Enterprise Information Integration

Unsupervised proposals for web information extraction

Hassan A. Sleiman

University of Sevilla, Spain

Doctoral Dissertation
Supervised by Dr. Rafael Corchuelo

December, 2012

First published in December 2012 by
The Distributed Group
ETSI Informática
Avda. de la Reina Mercedes s/n
Sevilla, 41012. SPAIN

Copyright © MMXII The Distributed Group
http://www.tdg-seville.info
contact@tdg-seville.info

Classification (ACM 1998): D.2.11 [Software Architectures] Domain-specific architectures; H.1.2 [User/Machine Systems] Human information processing; H.3.4 [Systems and Software] Performance evaluation; H.3.5 [Online Information Services] Web-based services; H.3.m [Information Storage and Retrieval]: Miscellaneous-Data Extraction, Wrapper Generation; I.2.6 [Learning] induction; I.5.2 [Design Methodology] Pattern analysis; I.7.5 [Document Capture] Document analysis.

Support: Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, TIN2010-09988-E, and TIN2011-15497-E).

University of Sevilla, Spain

The committee in charge of evaluating the dissertation presented by Hassan A. Sleiman in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Software Engineering, hereby recommends ____________________ of this dissertation and awards the author the grade ____________________________________.

____

Miguel Toro Bonilla
Catedrático de Universidad
Universidad de Sevilla

____

Juan Pavón Mestras
Catedrático de Universidad
Universidad Complutense de Madrid

____

Emilio Corchado Rodríguez
Catedrático de Universidad
Universidad de Salamanca

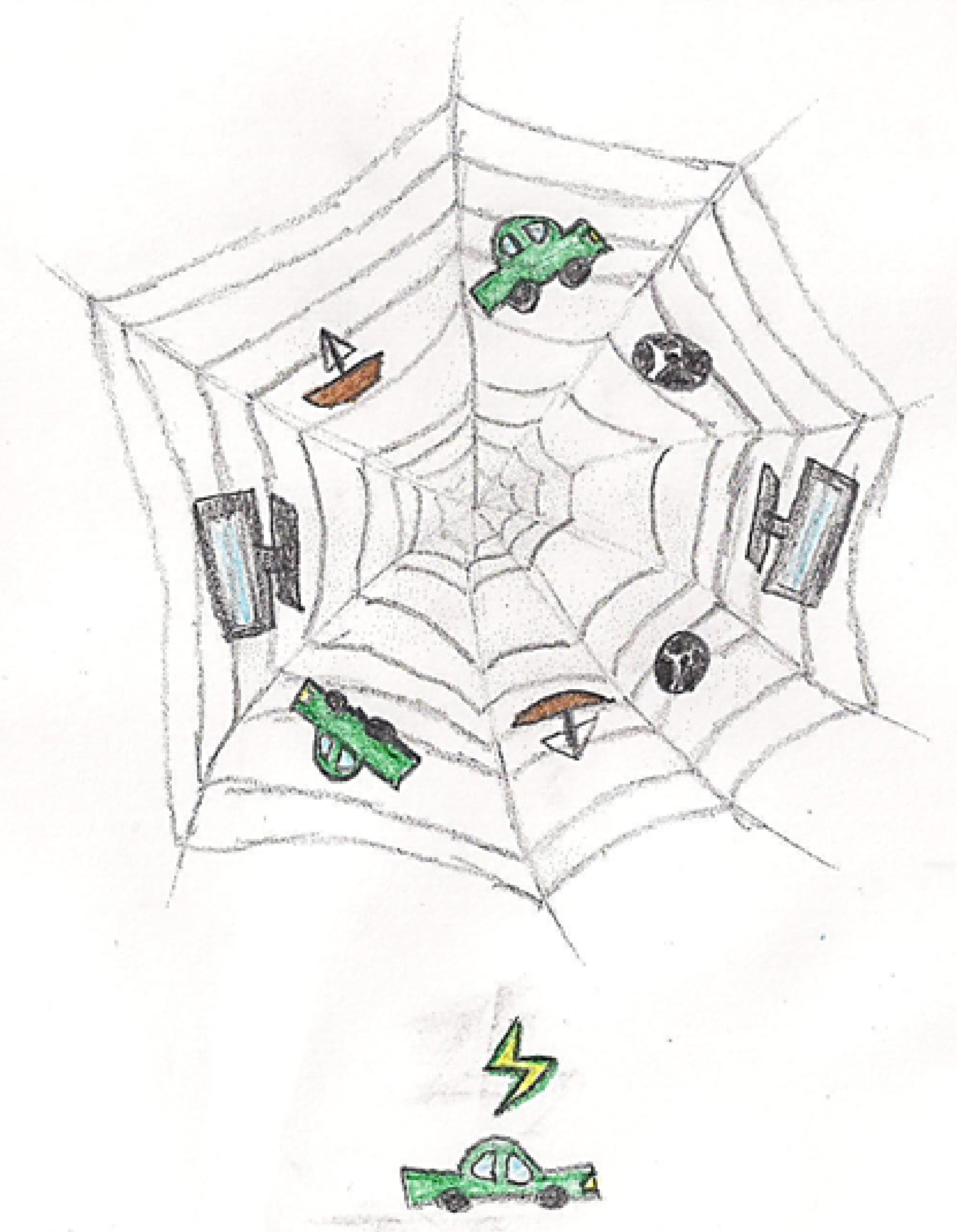
____

Alberto Pan Bermúdez
Profesor Contratado Doctor
Universidad de A Coruña

____

Carlos Molina Jiménez
Research Associate
Newcastle University

To put record where necessary, we sign minutes in ____________, ________________.



Extracting information of interest from the World Wide Web, by Alberto, aged twelve.

To my family.

Contents

III Our Proposal

IV Final Remarks

List of Figures

List of Tables

List of Programs

Acknowledgements

A little knowledge that acts is worth infinitely more than much knowledge that is idle.

Gibran Kahlil Gibran, Lebanese artist, poet, and author (1883-1931)

Some people believe that when we are born or when we die, the most important moments of our life flash before our eyes. I do not believe in that, but I do believe that at the end of a Philosophy Doctor Thesis, the important moments of the PhD are remembered. Some of these moments are full of sadness and others are full of happiness, but they both share a unique message: cheer up, you can do it!

I feel too lucky since Dr. Rafael Corchuelo has supervised my PhD. He is even more than a supervisor, he is a friend. I owe sincere thankfulness to Rafael for his support, understanding, encouragement, comments, dedication, and guidance during the last years. I am sure that my achievements in these years would not have been possible without him. Thank you.

It was a great pleasure to work during the last years with the TDG family in Sevilla and Huelva. They created a suitable work environment, which provided me with valuable feedbacks for this dissertation. I also would like to thank the AIFB and FZI families from the KIT with whom I spent the last three months of this dissertation. The comments and the experience they provided helped me acquire an important knowledge for my future. I would like to thank the external reviewers, namely: Dr. Rafael Z. Frantz and Dr. Fabricia Roos Frantz, whose comments helped us improve this dissertation.

I could not forget to thank all of my friends and family members for their support during the last years. Finally, and most important, I would like to thank my parents and my brothers, to whom I dedicate this dissertation. They all have been a constant source of love, concern, support, and strength all these years. Thank you all.

Abstract

You know that the beginning is the most important part of any work.

Plato, Greek philosopher (427-347BC)

The goal of Enterprise Information Integration is to provide a uniform access to multiple data sources, which should be viewed as if they were a unique and integrated database. Wrappers are software modules that aim to provide an API that abstracts developers away from the details required to simulate a human interacting with a search form and transforming the results into structured data. One of the key components of a web wrapper is the information extractor, which is used to extract and structure information from web documents. The literature provides many techniques to learn information extractors from samples, but none of them is universally applicable. We focus on unsupervised techniques to learn extraction rules and on heuristic-based information extractors that do not rely on rules. The problem that we address in this dissertation is how to reduce the costs of developing an information extraction proposal, how to compare techniques homogenously, and how to extract information using efficient and effective techniques. Currently, the literature does not provide any frameworks to help software engineers devise and implement new information extraction techniques for semi-structured web sites, and current unsupervised information extraction techniques suffer from some drawbacks that hinder their applicability in practice. In this dissertation we present a reference architecture and an accompanying framework to help software engineers devise new information extraction techniques for semi-structured web documents. Furthermore, we propose two unsupervised information extraction techniques that have proven to be very effective and efficient in practice.

Resumen

Sabes que el inicio es la parte más importante de cualquier trabajo.

