France region. Within its administrative limits (the 20 arrondissements), the city had 2,234,105 inhabitants in 2009 while its metropolitan area is one of the largest population centres in Europe with more than 12 million inhabitants.

An important settlement for more than two millennia, by the late 12th century Paris had become a walled cathedral city that was one of Europe's foremost centres of learning and the arts and the largest city in the Western world until the turn of the 18th century. Paris was the focal point for many important political events throughout its history, including the French Revolution. Today it is one of the world's leading business and cultural centres, and its influence in politics, education, entertainment, media, science, fashion and the arts all contribute to its status as one of the world's major cities. The city has one of the largest GDPs in the world, €607 billion (US$845 billion) as of 2011, and as a result of its high concentration of national and international political, cultural and scientific institutions is one of the world's leading tourist destinations. The Paris Region hosts the world headquarters of 30 of the Fortune Global 500 companies[5] in several business districts, notably La Défense, the largest dedicated business district in Europe.[7]

Centuries of cultural and political development have brought Paris a variety of museums, theatres, monuments and architectural styles. Many of its masterpieces such as the Louvre and the Arc de Triomphe are iconic buildings, especially its internationally recognized symbol, the Eiffel Tower. Long regarded as an international centre for the arts, works by history's most famous painters can be found in the Louvre, the Musée d'Orsay and its many other museums and galleries. Paris is a global hub of fashion and has been referred to as the "international capital of style", noted for its haute couture tailoring, its high-end boutiques, and the twice-yearly Paris Fashion Week. It is world renowned for its haute cuisine, attracting many of the world's leading chefs. Many of France's most prestigious universities and *Grandes Écoles* are in Paris or its suburbs, and France's major newspapers *Le Monde*, *Le Figaro*, *Libération* are based in the city, and *Le Parisien* in Saint-Ouen near Paris.

Paris is home to the association football club Paris Saint-Germain FC and the rugby union club Stade Français. The 80,000-seat Stade de France, built for the 1998 FIFA World Cup, is located in Saint-Denis. Paris hosts the annual French Open Grand Slam tennis tournament on the red clay of Roland Garros. Paris played host to the 1900 and 1924 Summer Olympics, the 1938 and 1998 FIFA World Cup, and the 2007 Rugby World Cup. The city is a major rail, highway, and air-transport hub, served by the two international airports Paris-Charles de Gaulle and Paris-Orly. Opened in 1900, the city's subway system, the Paris Métro, serves 5.23 million passengers daily. Paris is the hub of the national road network, and is surrounded by three orbital roads: the Périphérique, the A86 motorway, and the Francilienne motorway in the outer suburbs.

(a) Page with plain text content



Country statistical profile: France

Keywords: France

Publication Date: 15 Nov 2013
Update frequency: Annual
DOI: 10.1787/20752288-table-fra
Also available in: French

Click to Access: WEB PDF XLS READ Hide / Show

Country statistical profiles: Key tables from OECD - ISSN 2075-2288 - © OECD 2013

Country statistical profile: France 2013

	Unit	2005	2006	2007	2008	2009	2010	2011	2012
Production and income									
GDP per capita	USD current PPPs	29 554	31 385	33 126	34 167	33 794	34 408	35 505	36 249
Gross national income (GNI) per capita	USD current PPPs	30 017	31 946	33 704	34 769	34 418	35 108	36 261	36 872
Household disposable income	Annual growth %	1.1	2.4	3.0	0.2	1.2	1.0	0.7	..
Economic growth									
Real GDP growth	Annual growth %	1.8	2.5	2.3	-0.1	-3.1	1.7	2.0	0.0
Net saving rate in household disposable income	%	11.1	11.2	11.7	11.7	12.6	12.1	12.2	..
Gross fixed capital formation	% of GDP	4.4	4.0	6.3	0.3	-10.6	1.4	2.9	-1.2
Economic structure - value added									
Agriculture, forestry, fishing: share of real value added	%	2.0	1.8	1.9	1.8	1.5	1.8	1.9	2.0
Industry: share of real value added	%	15.4	14.8	14.3	13.6	13.0	12.8	12.7	12.5
Services: share of real value added	%	28.7	29.6	29.9	29.8	29.7	29.9	30.3	30.4
Government deficits and debt									
Government deficit	% of GDP	-3.0	-2.4	-2.8	-3.3	-7.6	-7.1	-5.3	-4.8
General government debt	% of GDP	78.9	73.9	73.0	79.2	91.4	95.5	99.2	109.3
General government revenues	% of GDP	50.6	50.6	49.9	49.9	49.2	49.5	50.6	51.8
General government expenditures	% of GDP	53.6	53.0	52.6	53.3	56.8	56.6	55.9	56.6
Expenditure									
Public expenditure on health	% of GDP	8.6	8.5	8.4	8.5	9.0	9.0	8.9	..
Private expenditure on health	% of GDP	2.5	2.5	2.5	2.6	2.7	2.7	2.7	..
Public social expenditure	% of GDP	30.1	29.8	29.7	29.8	32.1	32.4	32.0	32.5
Private social expenditure	% of GDP	3.0	3.0	2.9	2.9	3.1	..	..	..
Public pension expenditure	% of GDP	12.4	12.4	12.5	12.9	13.7	..	..	..
Private pension expenditure	% of GDP	..	..	..	..	0.4	0.4	..	..
Net official development assistance (Aid)	% of GNI	..	0.47	0.38	0.39	0.47	0.50	0.46	..

(b) Page with semi-structured content

Figure 1.1: Information organization on web pages

1.3 Challenges

Despite all the achievements in the various research areas of Web information retrieval, some of which have been successfully applied in search engines and commercial sites, there are still some challenges to be addressed.

In crawling, one of the main shortcomings of web crawlers is that they usually lack the ability of performing form submissions to access content behind HTML forms. This hidden web content is referred to as the Deep Web and is estimated to be several orders of magnitude larger than the surface pages crawled and indexed by search engines.

Crawlers efficiency is also an objective that faces some obstacles, mainly because crawlers should have a means to distinguish relevant from irrelevant pages to optimize the effort spent on following links and downloading pages.

Another aspect that limits the coverage of crawlers is scalability. The size of the Web is estimated in billions of indexed web pages, with the Power Law distribution shown in Figure 1.2. Given these facts, a crawling operation may take days, or even weeks, which means that some interesting niche pages in the long tail may not even be reached, and the collected pages may not necessarily ensure information freshness.

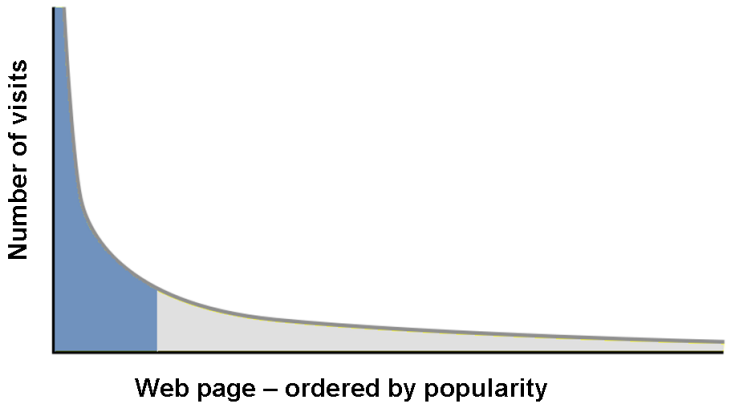


Figure 1.2: Power law distribution of web pages

As for data extraction, the challenges consist in dealing with both the layout and the content of the pages publishing the data. Some examples are:

- Optional attributes, which can be omitted altogether, or have a label present with a null or default value
- Different ordering or placement of attributes for the same type of records in a web site

- Attributes with multiple values (e.g., several movie directors listed, or a price listed in different currencies)
- Typographical errors and value inconsistencies
- Support of JavaScript and HTML frames
- Errors in HTML code (e.g., missing closing tags)
- Need for domain experts input and/or human supervision in general

Figure 1.3 illustrates 2 pages displaying book information, with the value for the `short description` attribute set to "Undefined" in one example (top), while being empty in another (bottom), the `Published` optional attribute being omitted on the bottom page, `Categories` having several values, and the placement of the attributes remarkably different on the 2 pages. This is nonetheless the case of 2 pages from the same web site. When data extraction is extended to cross-site techniques, it becomes even more challenging.

Once the values are extracted from web pages, a data cleaning process is necessary to resolve inconsistencies and overcome the heterogeneity of data before integrating it into a unified structured schema. Challenges arise in truth discovery efforts and detecting copiers, aligning attributes, filling missing values, etc.

Another important consideration is taking into account the constantly changing nature of the data sources. Since web content is continuously growing and being updated, changes in page layout can cause generated extraction rules to no longer extract data correctly. Challenges reside in finding ways to define when an extraction rule is "broken" and if it needs to be redefined, and also finding an acceptable trade-off between the frequency of verification and repair and the frequency at which the data sources are being modified.

We address in our current work some challenging aspects of crawling and data extraction, while data integration and wrappers maintenance may be interesting extensions to consider for future work.

1.4 Contribution

Motivated by the challenges raised in the Web information retrieval field, we present in this work 2 complementary approaches that aim to improve efficiency in locating relevant pages in large data-intensive web sites and performing data extraction on these pages.

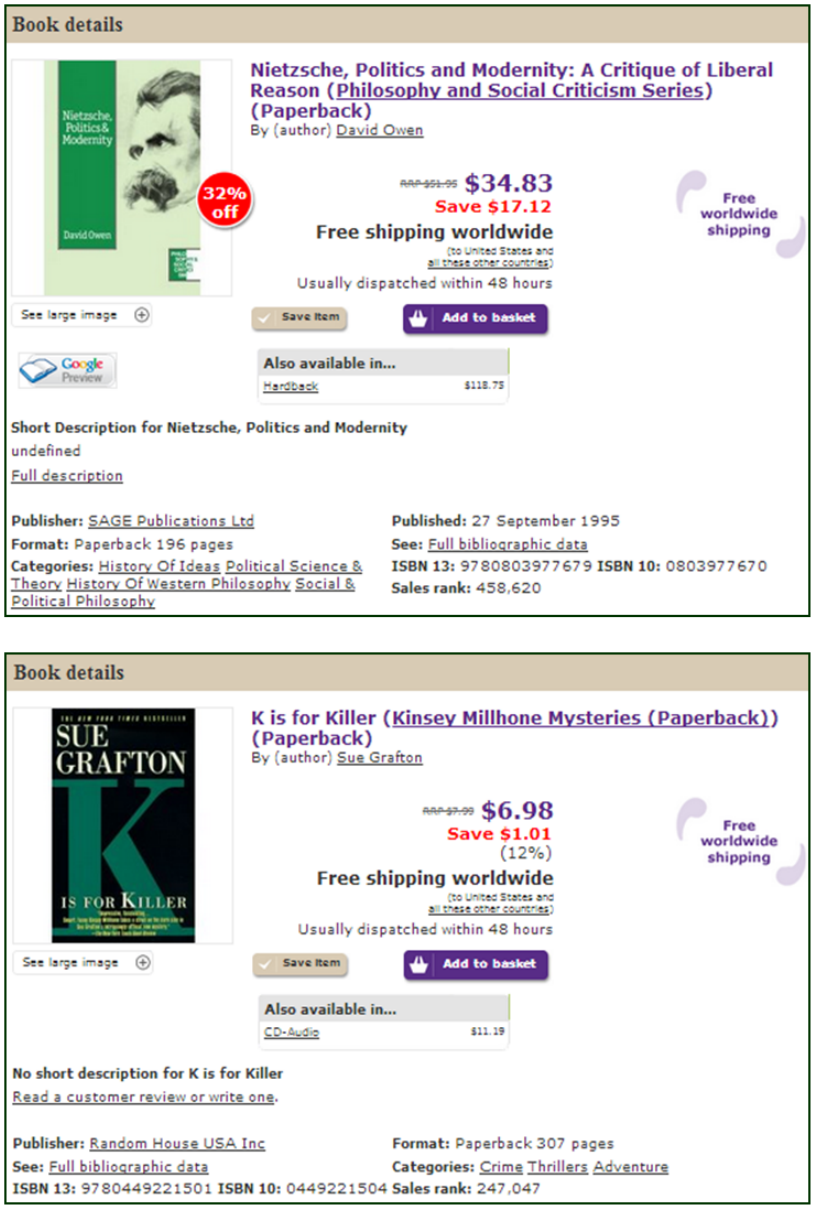


Figure 1.3: Optional attributes

In the first part, we propose a new methodology for a synergic specification of crawling and wrapping tasks on large data-intensive web sites. Starting with the observation that this type of web sites usually contains several categories along different verticals, with each category listing many hundreds or thousands of entity pages in that vertical domain, it is clearly important to be able to reach the entity pages of interest for data extraction without having to crawl the entire web site.

This is illustrated in Figure 1.4, where a cars web site hierarchy shows how a user can browse from the home page to various pages listing car by brand, and then from a list page can click on any of the car links to get to the car detail page. The hierarchy also shows that from the same home page, there are links to other categories like car news, videos, or upcoming shows, etc., which are not interesting for the data extraction task.

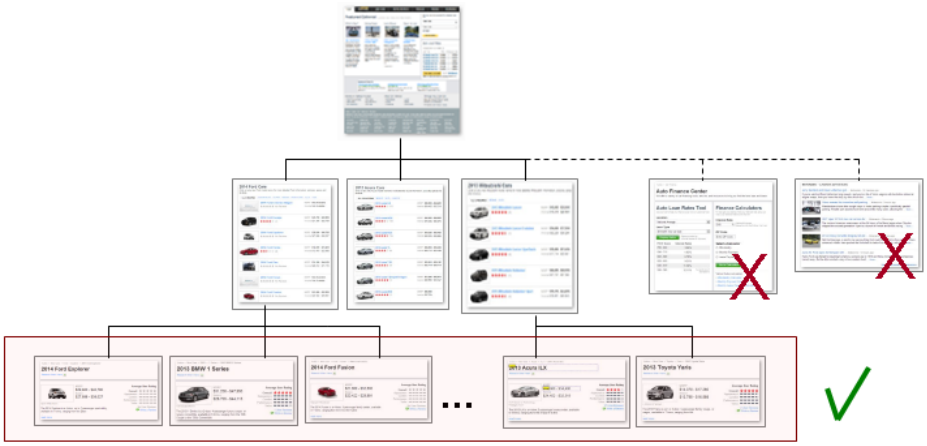


Figure 1.4: Large web site hierarchy

In order to crawl only the relevant subset of pages, our approach derives a model for the web site on the intensional, extensional, and constructional levels. This model describes the topological structures of the site's navigational paths as well as the inner structures of the HTML pages. Deriving this web site model is facilitated by exploiting structural and content redundancy present in large web sites. Moreover, the approach tackles the two problems of crawling and wrapping concurrently during inference and even during execution, such that data extraction takes place while the crawler is collecting pages at the different levels of the derived web site structure. With the system operation based on tracking and learning from the browsing activity of the non-expert user, that user is not required to spend any effort on crawler or wrapper specification tasks.

During its operation, the system also collects valuable content from the web site

and information about the domain of interest itself. In the second part of this work, we take advantage of this gathered knowledge and highlight the existence of information redundancy among entities on the Web, to introduce an efficient technique that finds other large data-intensive web sites offering pages on the entity of interest. The newly discovered entity pages can then be downloaded to extract instance data on them and extend, enrich, or validate the data already collected from the training site.

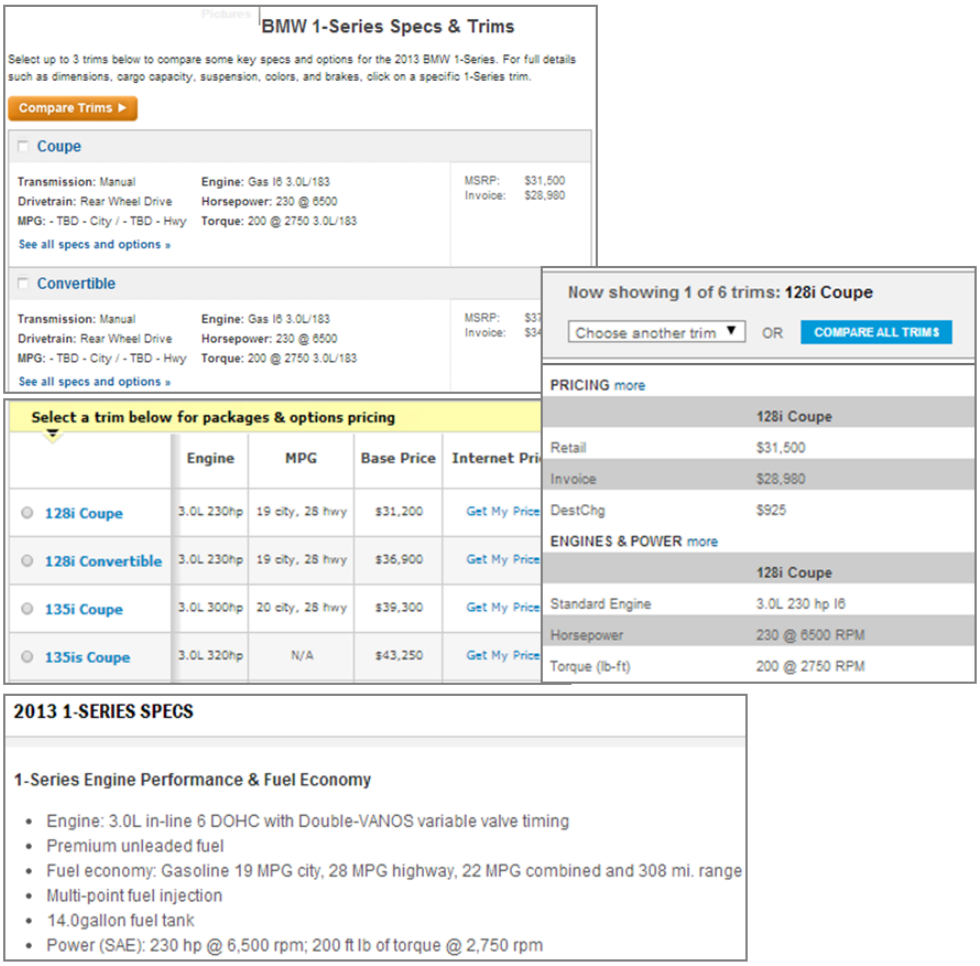


Figure 1.5: Information overlapping in car instances

To give an example of information redundancy, Figure 1.5 shows 4 excerpts of semi-structured entity pages for the same BMW instance of the `Car` entity, coming from 4 different web sites. Our approach aims to start with instances from one web site and discover unseen similar web sources, with a targeted search and filter operation based on the large amount of overlap on the Web among sources in the same vertical domain. This task is fully automatic and can be executed independently of the web

site on which the inference was performed. In fact, we propose to generate specific queries that are sent to a search engine and perform a redundancy detection analysis on the returned results. The identification and incorporation of data-intensive pages pertaining to the same domain is then made possible by a semantic, syntactic, and structural evaluation of web pages collected from the filtered search results.

1.5 Thesis Outline

This document is organized as follows:

Chapter 2 gives a wide overview of the state-of-the-art in Web information retrieval regarding various techniques for crawling and information extraction and integration.

In Chapter 3, we present our synergic crawling and data extraction approach for large semi-structured web sites. This approach exploits content and structural redundancy within a web site to guide and enhance the inference process.

We rely on the domain knowledge collected during this process to introduce in Chapter 4 a vertical domain explorer that can expand the gathered information by discovering new similar web sources based on entity redundancy across web sites.

Finally, a chapter on conclusions and future work concludes this document and presents some potential research directions to explore.

Chapter 2

State of the art

2.1 Web Information Extraction

The field of Web data extraction and integration has been an active research area in recent years. From the improvements to search engines, to vertical search tools, to truth discovery efforts, the potential benefits targeted by researchers are many. However, given the massive amount of data that exists on the Web, the unstructured and imprecise nature of a sizeable portion of this data, and the fact that structured data mainly resides hidden behind web forms, various challenges stand before effectively extracting relevant data that the user is trying to find.

From a general perspective, posing rich queries over structured data sources requires combining and re-purposing the large amount of structured data on the Web. However, the challenges remain in locating relevant data sources, cleaning and extracting the relational data from the adjacent HTML, as well as capturing data semantics and implicit pertinent information from the surrounding text. In many cases, defining and maintaining a global integrated schema is not feasible because of the continuously changing nature of this large number of sources. On the other hand, since specialized search engines limit the query expressiveness with the use of keywords, other alternatives are needed.

More specifically, on the web data extraction topic, wrapping is a crucial step for accessing data sources on the Web, as it translates textual HTML back into a relation form to deliver effective query responses. Minimizing error in this step is strongly desired. In addition, attribute labels are useful for deriving correspondences among elements in different forms. Therefore, automatically extracting labels with high accuracy is important for information retrieval and integration techniques such as hidden web crawlers and metasearchers.

Accessing Deep Web content has its own special considerations. Different approaches tackle the fact that Web crawlers used by search engines usually lack the

ability of performing form submissions to access content behind HTML forms. They aim at locating forms and surfacing Deep-Web content, since this content represents a substantial part of the structured data on the Web and is thus important for search engine coverage

After extracting the data, the challenge rises in the integration part where finding large scale methods to identify and integrate fresh data from data-rich pages is needed. Combining data comes useful for users who want to find information in alternative or comparative sites. Moreover, the observation of redundancy among web pages is another motivation for the development of scalable techniques enabling search engines to extract and integrate structured or semi-structured data from large sets of web sources.

Extraction rules maintenance is another task to be considered, as it is important to make sure that wrappers are functional over time with the modifications and updates continuoustly brought to web pages.

In addition, it is worth noting that it is not uncommon for various web sites to provide conflicting data. With web data being imprecise, characterizing its uncertainty becomes an important step for data integration. Resolving data conflicts and discovering true values have thus become areas of interest in the context of information retrieval.

In this chapter, we introduce some research works addressing these various problems related to Web data extraction.

2.2 Crawling Web Sources

Millions of users, each with different needs, access the Web continuously and pose millions of search queries. Standard search engines need to find the relevant information among the billions of web pages available online. These search engines rely on crawlers or spiders to find and download web pages that are then indexed and used to answer queries. A few works [CMM05, KLHC04, LNL04] have addressed the automatic discovery of content pages by finding paths to all generic content pages within a web site. The work by [VdSdMC06] uses URL patterns to specify crawlers and adopts a breadth-first strategy to compare the site's pages DOM trees with a provided sample page. Scalability is a known challenge for these processes and therefore, the collected pages may suffer from a relative low coverage given the size of the Web and may not necessarily ensure information freshness.

Topical Crawlers

Rather than attempting to crawl all possible documents in an attempt to answer all possible queries, focused crawlers have emerged to restrict the set of pages collected by introducing some constraints. Only pages satisfying specific properties are downloaded by the crawler. These properties can range from predicates applied to pages URL before downloading them, to properties pertaining to the topic of the page content.

[CVdBD99] addresses focused crawlers and establishes how they analyze the crawl boundary to prioritize and follow relevant link paths on the Web and avoid crawling sections of the Web that are of little or no relevance to the crawl objectives. This results in optimizing the collection and indexing of documents, which translates into hardware and network resource savings, as well as a greater possibility to keep the crawl up-to-date. They also describe how the evaluation of pages can be achieved with a semi-supervised classifier that learns relevance and popularity by example.

Similarly, [Cha02] highlight the same difficulties for building domain-specific search engines and propose to predict whether a URL points to a relevant Web page to avoid downloading those that are irrelevant or low-quality. With a neural network model they compute a score for prioritizing URL downloads and estimating the relevance of a page. Features such as the domain relatedness of terms appearing in the page title and content, in addition to incoming and outgoing link analysis, are used by the classifier for web page filtering.

In a more recent work, [PS11] suggest to improve the performance of focused web crawling by implementing a document parser that analyzes the structure of the HTML DOM tree of the downloaded documents. The parser relies on the occurrence of specified tags to determine the priority of further downloads and then communicates its results to the download scheduler that fetches the pages from the Web.

2.3 Wrapper Generation

Extracting information with clear structure and semantics is of great interest to search and commercial engines in their quest to respond to user needs with more specific and relevant data. Because of nature of the content published on the Web, the quest for structure and semantics from this content is mostly performed by external agents, without the active involvement of the web sites. For that, selecting and retrieving the information of interest is needed once the web sources containing these information are collected. To this end, wrapper generation techniques have gained a lot of attention in research. A web wrapper is mainly a program that extracts content embedded

in the HTML of a web source and delivers it in a structured format that is more adapt for machine processing. Wrappers are generated for each different web source, such that they represent a mapping between the data source and a structured data repository, to populate the repository with the implicit objects contained in the data source [LRNdST02].

Several classification schemes exist for web data extraction approaches [FDMFB12, CKGS06, FMM+04, LRNdST02, Eik99]. We discuss some examples here according to their degree of automation. In fact, these information extraction tools require the creation of extraction rules, which is performed either by manual, semi-automatic, or fully automated approaches. While manual approaches favor higher precision, semi-automatic and fully automatic approaches aim to improve effectiveness and minimizing user effort by learning from web site structure and content.

Manual Wrapper Generation

The first category of wrappers is based on manual specification of extraction rules. An expert user can use programming languages to create extraction rules, which are commonly expressed as XPath or regular expressions. To facilitate the expert's task, [SA99] implement a Java toolkit with a graphical interface to support the manual creation of wrappers. Manually tailored wrappers generally offer high precision but do not scale well for large data extraction efforts, due to the required effort and expertise.

Semi-Automatic Approaches

In order to overcome the technical challenges of manual wrapper generation, many semi-automatic approaches have been developed. They aim to reduce user interventions, but the process remains mostly template and web site dependent and requires annotated training pages. Essentially, extraction rules are inferred automatically by the system, while the user's role is to supervise by providing examples or revising patterns on the browser-rendered content, instead of working directly on the HTML of the page. Some of the early works include [Kus97, Sod99, Kus00] introduce machine learning techniques for wrapper induction on a sample set of pages, to ensure larger scalability to different web sources and better noise tolerance. They evaluate the efficiency of different classes of wrappers with different levels of expressiveness.

However, one of the main bottlenecks to address in supervised learning is the selection of which documents to label. To resolve this, various active learning selection strategies have been proposed and evaluated in different extraction settings [FK03, SC08, RC11, LC06]. An extensive survey is presented in [Set10]. [MMK03] introduce multi-view learners, where they combine results from disjoint sets of fea-

tures, categorized as strong views or weak views. Strong views are specific to the target concept and are considered sufficient for learning, while weak views bring additional but more general information. An agressive co-testing algorithm evaluates the results from the rules learned in each view, then applies an active learning approach to query the user on the unlabeled example where the rules disagree. [IS06] propose a training interface backed by an active learning framework to increase the accuracy and robustness of wrappers that extract data records on listing pages. Through this interface, the user supplies the definition of attributes of interest and provides a training tuple. With a powerful extraction language that captures various predicates related to the HTML elements, along with a ranking algorithm, the system is able to infer wrappers based on the one or two provided examples and the set of unlabeled verification pages.

Moving from a web site-centered to a wider domain-centered view, [CCZ07] note that an average of 64% to 75% of attributes are overlapping between two sources from the same domain. With the fact that a field may be hard to extract in one source but obvious in another, they propose a system, Turbo wrapper, which is a collaborative wrapping tool where multiple sources collaborate to mutually correct their extraction errors. The authors consider that each source adopts a submodel of the global domain schema, so each wrapper for a given source attempts to decode this source model and return an output to be used by other wrappers to reinforce the wrapping. Such approach has been generalized to deal with a larger number of sources on Web scale and automatically discover attributes for data extraction, as discussed in Section 2.5.

Furthermore, some more ambitious efforts address wrapper generation for information extraction on a web scale. [DKS11] present an algorithm that targets noise-tolerant data extraction. Instead of human labeling, they use compiled dictionaries for specific domains to annotate pages, and remedy for the possible noisy annotations with domain-specific knowledge and probabilistic and ranking models. [GMM+11] also implement a web scale data extraction system. They applies clustering techniques on detail entity pages according to template similarity, and rely on several user inputs for annotation and rule learning for the structurally diverse sample pages.

For all these supervised methods, interaction with the user mainly increasing wrappers reliability. The time and number of examples required for learning are common indicators for a wrapper's efficiency, in addition to running complexity and site coverage.

Automatic Approaches

Although a great improvement over manual wrapper generation, semi-automatic approaches still require human effort and labeled training sets. On the other hand, automated systems have emerged to perform data extraction from unseen pages, without the need for human labeling or wrapper specification. The most renowned in the state-of-the-art are RoadRunner, ExAlg, and FiVaTech [CM08, CM06, CMM+01, AGM03, KC10, CHL03]. These systems target a formal definition and unsupervised derivation of web pages template. By identifying the page schema components and patterns, they are able to express complex extraction rules that can deal with flat records as well as semi-structured data listed in nested format. In [ZSWW07], template detection is combined with wrapper generation by applying a wrapper-oriented page templates clustering, in order to optimize results accuracy. Even though these systems improve the level of automation, they still present some limitations. For example, they may require considerable preprocessing of the pages HTML, and they make certain assumptions about the structure of the pages. They also lack the semantics of the extracted data and thus are not able to distinguish relevant from irrelevant data. Due to the insufficient accuracy and possible incorrect extraction of noisy data, they require a post-processing step for data cleaning.

Comparing Wrappers Performance

As an ending note, wrappers in their different implementations are commonly evaluated for expressiveness and adaptability to a multitude of web pages, in addition to robustness over changing layouts and scalability. However, comparing different wrapping techniques in the literature remains a real challenge, mainly because of the differences in the adopted techniques and the lack of a common evaluation test standard.

2.4 Wrapper Maintenance

Generating wrappers that extract data with high accuracy on a set of sample pages or on all pages of a web site is only an intermediate step in the information retrieval process. In fact, web content is continuously growing and being updated, and very often, changes in page layout can cause wrappers' extraction rules to no longer extract data correctly. This motivates monitoring the sources and conducting wrapper verification on regular intervals to specify pages or data features that change and determine if the wrapper is failing [LK09]. The validity or failure of a wrapper on a given web site is usually deduced based on statistical estimations conducted on the extracted data,

and can possibly be enforced by the definition of patterns and integrity constraints on the extracted attributes. A detected failure prompts a wrapper repairing task, which, just like wrapper generation, can range from manual, to semi-automatic, to fully automated. The faster wrapper systems can find differences between two versions of a web page and adapt automatically to the modifications, the more robust and reliable the data extraction is considered. While some approaches for wrapper repair rely on aligning structural and content data between the page used to define the wrapper and the new version that caused it to fail, most other approaches tend to apply learning algorithms.

Examples of schema-guided wrapper maintenance are [MHL03] that rely on more stable components like syntactic patterns, annotations, and hyperlinks to recognize data items on changed pages, and [FB11] that propose a DOM tree matching algorithm to align HTML elements and attribute values for wrapper reinduction.

On the other hand, learning approaches use the values and features of the training examples from the wrapper generation phase to identify the location of the values of interest on the new modified page layout and fix the wrappers accordingly. For [LLD10], past data can serve to train a classifier based on features such as extracted value content, length, data type, pattern, and attributed label. The trained classifier is then used to identify the region of extraction and the element of interest on the new page, for which new wrappers are generated. [CRB06] note that wrapper maintenance costs are far superior to the creation cost for a content integrator solution in a commercial context, and propose to automate wrapper maintenance to reduce the costs by considering content and context features to characterize the labeled tokens in a page. By including preceding and succeeding tokens, in addition to syntactic and semantic features, their classifier tries to automatically relabel the new page content. Based on the outcome of the operation, wrappers can be repaired or reinferred from scratch. In a similar way, [LMK03] describe their machine learning algorithm that exploits available structural information, augmented with content-based patterns. In fact, the algorithm finds statistically significant patterns in the extracted tokens of positive examples and creates a word-level type hierarchy tree. This is later used to classify extracted data in the wrapper verification step, along with the density distribution of detected data types. When comparison results do not match expected patterns, the page template is learned again and automatic labeling is used for wrapper reinduction.

Although automatic wrapper repairing can reduce considerably the number of human interventions, nontheless, identifying false positives in the verification results and initiating a reinference of the wrappers for a modified web site remain generally the wrapper designer's task.

2.5 Scalable Automatic Approaches for Attribute Discovery

The many automatic approaches proposed for web data extraction have as a main goal scalability to the huge number of semi-structured sources on the Web. These approaches attempt to discover data fields on the pages with little or no manually labeled training pages, mostly by confronting different data-intensive web sites and different entity pages within these web sites. They generally start with the hypothesis that there exists a domain model and that attributes in data-intensive sources belonging to the same domain are shared or complementary. Therefore, some target extracting preselected attributes, while others aim to discover entity attributes dynamically within and across web sites. Essentially, in order to scale to the size of the Web and extract data on any page from a given vertical, a system needs to eliminate the need for human intervention. This is confronted with many challenges when dealing with unseen pages, such as the variations encountered among values of the same attribute due to differences in formats, or using abbreviations, for example. Moreover, there is the variation in layouts even with the same web site general template, and more importantly that among pages from different web sites. A page layout is a work of creativity in rendering information that is addressed to the human cognitive capabilities. Therefore, an automated system needs to learn the existence of these variations and implement a way to handle them. Yet another challenge is noisy content published on the Web. This requires techniques for truth detection and data cleaning, which we discuss in Section 2.7.

Among the automated approaches developed in recent years, [CS08] propose to use training data from a combination of various web sites, to extend the extraction of entity records to new web sites, without human supervision. Their approach aims to learn an extraction model for entity pages with semi-structured data in a given domain, but specifies the domain attributes of interest in advance during the manual labeling phase of the training examples.

[HCPZ11] later build on this model to introduce a solution that takes one labeled example site and is able to extract data from unseen sites in the same vertical. For their system to learn patterns and overcome the variations among pages, they propose to use weak but general learning features about content, context, and layout, to learn vertical knowledge from the seed site. They then apply this knowledge to a given new site to identify attribute values on its pages. Site-level statistics and constraints are used to refine the results. Unlike the approaches that are based on deducing the page template and exploiting DOM tree representations, their approach is more aware of the semantics of the vertical domain through the extraction of its related attributes. However, they still rely on hand-crafted regular expressions to tag attribute values on

the seed site.

On the other hand, [SWL+12] propose a framework for automatically discovering attributes and extracting their values on unseen pages in a vertical domain, without any manual labels. Their approach consists in progressively learning what they describe as "credible" attributes from structural, inner-site, and cross-site features taken from a number of pages in enormous web sites. Using the attributes they identified, they proceed to items discovery and extraction on the pages, where items consist of actual attribute-value pairs. With a set of attribute values assumed available, the new values discovered are integrated into the final results only if they appear in high frequency in the existing lexicon. Operating without human intervention, their approach is dynamic and effective and prove to even detect unseen attributes in most vertical domains. It still has some limitations mainly concerning the pre-determined attribute label and value positions, and it lacks the ability to classify relevant pages before processing them.

Finally, it is worth noting that text segmentation techniques have also gained interest in data extraction research [COdS+11, CdSGdM10, ZMR08]. These approaches aim mainly to convert implicitly semi-structured information displayed as continuous text possibly without any delimiters, into structured records. This has been applied for instance to listings of postal addresses, recipes, bibliographic citations, etc. While human training is not necessarily required, these approaches exploit pre-existing domain data to associate the identified segments in the analyzed string with domain attributes, by using learning techniques (CRFs), or by applying matching strategies. Combined with the existent content knowledge, they mainly rely on structural features like positioning and sequencing of attribute values to extract data records.

2.6 Surfacing the Deep Web

One of the shortcomings of web crawlers used by search engines is that they usually lack the ability of accessing content behind HTML forms, as they usually lack the ability to perform form submissions. Nonetheless, web content lying in structured and unstructured sources and accessible only through query forms is estimated to be several orders of magnitude larger than the surface pages crawled and indexed by search engines. This hidden web content is referred to as the Deep Web, and is usually revealed by dynamically retrieving content from a database and filling it in a template structure to be rendered in the browser. Since this content represents such a substantial part of the data on the Web and is thus important for search engines coverage, various efforts have been presented to make it more accessible.

The approaches in [MKK+08] and [MAAH09] for surfacing deep-web content

are based on a system that precomputes submissions for HTML forms and adds the resulting HTML pages to a search engine index. To index content behind HTML forms in an automatic, scalable, and efficient way, it is required to select input values for text search inputs in web forms and identify inputs that accept only values of a specific type. In order to efficiently navigate the search space of possible inputs to identify those that generate relevant URLs, implementing mediator forms for each domain with semantic mappings is costly and does not scale. Therefore, they highlight the following requirements for surfacing:

- keep the approach fully automatic with no site-specific scripting
- decide which inputs to fill
- find appropriate values for these inputs

Based on these points, the authors' contribution consists in developing an informativeness test to evaluate and select query templates, an algorithm to identify input combinations that generate fewer URLs and higher coverage, and an algorithm for predicting appropriate values for text boxes. Their aim is to maximizing the coverage of the underlying database, that is, the number of records retrieved, with a bounded number of form submissions. They distinguish between selection inputs and presentation inputs that can be present on the forms. Some inputs are binding and they are the ones of interest, since by their different values multiple form submissions are generated. The rest are referred to as free inputs. For generating query input values, seeds are selected from words on the form page and restrictions are dynamically applied on the maximum number of keywords for a text box. From the returned results, they identify input combinations that generate informative content and add the corresponding URLs to the search engine index.

Another approach to crawl the Hidden Web has been proposed by [RGM00, RGM01]. The architectural model for their crawler, named HiWE for Hidden Web Exposer, targets pages that contain web forms. Assisted by some initial user's input, it addresses resource discovery and content extraction from the Deep Web for a specific task in a particular domain. To do that, the system derives an internal representation of the form to which queries are to be submitted. It assumes an existing database rich with content relevant to the task or domain. The content of this database is used to formulate the queries, after the detected elements on the form are matched to the database attributes. Then a response analysis module evaluates the pages resulting from the form submissions to distinguish pages containing search results to be kept, from those containing error messages to be discarded.

To locate pages containing web forms that can be exploited for Deep Web exploration, [BF07] propose novel focused crawling strategies that retrieve entry points to hidden-Web sources. Their crawler applies learning techniques to automatically adapt and improve its link exploration process. This is followed by an evaluation and filterning mechanism that controls the relevance of the found forms to the vertical domain of interest. To expose the database content behind these forms, the authors propose keyword-based interfaces to issue crawling queries [BF10], and build a deep-web search engine, DeepPeep, which has gained worldwide interest [BNN+10, BNN+09].

To facilitate information retrieval from the Deep Web and integration techniques, the problem of automatically identifying and extracting form labels has gained importance in research. This problem poses several challenges: first, labels can be placed in different positions with respect to form elements; also, in dynamic forms, content and layout can change based on user selections; finally, one label can be associated with many form elements, or it can be placed outside the form tags in HTML.

While previous approaches relied on manually specified label extraction rules or heuristics, the work in [NNF08] introduces a technique that uses a learning classifier, with a reconciliation step to boost extraction accuracy. The authors present a hierarchical framework that extracts information from HTML and visual layout. Given a set of web forms with labels appropriately identified, their classifier learns the patterns for label placement. A training phase consists of a manually derived set of label-element mapping, with a first classifier (Naïve Bayes) and a second classifier (Decision Tree). A following extraction phase uses a pruning selector and a mapping selector to identify correct and incorrect matches, then applies a mapping reconciliation step on elements with multiple or no labels. When generating candidate mappings, a proximity-based heuristic is applied between an element and textual labels in its vicinity. To identify correct mappings, various learning techniques are evaluated, relying on different features such as element type, font, alignment, distance, etc. Recall, precision, and F-measure scores for each techniques are reported, proving the improvements they bring over the state-of-the-art.

2.7 Information Integration

The Web can be viewed as a rich information repository where users go not only to find but also to publish data. For this data to be machine-readable, it is collected from online sources and stored into structured format. After that, given its heterogeneous nature, an integration process is necessary to make it usable for effective query answering and other automated tasks.

Data extraction efforts have been introduced in Section 2.1. In this section, we de-

scribe some research achievements in the area of data integration, and highlight some of the open challenges. Many efforts share the goal of constructing an entity aware search engine on which it is possible to run precise queries, in contrast with existing search engines that return relevant documents based mainly on keyword matching.

Best Effort Approaches

Some approaches propose best-effort integration, favoring automation, scalability, and efficiency. [CC07], for example, propose a solution that addresses both how users formulate queries and how the search engine returns result, based on large scale on-the-fly integration of online data. This aims to provide the user with the most precise information needed, instead of having to swift through documents in search for the data. In their model, results returned from their search engine can consist of information combined from multiple sources. Existing integration systems are mostly limited to vertical domains as they rely on domain-specific wrappers to collect data. The authors sustain that these systems are not capable of scaling to the expanding heterogeneous web sources and user needs. In response to these needs, they propose an entity search system where the user can specify which entities she's interested in finding and possibly provide some context information to be matched as well. These are provided as input to the entity search engine in the form of tokens, distinguishing keywords, context information, patterns, and entity references with special identifiers. Search results consist of the actual entities in addition to the supporting pages for further references. These results can be found in any web corpus, assuming the sources are tagged so that corresponding entities are extracted on them, coupled with a confidence probability. This allows to return results with merely a best-effort matching on their semantics and a probabilistic integration combined with a calculated likelihood of their relevance, which is the basis for their ranking.

Another best-effort approach is proposed in [MFH10], to query and integrate structured web sources on-the-fly. Arguing that, on one hand, defining and maintaining a global integrated schema for structured sources is not feasible because of scalability and the changing nature of content, structure, and number of pages, and on the other hand, creating specialized search engines limits the query expressiveness with the use of keywords, their solution introduces scalable best-effort algorithms for handling large numbers of sources. A structured web search engine is built upon web pages containing HTML tables, periodically gathered by a focused web crawler. When a user writes an SQL query, a set of query rewritings is performed, applying a best match to determine the correspondence of the requested attribute names with colum names extracted from web pages. This approach also relies only on string

similarities among column names to determine connections among different extracted sources and combine their data into ranked result records. It can potentially be improved by taking into account user feedback on the quality of query rewritings and attempting to use both source metadata and contents to determine source compatibility.

[NFP+11] also introduce a system to support product search engines or shopping verticals by extracting and integrating data at a web scale. In fact, their product synthesis architecture consists mainly of two large functional components, the first for offline learning and the second for offer processing at runtime. Relying on a product catalog that they build from historical offers during the offline learning phase, they create attribute-value pair correspondences from data in HTML tables on merchant offer pages, specified for each pre-defined entity or real-world object represented in the catalog. At runtime, schema reconciliation is applied to newly encountered product offers to associate them with a pre-defined category and match the information on the page to the schema attributes definition. When this is not possible, a new catalog product is created and specified with its attribute associations. This is followed by a value fusion mechanism that selects representative values for each attribute.

Matching to Domain Schema

In various works in the literature, it is noticed that data integration efforts assume that there exists a domain schema listing the attributes for a given conceptual entity in that domain. On one hand, some techniques address how to match data extracted on web sources onto entity attributes when that schema is given or manually specified, such as in the examples presented above. On the other hand, more general techniques attempt to find matchings among data coming from heterogeneous sources, in order to deduce the composition of the global domain model.

In this latter scope, the work of [CC08] introduces holistic schema matching for finding the correspondence of the data fields among multiple sources. They address combining data from query results to facilitate the task of users who want to obtain information in alternative or comparative sites. An obtained integrated data source would also be valuable for vertical search engines and other applications. Their solution matches multiple sources holistically to eliminate accumulated errors in pairwise matching. Data fields are matched based on content similarity of their data labels and values. With cross-source integration, they observe that some error matches or mismatches can get recovered, assisted with domain constraints that are represented as predicates with uncertainty. These domain constraints are themselves automatically derived, therefore they are only reliable when many sources are observed to-

gether. Because the model is based on matching entity instances, if more sources are observed, the schema derived from the input data is believed to describe more effectively the whole domain. The algorithm they introduce, HoliMatch, applies then an iterative optimization that alternates from matching to schema and from schema to matching, to label elements with appropriate domain fields. It operates under the consideration that, when a domain schema is given, the matching among a group of fields can be derived, and vice versa. Their experiments that compare HoliMatch with multiple-source matching strategies, such as clustering (ClustMatch), extended pair wise matching (ChainMatch), and progressively adding sources (ProgMatch), record satisfactory results and emphasize that a good extractor is important for schema matching and overall data integration.

Similarly, [BBC+10a] and [BBC+10b] present their solution that exploits cross-source redundancy of information in structured and semi-structured sources on the Web to map their content onto a hidden conceptual relation for their vertical domain. They combine data extraction and integration techniques, in an effort to exploit mutual dependencies and benefits between the two activities for improving scalability and accuracy of overall output results. Since redundancy can be observed among sources listing instances of one domain entity both at the schema level (shared attributes) and at the content level (instance attribute values), their approach attempts to apply unsupervised schema matching on these sources without apriori knowledge about the domain. Redundancies are exploited locally in evident regularities within the same web site, and globally in overlapping data among different sources. In one phase data is extracted using unsupervised wrappers, then in a second phase an original data matching technique is applied to identify data mappings for extracted attributes. Results of data mapping are used in return to validate and improve the wrappers extraction rules. Because no labels are associated with extracted attributes, matching relies on attribute values, together with the fact that non-identical attributes from the same source have different semantics. With these constraints, a novel algorithm, SplitAndMerge, is proposed for source integration. It proves to outperform a previous naive matching approach where a matching score is associated with each attribute against all the mappings derived for it.

Probabilistic Schema Matching

Even though a large amount of redundancy is detected in the content and schema attributes across semi-structured web sites in the same vertical domain, these web sites exhibit nevertheless an amount of inconsistencies not to be ignored. Therefore, many studies have addressed the quality of extracted data, or proposed techniques

for data cleaning and reconciliation. Furthermore, the concepts of uncertainty in data integration and probabilistic schema mappings have been introduced in [DHY09], which lead the way to other efforts in this direction.