Platón, Filósofo griego (427-347 AC)

El objetivo de la integración de información empresarial es ofrecer un acceso uniforme a multiples fuentes de datos, que deben ser tratados como una base de datos única. Los wrappers web son módulos software que tienen como objetivo ofrecer una API para abstraer a los desarrolladores de los detalles requeridos para simular el comportamiento de una persona con los formularios y para transformar los resultados a datos estructurados. Un componente clave en un web wrapper es el extractor de información, que se usa para extraer y estructurar la información de los documentos web. Existen en la actualidad muchas técnicas para aprender las reglas de extracción de información, pero ninguna de ellas es aplicable universalmente. En esta tesis doctoral, nos centramos en las técnicas no supervisadas para aprender estas reglas y los extractores de información basados en heurísticas que no utilizan reglas. Los problemas que estudiamos en esta tesis doctoral son la forma de reducir los costes de desarrollo de las técnicas de extracción de información, la forma de comparar estas técnicas de una forma homogénea y cómo extraer información usando técnicas de extracción eficientes y efectivas. Actualmente, no existe ningún framework para ayudar a los ingenieros del software a diseñar e implementar nuevas técnicas de extracción de información para sitios web semi-estructurados; además, las técnicas no supervisadas existentes tienen diversos problemas que afectan a su aplicación en la práctica. En esta tesis doctoral presentamos una arquitectura de referencia acompañada de un framework para ayudar a los ingenieros del software a desarrollar nuevas técnicas de extracción de información para documentos semi-estructurados. Además, proponemos dos técnicas no supervisadas para la extracción de información que han demostrado ser muy efectivas y eficientes en la práctica.

Part I

Preface

Chapter 1

Introduction

The beginning of all knowledge is the discovery of something interesting
that we do not understand.

Frank P. Helbert, American author (1920-1986)

Our goal in this dissertation is to report on our work to create an information extraction reference architecture and on two unsupervised information extraction techniques. This chapter introduces this dissertation. It is organised as follows: in Section §1.1, we first introduce the context of our research work; Section §1.2 presents the hypothesis that has motivated it and states our thesis; Section §1.3 summarises our main contributions; Section §1.4 introduces the collaborations we have conducted throughout the development of this dissertation; finally, we describe its structure in Section §1.5.

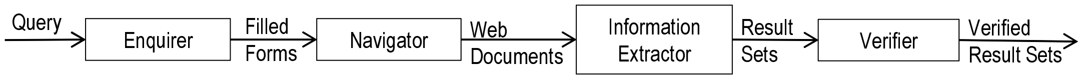


Figure 1.1: *Components of a typical web wrapper.*

1.1 Research context

The idea behind Enterprise Information Integration, or EII for short, is to have a target schema that integrates the data managed independently by several applications, so that they can be seen as if they were a large database [61]. Wrappers are pivotal to EII, since they allow to have access to an application's data so that it can be integrated.

Our focus in this dissertation is on web applications that do not provide a programmatic or data-oriented interface, but a user interface only, which is typically the case of many web applications. Integrating them is challenging insofar building a wrapper amounts to writing a component that emulates a human interacting with them.

A typical web wrapper is composed of the components in Figure §1.1, namely: the enquirer takes a user query as input and maps it onto the appropriate search forms provided by a web application; the navigator cares of submitting the filled forms provided by the enquirer and navigating through the results to fetch web documents that are relevant to the user query; the information extractor is responsible for extracting relevant information from these documents. Since the previous components rely on artificial intelligence techniques, they are likely not to work well if the web site for which they have been trained undergoes redesign changes. This is the reason why a wrapper should incorporate a final component to verify the information returned by the information extractor.

In this dissertation, we focus on information extractors, which constitute a vast research field in which there are a plethora of proposals. The common theme is to help transform web documents into structured information, i.e., data for which there is an explicit model. This idea is not new at all. In 1950, Zellig Harris suggested that it would make sense to reduce documents to a tabular structure, as a means to provide an abstract with relevant facts only. Sager [129] devised one of the earliest materialisations of Harris's ideas in the context of medical documents. With the advent of the Web in

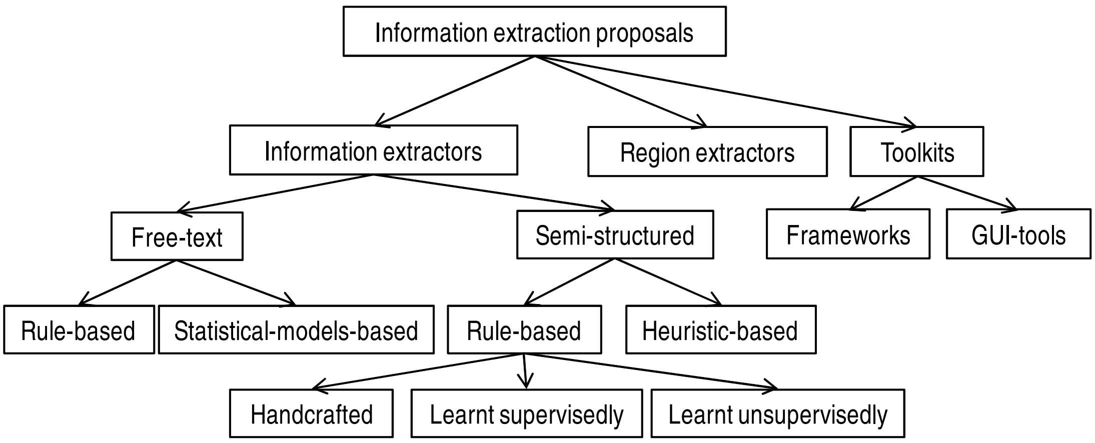


Figure 1.2: *Proposals related to information extraction.*

the early 90s, the problem attracted an increasing number of researchers, first in the context of the well-known Message Understanding Conference series, or MUC conferences for short, and later in the context of the SIGMOD, WWW, VLDB, and CIKM conferences, to mention a few.

Figure §1.2 provides a taxonomy of the many proposals that have been presented in this context. They can be classified into information extractors, region extractors, and toolkits. Unfortunately, none of these proposals is universally applicable, which makes this quite an active research area [34]. In the following paragraphs, we provide an insight into each of the categories in our taxonomy.

Information extractors: The emphasis of these earliest attempts to address the problem focused on documents that were written in natural language (free-text documents), including telegraphic language. Note that a free-text web document usually contains HTML tags that provide a little structure, e.g., <h1> tags to typeset a title, <div> tags to typeset the authors, <p> tags to typeset paragraphs, and <strong> or <emph> tags to emphasise some pieces of text. Unfortunately, this little structure is not enough to characterise the information to be extracted; that is, natural language processing techniques are required to extract relevant information from these documents.

This field attracted the attention of many authors, who devised powerful natural language processing proposals. According to Turmo and others [158] and Sarawagi [131], roughly half the information extractors from free-text documents are based on rules [2, 11, 19, 49, 74, 83, 127, 150, 151, 159, 166] and the other half are based on statistical models [23, 24, 29, 47, 50, 51, 79, 113, 121, 128, 132, 137, 155, 168, 172].

As the web gained popularity, it progressively turned into a platform through which most companies provide information about their catalogues of products and/or services. It is very common that these catalogues are generated using server-side templates, which results in so-called semi-structured documents in which the information of interest is rendered as attributes in lists or tabular forms. In these documents, HTML tags provide far more structure than in free-text documents since the pieces of information to extract are usually enclosed within formatting tags. Such pieces of information are usually referred to as attributes or slots in this context.

The proposals in this field can be classified into two categories, namely: rule-based and heuristic-based information extractors. Rule-based information extractors rely on so-called extraction rules, which can be handcrafted [9, 30, 55, 62, 112, 124, 130], learnt using supervised techniques [17, 20, 28, 48, 57, 71, 73, 87, 93, 118, 151, 157], or learnt using unsupervised techniques [7, 10, 18, 21, 32, 56, 72, 81, 101, 103, 104, 162, 169, 173].

Information extractors that rely on extraction rules do not usually adapt well to changes to the Web. Note that once a set of extraction rules is handcrafted or learnt, the Web keeps evolving and it is not uncommon that changes may invalidate the existing extraction rules. This motivated some authors to work on proposals to maintain extraction rules (semi-) automatically [22, 91, 96, 110, 123, 125]. Contrarily, others worked on unsupervised proposals that do not rely on extraction rules. These proposals are referred to as heuristic-based information extractors; they are based on a number of hypothesis and heuristics that have proven to work well in many cases [6, 38, 59, 98, 106, 136, 152]. Note that changes to a web site do not usually have an impact on these extractors since they analyse every new web document independently from the previous ones.

Region extractors: As the complexity of typical web documents increases, information extractors have to analyse more and more irrelevant regions, which has an impact on both efficiency and effectiveness [80, 161, 167]. This has motivated a number of authors to work on region extractors as a means to relieve information extractors from the burden of analysing the regions of a web document that are not likely to contain any relevant information [142]. A region in a web document is an HTML fragment that shows information about one or more related items when it is rendered on a web browser. A region can be an individual record (that is, a collection of related attributes), a data region (which encompasses a series of data records), or an ancillary region (which refers to headers with navigation menus, footers with contact and corporate information, or sidebars with advertisements, to mention a few). The majority of region extractors focus on data records and data regions; only a few attempt to extract other regions.

The difference between information extractors and region extractors is that the former focus on extracting and structuring data records and their attributes, whereas the latter focus on identifying the HTML fragments that contain this information. Yi and others [167], Wang and Lochovsky [161], and Kang and Choi [80] have confirmed experimentally that using region extractors has a positive impact on both the effectiveness and the efficiency of information extractors; it is not surprising then that some recent proposals for information extraction incorporate a built-in region extractor [98, 101, 136, 162, 169]. The literature records an increasing number of proposals in this area [14–16, 39, 80, 97, 100, 111, 119, 120, 134, 161, 163, 170, 171].