One example that deals with conflicting data is [BCMP11]. They propose a Bayesian model to evaluate the quality and uncertaintly of data extracted from the Web. In fact, a probability distribution can be derived for the values published. Also, a probability distribution can be used to estimate the accuracy of a data source, or the likelihood of it providing correct values for an object. By taking into consideration factors such as source accuracy, consensus over extracted values, and factoring in the detection of copiers, they propose techniques with various complexity to estimate truth probabilities in the data. Their study leads to the interesting observation that web ranking models seem not to have any correlation with sampled source accuracy.

In a similar effort to resolve data conflicts and discover true values, the authors in [DBES09] present an approach that considers dependence between data sources and accuracy of sources. With the observation that when one source copies from another, the two are likely to share some erroneous values that are not found on other sources, they propose Bayes models that compute the probability of any two data sources being dependent, and associate objects with a probability distribution for various possible values in the domain under study. With further probability analysis, their algorithms are able to distinguish between independent sources and copiers, and even separate good independent sources from bad ones, and benevolent from malicious copiers. Tested both on synthetic and real-world data, the results of their models can be regarded as a probabilistic database and prove to increase the accuracy of truth discovery for data integration.

Comprehensive Solutions

Finally, some efforts have addressed comprehensive systems that put together techniques for data extraction, data integration and cleaning, and also offer a search interface to access this data.

The most renowned system in recent literature is OCTOPUS [CHK09] (along with its predecessor OMNIVORE [Caf09]). It is a user-oriented comprehensive solution that offers all the functionalities of integrating data extracted from web sources, to searching the structured data repository, and even extend the existing data sets with information found on the Web. Given an already extracted set of relations, a user can then exploit the search functionality by providing keywords. The system returns clusters of records, sorted by relevance to the query. Furthermore, the user has the ability to refine the results by specifying a semantic context to limit the scope of the

returned selection. Moreover, an `extend` option allows to complement a relation with attributes joined from another extracted relation. Octopus offers some additional features as well: selection, projection, column-add, column-split, and column-rename operations are possible manual changes that the user can perform. The complete solution is based on several possible algorithms for each functionality it implements, offering tradeoffs between accuracy and computational overhead.

2.8 Entity Discovery

Going beyond raw data collection and integration, entity discovery has been a subject of interest to many research groups in information extraction. These works aim either to identify all possible entities and relationships when parsing a given document (described as open or input-oriented information extraction), or to find all those entities and relationships in any choice of sources, that are instances of a predefined set of relations (closed or output-oriented information extraction). Various techniques have been developed and applied to unstructured or semi-structured web sources, with a variation in performance results.

For analyzing unstructured text inputs, statistical machine learning is combined with natural language processing (NLP) techniques [YCB+07, ECD+04]. These include techniques such as Part-of-Speech (PoS) tagging that identifies words' grammatical role, and semantic role labeling (SRL) that maps parts of a sentence to the logical role they represent, be it location, time, amount, seller, buyer, or other. The main drawbacks are that PoS tagging relies on lexical dependency parsers that are computationally expensive, whereas SRL lacks the training corpora needed to cover a variety of domain-specific semantics.

In [WT10], the authors present a technique for knowledge harvesting from semi-structured and natural-language online sources such as Wikipedia. Their goal is to construct a comprehensive knowledge base of facts by extracting semantic classes, mutual relations, and temporal contexts of named entities. With the constructed machine-readable knowledge base, they aim to support expressive and precise querying and enhance many applications, such as entity disambiguation, semantic search, natural language question answering, machine translation, etc. In their work, they adopt state-of-the-art achievements related to knowledge harvesting and they also present the main problems and challenges still to be considered.

Harvesting entities/classes consists in collecting individual entities and associating them with classes organized in a hierarchy of superclasses and subclasses to build a taxonomy. In their YAGO system [SKW07, SKW08] for example, this is achieved by automatically discovering entities in Wikipedia and mapping them to one or mul-

tiple predefined YAGO classes as Wikipedia classes are noisy, and using WordNet to determine synonyms and head words in string expressions. The main challenges in this case are dealing with category classes that are not clean and solving entity disambiguation. These challenges are partially addressed when the knowledge base is rich with information and when other strings are available in the surrounding context. Finding a way to automatically discover new entities as they appear on the Web in order to enrich the knowledge base remains a subject for study however.

On the other hand, harvesting relational facts is achieved with various techniques.

- Rule-based fact gathering relies on wrappers when the web sources present enough structure for learning. Extraction is then done by regular expressions over the sources' DOM trees. This can achieve high precision but there is room for improvement in recall.

- Pattern-based fact gathering in free text applies text patterns and constraints extracted from natural language to relate subjects and objects of certain verbal phrases. For example, this allows to extract *instanceOf*, *partOf*, and causal relationships from common syntactic patterns in the text. While obtaining high recall is possible, this approach suffers from lower precision.

- Learning and reasoning methods apply consistency constraints by combining first-order logic rules with probabilistic models. Moreover, type checking in binary relations allows the verification of the hypotheses on the extracted facts.

Challenges in extracting relational facts consist first in scalability when filtering relevant sources from non-relevant sources in rule-based approaches, and scalability of the fact extraction in general, due to the fast growth of online content. In addition, balancing hard constraints that have to be enforced and soft constraints that may be violated by some instances is still an open research issue.

Finally, the authors present their additional contribution concerning temporal knowledge, which aims to extract validity time of facts in time-dependent relations for a time-aware knowledge base. However, due to the narrative nature of language, temporal data extraction still requires a considerable consistency reasoning effort in post-processing. Another challenge is that temporal knowledge lacks an efficient query language to explore the collected information with predicates expressing duration, overlap, and approximate intervals of time. The current limitations constitute a motivation for active research activities in the area of entity and relationship knowledge harvesting.

In a different, more relaxed effort, [LDF11] adopt an approximate entity extraction approach. Starting with predefined entities in a dictionary, the approach analyzes

documents to find all substrings that approximately match them. This aims to improve recall, as exact matching achieves high precision but fails to deal with string typing variations. Thus they present a unified framework that supports various string similarity/dissimilarity functions, such as jaccard, cosine, dice, edit similarity, and edit distance, to deal with typographical or spelling errors that would otherwise prevent exact matching with dictionary entities. Based on the overlap between a string in the document and a dictionary entry, the two are considered *similar* or not. Their approach, which also uses shared computation on overlapping substrings, and pruning techniques on unnecessary candidate pairs in the document, achieves high performance results on real datasets.

More recently, an interesting initiative has been presented in [FGS12, FGG+12] towards a fully automatic entity recognition in vertical domains. By relying on an extensive domain knowledge, they propose the DIADEM engine that is able to explore, identify, and analyze form fields, query results, and entity detail pages. Their system navigates a web site and extract objects listed in it, without human supervision. The key to identifying objects and their attributes in a first phase is knowing the entities and relations in the given domain, in addition to their phenomenology (how they occur on the web site). In a second phase, domain constraints in form of logical rules are applied for reasoning on the extracted knowledge. Essentially, they trade-off the web-scale scope and limit their work to specific well-described domains in order to guarantee higher accuracy and automation. The data extraction results from their initial prototype seem promising.

2.9 Semantic Annotation

Another technique for unsupervised information extraction is using semantic annotation. These methods rely mostly on manual or automatic annotations using either Microformats, Microdata, or RDFa. [HFV+11], for example, propose to automatically find reviews, identify review attributes, and annotate them using Google's RDFa vocabulary, by applying a combination of text statistics, named entity recognition (NER) techniques, and regular expression patterns.

On the other hand, [EMS10] apply token classification scores along with tag confidence scores in various active-learning strategies for sentence-specific annotations using a pre-specified tagset.

2.10 Conclusion

In this chapter, we gave an overview of some main achievements in the field of Web information retrieval. In particular, state-of-the-art systems and approaches were presented to illustrate how the problems of crawling, wrapper generation and maintenance, scaling to the Web and the Deep Web, and data integration are addressed. In addition, some techniques that exploit semantics to extract information were also summarized. In the remainder of this document, we present our contribution related to crawler and wrapper generation and the methodology adopted to locate data-rich web sources describing domain entities in large web sites.

Chapter 3

Synergic Crawling and Data Extraction

Large data-intensive web sites contain information of interest to search engines, web applications, and various online service providers. These sites often present structural regularities, embedding content into predefined HTML templates using scripts. Regularities are also apparent at the topological level, with similar navigational paths connecting the pages obeying to a common template. In this work, we extend the work introduced in [BCM08], to address two related issues for capturing useful information from structured large web sites: first, the pages containing the information of interest need to be downloaded; second, the structured content needs to be extracted by web wrappers, i.e., software modules that collect page content and reorganize it in a format more suitable for automatic processing than HTML.

In general, crawlers and wrappers generation have been studied separately in the literature. Numerous tools exist for generating wrappers, with different levels of automation. They usually aim at extracting information from semi-structured data or text, and they use to that effect scripts or rule-based wrappers that rely on the structure or format of the source HTML. In some cases, data extraction is based on ontologies or NLP techniques [SWL09, YCB+07, ECD+04]. Concerning the level of automation, hand-coded wrappers require a human expert, which becomes a cumbersome task for data extraction on large data-intensive web sites. Fully automatic wrappers have also been implemented [BCM08, ZL06], but they necessitate considerable data post-processing and may suffer from lower accuracy. In semi-automatic wrapping, the main focus has been on learning approaches that take several positive and negative labeled examples as input.

Various tools also exist to crawl web pages and entire web sites. Popular techniques start with seed URLs and either search on the pages for hyperlinks matching certain patterns, or consider all encountered hyperlinks and then apply selection

and revisit techniques for downloading target pages based on content or link structure [CVdBD99, JSHA06]. For deep Web crawling, some work has been proposed to obtain and index URLs resulting from different form submissions, e.g., [MKK+08]. However, the production of crawlers for structured web sites remains a subject with large room for improvement.

When it comes to large web data-intensive sites, it is commonly useful to extract data from a subset of the pages, in general pertaining to vertical domains. For example, in a finance web site, one may be interested in extracting company information such as shares, earnings, etc. In this case, there is no need to download all the site's pages about market news, industry statistics, currencies, etc. Thus we propose a new approach in which the two problems of crawling and wrapping are tackled concurrently, where the user indicates the attributes of interest while making one browsing pass through the site hierarchy. The specifications are created in the same contextual inference run. Moreover, the execution of the wrappers takes place while the crawler is collecting pages at the different levels of the derived web site structure. The mutual benefits are manifested as the produced simple wrappers extract the specific links followed by the crawling programs, and the crawlers are in turn used to obtain sample pages targetted for inferring other wrappers. We have developed a working system that offers these capabilities to non-expert users. It is based on an active-learning inference engine that takes a single initial positive example and a restricted training set of web pages. This active learning engine runs an internal evaluation on the extraction performance, using the knowledge acquired from the available sample set. It applies uncertainty sampling techniques to select query pages to propose to the user for feedback and refine the output. In this context, we also define the *Sampling Problem*, a problem often undervalued in the literature, which arises when the training sample set is biased. We show how it is mitigated by our approach when collecting a representative training set for the inference process.

Throughout this chapter, we consider an example web site that offers financial information on companies. We are interested in extracting from company pages data about stock quotes that is accessible in two different navigation sequences in the site topology: first, the home page displays companies' initials as links. Clicking on a letter leads to a listing of all the companies with this given initial. Each company name in turn links to a page containing the data to extract. Second, by filling and submitting a form on the home page, one reaches index pages grouping the companies by financial sector. Index pages are paginated, and by following the links provided in each paginated result list, the target company pages can be finally reached. We refer to this example while presenting our web site abstract model, on which we build the

interleaved crawling and wrapping definitions.

The rest of this chapter presents first our web site abstract model on which we build the interleaved crawling and wrapping definitions (Section 3.1), then our crawling algorithm (Section 3.2), the extraction rule classes defined for various types of wrappers (Section 3.3), the wrapper and assertion constructs that enhance our synergic crawling and wrapping tasks (Section 3.4), and a formalization of the sampling problem with a our proposed approach to collect a sample set (Section 3.5).

3.1 An Abstract Model for Describing Large Web Sites

To access data from data-intensive web sites, we aim to reach these data on pages collected by a crawler using wrappers tailored to the user's information needs. We note that large web sites are composed of hundreds of pages that can generally be grouped in few categories, such that pages of the same category share a common HTML template and differ in contents. These sites also exhibit topological regularities in the navigational paths that link the pages and page categories. By capturing these regularities, we describe large web sites with an abstract model of three interrelated levels: the *intensional, extensional,* and *constructional* levels.

Intensional Level Description

Our intensional model defines two main constructs for building schemes: the *Page-Class* construct describes a set of similar pages of the same category, while the *Request-Class* construct models a set of homogeneous *requests* (GET or POST) to navigate from the pages of one Page-Class to those of a consecutive Page-Class. In our model, *Request-Classes are typed*, in the sense that each Request-Class specifies the Page-Class that it leads to.

The above concepts are illustrated in the graph of Figure 3.1: for our example site, the home page can be modeled by a singleton Page-Class Home from which depart two Request-Classes, ByInitial and BySector, leading to two Page-Classes, IndexByInitial and Sector. IndexByInitial models the index pages grouping companies by initials while Sector describes index pages grouping companies by financial sector.

Both Page-Classes, by means of Request-Classes Overview and Composition respectively, lead to the last Page-Class Company whose pages contain detailed information about companies, one company on each page. In particular, Request-Class Next models the link leading to another page instance of the same Page-Class Sector.

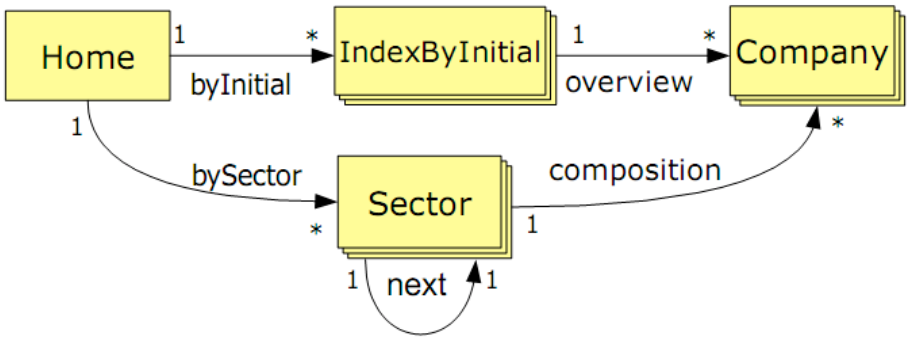


Figure 3.1: Intensional model example.

Extensional Level Description

An extensional listing of a Page-Class is a set of pages called Page-Class *support* that: (i) obey to a common HTML template and hence have similar structure; (ii) are about a common topic (each page represents a different instance of the same conceptual entity); (iii) are reachable by similar navigational paths, that is, by crossing pages that also correspond to predefined Page-Classes. Similarly, the extensional definition of a Request-Class is a set of requests called Request-Class *support* that: (i) lead to page instances of the same Page-Class; (ii) can be generated by clicking on comparable links or by different submissions of the same web form.

The two intensional constructs can be used together to build complex schemes, and to each scheme an extensional counterpart can be associated. That is, a scheme instance can be obtained by arranging the supports of every Page-Class and Request-Class involved into a graph that reflects the logical organization of the modeled site, where nodes represent page instances of a Page-Class in the scheme, while edges represent request instances of a Request-Class. We call an instance *empty* whenever every Page-Class (and hence every Request-Class) has empty support. An extensional graph for our example is partially shown in Figure 3.2.

Constructional Level Description

The constructional level in our model bridges the intensional and extensional descriptions by providing all the operative details needed to build the schema instances. Constructional elements are in fact the information that the system needs to start from the entry page, determine the possible navigational paths to follow, proceed to subsequent pages, and extract the attributes of interest to the user. These elements consist of: (i) for the *entry* Page-Class, the set $E = \{e_1, \ldots, e_n\}$ of addresses of the pages in its support (typically the URL e of the page is sufficient); (ii) for each Request-Class

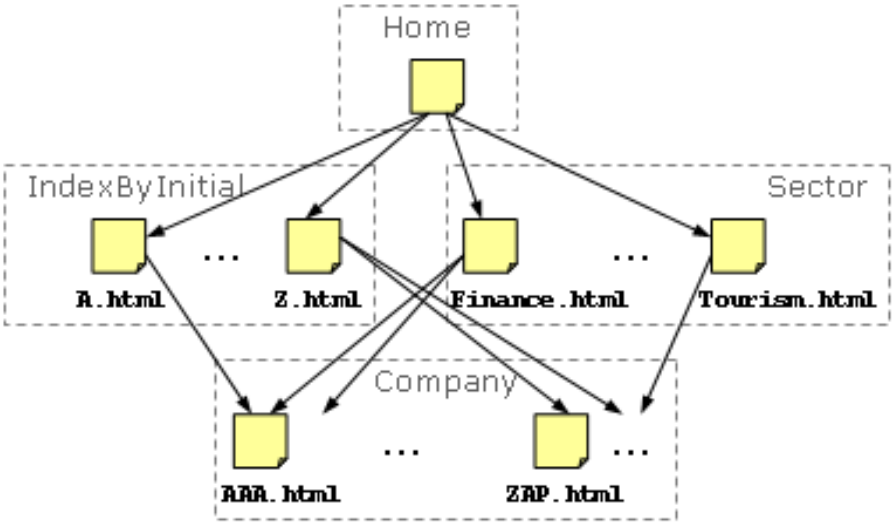


Figure 3.2: Extensional model example.

r, a function er^r that builds its support, given a page instance of the Page-Class from which it departs. We call this function an extraction rule since it extracts requests from pages, e.g., the initials hyperlinks on the home page. Two other constructional elements, *assertion* and *wrapper*, are added to our model in order to enhance the Page-Class constructs. They are detailed in Section 3.4.

3.2 Crawling Algorithm

Having defined our abstract model and how to build its instances, we explain how the crawler operates in this context. We first formalize the definition of navigational path, a main component for our crawling algorithm. On the intensional level, a navigational path $\mathcal{N} = (P_0 \cdot r_0^1 \cdot P_1 \cdot r_1^2 \cdot \ldots \cdot r_{h-1}^h \cdot P_h)$ is a sequence of Page-Classes and Request-Classes such that each Request-Class r_k^{k+1} is outgoing from Page-Class P_k and ingoing to Page-Class P_{k+1}, $k = 1..h-1$. It essentially corresponds to a path in the graph associated with the scheme. For our purposes, the interesting navigational paths start from an *entry* Page-Class (Home, in our example) and reach a *target* Page-Class containing the relevant data (Company). On the extensional level, the navigational path consists of *navigational trees*, where each tree is rooted at one entry page of the site and its descendants represent the pages visited by following the various requests in the support of the Request-Classes traversed.

A generalization of the user's sample browsing activity derives the navigational path definition as a sequence of constructional Page-Class and Request-Class ele-

ments. The crawler then finds all the navigational tree instances of this given path in the web site, in order to eventually download -strictly- all the pertaining pages.

Algorithm 1: Crawling algorithm based on the abstract model for large web sites

Input : A navigational path $N = (P_0 \cdot r_0^1 \cdot P_1 \cdot r_1^2 \cdot \ldots \cdot r_{h-1}^h \cdot P_h)$;
addresses $E = \{e_1, \ldots, e_n\}$ of P_0's pages;
a set of extraction rules
$\{er^{r_k^{k+1}} | k = 0, \ldots, h-1\}$

Output: The navigational trees T instances of N.

Let $T = \{t_1, \ldots, t_n\}$ **be** an empty instance of N;
foreach $e_i \in E$ **do**
  add the page with address e_i as root of t_i;
for $k = 0$ **to** $h - 1$ **do**
  foreach *page* $p \in \texttt{support}^{t_i}(P_k)$ **do**
    foreach *request* $r \in er^{r_k^{k+1}}(p)$ **do**
      Let d **be** the page obtained by means of r;
      insert d as a child of p in t_i;
      insert d in $\texttt{support}^{t_i}(P_{k+1})$;
return T;

$support^{t_i}(P)$	support of P in the navigation tree t_i
$er^{r_k^{k+1}}(p)$	set of requests associated with the r_k^{k+1} and extracted by applying the extraction rule er on page p

To do so, a crawler operates according to Algorithm 1. It starts with an input navigational path and an empty set of the corresponding navigational tree instances. Then it incrementally builds each instance by first adding a root page from the support of the entry Page-Class P_0, using to that effect the input addresses E. Subsequently, for each page p in the Page-Class under consideration, the algorithm uses $er^r(p)$ to build the support of the outgoing Request-Class r, that is, to extract on p the actual set of requests corresponding to r. These requests are sent to the web server in order to obtain new pages. The latter are added to the support of the Page-Class that constitutes the destination of the processed Request-Class. The crawler continues by iteratively picking each page p from the support of an already populated Page-Class and following its corresponding requests, once extracted. It thus incrementally builds other instances and the algorithm stops when all the requests have been sent.

3.3 Specification of Extraction Rules

To perform wrapping and crawling tasks in an interrelated mode, our system infers extraction rules. These rules are used in crawling to locate the requests to be followed,

and in wrapping to extract the relevant data from the downloaded target pages.

In Section 3.1, we have already discussed extraction rules for requests. We revisit the previous definition to also model wrappers for extracting relevant data from a page. An extraction rule er is then more generally defined as a function that locates and returns a set of strings s_i from the HTML code of page p: $er(p) = \{s_1, \ldots, s_k\}$.

In order to infer extraction rules, our algorithm takes as input a few positive examples provided by the user, highlighting the strings to extract (links or attributes). Eventually, only the rules compatible with collected user feedback are kept. Defining the class of inference rules to be generated constitutes then an important design choice. On one hand, a large and expressive class is more likely to include all the useful rules but may require many initial samples and a lot of user interaction before the correct rules are discerned. On the other hand, a limited and concise class of extraction rules requires less samples, and therefore less user interaction, but is also less expected to include correct extraction rules that are valid on all the pages. We address this tradeoff by opting for extraction rules based on XPath expressions and obtained from the union of three simple classes:

ABSOLUTE containing all absolute XPath expressions with positional predicates associated to each constituent tag, and generalized when a superset of several samples is needed.

URL-REGEX obtained by filtering the rules in the ABSOLUTE class with simple regular expressions, and used mainly for extracting links where the URL value obeys to a general pattern.

LABELED consisting of relative XPath expressions that make use of a fixed node considered part of the template to locate the string to extract.

These classes proved to work on more than one hundred real web sites with very good performance results, while maintaining simplicity and requiring very limited user effort to discern a correct rule.

3.4 Wrapper and Assertion Constructs

As mentioned, wrappers and assertions are constructs used in our constructional model to enhance the system's definitions of crawling and data extraction activities.

Wrappers are tools for extracting data values on web pages. Data wrappers generated by our system consist of rules taken from the LABELED class as they are less sensitive to variations on the page. They require that the node chosen as a reference be present in most of the target pages while being as close as possible to the data node.

We associate the wrapper element with the Page-Class construct so that when crawling to build a Page-Class support, if a wrapper definition is encountered, it is executed instantly on the downloaded page.