Toolkits: This category includes proposals that are intended to help end users build an information extractor or put an information extractor in production scenarios.

Toolkits can be classified into two subcategories, namely: software frameworks and user interfaces. Software frameworks aim to help users build their information extractors without starting from scratch [33, 45], whereas, user interfaces facilitate putting information extraction proposals into practice in production scenarios, i.e., they provide a wizard-like environment that guides users in tuning and deploying an information extraction technique [1, 12, 36, 46, 63, 94, 99, 102, 105, 114, 115].

1.2 Research rationale

In this section, we present the hypothesis that has motivated our research work in the context of information extraction from semi-structured web documents; we also state the thesis that we prove in the rest of the dissertation.

1.2.1 Hypothesis

Companies are increasingly relying on software applications to manage their data. Although these applications are very valuable on their own, most companies have realised that integrating web information with the information provided by these applications is even more valuable since this usually results in better support for business processes [84]. We think that more and more companies shall rely on an increasing number of such automated business processes, which shall require more and more web information to be integrated to support them.

Unfortunately, even though the technologies provided by the Service Oriented Architecture and the Semantic Web initiatives are helping cut web information integration costs down, a recent report by IBM [75] highlighted that 80% of the information on the Web is not structured, but in semi-structured or unstructured forms. Furthermore, Gartner [84] highlighted the importance of information extraction in the semantic connectivity technology trend. Another recent paper published in the SIGMOD conference [25] highlighted the high costs involved in developing and maintaining information extractors.

According to the previous argumentation, we formulate the following hypothesis:

> *Companies are increasingly interested in extracting information from the Web to enrich their business processes. Software engineers need a software framework to reduce the effort to develop information extraction proposals and to compare them homogenously.*

1.2.2 Thesis

Software frameworks have demonstrated to be a very useful tool to reduce the effort required to develop a software system and, thus, the costs involved [41]. In the context of information extraction, there are a number of frameworks that help software engineers reuse free-text components to devise and implement new information extraction techniques [33, 45]. Unfortunately, no such a software framework seems to exist in the context of web information extractors for semi-structured documents.

The Web has evolved and information extraction techniques that used to perform well a few years ago are facing problems in the current web: web documents are more and more complex, and incorporating information extraction proposals into business processes requiere both effectiveness and efficiency.

According to the previous argumentation, we formulate the following thesis:

> *It is possible to build an information extraction framework for semi-structured documents that provides a reference architecture for software engineers and new unsupervised techniques that achieve high effectiveness and efficiency on current web documents.*

1.3 Summary of contributions

Next we summarise the contributions we have made to prove our thesis.

CEDAR: This is a reference architecture and a software framework that helps software engineers design and implement new proposals in the field of information extraction from semi-structured web documents. To validate it, we have implemented four supervised information extraction techniques that got inspiration from other classical techniques in the literature. Furthermore, it was used to develop two new unsupervised information extraction techniques, namely TEX [143, 144], and Trinity [145]. We got the following publications regarding this contribution: [42, 138–142, 146–148].

TEX: This is a heuristic-based information extractor that extracts attributes from semi-structured web documents. It saves end users from the burden of annotating training examples to learn extraction rules and, consequently, from maintaining them. It works on malformed web documents since it does not require converting HTML code into XHTML or to build DOM trees. Furthermore, it can work on both single- and multi-record web documents. We got the following publications regarding this contribution: [143, 144].

Trinity: This is an unsupervised proposal to learn extraction rules. These rules are regular expressions that model the server-side template used to generate a number of web documents; these expressions contain capturing groups that help extract information from similar web documents. Since it is unsupervised, it does not require the user to provide any annotations, only to interpret the resulting rules. This facilitates maintaining them when a web site is redesigned. Our technique can work on malformed input documents and deal with both single- and multi-record web documents. We got the following publications regarding this contribution: [145].

In addition to the previous main contributions, we also got involved in a number of complementary contributions with other members of our research group: [35, 43, 44, 67, 68, 149].

1.4 Collaborations

Throughout the development of this dissertation, a three-month research visit was paid to the Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) at the Karlsruhe Institute of Technology (Germany). This visit was paid to the research group headed by Prof. Dr. Rudi Studer, under the supervision of Dr. Achim Rettinger. The focus was on presenting our research results, researching on how our work can be used to support the design and implementation of a technique to map semi-structured web documents onto ontological models using transducers.

1.5 Structure of this dissertation

This dissertation is organised as follows:

Part I: Preface. It comprises this introduction and Chapter §2, in which we motivate our research work and conclude that there is currently a lack of a software framework to support software engineers and researchers in devising and implementing their information extraction techniques for semi-structured web documents, and that current information extraction techniques have a number of drawbacks.

Part II: Background Information. It provides information about the software tools and technologies that are related to our research context. In Chapter §3, we describe the software frameworks that are currently available to build information extractors. In Chapter §4, we describe the unsupervised proposals in the literature that are based on extraction rules. In Chapter §5, we describe the proposals in the literature that are based on heuristics. In Chapter §6, we study the steps to evaluate an information extraction proposal.

Part III: Our Proposal. It reports on the core contributions we made with this dissertation. In Chapter §7, we report on CEDAR, our reference architecture and software framework. In Chapter §8, we introduce TEX, our heuristic-based information extractor. In Chapter §9, we present Trinity, an unsupervised technique to learn extraction rules.

Part IV: Final Remarks. This concludes the dissertation and highlights a few future research directions in Chapter §10.

Chapter 2

Motivation

One finds the truth by making a hypothesis and comparing observations with the hypothesis.

David H. Douglass, American physicist (1920-1986)

Web information extraction has been extensively researched. However, existing proposals in this field have a number of drawbacks that hinder their applicability to many current web documents. In this chapter, we present some important requirements for a web information extractor, we detail to which extent current proposals deal with these requirements, and we motivate the need for new proposals in this field. This chapter is organised as follows: in Section §2.1, we introduce it; in Section §2.2, we present the requirements of an information extractor in detail; in Section §2.3, we describe the current proposals; in Section §2.4, we discuss these solutions and we conclude that current proposals do not fulfill these requirements at a time; in Section §2.5, we introduce our contributions; finally, Section §2.6 summarises this chapter.

2.1 Introduction

The Web provides a huge amount of information and it is still growing [5, 58]. This unlimited repository has attracted the attention of many companies that have devised applications that consume and analyse this information [84]. Business processes are more and more commonly fed by Enterprise Information Integration systems that gather data from semi-structured web documents. Unfortunately, integrating this information into business processes is a costly task since web information is usually embedded in HTML tags and buried into other superfluous contents. This has motivated many authors to work on web information extractors that allow to extract relevant information from web documents and structuring it in formats that can be easily consumed by business processes.

In recent years, researchers and software engineers have realised that devising an information extractor from scratch is costly [25, 95, 107]. This has motivated the emergence of some software frameworks that aim to reduce development costs. However, as we discuss later, these frameworks focus exclusively on free-text documents, which makes them of little interest for researchers and software engineers who focus on semi-structured web documents.

Regarding information extractors, the literature provides a plethora of proposals, which range from information extractors whose extraction rules must be handcrafted to proposals to learn them supervisedly or unsupervisedly, including heuristic-based proposals that do not rely on rules. Due to the costs involved in handcrafting rules or using a supervised technique, the current focus of researchers is unsupervised rule learners or heuristic-based information extractors. We have identified a number of requirements that should be addressed by information extraction proposals. These requirements are intended to facilitate integrating these proposals in current enterprise information integration solutions and to achieve high effectiveness and efficiency.

2.2 Requirements

Devising and implementing an information extractor is not a simple task. It is challenging from both a conceptual point of view, since the array of techniques that are currently available make it difficult to innovate, and a software engineering point of view, since the algorithms involved are commonly intricate and difficult to give expression in a good design. We have identified a number of requirements that are necessary for researchers and developers when they devise or implement a new information extraction technique; the most important are the following:

(R1.1) Count on a software framework: A software framework implements a reference architecture that helps software engineers reuse pieces of software to solve common problems, so that they only have to focus on the details that are specific to their technique. Counting on a software framework helps software engineers develop information extractors for semi-structured web sites. This is an important requirement since, if it is not fulfilled, software engineers need to implement their proposals from scratch, i.e., they need to pay attention to a variety of details that are ancillary and common to many other proposals, but do not constitute the core of their research. Furthermore, the lack of such a framework may lead to a variety of terminologies, which makes communication amongst software engineers very difficult.

(R1.2) Count on an evaluation methodology: It is necessary to follow a consensus on how to study the evaluation results of different proposals from a comparative point of view. The lack of such requirement may lead to a situation in which making a decision on which technique is the most appropriate for a particular problem remains largely subjective. Counting on a common evaluation methodology allows the published results to be side-by-side comparable because the evaluation methodology used to compute them is the same for different techniques.

(R1.3) Count on an evaluation repository: It is necessary to count on an up-to-date and publicly available collection of datasets on which information extraction techniques can be tested. Such repository allows to compare published results side-by-side since they have been computed on the same datasets.

Regarding information extractors, themselves, our focus in this dissertation is on unsupervised rule-based or heuristic-based proposals. We have identified a number of requirements that the proposals should meet, namely:

(R2.1) Ability to work on current web documents: The Web is still evolving and typical web documents are growing in complexity. An essential requirement for the information extraction proposals is to be based on simple algorithms that can work correctly on recent web documents despite their complexity and errors.

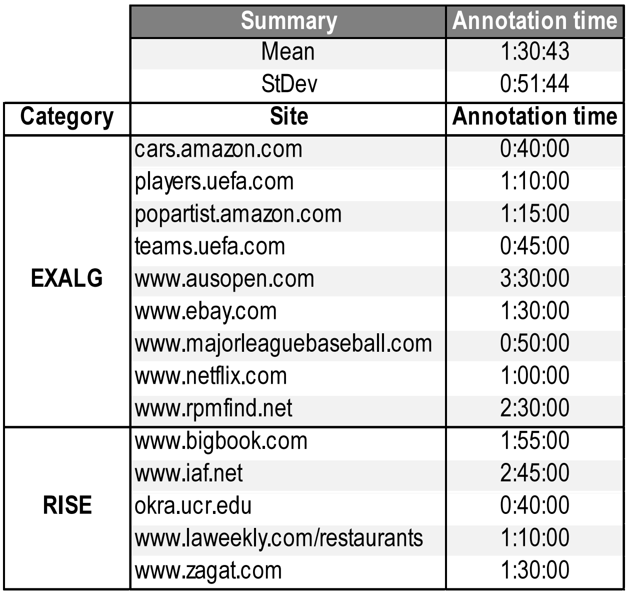
(R2.2) Work on text view: Real-world web sites usually contain errors in their HTML code. Working on DOM trees requires to convert HTML code that is very often malformed into correct XHTML and then into DOM trees. This requires to use an HTML cleaner, which is a heuristic-based tool that attempts to mend the errors in the HTML code; the problem is that mending an error may result in additional spurious errors, which is problematic insofar these new errors may lead to noise. Then, working on a text view is a requirement for information extractors since it helps reduce the numbers of possible errors during information extraction.

(R2.3) Work on both single- and multiple-record web documents: It is necessary for an information extractor to work on all kinds of semi-structured web documents. This is an important requirement since many proposals only work correctly on multi-record web documents by identifying repetitive patterns within a single document, which prevents them from being used in detail documents that provide a single record of information.

2.3 Analysis of current solutions

Despite the high number of proposals on information extraction in the literature, only few of them have focused on devising a reference architecture and a software framework to devise information extractors [1, 28, 33, 45, 63, 88, 154]. Some of these proposals claim to offer an environment in which information extractors can be developed and tested [1, 28, 63, 88, 154]. However, they were presented as tools and the description of their architecture is not clear. Furthermore, the majority of them are neither available nor maintained any more.

The most popular and up-to-date frameworks are GATE [33] and UIMA [45]. They both provide a reference architecture and an accompanying software framework that aim to help software engineers devise solutions to process large volumes of text. They both provide a number of components that help tokenise text, determine the grammatical role of a word in a sentence, analyse the structure of a sentence, and so on; they also provide kind of an engine in which these components can be composed in a reusable workflow. Unfortunately, they were devised to work on free-text documents,



	Summary	Annotation time
	Mean	1:30:43
	StDev	0:51:44
Category	**Site**	**Annotation time**
EXALG	cars.amazon.com	0:40:00
	players.uefa.com	1:10:00
	popartist.amazon.com	1:15:00
	teams.uefa.com	0:45:00
	www.ausopen.com	3:30:00
	www.ebay.com	1:30:00
	www.majorleaguebaseball.com	0:50:00
	www.netflix.com	1:00:00
	www.rpmfind.net	2:30:00
RISE	www.bigbook.com	1:55:00
	www.iaf.net	2:45:00
	okra.ucr.edu	0:40:00
	www.laweekly.com/restaurants	1:10:00
	www.zagat.com	1:30:00

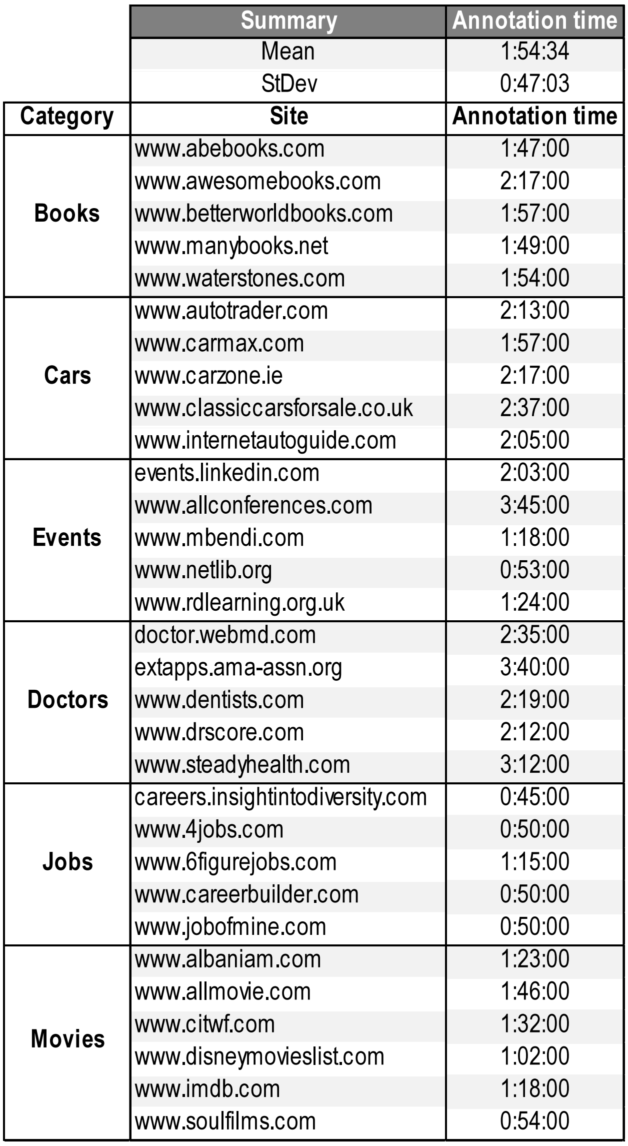
Table 2.1: *Time necessary to handcraft some extraction rules.*

which makes them of little interest in the field of semi-structured web documents.

The problem of evaluating information extractors has been addressed in several papers [3, 40, 90, 95, 126]. Their focus was on free-text information extractors and provided some guidelines only. Unfortunately, not a word on how to evaluate a semi-structured information extractor or to produce a ranking has been published.

Regarding evaluation repositories, there are some public ones that focus on free-text proposals [77, 156], another that focuses on semi-structured proposals [164], and another that is mixed [117]. There are also some ad-hoc repositories [6, 7, 31, 88, 154]. Unfortunately, the public repositories are no longer maintained, which implies that the documents they provide are commonly outdated and little representative of current web documents. Furthermore, most of the ad-hoc repositories have not been publicly released.

Many information extractors rely on extraction rules. These rules can be handcrafted [9, 30, 55, 62, 112, 124, 130]. The problem is that they require the user to analyse the web documents to handcraft rules, which is cumbersome, error-prone, and time-consuming, not to mention that the rules need be maintained. Table §2.1 illustrates the time we required to handcraft extraction rules for the datasets offered by two well-known public repositories [7,



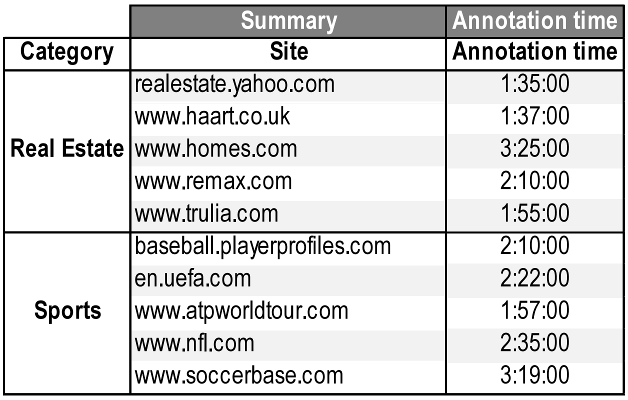
Summary	Annotation time
Mean	1:54:34
StDev	0:47:03

Category	Site	Annotation time
Books	www.abebooks.com	1:47:00
	www.awesomebooks.com	2:17:00
	www.betterworldbooks.com	1:57:00
	www.manybooks.net	1:49:00
	www.waterstones.com	1:54:00
Cars	www.autotrader.com	2:13:00
	www.carmax.com	1:57:00
	www.carzone.ie	2:17:00
	www.classiccarsforsale.co.uk	2:37:00
	www.internetautoguide.com	2:05:00
Events	events.linkedin.com	2:03:00
	www.allconferences.com	3:45:00
	www.mbendi.com	1:18:00
	www.netlib.org	0:53:00
	www.rdlearning.org.uk	1:24:00
Doctors	doctor.webmd.com	2:35:00
	extapps.ama-assn.org	3:40:00
	www.dentists.com	2:19:00
	www.drscore.com	2:12:00
	www.steadyhealth.com	3:12:00
Jobs	careers.insightintodiversity.com	0:45:00
	www.4jobs.com	0:50:00
	www.6figurejobs.com	1:15:00
	www.careerbuilder.com	0:50:00
	www.jobofmine.com	0:50:00
Movies	www.albaniam.com	1:23:00
	www.allmovie.com	1:46:00
	www.citwf.com	1:32:00
	www.disneymovieslist.com	1:02:00
	www.imdb.com	1:18:00
	www.soulfilms.com	0:54:00

Table 2.2: *Time necessary to annotate some web documents.*

117]. Note that we spent an average of one hour and a half per dataset and that this process needs be repeated every time a web site changes.