Assertions are Boolean predicates over web pages. They are formulated as a set of XPath expressions locating valid template nodes, and a page is said to satisfy an assertion if and only if it agrees with all the XPath expressions associated with the template of its respective Page-Class. As we relax extraction rules and allow them to extract a superset of the correct links, the system can detect and discard any *false positive* pages crawled by checking whether their template matches or not all the established assertions. Consequently, assertions replace the need for adding more expressive classes of extraction rules when the crawler's performance is not satisfactory, which means fewer rules produced in response to the examples provided by the user, and fewer subsequent examples required from the user to discern the correct rules [CM08].

3.5 Sample Set Selection

The input required for the semi-automatic generation of the crawling programs and annexed wrappers is provided through interaction of a non-expert user with our system through a browser-based interface. For the model generation, the system tracks user navigation and derives an *internal schema* from the user's browsing activity as per the model listed in Section 3.1, with the Page-Classes and Request-Classes of the web site along with the needed extraction rules. As for wrappers inference, the system generates extraction rules for the data selected by the user and evaluates them with the active learning module of the underlying inference engine. Then for any page with uncertainty, be it for the template or for the extracted values, the user is prompted to confirm the results, correct them, or even discard the page.

Sampling Problem

In order to produce correct and effective extraction rules, automatic and semi-automatic inference methods require samples with a certain quality of representativeness [CM08] that inexpert users cannot provide. Therefore the sample pages chosen by the tool to collect user feedback need to be representative of their corresponding Page-Classes. In addition, the navigational paths linking them together need to cover a rather wide variety of the possible paths in the selected informative domain. As a result, selecting sample pages introduces what we define as the *Sampling Problem*. This problem, often neglected, consists in determining which sample pages to choose in order to have "good" positive examples and guide an inexpert user in the inference process. Let us consider the following two scenarios:

Example 1: the downloaded sample pages all belong to one specific subset; say company pages from the Agriculture sector in our running example. Such pages would all contain the token "Sector: Agriculture".

Example 2: the downloaded sample pages share a particular variation in the template, such as the pages of top-ranked companies in our example containing a table with statistical data. The headers of this table constitute tokens that are not shared by other pages that do not rank first in their respective sector.

In these examples, sample pages may not be representative of all their Page-Class instances, as they contain some specific tokens that are not present in the remaining target pages. This can in turn affect negatively the crawler, were wrong template assertions derived from these tokens, as well as the data extraction process, were any relative rules to be built around these tokens. These points are summarized in Figure 3.3.

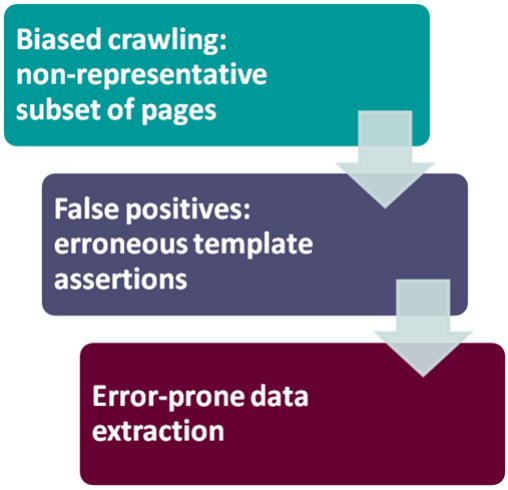


Figure 3.3: Sampling problem

Resolution

The manifestations of the sampling problem are: (i) in characterizing the valid pages template, (ii) in collecting good navigational path instances, (iii) and in generating accurate extraction rules for the data of interest. In the first case, the problem is to generate a template characterizing only relevant web pages and discard "false positives". A template consists of a set of valid tokens that are present in most of the sample pages at the same XPath location. The second case relates to the need of covering the domain's navigational paths with minimum bias, and is addressed by tracking and

generalizing the user's navigation at each step. The third case can then be resolved having collected a diversified page sample and derived a valid template. We propose the following approach to characterize valid page templates based on a statistical analysis of page contents and a learning algorithm requiring limited user effort:

- at each navigation level, consider all the page addresses obtained by generalizing selected links or form requests
- from this set of page addresses, which can be very large, choose a fixed percentage of random sample pages to download
- analyze tokens occurrence and data extraction results on the sample pages to train the classifier
- apply uncertainty sampling techniques to select query pages to propose to the user for feedback
- update tokens set, assertions, and extraction rules from user feedback

In theory, to cover all variations in page templates, one possible solution is to consider all the links at each level in the order of user navigation. The support of each Request-Class is fetched and the corresponding web page is crawled. Links pointing to paginated results are also followed and processed when applicable. Inference can then be performed on the entire set of pages from this navigation level. During the inference phase however, a similar process would burden the user with long waiting times for pages to get downloaded. This is due to the fact that data-intensive web sites are usually composed of hundreds if not thousands of pages that represent the entities in the given domain. Since one of our main goals is to minimize the load laid on the user, and keep her experience as close as possible to normal browsing and data labeling, we propose to choose a fixed percentage, which sets an upperbound on the number of downloaded pages and maintains an acceptable system response time from a user experience point of view.

With the proposed resolution technique, our system is able to collect and use a representative subset from the site pages to infer performant wrappers and crawlers relevant to the user's needs. At execution time, the constructional details, as described in Section 3.1, recorded in XML format, are used to guide the interleaved crawling and wrapping on the large web site. An example XML output file of the specifications generated by the system is shown in Figure 3.4. In the following sections, we describe in more details the process leading to the final definition of crawlers and wrappers, where a classifier is iteratively trained and active learning techniques are used when there is uncertainty in the system's inference rules.

```
<navigation>
<model name="FinancialCompanies">

<entrypoint url="http://financialdata.mywebsite.com" type="pageclass_0"/>

<pageclass type="pageclass_0">
    <requestclass type="requestclass_0" expectedtypes="pageclass_1">
    <clickall xpath="//HTML[1]/BODY[1]/DIV[3]/DIV[2]/DIV[1]/UL[1]/LI/DIV[3]/DIV[1]/A[1]"/>
    </requestclass>
</pageclass>

<pageclass type="pageclass_1">
    <assertion xpath="HTML[1]/BODY[1]/DIV[3]/DIV[2]/DIV[1]/DIV[2]/DIV/DIV[1]/DIV/P[4]/A[2]">
    <text><![CDATA[ Stock future ]]></text>
    </assertion>

    <extractor>
    <extraction-rule type="xpath" name="name" xpath="(//P[text()='Company Name:']/../H1)[1]"/>
    <extraction-rule type="xpath" name="price" xpath="(//SPAN[text()='Price:']/..)[1]"/>
    <extraction-rule type="xpath" name="change" xpath="(//SPAN[text()='Change:']/..)[1]"/>
    </extractor>
</pageclass>

</model>
</navigation>
```

Figure 3.4: Example XML output

3.6 Web Pages Analysis

We explained how our system collects a sample set of pages from the training web site. In this section, we describe first the characteristics of these pages, and then how these characteristics are used to guide the semi-automatic inference process.

Pages Characterization

Following our sampling technique, the collected set of pages can reasonably be considered to have the following properties:

- Relevant: the pages downloaded belong to the support of the target Page-Class(es) designated by the user. Therefore, they are of interest to the user and relevant to her information needs on the web site under consideration.
- Unbiased: the pages are collected according to our random selection technique, where all similar links at a given navigation level have the same probability of being selected and followed, which favors obtaining an unpolarized pool of pages.

- Representative: as a consequence of the previous two properties, we can say that the pages collected in the sample pool are representative of the overall set of pages in the Page-Classes that are of interest to the user.

Tokens Characterization

For the sample set of pages reached by following the generalized rule characterizing a Request-Class, we would like to numerically estimate the similarity in content and structure. In this way, the system would be able to automatically identify a page that does not belong to the Page-Class support or that contains major variations with respect to the rest of the pages. To do so, we individuate the leaf text nodes from the DOM trees of the pages, which we refer to as *tokens*. For each of these nodes, we keep a record of:

- The node's **content**: it represents mostly the text visible on the web pages when rendered in a browser.
- The node's **XPath**: it gives an indication on the text occurrence and location on the web page. The XPath in this case is an absolute expression that qualifies the complete path from the root of the document to the node under consideration, including along the way all the encountered node positions in the DOM tree. The constituents of an XPath expression are called *location steps*, where a location step is the part included between two consecutive "/" and specifying a tag name and its position.

The goal is to distinguish recurrent content from variable content by determining the presence or absence of a token t on a page p. Since each page in our target set generally displays data relative to one instance in the selected domain, the content coming from the instance record is variable, whereas template strings are more likely to be repeated on the different pages. Tokens' occurrence can then be useful to individuate common template strings from variable content proper to the instance. To account for minor variations on web pages and for repeated tokens (generally displayed by means of loops in the page source code), any two or more leaf nodes with the same text content are merged if their XPath can be generalized according to the model in Figure 3.5. They will be considered as the same token in later processing. Merging tokens is possible only when the XPath expressions have the same location steps count.

Tokens Filtering

For each page of the sample set, once its tokens have been extracted, we apply the tokens merging technique on them. Among these tokens, common template nodes

Rule 1:
tag[index]
tag[index]
⇒ tag[index]

Rule 2:
tag[index1]
tag[index2] (index1 ≠ index2)
⇒ tag

Rule 3:
tag1[index1]
tag2[index2] (tag1 ≠ tag2)
⇒ fail

Example:
//html[1]/body[1]/table[2]/tbody[1]/tr[1]/td[1]
//html[1]/body[1]/table[2]/tbody[1]/tr[2]/td[1]
⇒ //html[1]/body[1]/table[2]/tbody[1]/tr/td[1]

Figure 3.5: Generalization rules to merge equivalent tokens.

when identified can be used in crawlers and wrappers inference. As mentioned in Section 3.3, an extraction rule er is defined as a function that locates and returns a set of strings s_i from the HTML code of page p: $er(p) = \{s_1, \ldots, s_k\}$. We defined as LABELED rule an extraction rule consisting of a relative XPath expression. Such a rule makes use of a fixed node considered part of the template in order to locate the string to extract. This type of extraction rules is preferred when irregularities on the pages make absolute XPath expressions unuseful for extracting data, as the latter are prone to breaking with any variation of the location steps betwen the DOM root and node containing the string to extract. LABELED rules, when formulated relatively to a node that does not vary on the different pages, and that is close in the DOM traversal from the target node, prove to be more robust for data extraction.

We present below the statistical estimations computed to identify among the tokens set those that are useful for building LABELED rules.

We first define the **support** of a token t over a page p as follows:

$$S(t,p) = \begin{cases} 0 & \text{if } t \notin p \\ 1 & \text{if } t \in p \end{cases} \tag{3.1}$$

We denote as **occurrence vector** of a token t the vector containing the support value of t on each page of the sample set. Each location in the vector refers then to a specific page.

We define the **average support** of a token, $S_{avg}(t)$ as the average of the values

stored in its occurrence vector. In other terms, this represents the probability of t to appear on a given page. With N being the number of sample pages, we have

$$S_{avg}(t) = \frac{\sum_{i=1}^{N} S(t, p_i)}{N} \tag{3.2}$$

For our computations, tokens with $S_{avg}(t) < S_t$ are considered variable content and not part of the common template, where S_t is an empirically defined threshold. Variable content is discarded when selecting reference nodes for inferring extraction rules. Furthermore, we filter out banners, headers, footers, and other content that is not unique to the target Page-Class, as these are likely to produce erroneous extraction rules that do not necessarily point to the target attributes to be extracted. The remaining tokens with occurrence higher than the threshold constitute valid candidates for the generation of LABELED rules. Figure 3.6 shows two car instance pages with the recurrent content possibly classified as template nodes circled. The rest of the strings are considered variable instance content.

For a given attribute to be extracted, we select the frequent token that is closest in the DOM tree as a potential reference token for its the extraction rule. This choice further reduces chances of interposing variable location steps that may break the extraction rule when applied to other pages. However, uncertainty in the data extraction process may still occur.

3.7 Learning Approach

In order to minimize uncertainty in data extraction incurred by variations among pages of the same Page-Class, our system adopts an iterative learning approach that incorporates what's incrementally learned from the user's navigation and eventual input during the inference process.

As the user navigates through the web site pages indicating links to follow, she can highlight data of interest to extract. This triggers the system to generate extraction rules for these data attributes. However, before storing the rules in the output specifications of the model for later execution on the web site, the system runs an internal evaluation on their extraction performance using the knowledge acquired from the reduced sample set. If needed, the system requests user feedback to clear out any inconsistencies and refine the extraction rules.

To explain the process, we first present a few definitions.

We denote as **initial positive example** or p_0 the page on which the user highlights the data attributes of interest.

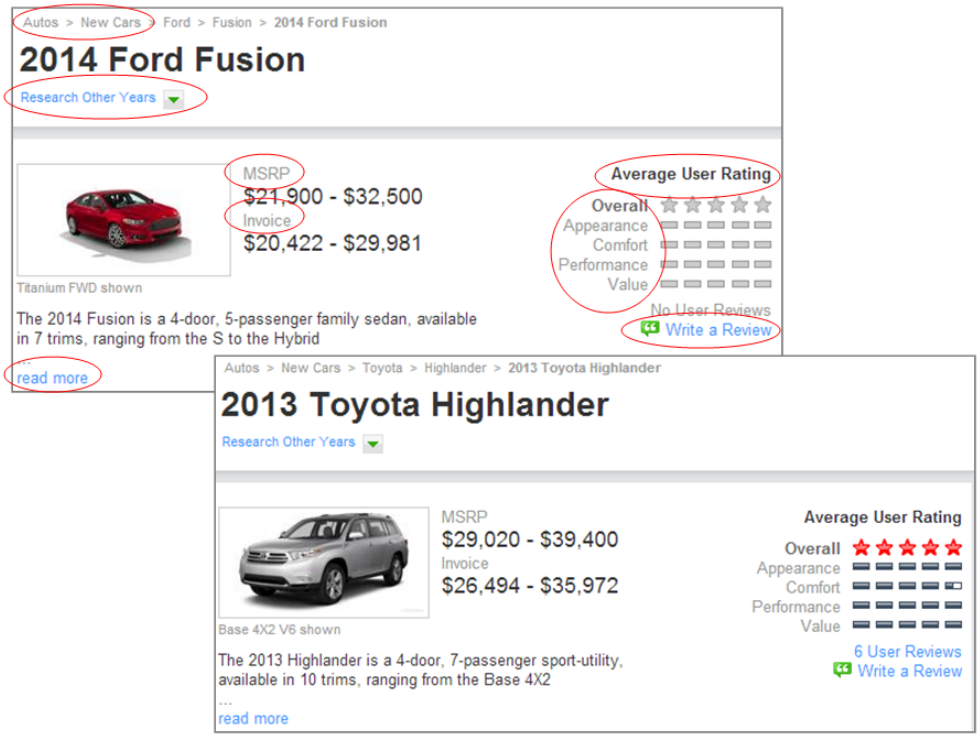


Figure 3.6: Identifying recurrent nodes

The other $N-1$ sample pages downloaded constitute the **unlabeled set** U, in the sense that the user is not initially required to specify the data attributes location on any of these pages.

We call P the set $U \cup p_o$. It is the **pool** of all downloaded target pages from a given Page-Class.

A **confidence score** S_C of a page p_i is the ratio of the weighted average support of its tokens with respect to all tokens collected over the representative sample set P.

$$S_C(p_i) = \frac{\sum_{j=1}^{T} S(t_j, p_i) * S_{avg}(t_j)}{\sum_{j=1}^{T} S_{avg}(t_j)}$$

where T is the total number of tokens.
This score gives a numerical estimate of whether the tokens commonly occurring on the pages in P are present on p_i or not.

A **rule extraction score** S_E is a number reflecting the count of values extracted by a given rule r_j on a page p_i. Since we are extracting single attributes, ideally a correct rule would extract one value when that value is present, and zero values when it is omitted on the page (optional field). We want to evaluate the correctness of the selected reference token. Therefore, we compute this number relatively to the presence of the reference token on the page. If the latter is not present on that page, we conventionally assign a score of 1.

$$S_E(r_j, p_i) = \begin{cases} \frac{Occurrence(Token_{r_j}^{p_i})}{Count(Values_{r_j}^{p_i})} & \text{if } Count(Values_{r_j}^{p_i}) > 0 \\ 0 & \text{if } Count(Values_{r_j}^{p_i}) = 0 \end{cases}$$

where $Occurrence(Token_{r_j}^{p_i})$ is the number of occurrences of the reference token for rule r_j on page p_i and $Values_{r_j}^{p_i}$ are the attribute values extracted by r_j on p_i that are not `Null`.

A **page null score** S_N is the ratio of `Null` extracted values on that page over the total number of attribute values to be extracted.

$$S_N(p_i) = \frac{Count(\texttt{Null}, p_i)}{Count(Attributes)}$$

Note that for the initial positive example page

$$S_E(r_j, p_0) = 1, \forall r_j$$

and

$$S_N(p_0) = 0$$

Finally, a **data type score** reflects the distance between the type of each positive example and the type of extracted data on the page in a defined type hierarchy.

In fact, recurring data types are observed in the information displayed on entity pages. A defined type hierarchy can be useful for the system to assess the homogeneity among the values extracted by a rule on the given set of pages, as well as compare the extracted values type with that of the example initially selected by the user. Extracted data is matched to one of the types in the hierarchy, starting with the most specific ones towards the tree leaves. It is then compared to the least common ancestor type of the user-specified positive examples for that attribute. The further the two types are in the hierarchy, the lower the score for the rule extracting that data. The type hierarchy specification is configurable and can range from very simple and generic to more elaborate and specific. A well-defined type hierarchy allows to better estimate the validity of extraction rule results. A sample implementation of the type hierarchy is illustrated in Figure 3.7. In this example, the Alphanumeric type is at the

root; Measure and Phrase are its subtypes. Percentage, Weight, and Temperature are examples of Measure subtypes. Further subdivisions of types have been abstracted by rectangles symbolizing child subtrees. Different hierarchy models can possibly be implemented.

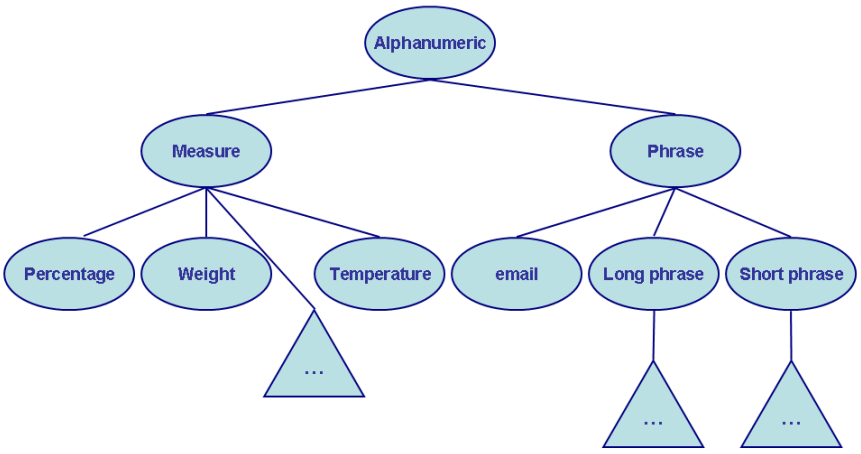


Figure 3.7: Sample type hierarchy

To summarize, when computed for the pages in the sample set, the various score definitions presented above can give an estimate of the following:

- The similarity in content of a page compared to the rest of the sample set P
- The conformity of data extraction on this page as compared to the initial positive example designated by the user.

These values constitute the features that will lead the system in its learning approach for the inference of robust extraction rules and will highlight unconforming results that may require particular attention.

Iterative Wrappers Evaluation

In our semi-automatic approach, one of the main goals is to minimize the user effort in terms of labeling data on different pages. Another main goal is to generate extraction rules starting with a scarce training set, and have them provide satisfactory results even when applied to the entire set of pages in the support of the target Page-Class.

Therefore, we need to assess how 'fitting' the inferred extraction rules are with respect to the user's initial information needs by applying them to the various pages in U. We reiterate that for our purposes, the set $P = U \cup p_0$ is assumed representative of the target Page-Class pages, as listed in Section 3.6, and thus we limit our computations to this set. In the spirit of active learning, the system is supposed to 'learn' the correct extraction rules on the 'unlabeled pages' (those not yet presented to the user) by using information from the ones already labeled. Moreover, if additional user feedback is needed during the inference process, the learner system, rather than the user, is expected to choose the *query page* that would be most informative for its learning process.

Since we start off with a training set composed of one element p_0, the problem is similar to online machine learning. Additional training data may come in later, one instance at a time. The main difference is that in our case we do not dispose of an example after using it for learning, because we are learning the rules for the same attribute as expressed by the extraction goals of the same user in one system run. That's why we opt for a cumulative approach that iteratively refines the learning parameters with each additional example collected from interacting with the user, until a stopping condition is met.

The combination of scores defined in the previous section constitute the basis for the system to estimate the relevance of the rules on the pages in U as compared to the user's information retrieval objectives. Based on the scores collected over the various pages, the system attempts to evaluate the rules performance on these pages and classify them as valid or invalid. The process is faced with important challenges with the presence of optional attributes and possible template variations, even on the most structured web sites. As a result, it is not possible for a system to automatically conclude with absolute certainty that extraction results not conforming to the positive examples are failures due necessarily to an invalid page or to invalid extraction rules. To address this problem, our system adopts a probabilistic classification reflecting the likelihood of the couple $\{p_i, r_j\}$ to be valid or not. Then, to resolve any classification uncertainties, the system requests user intervention.

Each time the system receives feedback from the user, it is incorporated with the labeled examples to enhance the classification model and extraction rules are refined. Rules evaluation on the remaining sample pages is then repeated, until no more uncertainties are encountered or a limit number of queries has been reached.

Active Learning with Uncertainty Sampling

The classifier that evaluates rules r_j on the pages p_i is implemented as an active learning module. In general, in a learning model, there are a number of independent variables representing observations from which the system learns. When evaluated together, they produce an outcome whose value depends on these variables' combination.

For our system, the variable observations are the various scores estimating a page's conformity to a common template structure and the data extraction results of the rules inferred for the attributes sought by the user.

The outcome represents the odds that the couple $\{p_i, r_j\}$ is valid. This is a continuous value that in turn can be translated into a binary output as "valid" and "non-valid" classes.

In supervised learning, a model's parameters (or coefficients) are initially fitted by means of training on existing examples, such that the error between the predicted outcome and the actual outcome is minimum. The model's hypothesis function thus obtained can then be used to predict the output when new unseen values of the independent variables are given.

The major constraint in our system is that the classifier is initially trained only with the positive example p_0 and a negative example p_n in order to fit the model's parameters. p_n can be any page not belonging to the target Page-Class, such as the home page, for example. This has the advantage of excluding cross-site tokens like menus, headers, and others, from the candidate tokens pool. The reason to have only one explicit positive example is mainly to minimize the user effort, and allow the system to subsequently decide if further training examples are needed.

However, the system can make use of the available unlabeled pages. It can apply the obtained hypothesis function on the remaining pages in U to predict their classification based on their content and extracted values, and estimate their conformity or variations. This fact favors a semi-supervised learning approach.