The costs involved have motivated many researchers to work on proposals to learn extraction rules using supervised techniques [17, 20, 28, 48, 57, 71, 73, 87, 93, 118, 151, 157]. Supervised techniques require the user to annotate a collection of sample web documents. The problem is that producing the an-



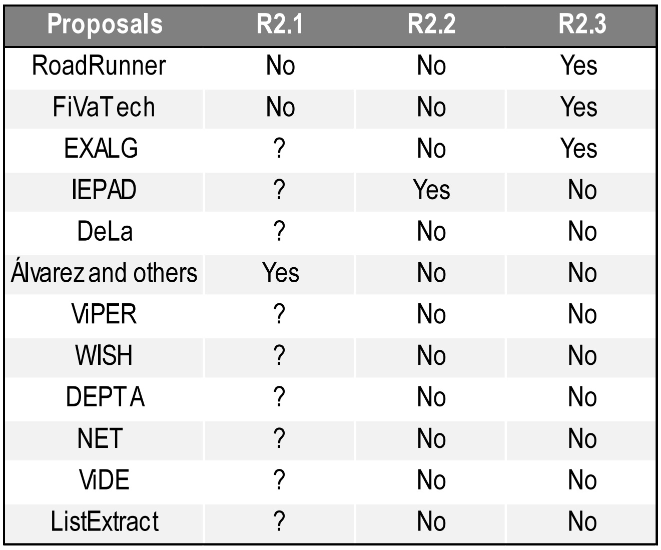
	Summary	Annotation time
Category	Site	Annotation time
Real Estate	realestate.yahoo.com	1:35:00
	www.haart.co.uk	1:37:00
	www.homes.com	3:25:00
	www.remax.com	2:10:00
	www.trulia.com	1:55:00
Sports	baseball.playerprofiles.com	2:10:00
	en.uefa.com	2:22:00
	www.atpworldtour.com	1:57:00
	www.nfl.com	2:35:00
	www.soccerbase.com	3:19:00

Table 2.2: *Time necessary to annotate some web documents. (Cont'd)*

notations is also cumbersome, error-prone, and time-consuming. Table §2.2 shows the time we spent at handcrafting the annotations for our collection of datasets. Each dataset consists of thirty web documents, and we spent roughly two hours at annotating each of them.

The effort required by the previous techniques has motivated many authors to work on unsupervised techniques to learn extraction rules: RoadRunner [32], FiVaTech [81], EXALG [7], IEPAD [21], and DeLa [162]. Unsupervised techniques seem to require less user effort because they learn extraction rules that extract as much information as possible, gives computer-generated labels to the information extracted, and it is the responsibility of the user to assign a meaning to these labels. This usually requires much less effort than handcrafting extraction rules or producing annotations since it usually requires just taking a quick look at the information groups extracted to discover their meaning.

Since extraction rules do not usually adapt well to changes to the Web, some authors worked on so-called heuristic-based proposals that do not rely on extraction rules, but are based on a number of hypotheses and heuristics that have proven to work well on a large number of web sources, namely: Álvarez and others' proposal [6], ViPER [136], WISH [72], DEPTA [169], NET [101], ViDE [103], and ListExtract [38]. These proposals build on the hypothesis that the data records introduce a repeated pattern in web documents, so they try to build the DOM tree of the input web document, and try to find this repeating pattern using different heuristics. This is problematic insofar real-world web sites usually contain errors in their HTML code, which are usually mended using additional heuristics. Furthermore, build-



Proposals	R2.1	R2.2	R2.3
RoadRunner	No	No	Yes
FiVaTech	No	No	Yes
EXALG	?	No	Yes
IEPAD	?	Yes	No
DeLa	?	No	No
Álvarez and others	Yes	No	No
ViPER	?	No	No
WISH	?	No	No
DEPTA	?	No	No
NET	?	No	No
ViDE	?	No	No
ListExtract	?	No	No

R2.1 = Ability to work on current web documents; R2.2 = Work on text view; R2.3 = Work on both single- and multiple-record web documents.

Table 2.3: *Comparison of current unsupervised information extractors.*

ing on the hypothesis that the data records introduce a repeated pattern in the web document does not work in case the input web document reports on a single item.

Table §2.3 summarises how the previous unsupervised proposals deal with the requirements we have identified. The first column in the table shows the name of each technique, and the others report on whether they meet the corresponding requirements or not. Since many proposals are not available for their evaluation, it was difficult to check if the techniques can deal or not with current web documents.

2.4 Discussion

The lack of a reference architectural proposal in the literature to guide software engineers in devising and implementing new information extraction techniques for semi-structured web documents amounts to little reuse; that is, the focus tends to blur because of irrelevant details. Existing proposals [33, 45] focus on information extraction from free-text web documents and do not support extracting information from semi-structured documents (R1.1).

The problem of evaluating information extraction proposals remains largely unexplored in the field of semi-structured web documents [81, 94, 163, 169, 170]. The majority of proposals in the literature simply present their results and compare them with a few more proposals, without checking if the conclusions are statistically significant, neither provide them any details on the evaluation methodology used (R1.2).

The lack of an up-to-date repository is another problem that researchers face when they are interested in evaluating their proposals [95]. The majority of current repositories are intended to evaluate free-text information extractors, whereas the few public repositories on the Web to evaluate information extractors for semi-structured web documents are outdated and no longer maintained (R1.3).

With the increasing complexity of web documents, unsupervised information extractors are becoming more and more attractive insofar they require less user intervention and thus smaller costs. Unfortunately, current proposals in the literature have a number of drawbacks that hinder their applicability in practice.

Unsupervised techniques that learn extraction rules build usually on the DOM trees of the web documents, which may reduce their performance due to the complexity of the current web documents (R2.1) and the possible errors introduced by the HTML cleaner (R2.2). Furthermore, the heuristic-based information extractors in the literature focus on extracting information from multi-record web documents, and none of them focuses on single-record web documents (R2.3).

2.5 Our proposal

We present CEDAR, a reference architecture to design, implement, and test information extractors for semi-structured web documents (see Chapter §7). It provides an abstract and reusable design that should allow software engineers and researchers face the development of a new information extraction technique without incurring the high costs of developing it from scratch (R1.1). To support our reference architecture, we have implemented a software framework and we have validated it by implementing four information extraction techniques that got inspiration from classical techniques in the literature; this helped us prove that CEDAR helps reduce development costs significantly. The framework provides a k-fold cross validator that allows to compare the results about a given proposal to others (R1.2). The framework also provides an up-to-date collection of 55 datasets that were gathered from 40 real-world web sites and two public repositories (R1.3).

We have also devised a heuristic-based information extraction technique called TEX to extract attributes from semi-structured web documents (see Chapter §8). TEX does not rely on DOM trees, but on quite an effective and efficient multi-string alignment algorithm (R2.1). Contrarily to the heuristic-based proposals in the literature TEX does not require the input web documents to be translated into DOM trees, i.e., it can work on malformed web documents without correcting them (R2.2), and does not require the document to contain multiple records (R2.3). It works on two or more web documents and compares them in an attempt to discover shared patterns that are not likely to provide any relevant information, but parts of the template used to generate the web documents.

Furthermore, we have devised a rule-based information extractor called Trinity to extract data records from semi-structured web documents (see Chapter §9). It tries to learn a regular expression that represents the server-side template that was used to generate the input web documents using an effective and efficient multi-string alignment algorithm that was inspired by TEX (R2.1). Contrarily to the unsupervised information extractors in the literature that learn extraction rules, Trinity works on malformed web documents without correcting them (R2.2) and does not require the web documents to be formatted using repetitive patterns (R2.3). Trinity uses an algorithm similar to that one used by TEX, but it learns extraction rules and the schema of the extracted information.

2.6 Summary

In this chapter, we have motivated the reasons for this piece of research work. We have analysed the requirements for developing information extraction techniques, and for the current information extraction proposals. We have concluded that current frameworks do not support developing information extractors for semi-structured web documents, and that current information extraction proposals do not meet some important requirements. This motivated us to work on this topic so as to advance the state of the art a step forward.

Part II

Background Information

Chapter 3

Software frameworks

I am the wisest man alive, for I know one thing: I know nothing.

Socrates, Greek philosopher (469–399 BC)

In this chapter, we present the frameworks in the literature that aim to help researchers and software engineers build their information extractors without having to implement their proposals from scratch. The chapter is organised as follows: in Section §3.1, we provide an overview on the proposals on the literature that provide such frameworks; in Section §3.2, we describe GATE, which is a reference architecture accompanied by a software framework to manage text documents; Section §3.3 presents UIMA, which is another reference architecture and framework to manage information sources; finally, Section §3.4 summarises this chapter.

3.1 Introduction

Information Extraction is the process of analysing unstructured or semi-structured documents and extracting relevant information in a structured format. Information extraction systems are usually composed of a set of components that are intended to read input documents, pre-process them, extract information, and structure it. They usually contain similar components and differ only in the extraction algorithms. This has motivated some authors to work on reference architectures and software frameworks to help researchers and software engineers build their information extractors without having to implement their proposals from scratch.

In the following we describe the state-of-the-art proposals in the literature that provide a reference architecture for information extraction.

3.2 GATE

Gaizauskas and others [52] introduced a system called LASIE in the context of the MUC conferences. Its goal was to extract named entities, i.e., names of people, countries, companies, equipment components, and so on. This system evolved progressively into GATE [33] (General Architecture for Text Engineering), which provides a number of tools, a reference architecture, and a number of components whose focus is on processing free-text documents.

The development tools include the following: GATE Embedded, which is a Java-based software framework that provides the foundation to every other tool or component; GATE Developer, which is an integrated development environment to develop new techniques to analyse free-text documents; GATE Teamware, which is a web-based collaborative platform for annotating free-text documents; Mímir, which is an indexing framework for text and annotations; GATE Wiki, which helps developers write their documentation collaboratively; GATE Cloud, which helps run GATE solutions from the GATE family on cloud computing infrastructures; and GATE Process, which is a set of UML activity diagrams that support GATE Teamware, GATE Developer, and GATE Embedded.

GATE provides several components for language processing tasks, such as parsers, morphological analysers, part-of-speech taggers, information retrieval tools, or information extraction components for different languages. These components rely on a number of linguistic resources that are known as

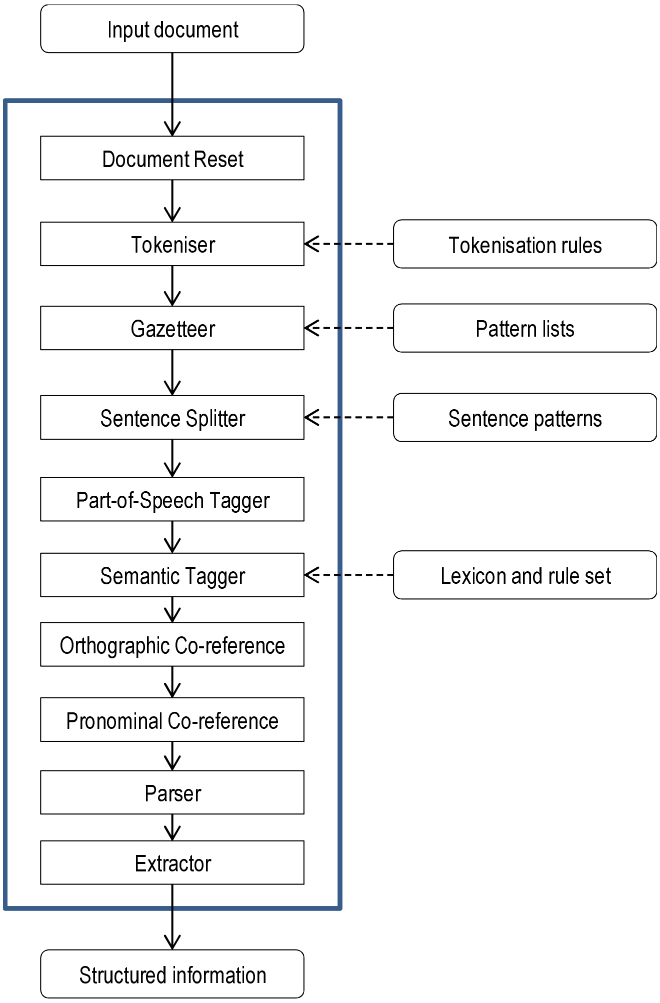


Figure 3.1: *GATE's information extraction components.*

CREOLE (Collection of REusable Objects for Language Engineering). The resources can be classified into three types, namely: language resources, such as lexicons, corporas, and ontologies; processing resources such as parsers and n-gram modellers; and visual resources like editing components.

The previous tools and components can be used to implement custom information extractors [109]. Notwithstanding, the latest version provides ANNIE [108], which is a collection of components that are specifically targeted to extracting information from free-text documents. Figure §3.1 illustrates the components of ANNIE and how they are usually connected in a workflow, namely:

Document Reset: It provides the users with methods to read different types of documents or to download them from a given URL. This component is usually run as the first step of an information extraction algorithm and outputs a document that is prepared to be used in other components of GATE.

Tokeniser: It provides methods to tokenise a document, which can be configured by means of rules. Each rule has a regular expression that describes a token and an annotation that is added to the annotation set of the document whenever the regular expression matches. The tokeniser has a default set of rules that distinguish between upper and lower case words, numbers, punctuation, special symbols, and space tokens. This component also provides a so-called English tokeniser that takes into account the intricacies of the English spelling, e.g., "`don't`" is tokenised as "`do`" and "`not`". The output is a document with annotations that reference the tokens found in the input document.

Gazetteer: This component takes a tokenised document as input and identifies the named entities within it. This component builds on name and pattern lists that help identify named entities. The output is a document in which every named entity found has been annotated appropriately.

Sentence Splitter: This component takes a tokenised web document as input and outputs a document in which the annotations reference the individual sentences of which it is composed. GATE provides two implementations of this component, namely: the simplest one is based on a number of heuristics and lists of abbreviations that prevent it from splitting a sentence at the trailing dot of a common abbreviation; the other is based on a list of regular expressions that help identify the end of a sentence more efficiently and precisely.

Part-of-Speech-Tagger: This component takes a tokenised web document, a lexicon document, a tagging-rule document, and the document output by the sentence splitter as input. It provides a tagger that adds a tag to every word in the input text; the tag describes the grammatical function of that word, e.g., substantive, adjective, verb, adverb, and so on. It is based on Hepple's technique [65] and uses a previously trained lexicon and rule set that can be modified if necessary. The output is a document with annotations that reference the words found in the input document and their grammatical functions.

Semantic Tagger: This component takes a document and the annotation documents produced in earlier steps as input. It provides methods to add some semantic tags to the annotations created by other components. The output is a document that contains an annotation with a semantic category of each word in the input web document. This component is essential for the co-reference components in ANNIE.

Orthographic Co-reference: This component, aka NameMatcher, takes a document and the annotations output by the gazetteer. It aims to identify named entities that are spelled differently, but actually refer to the same real thing. For instance, it can identify that "Mary Smith" and "Mrs. Smith" or that "International Business Machines Ltd." and "IBM" refer to the same actual entities. It modifies the input document to make these correspondences explicit.

Pronominal Co-reference: This component takes a tokenised document and annotations produced by the previous components as input. It provides methods to find the noun or noun phrase to which an anaphora refers; typical anaphora include pronouns and noun sentences. For instance, given the sentences "IBM reported on its profits yesterday; the company was satisfied.", this component is intended to find that "its" and "the company" refer to "IBM". The output is a document in which each pronoun is annotated with the noun or noun phrase to which it refers.

The previous components provide a foundation, but are not enough to extract structured information, only named entities. GATE provides two additional components for this purpose, namely:

Parser: This component aims to construct the syntax tree of a sentence, i.e., it identifies subjects, verbs, and complements and represents them in a hierarchical form that makes it explicit their grammatical structure. GATE provides several implementations of this component, ranging from a shallow one to full parsers, such as SUPPLE, RASP or the Stanford Parser.

Extractor: This component builds on extraction rules that are expressed in Prolog; they help walk the syntax trees output by the parser in order to locate the information on which we are interested. Note that this provides the foundations, again; other authors developed several information extraction techniques building on it [4, 26, 27, 122].

3.3 UIMA

Unstructured Information Management Architecture [45], or UIMA for short, is a framework that was intended to help analyse large amounts of free-text documents, video, and audio to extract relevant information. The project was initiated by IBM Research in 2001 to provide a common software architecture for developing techniques to work with free-text documents in several problems, such as natural-language dialogues, information retrieval, and information extraction, to mention a few. In 2004, IBM released the UIMA Software Development Kit (SDK), which allowed developers to build and deploy components for analysing free-text documents. In 2006, IBM made the UIMA SDK open source and it was accepted as an Apache incubation project. That same year, UIMA was transformed into a standard specification by OASIS; the open source reference implementation is now called Apache UIMA.