By applying the learned model on the unlabeled pages, our classifier computes the validity odds for each and outputs $Y = 1$ or $Y = 0$, such that $P(Y = 1|X)$ is the probability of the extraction results on a page being valid, while $P(Y = 0|X)$ is that of the extraction results on a page being invalid. In this case, X is the vector of features corresponding to be the different numeric scores presented earlier, namely S_C, S_E, and S_N, which we evaluate for each page. We note that these feature values are scaled between 0 and 1.

When the evaluated result on a page p_i is similar to the one from the positive example, the odds of validity are high and $P(Y = 1|X_{p_i}) \approx 1$. Querying the user on

p_i would not bring much additional information to the learning system. On the other hand, if the outcome for a page is not so close to the positive result, $P(Y = 1|X_{p_i})$ diminishes. A probability close to 0.5 indicates that neither validity nor invalidity can be asserted. In this case, that page is automatically picked to be shown to the user to clarify the uncertainty. The user can then provide feedback to confirm the extraction rule results, correct them, or even choose to discard the page altogether if it is not conform to the common template of the Page-Class. In particular, when the user indicates that an extraction rule needs correction, her input triggers the update of the system's selection of template nodes for the LABELED extraction rules. New rules are then generated with different reference nodes in compliance with all the feedback collected so far. Moreover, when the user discards a page, its tokens are removed from the valid candidate tokens repository to avoid any future generation of broken rules.

After the user feedback is recorded, the query page counts as a new positive or negative example and is added to the classifier's training set (and thus removed from U). The computation of the various scores on the remaining pages in U is repeated, as well as parameters fitting, in order to reassess the rules performance and the need for a new query. We note that to avoid incurring an excessive query load on the user, we set an upper limit of the number of queries allowed. However, we observed experimentally that few queries are usually enough to find well-performing rules. The quantified evaluation of results with respect to maximum queries allowed are presented in the experiments section.

3.8 Experiments

In this section, we summarize our experiments conducted with the system prototype, called CoDEC, implemented as a Mozilla Firefox extension. We use our prototype for generating specifications and executing them on real web sites to analyze the performance on a wide variety of heterogeneous topics, page templates, and navigational paths. We evaluate our experiments by computing the results' precision and recall. Moreover, the F-measure is also computed as

$$F = 2\frac{P * R}{P + R} \tag{3.3}$$

This value, ranging between 0 and 1, reports the harmonic mean of the precision *P* and recall *R*. For crawling, we evaluate the set of downloaded pages as compared to the set of actual target pages. Whereas for wrapping, we evaluate the values extracted by the generated rules with respect to the correct set of attribute values on the target pages. Larger *P*, *R*, and F-measure values imply better results.

Table 3.1: Crawling results summary

Total # web sites	Total # pages crawled	Precision	F-measure
100	208769	0.99	0.99

Table 3.1 sums up the experiment results of our crawling techniques on 100 different sites belonging to various domains. The simplest class ABSOLUTE was sufficient to extract most of the links leading to target pages. In few cases, where other links leading to non-target pages were located very close to the correct links, URL-REGEX extraction rules improved the precision by discarding the links that do not match an inferred URL pattern. One crawling limitation was the inability to discard target pages with the same template but different semantic entities, such as pages for coaches downloaded with those of players in a soccer web site.

For evaluating data extraction, we consider a subset of the previous sites evaluated for crawling, covering a variety of vertical domains. In Table 3.2, we summarize the number of different web sites and corresponding target pages on which the wrapping was performed and evaluated, in addition to the total number of attributes manually designated for data extraction, and that of the optional ones, grouped by vertical domain. Optional attributes are those that occur only on a fraction of the pages in the same Page-Class.

Table 3.2: Experiments summary

Domain	# sites	# pages	# attributes	# optional attributes
FINANCE	3	610	4	0
BASKETBALL	4	2310	7	1
SOCCER	4	1214	4	0
MOVIES	3	3046	7	6
RESTAURANT	3	5998	4	0
UNIVERSITY	2	3998	4	1

Table 3.3 lists the different domains along with the corresponding attributes selected for data extraction. Attributes that are optional are enclosed with parentheses.

Table 3.3: Extracted attributes by domain

Domain	Extracted attributes
FINANCE	Company Name, Open, Close, Volume
BASKETBALL	Team, Player, Height, Weight, Age, Position, (College)
SOCCER	Player, Height, Weight, Position
MOVIES	Title, (Time), (Director), (Distributor), (Rating), (Genre), (Release Date)
RESTAURANT	Name, Cusine, Phone, Address
UNIVERSITY	Name, (Type), Phone, Web site

Table 3.4 summarizes the results of testing our wrapping techniques with the listed attributes on the selected subset of web sites. In general, accuracy is very high for clean, well-structured page content. However, when several attributes on a page are optional, their inconsistent occurrence and the resulting variable attribute layout are likely to cause extraction rules to fail. This explains the observed low recall on Movies experiments, for example. Major variations in page templates also affect the performance of extraction rules, such as the case of pages in the Restaurant domain, where multimedia and dynamic content is commonly injected. Moreover, we noted that valid extraction rules cannot be generated when poor HTML formatting affects user selection of data.

Table 3.4: Wrapping results summary

Domain	Precision	Recall	F-measure
FINANCE	1.00	1.00	1.00
BASKETBALL	0.98	1.00	0.99
SOCCER	0.99	1.00	0.99
MOVIES	0.99	0.88	0.92
RESTAURANT	1.00	0.87	0.93
UNIVERSITY	1.00	0.98	0.99

All in all, the overall results collected on the different web sites support the effectiveness of our tool.

3.9 Related Work

The methodology we presented and the system that supports it combine techniques from crawling, wrapping, and machine learning. We present in this section some related work in each of these areas and list what differentiates our approach from those existing in the literature, where applicable.

Wrapper Inference

Data extraction for structured web sources has been widely studied, as shown in the various surveys on wrapper generation [LRNdST02, FMM+04, CKGS06, FDMFB12]. In Chapter 2, we gave an overview of numerous techniques available in the literature and several works on wrapper specification, inference, and evaluation of extraction rules with different levels of automation.

Moreover, some research efforts have emerged that rely on the DOM tree of the web pages to define wrappers using XPath expressions [Ant05, CCG06]. Other efforts aim to apply DOM tree alignment [CM06, KC10, HB02] to discover repeated patterns and detect attribute values for extraction in the HTML code of web pages.

On the other hand, our approach focuses on how to contextually specify, create, and execute simple and interrelated crawling and wrapping algorithms, rather independently from the underlying inference mechanisms of extraction rules. Additional, or potentially more complex, classes of extraction rules can be added to the system without modifying the principles of its operation.

However, the concept of extracting template tokens introduced in [AGM03] was adopted in our template tokens detection approach. In fact, the authors propose a model describing how in semi-structured web pages, data values typically coming from a database are encoded into the pages using templates. Their algorithm discovers sets of words with similar occurrence patterns in multiple pages and these words are considered to be the common template tokens. By determining the page template, it is possible then to automatically identify and extract attribute values fed into it on each page. In addition, equivalence classes are defined as large sets of tokens that occur together, with the same frequency, and appear on a large number of input pages. Such token sets are unlikely to be formed by chance and thus are considered as part of the template. Their method is even able to differentiate token roles on the page, using their occurrence context.

Our approach, on the other hand, does not implement this thorough template deduction algorithm, mainly for efficiency reasons during the user's navigation. We do however parallel their definition of equivalence classes by creating occurrence vectors for each text token to represent its occurrence frequency on each of the sample pages. Similar occurrence vectors are grouped together to form classes, which are in turn treated as one entity for further processing. The large support and size of a class determine its validity, while low values cause the tokens of the class to be disregarded. Classes also appearing on non-target pages are filtered out as well. In fact, the goal of our heuristic is mainly to improve the selection of reference tokens in constructing relative extraction rules. Therefore, only tokens that exhibit *invariant* properties on the target pages should be taken into consideration.

In [SFG+11b, SFG+11a], Sellers A. et al propose a formalism for data extraction by describing and simulating user interactions on dynamic sites. However, the declarative specifications are defined by an expert programmer and not derived from an actual user's navigation.

Also, [GMM+11] implement a web scale system for data extraction that generates extraction rules on structured *detail* pages in a web site. They apply extensive calculations for clustering pages according to template similarity and rely on several user inputs for annotation and rule learning. Their work is different in scope from ours since we propose a synergic crawling and wrapping that can cover various Page-

Classes in a web site while deriving information discretely from the user's browsing activity.

Crawling Techniques

Crawlers have been discussed in Section 2.2. In particular, a few works [CMM05, KLHC04, LNL04] have addressed the automatic discovery of content pages and pages containing links to infer a site's hierarchical organization. These approaches mainly aim at finding paths to all generic content pages within a web site, often with some limitations of specific hypotheses to allow automation.

In contrast, we aim at semi-automatically gathering pages from a selected class type of interest to a user, with a minimal human effort.

The work by [VdSdMC06] partially inspired our URL-REGEX class, as they use URL patterns to specify crawlers in their GoGetIt! system. However, our experiments show that many web sites cannot be crawled in this restrictive way alone. In addition, they adopt a breadth-first strategy to compare the site's pages DOM trees with a provided sample page, while our system works on a small set of sample pages presented to a user for feedback.

Active Learning

Early information extraction approaches started adopting supervised and unsupervised machine learning techniques for wrapper induction [Fre98, Kus00, Coh03], with tradeoffs between coverage and accuracy. Classifiers and statistical methods have been widely applied in different contexts to enhance the data extraction process. As we also presented in Chapter 2, learning-based data extraction spans both the semi-automatic and automatic research efforts. The development of these approaches came as a response to the inefficacy of hand-coded wrappers that are highly accurate but do not scale.

In recent years, active learning techniques have gained popularity among Web information retrieval solutions. Active learning is a sub-field of machine learning that offers greater accuracy with fewer training examples. In fact, [BHV10] state that active learning approaches may result in exponential improvements over the number of training samples with respect to traditional supervised approaches. Differently from unsupervised learning and automatic techniques where a lot of post-processing is needed, and from supervised learning and manual techniques where a human expert is required to provide a considerable effort, active learning is a middle-way solution that relies on positive and negative training examples and on exploiting user feedback for results refinement. The learning program however is allowed to choose the data

from which it learns. This means the active learner can pose selected queries to the user, mostly in the form of data labeling request, in order to improve its overall performance. When labeled instances are expensive to obtain, active learning aims to minimize the cost of labeled data by requesting as few labeled examples as possible. For these reasons, active learning techniques have been applied in many areas of Information Retrieval, such as annotating and labeling documents [AHVC11, SC08, LC06], wrapper generation [RC11, IS06, MMK03], and improving search results ranking by exploiting the user's iterative relevance feedback [TL11, LHZ+09].

Our system implementation in this chapter relies on an active learning engine that operates during the inference process in order to optimize the system's interaction with the user. This approach aims to achieve a better performance with less training. As described in Section 3.7, our learning scenario fits the *pool-based sampling*, where a small set of labeled data and a large pool of unlabeled data are available. With every new input, the active learner evaluates the informativeness measure on the entire pool before deciding which query to present to the user. This is in contrast with the 2 other common active learning scenarios, namely stream-based sampling, where instances are scanned sequentially, and membership query scenarios where arbitrary instances are presented to be labeled, a popular technique in natural language processing.

Informativeness measures in active learning differ according to the selected query strategy. Uncertainty sampling, which has been adopted in our system, queries the user on the instance about which the model is the least confident. For our implementation, logistic regression was applied to query the instance closest to the linear decision boundary. Other uncertaintly sampling techniques could opt for maximizing entropy, or reducing classification error, etc., depending on the instance data modeling and the available information. Our iterative training process has been inspired by [LHZ+09], where the authors iteratively train a logistic regression while incorporating the feedback from the user into the model. They use this technique however in the context of improving the relevance of image search results, whereas we apply it for improving the crawler and wrappers inference while minimizing the user's effort.

Other popular active learning query strategies include query by committee, expected model change, expected error reduction, variance reduction, and density-weighted methods. More details can be found in [Set10, FK03]. Choosing one strategy or the other also depends on the model of the sample space and the aim of the learning algorithm. Information density in particular was integrated into our model to complement uncertainty sampling by querying an instance that is *representative* of other instances in the distribution, in other words that is uncertain and also exhibits some similarities to other unlabeled instances. This is done to avoid spending effort on querying outliers

that do not have any impact on improving the model's accuracy [SC08].

3.10 Conclusion

We presented a new semi-automatic methodology for synergic crawling and wrapping in the scope of information retrieval. Our contributions can be summarized by: (i) a formalism to specify interleaved crawling programs and wrappers concurrently over structured web sites; (ii) the introduction of the *Sampling Problem*, which illustrates how randomly chosen samples can be biased and negatively impact the inference tasks; (iii) an approach to mitigate the effects of the sampling problem by requiring minimal effort from an inexpert user; (iv) and an experimental evaluation to validate our proposed techniques. Our experiments revealed encouraging results, and can be further improved with the potential inclusion of semantic assertions and a mechanism to deal with any optional attributes with changing location on the page. Moreover, the data gathered during inference is a rich source of information about the vertical domain addressed by the web site and about the entities listed on its target pages. This wealth of information can be exploited to initiate an automatic search for other similar sources on the Web and eventually complement, enrich, or validate the attribute values collected by the wrappers defined for the training site.

Chapter 4

Vertical Domain Explorer

Entities generally represent real-world concepts, such as a person (writer, basketball player, singer, etc.), a product (book, camera, movie, clothing, etc.), an organization, a business, or other. In large data-intensive web sites, sections related to an entity in a given vertical domain consist of a large number of data-rich pages, each displaying attribute values for one instance of the given entity. We refer to this set of pages, which in general share a common structure or template, as *entity pages*. Figure 4.1 shows for example 2 pages, each representing a distinct instance of the `Book` entity offered on a book selling web site. Common attributes for the `Book` entity, such as title, price, ISBN, publisher, publication date, etc., are listed on each page. The actual values displayed on each page pertain to one particular `Book` instance and usually originate from one record in the underlying database. Therefore we use *instance page* to refer to one individual example of the entity pages. Some have a label preceding the attribute value, while others, like the title and price, don't.

In Web data extraction, wrappers inferred on a large data-intensive web site are usually able to extract selected attribute values on a subset of the site's entity pages, if not on all of them. However, the attribute values collected by these wrappers for the various entity instances remain limited, in the best case, to the set of data stored in the underlying repository of the web site.

Ideally, to build a rich repository of instances of a given entity that serves the unlimited search needs of the users on the Web, data aggregators would like to collect all the possible instances available for that given entity and apply data extraction for its attributes on a large web scale. For this purpose, in order to complement, enrich, or validate the data gathered from the training site used to infer wrappers and/or crawling specifications, it is useful to extend it with instance data available on other sites that offer similar information on the same type of entities.

This requires performing two main operations:

1. Find other large data-intensive web sites offering pages on the given entity

2. Download the entity pages to extract instance data on them

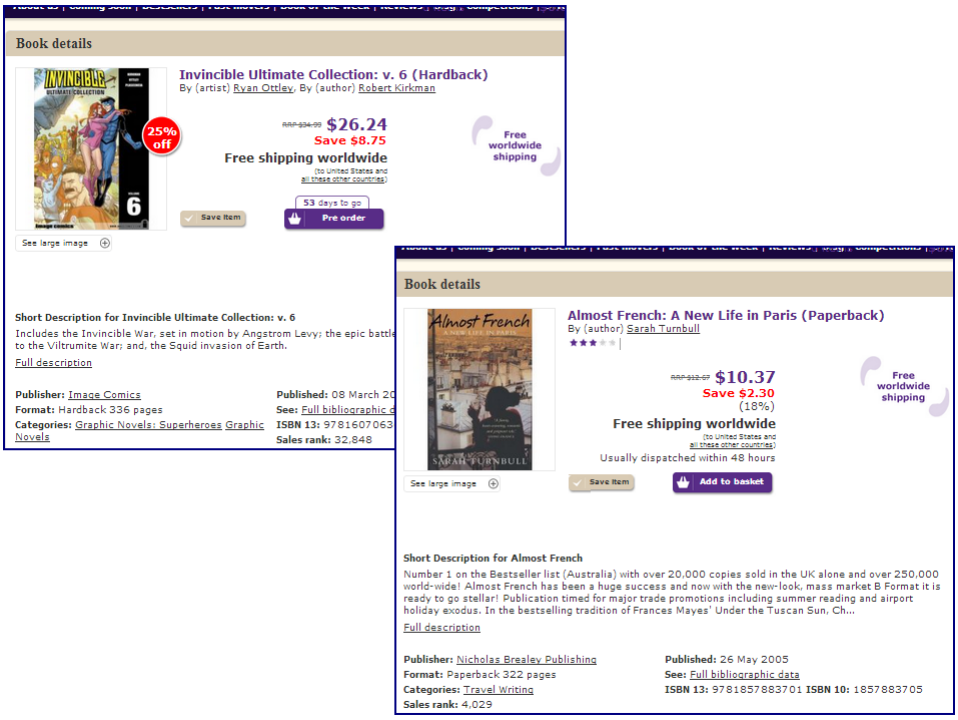


Figure 4.1: Data-rich instance pages of the Book entity

Going back to our `Book` entity example, a data aggregator or vertical search engine for the `Book` domain would not only need to extract all the different attribute values of the book instances within the training web site, but also find new data-intensive sites that display pages about books and download their data as well.

With the synergic crawling and data extraction system we presented in Chapter 3, the user indicated her interest regarding the entity information she wishes to extract and which subsection of a given large web site contains the entity instance pages. In particular, the user specified how to navigate the site hierarchy to arrive to the target pages, and identified attributes to extracts on these pages.

Additionally, during the inference process, the user's input allowed the system to collect valuable content from the web site and information about the domain of interest itself.

This knowledge is the starting seed to perform the first operation, exploring the Web in search for other similar entity pages.

Therefore, in this work we address the problem of locating large data-intensive web sites in a given vertical domain. To achieve this, we propose an approach that can be exploited by vertical search engines to enhance web page crawling and filtering by using domain knowledge in a pre-processing phase. Our approach uses the information collected during inference on a training site as input in order to find other large web sites or web site sections containing instance pages about the entity of interest. This is based on the fact that there is a large amount of overlap on the Web among sources in the same vertical domain [DMP12], so information we have from one site, or possibly more, can lead to overlapping information in sources not yet discovered. For example, if our training web site in the `Book` domain contains an instance of `Jane Eyre` book, an instance of `Oliver Twist`, and an instance of `Madame Bovary`, among others, and we are able to find another web site that also lists instance pages for these 3 books, it is very likely that this new web site is a data-intensive source in the `Book` domain (or contains a sub-section on books, if it is a more general web site covering various verticals). We aim to perform this search and discovery task in a fully automated way, independently of the web site on which the inference was performed. The pages found then individuate new and relevant targets for crawling and data extraction. The rest of this chapter presents our proposed automatic solution with its components that apply structural and semantic techniques to first search the Web with formulated queries, then efficiently filter the obtained results and identify relevant sources.

4.1 Targeted Search Approach

Given a large web site in a vertical domain, we presented in Chapter 3 a synergic method to locate entity pages and extract instance data on them, and described our CoDEC system that implemented it. The approach was facilitated by exploiting redundancies in the site's HTML structure, navigation paths, and content tokens.

When this approach needs to be extended to other large web sites, the first step is to locate these similar sites on the Web. One way is to run a manual search, which would not scale and carries the risk of user bias in the query formulation. We propose an automated way that builds on the information gathered during inference with CoDEC, and exploits wider redundancies in entity occurrences on the Web, in order to find new web site entity pages for a given domain. Once identified, these sites can eventually serve for data extraction and enriching the information repository, which in turn would enable further search for more web sites, until no new information is discovered.

This automated search is made possible by extracting, from the collected information repository, keywords that represent the domain, the entity of interest, and a

few selected instances, in order to run web queries and locate pages with overlapping information. The more frequently a web site overlaps with our repository instances and shows up in search results, the more likely it is a large data-intensive web site matching our target.

The lookup being launched through a search engine, the queries are likely to generate a large number of results. Many of these are not data-intensive entity pages, as the Web contains a huge amount of pages of various formats and content types, such as blogs, news, reviews, personal or corporate sites, which are of no relevance to our targeted approach. Thus the need to apply a filtering mechanism to confirm or discard a returned result page based on its relevance. With scalability in mind, this filtering task also needs to be automated and it can be performed at two levels: First, at the level of the returned URLs, to determine the subsets of search results to be further examined; second, the pages pointed to by these URLs need to undergo a check for semi-structured data formatting, in which case they can be considered candidates for eventual data extraction tasks.

All in all, our proposed system can be viewed as a controller that operates in different and complementary stages to determine where to possibly locate entity pages, which pages to download, their level of relevance to the targeted search, what display characteristics their content exhibits, and eventually where on the page to extract data attributes of interest.

Starting with an input collected while running the inference engine of the CoDEC system, the system then outputs a set of similar instance pages found on newly discovered sites on the Web, and points out specific content nodes on them.

This set of instance pages can be fed into a crawler to download the rest of instance pages on the discovered web site, and the identified content nodes can be used for extracting entity data into the vertical domain's repository. However, these extensions are not part of the work presented in the current document.

4.2 Proposed System Model

Entity instance pages are spread out on the Web, populating many large data-intensive web sites. Based on this fact, finding and collecting these pages in a repository for a pre-determined domain consists mainly of a targeted web search activity, followed by an appropriate result filtering process. We maintain that it is sufficient to identify in this way a subset of the relevant instance pages in a given large web site in order to efficiently feed a more extended focused crawling mechanism within that site.

To conduct the above activities, we propose a system that exploits the data and domain knowledge collected on a large web site during the synergic inference of crawlers

and wrappers. The gathered information is then used to facilitate discovering other potential large web sites in the same vertical domain and eventually extract instance data from their entity pages.

The system is conceived of four main connected components representing logical sub-activities that perform specialized and complementary tasks, as illustrated in Figure 4.2. The components' goal is to direct and enhance the entity page search and filtering process as follows:

- The input to the first component in the system is the information obtained from the training web site, in particular, the template tokens from the sample pages. This component then analyzes these tokens in combination with the domain identifying terms and with the help of a semantic evaluator to output keywords for the entity of interest (Section 4.3).
- Keywords are then passed to the query generator component that builds a series of boolean queries and sends them to a web search engine (Section 4.4).
- The URL results from the respective queries are passed as input to the following component, the URL analyzer. The latter applies a clustering algorithm on the set of URL results and keeps only relevant clusters for further analysis (Section 4.5).
- These are in turn passed to the page evaluator component that downloads the pages from the Web and identifies entity-related semi-structured content (Section 4.6).

When various entity instance pages are identified on a web site, a customized explorer or crawler can then find the page that links to them on that site and download all the remaining instances that it contains. In addition, the system keeps record of the location (containers) of relevant data on the pages, facilitating further data extraction efforts.

The following sections describe in more detail each system component and the functionality that it implements.