UIMA provides an architecture and an accompanying software framework that supports it. The framework provides a collection of components and a run-time environment in which developers can plug in and run their UIMA component implementations along with the other components to build and deploy their applications.

Users interested in using UIMA should define a so-called pipeline configuration. This pipeline contains the components that should run consecutively. An UIMA application is organised as a so-called Collection Processing Engine that consists of one or more UIMA Collection Readers, one or more UIMA Analysis Engines, and one or more Collection Consumers. The input is stored in an internal UIMA data structure called Common Analysis Structure (CAS). In Figure §3.2, we show a typical UIMA pipeline for information extraction. Note that there are different implementations for each type of component and other types of components can be used in the pipeline. In the following we present some more details on this pipeline:

Collection Readers: This group includes the components that allow to have access to information sources, read documents, gather metadata about them, and convert them into Common Analysis Structures that can be processed by the following components in the pipeline. Components in this group include a MySQL reader, an HTML reader, or a plain text reader, to mention a few.

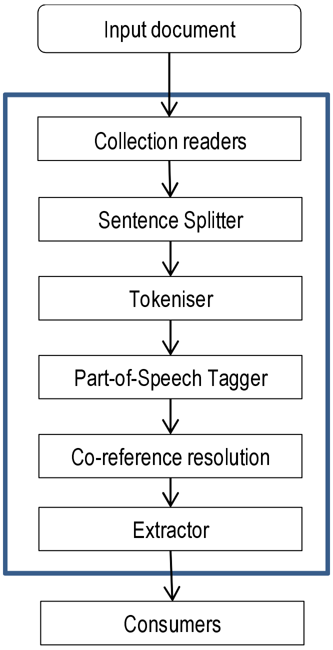


Figure 3.2: *Components used for information extraction in UIMA.*

Analysis Engines: This group includes the components that allow to analyse the input documents, namely: i) a sentence splitter, which splits the input document into individual sentences; ii) a tokeniser, which segments the sentences into tokens using blanks as separators; iii) a part-of-speech tagger, which annotates each token with its grammatical function; iv) a co-reference resolver, which performs anaphora resolution, i.e., makes it explicit which noun or noun phrase is referenced by each pronoun; v) an information extractor that works on the previous annotations and extracts information depending on the domain; authors have developed several information extraction techniques that can be used here [78, 153].

Collection Consumers: This group includes a number of components that work on the information extracted to display it, to serialise it, or to store it to a database, to mention a few examples.

3.4 Summary

The literature provides two reference architectures and frameworks for information extraction. Both are intended to manage large volumes of information and focus on free-text documents and provide a number of components that allow to read input documents, tokenise them, split them into sentences, perform anaphora resolution, and so on. Unfortunately, no support is provided to deal with semi-structured web documents.

Chapter 4

Unsupervised rule-based information extraction

Imagination rules the world.

Napoleon I Bonaparte, French military and political leader (1769–1821)

This chapter is devoted to presenting the state-of-the-art unsupervised proposals to learn information extraction rules. It is organised as follows: Section §4.1 introduces this chapter; Section §4.2 describes RoadRunner; Section §4.3 describes FiVaTech, Section §4.4 describes EXALG, Section §4.5 describes IEPAD, and Section §4.6 describes DeLa. Finally, Section §4.7 summarises this chapter.

4.1 Introduction

Handcrafting extraction rules is usually a difficult and laborious task. Furthermore, these rules tend to be difficult to maintain. This has motivated many authors to work on proposals that learn them automatically.

Earlier proposals in the field of information extraction from semi-structured web documents focused on learning extraction rules starting from annotated samples provided by a user. Producing these annotations is a time-consuming, cumbersome, and error-prone task. In fact, after extraction rules have been learnt for a collection of annotated web documents, a small change in the HTML code can make the information extractor to fail and, as a consequence, the annotation and learning phases have to be repeated.

The previous costly tasks motivated researches to work on learning these rules using unsupervised techniques. The idea is to make the rule learning process completely automatic, without relying on a prior knowledge about the input web documents and their contents. The proposals in this field are intended to learn a rule that describes the template that was used to generate a number of similar web documents, namely: RoadRunner [31], FiVaTech [81], EXALG [7], IEPAD [21], and DeLa [162], which we study in the following sections.

4.2 RoadRunner

RoadRunner [31] is intended to learn a union-free regular expression that describes the server-side template that was used to create the input web documents. It is based on the hypothesis that input web documents were generated using the same server-side template and that comparing these documents side-by-side allows to learn a regular expression that describes it.

The RoadRunner algorithm works as follows:

i. It applies JTidy to mend the HTML code, which is then tokenised using a text-tag tokenisation schema.

ii. The algorithm considers the first web document as an initial template and then compares it to the second input document. The comparison is performed sequentially, token after token. Therefore a mismatch can be of the following types: text to text, tag to tag, tag to text, or tex to tag. In the following paragraphs, we explain how each kind of mismatch is dealt with.

iii. A text-to-text mismatch indicates that there can be relevant data at the position where the mismatch occurs. It can be easily resolved by generalising both pieces of text to a variable that indicates that they must be extracted.

iv. Tag-to-tag, tag-to-text, or text-to-tag mismatches are more difficult to solve since they imply that there may be repeated or optional patterns in the template, i.e., similar subsequences of tokens that occur in sequence or similar sequences of tokens that do not occur in every input document. For the sake of brevity, a subsequence of tokens is also referred to as a fragment. Next we describe how each kind of pattern is detected.

v. To detect repeated patterns, the algorithm first searches for the closing tag that caused the mismatch and checks if its followed by the corresponding opening tag, e.g., if the mismatch is caused by tag <li>, then it searches for </li><li>. If such tags are found, then this is a strong evidence that there is a repeated pattern; in that case, the algorithm tries to identify every fragment that is repeated and substitutes them all by a regular expression.

vi. If no repeating pattern is found, the algorithm applies two steps to detect optional token sequences: i) the algorithm finds the mismatched token sequence that is located between two matching tokens; ii) the mismatched sequence is added to the template using an optional operator as long as it only occurs in the document.

4.3 FiVaTech

FiVaTech [81] is intended to learn a context-free grammar that represents the template used to generate the input web documents and the schema of the information in these documents. It is based on the hypothesis that the input web documents share the same template, that attributes of same type have the same path from the root of the DOM tree, and that similar subtrees contain the same attributes.

The FiVaTech algorithm works as follows:

i. The HTML code of the input web documents is mended and their DOM trees are built using JTidy.

ii. The algorithm tries to merge the DOM trees of the input web document into a so-called fixed/variant pattern tree.

iii. The algorithm iterates over the DOM tree nodes from the root to the leaves level after level. For each node, it collects its direct child nodes and performs four steps, namely: peer-node recognition, matrix alignment, pattern mining, and optional and disjunctive node identification.

iv. The peer-node recognition step creates a so-called peer matrix. The rows of this matrix contain the nodes that should be aligned; the columns contain the nodes in the same web document that occur at the same level in the DOM tree. It assigns a computer-generated label to each node in the matrix such that similar nodes share the same label. The similarity algorithm is a version of the one proposed by Yang [165], in which two nodes may match only if they occur at the same level in the DOM trees, two nodes may match although they have different tags, and two leaf nodes may match although they have different text values.

v. The algorithm aligns the peer matrix by iterating over its rows. For each row, it checks if it is aligned or not. If all the labels in a row are the same, then the row is considered aligned and the algorithm works on the next row; in other case, it tries to align the row by searching for the node(s) in the row that should be shifted down, and the number of rows to be shifted.

vi. Once the peer matrix is aligned, the algorithm converts it into a vector in which each row corresponds to a label in the aligned peer matrix.

vii. The algorithm works on mining patterns inside this vector and on constructing a pattern tree. It searches for repeating patterns in the vector starting from size one until no pattern occurs more than once. When a repeating pattern is found, the algorithm removes every occurrence, but the first one, and it is considered as a list of information. The algorithm also creates an occurrence vector for each label in the result pattern. The vector has a one if the corresponding label appears in an occurrence of the pattern in the peer matrix or a zero in other case.

viii. The algorithm considers that a label is optional if its occurrence vector contains a zero. Three rules are applied before a pattern can be considered optional: i) optional labels that have the same occurrence vector are grouped together; ii) optional labels that have complementary occurrence vectors are grouped together and considered disjunctive; iii) optional labels should be grouped with non-optional labels that occur before them in the pattern.

ix. After learning a template, FiVaTech works on learning an information schema. For each node in the pattern tree, if the node is repeated, then a set type is added to the schema, if the node is optional, then optionality is added to the schema, and the leaf nodes that in the pattern tree that are variant are considered as basic type.

FiVaTech can also be applied to identify the data regions in a web document starting from the schema thus learnt. It builds on the hypothesis that a data region contains several data records and that the node in the schema that identifies the data region is a node whose path from the root of the schema does not contain any nodes of type set.

4.4 EXALG

EXtraction ALGorithm [7], or EXALG for short, is intended to learn a regular expression that represents the server-side template that was used to generate a collection of web documents and the schema of the information in these documents. It is based on the hypothesis that tokens that have the same path from the root node and have the same context share the same role in the web document, that tokens that do not occur in many input web documents are part of the relevant information to be extracted, and that the tokens that occur several times in a large number of web documents are part of the template.