4.3 Finding Entity Keywords

Aside from search providers, large companies, and research groups that have the capabilities to crawl large portions of the Web and store them in their local repositories, any general effort to find content on the Web has to pass through a search engine by providing some keywords. In fact, the terms used in a query constitute the "interface" between the search need and the information indexed in the content repository of the

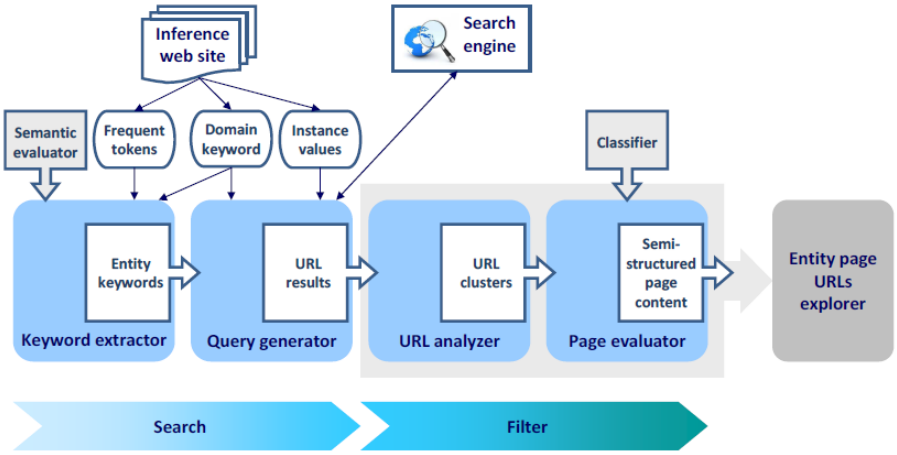


Figure 4.2: System components

search engine. Choosing "good" search terms usually leads to a better communication of the need, which minimizes the distance between what's sought and what's found. In addition to full or partial word matching, search engines today use implicit techniques and measures such as contextualization, approximation, search history, content freshness, PageRank, etc. to provide more useful results in a ranking order that would be relevant to the information need. This is acknowledged by most big search engines, such as Google [1] and Microsoft's Bing [2]. However, keywords remain essential to state the initial more or less specific intent of the search. Furthermore, when explicitly specified, a search engine can process the query as a boolean search and return only exact matching results among their indexed pages.

This section presents the keyword extractor component responsible of selecting the query terms that are likely to generate relevant results. In particular, results are relevant if they lead the system to discover new data-intensive large web sites covering the given domain entity.

Keyword Search

Most queries sent to a search engine return a very large number of results. Generally, a user running a web search would type in the query terms and browse through the results to determine what is relevant to her search needs. In our case, we would like to make use of search engines in order to find semi-structured instance pages for a given

[1] http://www.google.com/intl/en_us/insidesearch/howsearchworks/algorithms.html

[2] http://onlinehelp.microsoft.com/en-us/bing/ff808447.aspx

entity. Our result acceptance criteria combines then both the content of the page and the format in which it is presented.

For this operation, a user's manual intervention would not scale. For example, we find instances of commercial products, publications, famous people, movies, proteins, and others, present in hundreds if not thousands of pages in their respective web sites. The richer the site's repository, the more pages it lists for a given entity.

We contend that a web search for instance pages of an entity in a vertical domain becomes more efficient when the queries sent to the search engine are well formulated and when a subsequent mechanism is available to confirm or discard a returned result page based on its relevance. Consequently, in our automated approach, it is the system's responsibility to determine how to formulate the queries by choosing a combination of "useful" keywords that are likely to lead to new web pages for the entity of interest. In later sections, we also explain the system's role in automatically filtering returned search results.

Template Tokens

To automatically select *entity keywords* for our search queries, that is, keywords tightly associated with the particular entity of interest, we rely on the information derived from the training web site used during the synergic inference of crawlers and wrappers. In particular, we aim to distinguish the terms that are both quantitatively and qualitatively related to that entity.

When estimating fixed template tokens during the inference process, the system kept text nodes appearing frequently at the same location in the set of sample pages, while others appearing less than a threshold of frequency were disregarded. The underlying assumption was that entity instance pages are usually generated by a common template. Therefore, tokens appearing consistently across a sample set are likely to be part of the template, whereas tokens that are particular to one or few pages are more likely to be variable content.

Moreover, templates in large data-intensive web sites often contain the labels related to the attribute values being displayed for each entity instance. For example, the template for a book page may contain labels such as "Title", "Author", "Publisher", "Price", etc. Many of these labels are specific to the domain, while others may be more general. The frequently occurring text fields collected constitute then a good starting point to find candidate domain keywords related to the entities.

In a simplified approach, we consider individual words in the gathered text tokens and keep those with the highest aggregated occurrence. In this case, applying common

tf-idf measures are counter-intuitive since we are in fact looking for template words that occur frequently on all the sample page, and not the opposite.

Semantic Evaluator

Among the frequent words collected, we note that many are stop words or extra information present on the page but not related to the domain entity. This prompts to identify and discard them as they would not add any value if kept among the candidates and used in query keywords. On the contrary, they may contribute to obtaining irrelevant search results.

Since the domain identifier token is already given to the system as input, we propose to narrow down on entity vocabulary with the help of a semantic evaluator. An example semantic tool is the UMBC semantic similarity calculator [HKF+13] that combines Latent Semantic Analysis and WordNet knowledge.

The semantic evaluator takes two words or sets of words and returns a value that represents their semantic distance. The closer they are semantically, the higher the score returned by the evaluator. Thus the system is able to rate, for each of the words collected, its relationship to the domain identifier expression. The words that score highest are kept.

A further step that we introduce consists in taking the obtained words, filtered by highest semantic link to the domain, and cross-evaluating them. This results in retaining a subset of words that hold the highest semantic correlation among each other. In other words, the final keyword candidates set consists of terms that

- appear frequently on instance pages of the vertical domain,
- are closely related to the domain identifier string,
- and additionally are closely related among each other from a semantical aspect.

Such a collection of terms is automatically extracted from all the available sources. It replaces the need to manually select entity attribute labels for each domain without knowing the relevance or frequency of these labels when applying the search to the remaining pages on the Web.

Examples of resulting word sets for four different domains are shown in Figure 4.3a. For this illustration, sets of two entity keywords are then derived from the existing words pool, such that, along with the domain keyword, they have pairwise a high semantic correlation. Finally, the groups of relevant keywords output by the keyword extractor module for the four domains are then displayed in Figure 4.3b.

Book	NBA Player	Restaurant	University
paperback	player	hotel	college
publish	soccer	shop	campus
author	game	pub	faculty
bible	tennis	cuisine	school
write	sports	store	graduate
chapter	golf	food	student
stock	team	establishment	education
item	stadium	nightlife	teaching
note	season	drink	institutional
	jersey	club	institute
	complete	city	sciences
			undergraduate
			community

(a) Filetered keywords by domain

book, write, publish
NBA, player, game
restaurant, shop, establishment
university, school, education

(b) Retained relevant keywords

Figure 4.3: Entity keywords

4.4 Constructing Queries

Since we are interested in finding large web sites that contain entity instances of the same vertical domain, we build on the observation in [DMP12] that there is a significant amount of connectivity and redundancy in content among data sources within the same domain on the Web. The existence of overlapping entities among different sources permits the discovery of new web sites starting with entities already discovered and stored in a system's knowledge base, thus operating with a set-expansion approach.

Based on this fact, we propose to start with the entities gathered from web site we used for inference and conduct a search for overlapping entities on the Web. This in turn allows to expand the entity repository with non-overlapping instances found on the newly discovered sources or to consolidate the information already extracted.

The domain knowledge acquired on the training web site consists of the domain identifier terms, the entity keywords extracted as described in Section 4.3, and all the values of the entity attributes that are extracted on the instance pages by the inferred wrappers, which we refer to as the domain instances *information repository*. With this knowledge combined, we propose to build and execute search queries for different instances, which are likely to find other pages on the Web describing these respective instances. When some discovered instances are determined to belong to a new data-intensive web site, they can constitute the seeds for a crawler tailored to find the rest of the instances on that site.

We explain here how the queries are constructed for a targeted web search, while the processing of the query results is detailed in later sections. Each query is composed of the domain identifier terms, the entity keywords extracted for that domain, and

one record from the instance attributes stored in the information repository. Various queries, each for a specific instance in the repository, run in parallel. For each instance query, pages returned by a search engine are considered of higher relevance if they are semi-structured pages about the same or a similar entity instance. Therefore, to restrain the scope of the search we require that attribute values be matched on the result pages because they are part of the instance-related content. The identifying attribute is required, while for the other attributes, all or a subset can be added to the query terms. The rationale is that if we are interested to find instance pages including specific attributes for data extraction, having them in the search query is likely to return results that fit the information need. On the other hand, the domain and entity identifier terms are rather descriptive and are not necessarily expected to figure in the content of an instance page. Therefore, in each query composition, the attribute instances are combined with logic AND, while the domain and entity indicator terms are joined with the OR conjunction operator. The example in Figure 4.4 illustrates how a query is constructed.

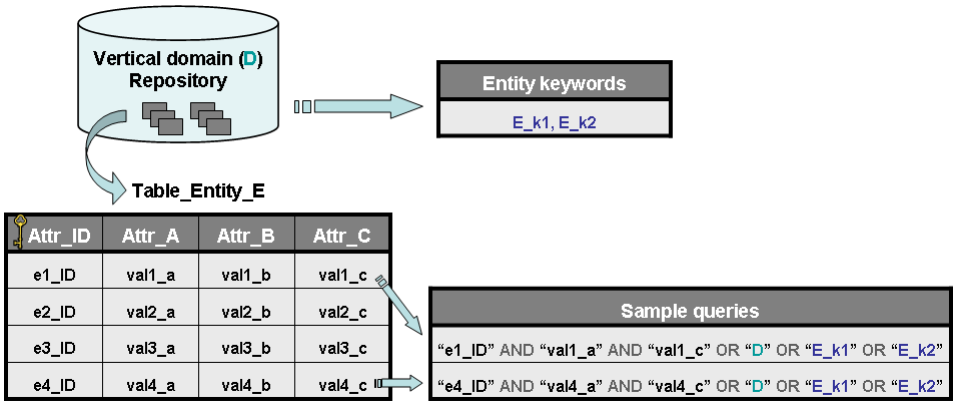


Figure 4.4: Query composition model

We note that we can relax the attribute exact matching requirement by using string approximation or stemming techniques, and this to accommodate variations due to abbreviations, formatting, spelling, etc. However, our experiments show that even with the simplified boolean match for the attributes we are able to find a large number of results.

As an example, consider we are interested in the book domain. The domain identifier term is `book`. The entity keywords derived from our inference web site are `publish` and `write`. A random instance extracted on the inference web site has the values `Great Expectations` and `Charles Dickens` for the title and author attributes, respectively, with the title attribute being the instance identifier in this case.

The query composition for this example is then:
`"Great Expectations" AND "Charles Dickens" "book" "publish" "write"`

The `OR` operator is implied by default. In a similar way, queries are constructed for a selected number of different entity instances in our database and each is sent to the search engine. Wrapper functions allow the abstraction of search engine variations when encoding queries, so the selection of a search engine is not bound by any constraints in the system's operation. Generally, additional parameters may be appended to the query construct to accomodate for common search engine options, e.g., number of results per page. The URLs of the search results are then collected and stored for each query, but results pages are not yet downloaded at this stage.

We note that since all the values of the query terms are taken as phrases, some attribute values are naturally more likely to yield more search results than others. This is due to the differences we mentioned, in the format for dates or measures, for example, or in the presence of abbreviations and spelling variations within the values of search phrases. This can be mitigated by diversifying both the instance values and the attribute selection for a wider search coverage.

4.5 Filtering URL Results

In our quest to discover large data-intensive web sites, we contended that an effective search would be more efficient when the queries are well formulated and when the system can automatically confirm or discard a returned result page based on its relevance.

In previous sections, we already discussed how the system chooses pertinent keywords and how queries are built using these keywords. In this section, we explain how the system goes about the numerous query results returned by the search engine, and how the first step of the automatic filtering is performed.

The main idea is to select only a useful subset of the URLs collected from different instance queries for further processing, instead of downloading all the web pages listed in the results. For that, we propose to combine the overall resulting URLs from a number of instance queries in order to identify which web site globally and frequently responds to our search for redundant entities. Since the same web site may appear in different query results with some variations in its URL constituent parts, this requires a way to compare URLs to assess their similarities and their possible belonging to a common sub-section of a web site. If a URL recurrence is found, based on similarity evaluation and not necessarily exact matching among URLs, the concerned subset of search results will be further examined.

Identifying Useful Redundancy

When a query is sent out to a search engine, the latter returns a very large number of results in response. These results commonly link to pages that contain a full or partial matching of the query terms.

Because of the entity redundancy principle that we discussed in Section 4.4, we expect that the same large web site may contain many instance pages overlapping with the instances used in the queries. Therefore, these instance pages are likely to appear in the different result lists of the distinct instance queries we run. In other words, the more overlap there is, the more frequently that web site will show up in the result sets. Equally, the more frequently a web site shows up in the result sets, the higher the probability of it being a large web site with content redundancy that responds to our search needs.

This brings to the question: how to determine automatically which of the pages listed by the search engine may be different instance pages belonging to a common web site? One possibility is to download all the pages and compare their templates and semantics [CMM05, Got08]. The problem in this approach is that it will employ too much processing time and bandwidth to download all the listed result pages while many of them do not respond to the requirements of (a) being semi-structured and (b) belonging to a large data-intensive web site. In addition, because we don't have any reference templates, and because search results can originate from any type of sites indexed by the search engine, the system would need to apply some heuristics and estimate an appropriate template distance to classify two pages as sharing a common template. These factors, if at all feasible, do not boost the search efficiency.

On the other hand, it is known that domain administrators adopt various conventions for encoding page URLs. Moreover, when the pages belonging to the same category are populated from a database through an automated script, as in the case of large web sites, some consistencies can be detected in their URL patterns.

We aim to exploit the similarities among URLs to group result pages into clusters, such that a cluster contains pages responding to distinct instance queries, but belonging to a single web site and having little dissimilarity in their URL patterns.

Each qualifying cluster can then be analyzed to decide whether it contains candidate pages belonging to a large web site.

There are some existing works that propose algorithms to cluster large web sites, such as [HRRC11, BDM11]. However, these approaches assume that the web site is already given and they try to discover its structure. Our goal is to estimate if pages with similar URLs, resulting from different instance queries, possibly belong to a large web site section describing an entity in the vertical domain.

Inspired by the approach in [BDM11] that combines URL analysis with some simple content features, we use URLs clustering as a starting point for a further web site exploration. In the next sub-section, we describe our clustering algorithm in more details.

URLs Clustering

By looking at a reduced sample set of URLs gathered from a results pool for queries in the book domain (Figure 4.5), a user can already spot some recurrences. For the system to operate on a large scale, it has to automatically determine which URLs have similar patterns and group them together.

http://productsearch.barnesandnoble.com/search/results.aspx?store=book&ATH=Jimmy%20Buffett
http://www.rakuten.com/prod/the-cold-war-a-new-history/31198770.html
http://www.abebooks.com/book-search/kw/fact%F3tum-charles-bukowski/page-1/
http://www.readprint.com/work-11523/Brief-Interviews-with-Hideous-Men-David-Foster-Wallace
http://www.alibris.com/Power-Multi-Level-Marketing-Mark-Yarnell/book/5267499
http://amblingbooks.com/books/view/emotional_alchemy_2
http://www.shelfari.com/books/8515328/The-Landscape-of-History
http://www.rakuten.com/prod/your-first-year-in-network-marketing/30327356.html
http://www.alibris.com/African-Cry-Jean-Marc-Ela/book/158642
http://productsearch.barnesandnoble.com/search/results.aspx?ATH=Mark+Yarnell
http://www.shelfari.com/books/373097/Stone-Rain
http://www.abebooks.com/book-search/title/sharpes-prey/page-1/
http://www.rakuten.com/prod/the-fatal-shore/30059832.html
https://www.facebook.com/benniebeltmouse
http://www.alibris.com/Emotional-Alchemy-Tara-Bennett-Goleman/book/15814741
http://amblingbooks.com/books/view/your_first_year_in_network_marketing
http://books.google.co.in/books/about/Mythology.html?id=GdEnAAAAYAAJ
http://productsearch.barnesandnoble.com/search/results.aspx?store=book&ATH=Mark%20Yarnell
http://www.readprint.com/work-54434/Midnight-s-Children-Salman-Rushdie-Vintage-Classics-Salman-Rushdie

Figure 4.5: Sample query result URLs

For this reason, a clustering approach seems applicable since cluster analysis is generally adopted to find associations based on elements similarity, especially when the number and type of element subclasses are not necessarily known in advance. The latter are then determined by the algorithm itself. Clustering techniques use in fact unsupervised pattern recognition to partition a given set of elements into subsets, referred to as clusters. Elements in a cluster are expected to be highly similar among each other and to differ from those in another cluster.

We opt for a hierarchical agglomerative clustering (HAC) algorithm to process the result URLs collected from all the instance queries and group the similar ones into clusters. The HAC algorithm was introduced in [CGK78] and is a rather simple "bottom-up" technique that fits the nature of our data set, where the percentage of results to be merged into clusters is not very high with respect to the entire set of URLs returned by the search engine for the instance queries. In fact, in HAC, each observation starts out as a separate cluster. In subsequent iterations, pairs of clusters are combined if the predefined measure of distance between them is minimal, i.e., the closest pairs are combined into a single cluster. In this way, a hierarchy is built by starting with clusters of individual elements and successively moving towards a single cluster containing all elements.

Therefore, we need to specify for our problem the metric that defines a distance measure between two URLs. A URL can commonly be broken into different parts, such as protocol, host, port number, path, query string, etc., some of which are optional [3]. An example is shown in Figure 4.6 for illustration purposes. The scheme is commonly referred to as *protocol* and the fragment is also known as *ref* or *reference*.

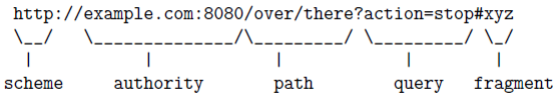


Figure 4.6: URL structure

To attribute a numerical notion of resemblance to URLs, we define the distance, or dissimilarity, between two URLs based on the parts they have in common and those that are different, as shown in Algorithm 2.

For the successive iterations, a linkage criterion needs to be defined for the algorithm to compute the distance between two sets of elements as a function of the distance of the elements these sets contain. One possible function is the minimum distance between elements of each cluster. This is referred to as single-linkage clustering. For clusters $C1$ and $C2$, their distance is:

$$\forall x \in C1, \forall y \in C2 : d(C1, C2) = min(d(x, y))$$

where x and y are URLs. Hence, at a given iteration, the two clusters separated by the shortest distance are merged. This implies that for a subsequent iteration, the minimum distance between clusters is larger than that at the previous step. A stopping condition for the algorithm can then be either a threshold that says that clusters have

[3] Detailed information on the URL specifications can be found on http://www.ietf.org/rfc/rfc1738.txt

Algorithm 2: Measuring Dissimilarity between Two URLs

Input : URLs $\{u_1, u_2\}$
Output: The distance D between u_1 and u_2

if $u_1.authority == u_2.authority$ **then**
    Let D **be** the number of different parts between $u_1.path$ and $u_2.path$;
    if $u_1.protocol$!= $u_2.protocol$ **then**
        D++;
    if $u_1.query$!= $u_2.query$ **then**
        D++;
    if $u_1.ref$!= $u_2.ref$ **then**
        D++;
else
    D = **INFINITY**;
return D;

become too distant to be merged, or, when applicable, a predefined number of clusters to reach.

Applying HAC to the collection of instance search results, each observation, which corresponds to URLS in our case, starts out as a separate cluster. URLs with the smallest dissimilarity are then grouped in the following steps. Based on the URL distance metric, we set our stopping condition as a rather small integer, between 1 and 3. In fact, larger variations in URL parts are not interesting for our analysis based on finding redundancies in URL format, and later in page content. In particular, when two URLs that have a different authority, for example, it is not even necessary to compare the rest of their components as they do not come from the same web site, so obviously not the same entity page template. Their dissimilarity is thus maximal.

From the clusters generated by HAC, we can consider as valid candidates for instance pages those URLs that:

- occur in multiple distinct instance search results
- originate from the same web site
- share a pattern with low dissimilarity value (inferior to a selected threshold)

URLs satisfying these properties make it through the first filtering step of the system and are then subject to further examination to determine whether they constitute semi-structured instance pages from a data-intensive web site. On the other hand, URLs occurring only occasionally or not matching any cluster (other than their own singleton) are not considered of interest and their pages are not downloaded.

4.6 Page Evaluation

To our day, web search engines still use full or partial keyword matching techniques when finding results to a proposed query, as mentioned in Section 4.3. That is, a page most likely appears in the search results list because it contains all or a subset of the terms present in the input query. Other considerations, as discussed in the same section, are applied by search engines in order to rank the appearance of a page in their output list.

In Section 4.5, we described how our system clusters similar URLs collected from query results and kept only those with sufficiently redundant occurrence among various queries. This constitued the first part of our result filtering process. In this section, we describe how pages at the resulting URLs undergo the second stage of filtering through our system. The URL clusters deriving from the URL filtering component are passed as input to the next component in the system, the page evaluator. The responsibility of the latter is to determine whether the pages at the given URL addresses contain semi-structured data sections that can be of interest for the data extraction task. In fact, we reiterate the goal of the system in finding large data-intensive web sites in the vertical domain. We adopt as a reasonable pre-condition the fact that pages belonging to large data-intensive web sites are generated by regular templates with some level of structure. This assertion implicates that pages not contaning any data sections with structure can be discarded for our purposes. For each URL in the clusters returned by the previous system component, the corresponding web page is downloaded to be further analyzed.

In order to separate interesting from non-interesting pages, we proceed in three steps:

- Locate relevant fragment(s) on the page
- Extract features from page fragment(s)
- Classify the page based on extracted features

The following sub-sections describe these steps in more detail.

Locating Relevant Content

Given a downloaded page, the first task in this component's process consists in identifying in its content the fragments that are to be evaluated with regards to the originating search purposes. Web pages usually consist of one or more main content section, in addition to less interesting sections such as banners and menus. In large

data-intensive web sites, instance web pages have a section that displays the entity attributes in a semi-structured format. Although this is generally located in the main content section, some sites adopt a box notation, in a side or top frame for example, while leaving the center frame on the page layout for more detailed information. Figure 4.7 shows an example of a restaurant instance and another of a movie instance, where a combination of semi-structured data and plain long text co-appear on the page, in addition to other less focal page content.

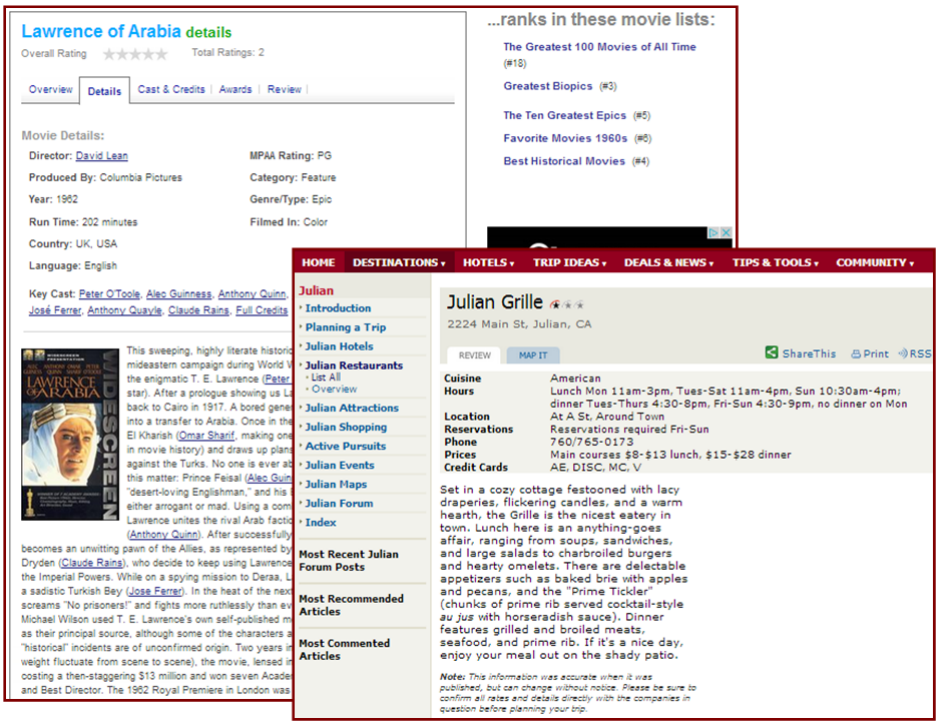


Figure 4.7: Mixed content instance pages

Since our data extraction goal is to retrieve entity attribute values where available in semi-structured sections, we need first to locate the HTML container in which they are displayed on the page. While some works address vision-based analysis and DOM tree alignment to delimit HTML containers for data extraction, we opt for a less complex approach. Instead, we start with the query that generated the result page that is being examined and we search for the least common ancestor HTML container of the instance attribute values in that query on the given result page. If one or more of the attribute values appear multiple times on the web page, several parent containers may be identified. These different container elements are not necessarily exclusive, meaning that one can possibly be a child of another. Moreover, the least common ancestor element may happen to be the root node of the page's DOM tree.

Depending on the occurrence of the attribute values, the following are the possible scenarios that can be encountered:

- Attribute values are located in one page fragment. In this case, the corresponding HTML container is returned for further analysis.
- Attribute values are located in several page fragments. In this case, a list of HTML containers is returned and each will be analyzed separately.
- Attribute values are not found on the page. We combine with this case when attributes are only partially matched. This means no valid container is found with all the values of interest and the given result page is then discarded.

The next task is to analyze each HTML element where these attribute values appear to determine whether it has the structural features generally encountered in semi-structured instance page templates.

Features Extraction and Page Classification

A human user is able to tell if a page contains a semi-structured data section that can be exploited for attribute extraction by simply looking at its rendering in a web browser. In fact, the instance attribute terms may appear in various parts of the result page. However, occurrences in text sections such as in reviewers comments, detailed descriptions, and the like, are not of any particular interest for our data extraction purposes. We are rather interested in finding redundancy with the instance attribute values displayed in a semi-structured layout. To filter out false positives, and given the large amount of web pages processed by our system, it is not efficient to have a person examine each page and determine whether it has any relevant semi-structured data section describing an entity instance or not.

Therefore, given an HTML fragment, the automatic page filtering sequence proceeds to analyze it with respect to some structural and content features that are commonly observed when entity instances are displayed on data-intensive web sites. Recurrent characteristics have been visually identified and used to train a classifier in order to automatically distinguish structured from non-structured content, by simulating a user's discernment task for each of the result fragments. Namely, we look into the length of the text, recurrence of characters such as the column, usage of lists or table cells, and other aspects of the HTML formatting. These features are extracted and stored for every container found and are evaluated in the third step for classifying the fragment. Initially, the classifier is trained with a large set of positive and negative examples, that is, with a mixed set of semi-structured and non semi-structured HTML

container elements. It is then applied to the obtained result page fragments. A result page is boosted as a potential instance page if at least one of its extracted fragments is classified as semi-structured. Otherwise, the page is not considered a valid candidate and is discarded.

The output from the page evaluator component consists then of those search result pages, already retained in a URL cluster, where the respective query attribute terms occur in a likely semi-structured layout. These are the final candidate instance pages of the system. Clusters with a larger number of classified candidate pages will have higher priority to be processed by the site explorer for crawling more instances from the discovered web site. This will serve to populate the vertical domain repository.

A detailed evaluation of the system's performance and the obtained results is provided in Section 4.7.

4.7 Experiments

After presenting our approach to discover new web sites in a vertical domain, in this section, we describe our implementation of the Vertical Domain Explorer system *(VerDE)*. We also describe the experiments we conducted and list the results obtained. Finally, some analysis and comments are proposed related to these results.

System Implementation

VerDE system is implemented in Java as a set of packages that reflect all the functionalities of the various components introduced in Sections 4.3 through 4.6. The information previously collected from the inference training web sites, which constitutes the information repository, is stored in a MySQL database, and so is the output of VerDE system.

A preprocessing module analyzes the tokens extracted on the training web site for their frequency of occurrence and semantics. It also discards stop words that bring no additional value to the keyword search. For the semantic evaluator, as described in the keyword extraction component (Section 4.3), we use the semantic similarity service provided by the University of Maryland, Baltimore County (UMBC). Their technique combines Latent Semantic Analysis (LSA) and knowledge extracted from WordNet to evaluate word similiarity [HKF$^+$13]. For any given words or phrases, the semantic similarity tool returns a value that represents their semantic distance. This numeric score decreases with increasing distance. The threshold to differentiate words that are close semantically from those that are distant is determined empirically in VerDE.

As for interfacing with search engines in the query generator component, we mentioned in Section 4.4 that it is abstracted by the wrapper functions that we imple-

mented. This choice provides the flexibility to select any of the major search engines to run the queries for the sake of preference or comparison.

The URL analyzer component includes a clustering module. Therefore, we opt for the Hierarchical Agglomerative Clustering (HAC) algorithm with single linkage, and base our implementation on the Java library provided by the *Sape* research group at the University of Lugano [4]. Support scores are computed for each URL results cluster to reflect their coverage of the different instance queries sent to the search engine. Clusters with high score mean that the web site contains many overlapping pages with the entity instances of the search. On the other hand, clusters with a score below a set threshold are not processed any further.

Finally, the page evaluation component requires a classifier to assess the structure of the page content where the instance attributes are located. Because of the binary nature of the expected output (semi-structured content or not), the classifier is implemented as a logistic regression. The training set consists of HTML content collected from pages on the Web and reflecting both semi-structured and non-semi-structured properties, as positive and negative examples respectively. Eight features related to text length, punctuation, and HTML formatting are identified for our supervised learning algorithm. For a model selection that is less prone to overfitting or underfitting, we fit the logistic regression hypothesis parameters on a subset of the collected training examples. Then we fit the degree of polynomial on a different cross-validation subset. Finally, we tune the regularization parameter such that the test error is minimal when evaluated on unseen samples in the third and last test subset. We report the accuracy scores listed in Table 4.1 for the performance of our classifier model.

Training set accuracy	81.82%
Cross-validation set accuracy	93.75%
Test set accuracy	100.00%

Table 4.1: Classifier accuracy

The classifier with the fitted model parameters is then used in the Page Evaluator component of the system to categorize the page fragments that are found to contain the relevant search data.

Finally, all the various VerDE components are implemented and integrated to smoothly deliver the functionalities of the automated search and filter approach that the entire system builds on.

[4] `http://sape.inf.usi.ch/hac`

Experiment Results

We evaluate the results obtained from VerDE by running experiments in 4 vertical domains:

- Restaurant
- University
- Book
- NBA player

For any inference web site in a given vertical domain, the information collected and stored in the repository contains the frequent tokens found on the entity pages and the records of attribute instances extracted by the inferred wrappers on the target pages.

Once retrieved from the inference site tokens, entity keywords can be stored for future use to enhance performance. When a different inference site is used, and thus a different set of tokens is processed from the information repository, the entity keywords extraction is launched again.

On the other hand, instance attribute values are fetched from the repository at every experiment run. Given the number of instances to retrieve, random records are selected from the database to formulate queries during the system execution. The system can include in the built queries any or all of the attributes stored for a given entity. A match for these attribute values is then sought in the search results. Attributes used in the queries for each domain are shown in Table 4.2. For evaluation purposes, we include in our query construction an attribute value that can be considered as entity identifier, whose label is shown in bold font in 4.2. We also select a variable second attribute from those shown in normal font. The number of instances and the attribute selection for each experiment run can be specified in the experiment configuration.

Table 4.2: Query attributes by domain

Domain	**Query attributes**
RESTAURANT	**Name**, Cuisine, Address, Phone
UNIVERSITY	**Name**, Type, Phone
BOOK	**Title**, ISBN, Author, Publisher
NBA PLAYER	**Name**, Team, Height, Weight

For the experiments run on the VerDE system, we calculate the the percentage of examined pages with respect to the total URLs collected from search engine results. These represent the pages that contain the entity search terms and are considered for

further examination by the classifier to detect semi-structured properties. The percentages are listed in Table 4.3.

Table 4.3: Percentage of examined pages by domain

Domain	% Pages examined
RESTAURANT	3.32
UNIVERSITY	25.89
BOOK	8.03
NBA PLAYER	26.32

We also compute the percentage of results positively classified as semi-structured by the system with respect to those retained for examination. The values are listed in Table 4.4

Table 4.4: Percentage of pages classified as semi-structured

Domain	% Positive pages
RESTAURANT	62.07
UNIVERSITY	88.60
BOOK	74.70
NBA PLAYER	78.73

We note in Table 4.5, for each vertical domain under consideration, the precision for the semi-structured sources automatically classified by the system.

Table 4.5: Precision of results by domain

Domain	Precision
RESTAURANT	0.83
UNIVERSITY	0.91
BOOK	0.83
NBA PLAYER	0.84

To compute the recall, one would have to go manually over the hundreds or thousands of URLs returned by the search engine and verify if the system missed any positive results. Therefore, due to the huge amount of results returned by the search engine, the recall value was not computed for our experiments. However, an estimate obtained by a manual verification on a sample subset of URLs gave a recall value of 73.4%.

Moreover, to illustrate the variation in the obtained results depending on the attributes selected for the queries, Table 4.6 lists the precision values for 3 different attribute combinations.

Finally, Table 4.7 lists the number of distinct semi-structured sources discovered for each domain. They correspond to a total of 172 distinct authorities.

Table 4.6: Precision by query attributes in the Restaurant domain

RESTAURANT	Name, Cuisine	Name, Address	Name, Phone
Precision	0.82	0.86	0.67

Table 4.7: Semi-structured sources automatically discovered

Domain	# Pages
RESTAURANT	54
UNIVERSITY	342
BOOK	245
NBA PLAYER	469

In the next sub-section, we explain our findings.

Result Analysis

In total, 10104 URLs were collected during the experiments, which were reduced to 9465 distinct URLs once duplicate results were removed. Only about 15% of them became part of clusters with URL redundancies. Of these, 1395 pages in total contained the attribute values of the query search terms and were examined by the classifier for semi-structured format filtering. This translated into a considerable effort saved from downloading unuseful pages. Furthermore, 1110 pages, or 11.73% of the URLs collected from search engine results, were classified as semi-structured. This percentage came in close agreement with [NWM07, NMS$^+$07], where it was statistically estimated that 12.6% of randomly crawled pages are semi-structured product pages. This was also equivalent to 79.6% of the pages that passed the first phase of automatic filtering being classified positive in the second filtering phase, which highlighted the benefit of the URL pre-processing step.

The experiments precision results for each domain shown in Table 4.5 are satisfactory for a completely automated approach, with an overall system precision of 0.84. However, we note that few results were returned and a poor accuracy was observed on numerical attributes, e.g., `ISBN`, `height`, `weight`, `phone numbers`, mostly due to wide variations in formatting on the Web and our heuristics being based on exact matching. In particular, Table 4.6 shows a low precision for results related to queries with the numeric `phone` attribute, as compared to those with `address` or `cuisine` textual values. However, diversifying the attribute selection in queries for a given entity can compensate any loss in results coverage. For example, queries with books `title` and `publisher` would find matching sources that queries with `title` and `ISBN` numeric values did not find. Another option to remedy to this problem would be to relax the boolean search with approximation or regular expressions when matching

values on pages.

Another observation is that the number of discovered sources is not overwhelming. This can be explained first by the fact that few entity instances were used in each experiment as query material to the search engine, in order to improve efficiency in evaluating the system model. However, with a set of queries of 5 to 50 different instance queries, 245 sources were discovered in a few minutes in the BOOK domain. It is a great improvement over a manual experiment we conducted, where it took a team of 3 people a few hours to locate 100 relevant BOOK sources. Other explanations can be related to the high number of non-entity pages returned by search engines, especially in the RESTAURANT domain where touristic sources, advertisements, and blogs are popular and abundant, as well as to the overlap of results returned over different experiments, which do not bring any added value. Finally, it is worth mentioning that a large portion of the semi-structured web content originates from the Deep Web, which is rarely indexed by traditional search engines, and therefore not accessible to our system.

Overall, the results translate into a clear optimization in the effort of exploring web sites for crawling and data extraction. In consequence, the site explorer module can eventually focus its processing only on likely candidates of large data-intensive web sites.

4.8 Related Work

The vertical domain explorer (VerDE) system we presented touches on many subjects in data extraction addressed by several research contributions. We discuss some of them as they relate to the system's functionality and to the four system components.

Focused Crawling and Vertical Search

In Section 2.2, we discussed how crawlers find and download web pages that are then indexed and used by search engines. The research work in [CVdBD99, Cha02, STS+03, PS11] presents focused, and more specifically, topical crawlers that filter the portions of the Web to be crawled. This has put forward challenges in the ability of the crawler to distinguish relevant documents from non-relevant documents and that of selecting non-biased crawl seeds. Learning algorithms and matching specific keywords in content and metadata have been commonly adopted, in addition to putting restrictions on the domain name of the web site to crawl, or on the link distance from the start page, but the problem remains open for study to improve performance and accuracy.

However, topical crawlers do not necessarily find large data-intensive web sites. In fact, these crawlers can tumble on many sites relevant to the topic but not fitted for data extraction purposes. As a result, they do not add value in enriching, checking or extending repository of entity attributes, while downloading and analyzing all pages crawled to determine their structure would even require a double effort: first to train the focused crawler, then to also post-process the documents. Our approach on the other hand relies on filtering the URLs to be downloaded by comparing parallel query results. By identifying redundancies, it tries to locate large web sites and consequently, automatically selects candidate seeds that are likely to yield large amounts of entity pages matching the crawling objectives. It also allows to limit the crawl to sections of the web site that satisfy constraints both on content and format.

Information Retrieval for Vertical Search Engines

In recent years, the need for vertical search engines dedicated to topical services like news, scholar publications, finance, shopping, biological and medical information, etc., has motivated efforts for highly accurate information retrieval techniques in this area. In fact, such vertical search engines allow users to search for information within only a subset of documents on the Web, mainly those that are relevant to a pre-determined topic. Domain-centric search engines have many advantages over document search, since they can maintain a more complete database of the entities in the domain by reaching the long tail of web sites [DMP12]. Moreover, given their specialized content nature, they are able to provide customized search features over the pre-determined vertical domain and they overtop standard search engines when it comes to coverage, accuracy, and freshness of data.

After the focused crawling process to locate pages on the Web from the specific vertical domain, it is necessary to extract from these pages the information related to the domain entities, integrate these information into a data warehouse, and offer the possibility to perform object-level search. It is also important to ensure the automation and scaling of data extraction from the collected structured or semi-structured web sources. Therefore, various research activities address these topics in their effort to enable object search in vertical domains. [NWM07, NMS+07] explore this paradigm and the technologies used to build their object search engines in two distinct vertical domains, namely academic and product search. Their topical crawler fetches pages, which are then classified into pre-determined categories corresponding to real-world objects. On these pages, values are extracted for the pre-determined entity attributes. This is followed by integrating the information into a warehouse, and ranking their popularity and relevance. They statistically estimate that 12.6% of

randomly crawled pages are product pages. This echoes the estimation we also report in our result findings.

[NFP+11] also consider the issues of data extraction, schema reconciliation, and data fusion, in building a system that synthesizes products for shopping verticals or product search engines. From the merchant offer pages, they identify name-value pairs of product attributes listed in HTML tables, and either match the product to an existing catalog's category in the global schema, or create a new catalog item if no match is found.

A discovery-based approach for vertical domains is introduced in [HCPZ11]. Starting with one labeled example site from a given vertical, their solution aims to train a system to extract data from unseen sites in that same vertical, independently of the domain. The system training relies on content and layout features to learn page-level, site-level, and attribute information, which allows it to identify similar attribute values on a new unseen site pages.

In their proposed system,[SWL+12] exploit entity redundancy in different web sites to learn entity attributes from inner-site and cross-site features. This enables a dynamic and automatic discovery and extraction of structured data from web pages belonging to the same vertical.

In all these works, the emphasis is on facilitating and optimizing information retrieval in given verticals. However, input pages are provided either by topical crawlers or by manually specifying the web source to analyze. No optimization is performed at the search and download stages to target only the pages that represent data-intensive and semi-structured entity pages, which would reduce considerably the amount of later page processing. On the other hand, our approach discovers new data-intensive site automatically. Due to this automation, we reduce human effort and also avoid unnecessary page downloads, which makes our approach fit to handle in an efficient way the huge amount of entity pages and rich data-intensive sites on the Web. We can thus consider VerDE as a preceding module for these systems, meaning that once we automatically discover new sites with VerDE, the discovered sites can be treated by any of these data extraction systems to identify and extract attribute-value pairs on the pages. Moreover, VerDE is able to individuate the HTML containers where the semi-structured data are displayed, based on the values sent in the query and retrieved from the training web site. This can resolve some extraction obstacles the other systems face where an attribute like "Book title" appears several times on the page with different values, as generally happens with various recommended instances listed on the same page as the main entity instance itself. An example of such pages is shown in Figure 4.8. When the Title attribute `Romeo and Juliet` and the Pub-

lisher `Sterling Juvenile` are the instance values in the query, for instance, the HTML section of interest identified by VerDE for further processing is delimited with the dotted rectangle. It represents the smallest common ancestor node containing both values. Consequently, other recommended books on the lower part of the page are of no relevance to our system, whereas with other solutions they may get matched to attributes in the `Book` entity schema and cause incorrect extraction for the `Romeo and Juliet` instance.

Figure 4.8: Sample instance page with listed recommendations

Entity-Aware Search Engines

One approach that have similar goals to ours is [BCMP08], where the authors run an automatic search for web pages that are instances of a given entity, which can be exploited by domain-specific search engines or faceted search applications. Their system takes as input a few sample pages from distinct web sites pertaining to the same entity. For sample pages belonging to the same web site, they propose to find other instance pages in the same large web site. With the various initial sample pages, they also propose to discover other instance pages in new web sites. This is achieved by first analyzing the overall structure of entity pages in the same web site to detect regularities, based on hyperlink locations. Then by intersecting in-site and cross-

site terms, they derive entity and domain keywords. These keywords are combined with link anchors pointing to instance pages in order to run search queries for each instance. Each URL in the query results is analyzed by exploring its web site to find similar pages and derive their common template. The derived template, when found, is then checked for common entity keywords to determine whether it represents instance pages. If the outcome is positive, the web site is then re-explored for finding and crawling those instance pages.

While both our work and theirs address the domain-independent page gathering task, the two approaches differ in several aspects:

- They need to analyze several similar web sites to perform quantitative keyword analysis and bootstrap the system. Our approach is based on data from one training web site and uses both quantitative and qualitative semantic evaluation to determine meaningful keywords for the domain under consideration.

- Their system determines instance pages based on entity keyword appearance and hyperlinks location. This may lead to false positives in web sites that belong to the domain examined but do not necessarily offer semi-structured data-intensive pages. On the other hand, our page filtering is based on a trained classifier that takes into consideration content, structure, and formatting and applies the evaluation to identified sections of interest on the page, which also reduces noise.

- For each URL result returned by the search engine, they run a full web site evaluation, which is not a trivial task, especially when the URL belongs to a large web site composed of thousands of pages. The web site exploration is even repeated if a derived template evaluates as a potential instance page. In contrast, we exploit apparent redundancies in search results and minimize the number of candidate URLs before they are processed any further. The second step of page classification avoids exploring a web site if no semi-structured content is detected. Furthermore, having identified other instance pages in the same web site cluster facilitates the task of finding the relevant sub-section in that large web site that relates to the entity of interest.

A different but related work is presented in [WT10], which presents a knowledge harvesting technique aiming to construct a comprehensive knowledge base of facts. These techniques exploit semi-structured and natural-language online sources such as Wikipedia with the goal of extracting semantic classes, mutual relations, and temporal contexts of named entities to support expressive and precise querying. For example,

they try to find all entities in Wikipedia corresponding to predefined classes and subclasses, such as scientists, politicians, movies, or all those matching binary predicates, such as *graduatedAt(Person, University)*. Their information extraction interest is not in collecting and integrating a variety of raw data to fit in a given domain schema, but rather gathering entities with their semantic types and corresponding relationships in an "output-oriented" approach, in other words, gathering instances for a given set of relations. In this context, the semi-structured nature of data-intensive pages cannot potentially be exploited in a pattern-based extraction of facts without a substantial postprocessing of the output.

Visual Cues for Data Extraction

Some recent approaches in data extraction address wrapper generation using visual content features from the web sources. These techniques parallel those that rely on DOM trees, which are quite simple but can lack accuracy because of page template variations. Visual-based wrappers, on the other hand, rely on the rendering information from the browser, like position, alignment, font, images, etc. to separate noise regions and locate data of interest [GHTG12, SL05, LNL01, LMM10, LLO07].

Such approaches commonly examine the resemblance of data records to build a block tree, then proceed to data record extraction and data item extraction, assuming the relevant information block is centered in one main region on the page.

They are mostly efficient if a page displays multiple data records with recurring structure, as in search engine result pages or record pages from the deep web, but remain nonetheless time-consuming in their processing.

In contrast, our page evaluator component identifies the block containing relevant data from the occurrence of the sample instance values already available. Therefore, analyzing the full content of the page is not necessary in this case. Whether the page contains repetitive records or not does not have any effect, as our trained classifier examines only the delimited section and determines if it is semi-structured in order to be exploited for data extraction.

4.9 Conclusion

In this chapter, we presented an approach to automatically and efficiently locate large data-intensive web sources in a given vertical domain, by starting with knowledge gathered from a training site and exploiting redundancies in entity occurrences on the Web. The system prototype implemented is composed of 4 logical components: a keyword extractor and an automatic query generator for the search tasks, then a URL analyzer and a page evaluator for the filter step. Our experiment results show a great

advantage in using the automatic 2-stage filtering before exploring web sites returned by search engines, and a high level of precision of our classifier. Future work can address the implementation of the web site crawler that exploits output page clusters with high coverage, in addition to application of data extraction techniques on the semi-structured containers identified on the pages.

Conclusion and Future Work

In this dissertation, we presented our contributions in common research fields of Web information retrieval. In particular, an innovative approach for synergic definition and exection of crawling and wrapping programs was evaluated, and was exploited to guide a proposed search technique for new web sources displaying semi-structured information about the same type of domain entities. These approaches build on the information redundancy observed within a large web site on one hand, and among various web sites on the Internet on the other hand. They constitute important building bricks for constructing vertical search engines and aggregator sites, while remaining domain independent, and they focus on keeping the user effort to a minimum.

Summary of Contributions

We started in Chapter 1 by introducing key aspects of the Web information retrieval research area and the main lines of work associated with it. We defined its goal, which is finding solutions to automatically discover implicit structure and semantics on web pages and turn the textual or semi-structured content they display into machine-usable structured data. This in turn allows complex queries to be run to respond better to users' information needs. We narrowed down the focus on semi-structured data extraction and large data-intensive web sites. The structure and content of these web sites reveal a large amount of *information redundancy*, which is the main thread that binds together the 2 central parts of our work. We also highlighted the main challenges encountered in the various tasks of an information retrieval activity, parts of which underline our motivation to address the open problems in this field.

In Chapter 2, we presented an in-depth review of the state-of-the-art related to the problems of crawling, wrapper generation and maintenance, and data integration are addressed. Innovative methods were described for truth discovery and for characterizing web data uncertainty when sources offer conflicting information. In addition, we gave an overview of some solutions that make information retrieval scalable to the Web and the Deep Web, in addition to some efforts that extract semantics and discover entity information on web pages.

In Chapter 3, we defined for a given data-intensive large web site, an *abstract model* describing it on the intensional, extensional, and constructional levels. Using this model, we introduced a new methodology to specify and execute crawlers and wrappers in a synergic way. A crawler is defined in such a way that only sections of the web site that are relevant to the data extraction task are downloaded, with support for pagination. Wrappers are defined both to extract navigation links for crawling, and for extracting data attributes while the pages are downloaded by the crawler. The 2 operations complement each other and operate in an interleaved way. We also implemented a prototype called CoDEC that derives these specifications from tracking an *inexpert user's* navigation and interaction on the site. We assured low overhead for the user by adopting an *active learning* approach, where the system learns from the user's input and prompts for more input only in case it encounters uncertainty with the data extraction results on a given page. The system then refines its learning of both the page template components and the inferred extraction rules. In this work, solving the *sampling problem* was also a crucial step to avoid bias in the training set, which would also affect the rules inference. The evaluation of the system through the implemented prototype showed satisfactory results with a minimum user effort.

Chapter 4 introduced a complementary approach that exploits the knowledge gathered while performing inference on a large web site, and is able to efficiently and *automatically* locate other large data-intensive web sites offering pages on the entity of interest. The proposed technique is based on the existence of information redundancy among entities on the Web and uses an automatic mutli-step *search and filter* mechanism to find relevant pages without having to download thousands of non-relevant ones suggested by a search engine. Our approach is also able to determine the location of semi-structured data attributes on the newly discovered entity pages with high accuracy. Data extracted on those pages would be useful in extending, enriching, or validating the data already collected in a vertical domain repository. After presenting the architecture of our system, we explained the design and operation of its 4 logical components, namely a keyword extractor and an automatic query generator for the search tasks, then a URL analyzer and a page evaluator for the filter step. To support the development of our models and methods, we relied on the implementation of a software prototype, named VerDE, which also served for the evaluation of the proposed approach. Our experiment results show a great advantage in using the automatic 2-stage filtering before exploring web sites returned by search engines, and a high level of precision of our classifier.

Finally, it is worth noting that both systems are *domain-independent* in nature and can be applied to large web sites from any given domain, regardless of their organiza-

tion, and content characteristics.

Future Work

As discussed their respective chapters, the 2 systems we developed leave some potential for improvement and extended functionality implementation. The CoDEC system could benefit from the inclusion of semantic assertions to distinguish pages describing entities that are close in structure and type but different in their semantic connotation, such as coaches and players, used cars and new cars, etc. Also, a mechanism is needed to deal with any optional attributes with changing location on the page. This challenge is one of the many open challenges in Web information retrieval highlighted in Chapter 1.

As for VerDE system, future work can address the integration of existing attribute discovery and segmentation techniques to extract the values from the identified semi-structured containers on the pages.

We note that while we studied the problem of locating and extracting semi-structured information on large data-intensive Web sites, we identified several general limitations that still constitute a challenge in the area of Web information retrieval.

In particular, the main open problem is how to generate a golden set of attributes and values to verify the obtained results and evaluate generated extraction rules. Many approaches rely on human efforts to check or annotate data sets, which is an expensive solution that does not fit the Web scale.

Moreover, with the diversity of implemented methods and proposed techniques in the literature, there is no common standard way for performance evaluation in order to conduct a fair comparison. The adopted metrics for precision and recall are able to quantify the performance of a system, but they remain linked to the experiments setup and the variety of data existing on the Web.

Deriving a global schema model covering all possible attributes of a given domain entity at Web scale also remains a challenge. Existing models are either manually generated or discovered from the finite sources fed into an data extraction system.

Also, extending the schema to allow record linkage between different sources would be an important feature for the completeness of results returned in response to a user query, as well as refining returned results that have a datestamp to determine which are current and which are entities from past years. This brings to the fact that a lot of work is still to be done concerning the rapid evolution of web sites and data and dealing with their changes over time.

Another open area for research is determining semantics and exploiting context information for attribute disambiguation and for improving the accuracy of entity dis-

covery.

Finally, an open challenge, or rather opportunity, remains in applying the advanced and innovative information retrieval techniques to develop specialized query systems and a wider range of performant vertical search engines, covering the numerous vertical domains that make the Web such a valuable source of information.

Bibliography

[AGM03] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 337–348. ACM, 2003.

[AHVC11] Vamshi Ambati, Sanjika Hewavitharana, Stephan Vogel, and Jaime Carbonell. Active learning with multiple annotations for comparable data classification task. In *Proceedings of the 4th Workshop on Building and Using Comparable Corpora: Comparable Corpora and the Web*, pages 69–77. Association for Computational Linguistics, 2011.

[Ant05] Tobias Anton. Xpath-wrapper induction by generalizing tree traversal patterns. In *Lernen, Wissensentdeckung und Adaptivitt (LWA) 2005, GI Workshops, Saarbrcken*, pages 126–133, 2005.

[BBC+10a] Lorenzo Blanco, Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Exploiting information redundancy to wring out structured data from the web. In *Proceedings of the 19th international conference on World wide web*, pages 1063–1064. ACM, 2010.

[BBC+10b] Lorenzo Blanco, Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Redundancy-driven web data extraction and integration. In *Procceedings of the 13th International Workshop on the Web and Databases*, page 7. ACM, 2010.

[BCM08] Claudio Bertoli, Valter Crescenzi, and Paolo Merialdo. Crawling programs for wrapper-based applications. In *Information Reuse and Integration, 2008. IRI 2008. IEEE International Conference on*, pages 160–165. IEEE, 2008.

[BCMP08] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Supporting the automatic construction of entity aware search engines.

In *Proceedings of the 10th ACM workshop on Web information and data management*, pages 149–156. ACM, 2008.

[BCMP11] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Characterizing the uncertainty of web data: models and experiences. In *Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality*, pages 1–8. ACM, 2011.

[BDM11] Lorenzo Blanco, Nilesh Dalvi, and Ashwin Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *Proceedings of the 20th international conference on World wide web*, pages 437–446. ACM, 2011.

[BF07] Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of the 16th international conference on World Wide Web*, pages 441–450. ACM, 2007.

[BF10] Luciano Barbosa and Juliana Freire. Siphoning hidden-web data through keyword-based interfaces: Retrospective. *Journal of Information and Data Management*, 1(1):145, 2010.

[BHV10] Maria-Florina Balcan, Steve Hanneke, and Jennifer Wortman Vaughan. The true sample complexity of active learning. *Machine learning*, 80(2-3):111–139, 2010.

[BNN+09] Luciano Barbosa, Hoa Nguyen, Thanh Nguyen, Ramesh Pinnamaneni, and Juliana Freire. Deeppeep: A form search engine. 2009.

[BNN+10] Luciano Barbosa, Hoa Nguyen, Thanh Nguyen, Ramesh Pinnamaneni, and Juliana Freire. Creating and exploring web form repositories. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1175–1178. ACM, 2010.

[Caf09] Michael J Cafarella. Extracting and querying a comprehensive web database. In *CIDR*, 2009.

[CC07] Tao Cheng and Kevin Chen-Chuan Chang. Entity search engine: Towards agile best-effort information integration over the web. In *CIDR*, volume 2007, pages 108–113, 2007.

[CC08] Shui-Lung Chuang and Kevin Chen-Chuan Chang. Integrating web query results: holistic schema matching. In *Proceedings of the 17th*

ACM conference on Information and knowledge management, pages 33–42. ACM, 2008.

[CCG06] Julien Carme, Michal Ceresna, and Max Goebel. Web wrapper specification using compound filter learning. In *Proceedings of the IADIS International Conference WWW/Internet*, volume 2006, 2006.

[CCZ07] Shui-Lung Chuang, KC-C Chang, and Cheng Xiang Zhai. Collaborative wrapping: A turbo framework for web data extraction. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1261–1262. IEEE, 2007.

[CdSGdM10] Eli Cortez, Altigran S da Silva, Marcos André Gonçalves, and Edleno S de Moura. Ondux: on-demand unsupervised learning for information extraction. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 807–818. ACM, 2010.

[CGK78] K Chidananda Gowda and G Krishna. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognition*, 10(2):105–112, 1978.

[Cha02] Michael Chau. Spidering and filtering web pages for vertical search engines. In *Proceedings of the Americas Conference on Information Systems, AMCIS*, 2002.

[CHK09] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101, 2009.

[CHL03] Chia-Hui Chang, Chun-Nan Hsu, and Shao-Cheng Lui. Automatic information extraction from semi-structured web pages by pattern discovery. *Decision Support Systems*, 35(1):129–147, 2003.

[CKGS06] Chia Hui Chang, Mohammed Kayed, Moheb R Girgis, and Khaled F Shaalan. A survey of web information extraction systems. *Knowledge and Data Engineering, IEEE Transactions on*, 18(10):1411–1428, 2006.

[CM06] Valter Crescenzi and Paolo Merialdo. Efficient techniques for effective wrapper induction. In *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, pages 47–47. IEEE, 2006.

[CM08] Valter Crescenzi and Paolo Merialdo. Wrapper inference for ambiguous web pages. *Applied Artificial Intelligence*, 22(1&2):21–52, 2008.

[CMM+01] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118, 2001.

[CMM05] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering web pages based on their structure. *Data & Knowledge Engineering*, 54(3):279–299, 2005.

[COdS+11] Eli Cortez, Daniel Oliveira, Altigran S da Silva, Edleno S de Moura, and Alberto HF Laender. Joint unsupervised structure discovery and information extraction. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 541–552. ACM, 2011.

[Coh03] William W Cohen. Learning and discovering structure in web pages. *IEEE Data Eng. Bull.*, 26(3):3–10, 2003.

[CRB06] Boris Chidlovskii, Bruno Roustant, and Marc Brette. Documentum eci self-repairing wrappers: performance analysis. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 708–717. ACM, 2006.

[CS08] Andrew Carlson and Charles Schafer. Bootstrapping information extraction from semi-structured web pages. In *Machine Learning and Knowledge Discovery in Databases*, pages 195–210. Springer, 2008.

[CVdBD99] Soumen Chakrabarti, Martin Van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11):1623–1640, 1999.

[DBES09] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Integrating conflicting data: the role of source dependence. *Proceedings of the VLDB Endowment*, 2(1):550–561, 2009.

[DHY09] Xin Luna Dong, Alon Halevy, and Cong Yu. Data integration with uncertainty. *The VLDB Journal–The International Journal on Very Large Data Bases*, 18(2):469–500, 2009.

[DKS11] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230, 2011.

[DMP12] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment*, 5(7):680–691, 2012.

[ECD+04] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Web-scale information extraction in knowitall:(preliminary results). In *Proceedings of the 13th international conference on World Wide Web*, pages 100–110. ACM, 2004.

[Eik99] Line Eikvil. Information extraction from world wide web-a survey. 1999.

[EMS10] Andrea Esuli, Diego Marcheggiani, and Fabrizio Sebastiani. Sentence-based active learning strategies for information extraction. In *IIR*, pages 41–45, 2010.

[FB11] Emilio Ferrara and Robert Baumgartner. Design of automatically adaptable web wrappers. *arXiv preprint arXiv:1103.1254*, 2011.

[FDMFB12] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: a survey. *arXiv preprint arXiv:1207.0246*, 2012.

[FGG+12] Tim Furche, Georg Gottlob, Giovanni Grasso, Omer Gunes, Xiaoanan Guo, Andrey Kravchenko, Giorgio Orsi, Christian Schallhart, Andrew Sellers, and Cheng Wang. Diadem: domain-centric, intelligent, automated data extraction methodology. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 267–270. ACM, 2012.

[FGS12] Tim Furche, Georg Gottlob, and Christian Schallhart. Diadem: Domains to databases. In *Database and Expert Systems Applications*, pages 1–8. Springer, 2012.

[FK03] Aidan Finn and Nicolas Kushmerick. Active learning selection strategies for information extraction. In *Proceedings of the International Workshop on Adaptive Text Extraction and Mining (ATEM-03)*, pages 18–25, 2003.

[FMM+04] Sergio Flesca, Giuseppe Manco, Elio Masciari, Eugenio Rende, and Andrea Tagarelli. Web wrapper induction: a brief survey. *AI Commun.*, 17(2):57–61, 2004.

[Fre98] Dayne Freitag. Information extraction from html: Application of a general machine learning approach. In *AAAI/IAAI*, pages 517–523, 1998.

[GHTG12] Pui Leng Goh, Jer Lang Hong, Ee Xion Tan, and Wei Wei Goh. Region based data extraction. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 1196–1200. IEEE, 2012.

[GMM+11] Pankaj Gulhane, Amit Madaan, Rupesh Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeepkumar Satpal, Srinivasan H Sengamedu, Ashwin Tengli, and Charu Tiwari. Web-scale information extraction with vertex. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1209–1220. IEEE, 2011.

[Got08] Thomas Gottron. Clustering template based web documents. In *Advances in Information Retrieval*, pages 40–51. Springer, 2008.

[HB02] Ajay Hemnani and Stephane Bressan. Information extraction-tree alignment approach to pattern discovery in web documents. In *Database and Expert Systems Applications*, pages 789–798. Springer, 2002.

[HCPZ11] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 775–784. ACM, 2011.

[HFV+11] Frederik Hogenboom, Flavius Frasincar, Damir Vandic, Jeroen van der Meer, Ferry Boon, and Uzay Kaymak. Automatically annotating web pages using google rich snippets. 2011.

[HKF+13] Lushan Han, Abhay Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. Umbc ebiquity-core: Semantic textual similarity systems. *Atlanta, Georgia, USA*, page 44, 2013.

[HRRC11] Inma Hernandez, Carlos R Rivero, David Ruiz, and Rafael Corchuelo. A tool for link-based web page classification. In *Advances in Artificial Intelligence*, pages 443–452. Springer, 2011.

[IS06] Utku Irmak and Torsten Suel. Interactive wrapper generation with minimal user effort. In *Proceedings of the 15th international conference on World Wide Web*, pages 553–563. ACM, 2006.

[JSHA06] M. Jamali, H. Sayyadi, B.B. Hariri, and H. Abolhassani. A method for focused crawling using combination of link structure and content similarity. In *WI 2006*, pages 753–756. IEEE, 2006.

[KC10] Mohammed Kayed and Chia Hui Chang. Fivatech: Page-level web data extraction from template pages. *Knowledge and Data Engineering, IEEE Transactions on*, 22(2):249–263, 2010.

[KLHC04] Hung-Yu Kao, Shian-Hua Lin, Jan-Ming Ho, and Ming-Syan Chen. Mining web informative structures and contents based on entropy analysis. *IEEE Trans. Knowl. Data Eng.*, 16(1):41–55, 2004.

[Kus97] Nicholas Kushmerick. *Wrapper induction for information extraction*. PhD thesis, University of Washington, 1997.

[Kus00] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1):15–68, 2000.

[LC06] Loïc Lecerf and Boris Chidlovskii. Document annotation by active learning techniques. In *Proceedings of the 2006 ACM symposium on Document engineering*, pages 125–127. ACM, 2006.

[LDF11] Guoliang Li, Dong Deng, and Jianhua Feng. Faerie: efficient filtering algorithms for approximate dictionary-based entity extraction. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 529–540. ACM, 2011.

[LHZ+09] Na Luo, WeiWei Hu, Jin Zhang, Tao Fu, and Jun Kong. Applying iterative logistic regression and active learning to relevance feedback in image retrieval system. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 1277–1282. IEEE, 2009.

[LK09] Kristina Lerman and Craig A Knoblock. Wrapper maintenance. In *Encyclopedia of Database Systems*, pages 3565–3569. Springer, 2009.

[LLD10] Wei Luo, Qingzhong Li, and Yanhui Ding. An approach based on extracted data for wrapper maintenance. In *Pervasive Computing and Applications (ICPCA), 2010 5th International Conference on*, pages 88–92. IEEE, 2010.

[LLO07] Longzhuang Li, Yonghuai Liu, and Abel Obregon. Visual segmentation-based data record extraction from web documents. In *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, pages 502–507. IEEE, 2007.

[LMK03] Kristina Lerman, Steven Minton, and Craig A Knoblock. Wrapper maintenance: A machine learning approach. *J. Artif. Intell. Res.(JAIR)*, 18:149–181, 2003.

[LMM10] Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vide: A vision-based approach for deep web data extraction. *Knowledge and Data Engineering, IEEE Transactions on*, 22(3):447–460, 2010.

[LNL01] Yi Li, Wee Keong Ng, and Ee-Peng Lim. Vide: A visual data extraction environment for the web. In *Database and Expert Systems Applications*, pages 577–586. Springer, 2001.

[LNL04] Zehua Liu, Wee Keong Ng, and Ee-Peng Lim. An automated algorithm for extracting website skeleton. In *DASFAA*, pages 799–811, 2004.

[LRNdST02] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran Soares da Silva, and Juliana S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.

[MAAH09] Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, and Alon Halevy. Harnessing the deep web: Present and future. *arXiv preprint arXiv:0909.1785*, 2009.

[MFH10] Sergio Mergen, Juliana Freire, and Carlos A Heuser. Querying structured information sources on the web. *International Journal of Metadata, Semantics and Ontologies*, 5(3):208–221, 2010.

[MHL03] Xiaofeng Meng, Dongdong Hu, and Chen Li. Schema-guided wrapper maintenance for web-data extraction. In *Proceedings of the 5th ACM international workshop on Web information and data management*, pages 1–8. ACM, 2003.

[MKK+08] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proc. VLDB Endow.*, 1(2):1241–1252, August 2008.

[MMK03] Ion Muslea, Steven Minton, and Craig A Knoblock. Active learning with strong and weak views: A case study on wrapper induction. In *IJCAI*, volume 3, pages 415–420, 2003.

[NFP+11] Hoa Nguyen, Ariel Fuxman, Stelios Paparizos, Juliana Freire, and Rakesh Agrawal. Synthesizing products for online catalogs. *Proceedings of the VLDB Endowment*, 4(7):409–418, 2011.

[NMS+07] Zaiqing Nie, Yunxiao Ma, Shuming Shi, Ji-Rong Wen, and Wei-Ying Ma. Web object retrieval. In *Proceedings of the 16th international conference on World Wide Web*, pages 81–90. ACM, 2007.

[NNF08] Hoa Nguyen, Thanh Nguyen, and Juliana Freire. Learning to extract form labels. *Proceedings of the VLDB Endowment*, 1(1):684–694, 2008.

[NWM07] Zaiqing Nie, Ji-Rong Wen, and Wei-Ying Ma. Object-level vertical search. In *CIDR*, pages 235–246, 2007.

[PS11] Ahmed Patel and Nikita Schmidt. Application of structured document parsing to focused web crawling. *Computer Standards & Interfaces*, 33(3):325–331, 2011.

[RC11] Parisa Rashidi and Diane J Cook. Ask me better questions: active learning queries based on rule induction. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 904–912. ACM, 2011.

[RGM00] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. 2000.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *27th International Conference on Very Large Data Bases*, 2001.

[SA99] Arnaud Sahuguet and Fabien Azavant. Web ecology: Recycling html pages as xml documents using w4f. *Database Research Group (CIS)*, page 24, 1999.

[SC08] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1070–1079. Association for Computational Linguistics, 2008.

[Set10] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52:55–66, 2010.

[SFG+11a] Andrew Jon Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. Oxpath: little language, little memory, great value. In *Proceedings of the 20th international conference companion on World wide web*, pages 261–264. ACM, 2011.

[SFG+11b] Andrew Jon Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. Taking the oxpath down the deep web. In *EDBT*, pages 542–545, 2011.

[SKW07] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[SKW08] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[SL05] Kai Simon and Georg Lausen. Viper: augmenting automatic information extraction with visual perceptions. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 381–388. ACM, 2005.

[Sod99] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine learning*, 34(1-3):233–272, 1999.

[STS+03] Sergej Sizov, Martin Theobald, Stefan Siersdorfer, Gerhard Weikum, Jens Graupmann, Michael Biwer, and Patrick Zimmer. The bingo! system for information portal generation and expert web search. In *CIDR*, 2003.

[SWL09] Weifeng Su, Jiying Wang, and Frederick H Lochovsky. Ode: Ontology-assisted data extraction. *ACM Transactions on Database Systems (TODS)*, 34(2):12, 2009.

[SWL+12] Dandan Song, Yunpeng Wu, Lejian Liao, Long Li, and Fei Sun. A dynamic learning framework to thoroughly extract structured data from web pages without human efforts. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, page 9. ACM, 2012.

[TL11] Aibo Tian and Matthew Lease. Active learning to maximize accuracy vs. effort in interactive information retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 145–154. ACM, 2011.

[VdSdMC06] Márcio L. A. Vidal, Altigran Soares da Silva, Edleno Silva de Moura, and João M. B. Cavalcanti. Gogetit!: a tool for generating structure-driven web crawlers. In *WWW*, pages 1011–1012, 2006.

[WT10] Gerhard Weikum and Martin Theobald. From information to knowledge: harvesting entities and relationships from web sources. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 65–76. ACM, 2010.

[YCB+07] Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007.

[ZL06] Yanhong Zhai and Bing Liu. Structured data extraction from the web based on partial tree alignment. *Knowledge and Data Engineering, IEEE Transactions on*, 18(12):1614–1628, 2006.

[ZMR08] Chang Zhao, Jalal Mahmud, and IV Ramakrishnan. Exploiting structured reference data for unsupervised text segmentation with conditional random fields. In *SDM*, pages 420–431. SIAM, 2008.

[ZSWW07] Shuyi Zheng, Ruihua Song, Ji-Rong Wen, and Di Wu. Joint optimization of wrapper generation and template detection. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 894–902. ACM, 2007.