The EXALG algorithm works as follows:

i. The algorithm mends the HTML code and builds the DOM trees of the input web document, tokenises their mended HTML code, and computes the path from the root to each token in the web documents.

ii. The algorithm computes an occurrence vector for each token in the input web documents. The i-th position of this vector contains the number of occurrences of the corresponding token in the i-th document. Two tokens are considered equal if they have the same text and the same path from the root.

iii. It then creates so-called equivalent classes, which are subsets of tokens that have the same occurrence vectors.

iv. Equivalence classes whose number of tokens is less than a threshold or whose total number of occurrences in the input web documents is less than a given threshold, are not considered since they are supposed to be part of the relevant information.

v. An equivalence class is said to be empty if its tokens always occur consecutively without any separators between them. EXALG uses non-empty equivalence classes to learn the template using a recursive algorithm. For each non-empty equivalence class, the algorithm constructs a template for the token at position i and the separators between it and the next token in the equivalence class, if any. The final template is the concatenation of the templates learnt for each equivalence class.

vi. The algorithm checks if there are some patterns that can be identified in each occurrence of the equivalence class. These patterns are used to identify basic, set, optional, and disjunction types inside the template and to construct the schema of the information.

4.5 IEPAD

IEPAD [21] is an information extractor that is based on a pattern discovery technique. It is based on the hypothesis that relevant information in a web document is often displayed regularly and closely together using repetitive patterns, and that they include at least two data records. The IEPAD algorithm works as follows:

i. The algorithm takes a web document that contains two or more data records as input.

ii. The input document is tokenised using TAG and TEXT tokens, but this is not an intrinsic feature of the proposal. Other tokenisation schemes can be used, as well.

iii. Each type of token is encoded as a binary string, which produces a sequence of 0s and 1s.

iv. This sequence is used to create a structure called PAT tree [116], which identifies a unique path to each token (a unique prefix).

v. The algorithm now counts the occurrence and the reference positions in the leaf nodes of the PAT tree. This helps know how many times a pattern is repeated and where it occurs.

vi. The patterns thus detected are now filtered by calculating three parameters: regularity, compactness, and coverage. These parameters are now compared with their respective thresholds.

vii. It uses a centre-star multiple string alignment algorithm [54] to generalise the detected patterns.

viii. These patterns are shown to the user in order to select the target pattern thay extracts the relevant information from the input web documents.

4.6 DeLa

Data extraction and Label assignment, or DeLa [162], was inspired by IEPAD [21]. It is based on the hypothesis that a web document contains multiple data records, that these data records are displayed together, and that they create a repetitive pattern inside the web document.

The DeLa algorithm works as follows:

i. The algorithm takes a web document that contains two or more data records as input.

ii. It builds the DOM tree of the input document and tokenises its content using TAG and TEXT tokens.

iii. The data region is located using a region extractor [161] that compares the DOM trees of resulting web documents and discards similar nodes since they are supposed to contain irrelevant information.

iv. The algorithm works on detecting repeating patterns using a structure that is calles suffix tree and is similar to the one used by IEPAD [21], called suffix tree. A suffix tree contains all possible paths to obtain each type of tokens, which are the leaves of this tree. It is used to detect repeated patterns in the input token sequence. These patterns are represented in an additional structure that is called pattern tree.

v. The patterns thus detected but one are removed from the original input sequence of tokens and the process is repeated again by creating a new suffix tree and adding detected patterns as children to the previously detected ones. The learning process ends when every repeating pattern is detected.

vi. Once the patterns are detected, string alignment is used to detect optional tokens and mark them as optional in the regular expressions.

4.7 Summary

In this chapter, we have presented the unsupervised proposals that are intended to learn an extraction rule that represents the template that was used at the server-side to generate the input web documents. The proposals in this field take a collection of input web documents and work on identifying shared patterns that are shared amongst them. These patterns are supposed to be part of the server-side template that generated them. The techniques used for this purpose range from multi-string alignment, tree similarity and an ad-hoc matrix alignment technique that is used to detect optional and repetitive patterns, and statistical techniques to differentiate the role of individual tokens.

Chapter 5

Heuristic-based information extraction

By three methods we may learn wisdom: First, by reflection, which is noblest; second, by imitation, which is easiest; and third by experience, which is the bitterest.

Confucius, Chinese teacher, editor, politician, and philosopher (551-479 BC)

Heuristic-based information extractors rely on a collection of predefined rules and heuristics that have proven to perform well in practice. In this chapter we survey the heuristic-based information extraction proposals in the literature. It is organised as follows: Section §5.1 introduces the heuristic-based information extractors; Sections §5.2–§5.8 describe the proposals in this field, namely: Álvarez and others' proposal, ViPER, WISH, DEPTA, NET, ViDE, and ListExtract; finally, Section §5.9 summarises this chapter.

5.1 Introduction

Heuristic-based proposals are very appealing from a practical point of view since they need not be configured to deal with a particular class of documents. Instead, they build on a number of heuristics that have proven to extract the information of interest in many common cases [6, 38, 72, 101, 103, 136, 169].

The existing proposals work on one input web document and search for repetitive patterns that hopefully identify the regions where the relevant information resides; implicitly, they assume that the input web documents contain similar information records since they all rely on finding repetitive patterns within a web document.

5.2 Álvarez and others' proposal

The proposal by Álvarez and others [6] is intended to extract data records and their attributes from an input web document. It builds on the hypothesis that a web document contains a unique data region, that the data records are usually under the same node in the DOM subtree that represents the data region, that these data records are similar, that each data record is composed of several sibling DOM subtrees, and that the same attributes in different data records have the same tag path.

Below we describe the algorithm by Álvarez and others:

i. It takes one web document that contains several data records as input and builds its DOM tree.

ii. For each text node in the DOM tree, it computes its tag path and uses it to group them. Each node in this tree is initially assigned a zero score that shall later be used to identify the data region.

iii. For each pair of nodes in each group of text nodes, the algorithm computes their deepest common ancestor, and increases the score of this common ancestor node in the DOM tree by one. The node that has the highest score, is considered as the root node of the main data region. To prevent large menus from being considered data regions, the authors use a simple heuristic: they require the data region to contain some of the key words in the user query that generated the input web document.

iv. The algorithm now works on breaking the data region into data records by converting each direct child node of the data region root node into a string and then calculating a similarity matrix between these strings using a string edit distance algorithm. This matrix is used to assign each child node to a cluster and a code to each cluster.

v. It then creates a sorted list that contains the codes of the clusters and searches for a combination that separates the data records. For this purpose, the algorithm creates multiple record candidates by partitioning the list into every possible sublist. Each sublist is called a candidate record list.

vi. For each candidate record list, the algorithm computes the similarity between each pair of nodes in the list and assigns an average similarity score to this list. The candidate record list that has the highest similarity average score is considered as the list of data records since data records are supposed to be similar.

vii. A string alignment algorithm is applied now to separate the attributes inside each data record. This is performed by converting data records subtrees into string sequences which are then aligned using a variation of the center star approximation algorithm for string alignment [54].

viii. If the first and the last data record in a candidate record list cannot be aligned to the others, then they are discarded since they are usually the header and the footer of the data region.

5.3 ViPER

Visual Perception-based Extraction of Records [136], or ViPER for short, is intended to extract data records and their attributes from a web document. It builds on the hypothesis that web documents contain one or more data regions, that data regions are centered in the web document, that they cover a large part of the web document, that each data region contains repetitive patterns, and that these repetitive patterns are the data records. The algorithm tries to improve on the MDR [100] algorithm for region extraction by considering repeated patterns, and visual information for separating data records and extracting their attributes.

Below we describe the ViPER algorithm:

i. It takes a web document that contains two or more data records as input. It build its DOM tree, renders it, and computes the coordinates of the rendering box for each node in the DOM tree. Then, scripts, style tags, and tag attributes are removed to create a so-called restricted tag tree, which is serialised into a so-called plain tag sequence structure.

ii. The algorithm now searches for so-called tandem repeats using a suffix tree; such repeats are repeated sequences of tokens that happen inside a plain tag sequence structure.

iii. It creates a similarity matrix in which the string representations of the nodes in the DOM tree are compared using a string edit distance algorithm. Tandem repeats are used in this step to reduce the costs of calculating the similarity matrix.

iv. The algorithm now analyses the similarity matrix to identify the sibling nodes that are similar. Each group of similar sibling nodes are considered as a data region candidate.

v. It now works on separating the data region candidates into data records by searching for the separators inside them. It uses the rendering information to sketch up a diagram based on the projection profiles of the rendering boxes. The valleys between peeks in this diagram are considered as potential data records separators.

vi. Once candidate data regions are identified and separated into data records, they are filtered to keep only the data regions. Candidate data regions are weighted based on their visual location, i.e., they should be centered and cover a large part of the web document. The key words in the query that generated the input web document can be also used to weight the regions.

vii. The algorithm aligns the data records to extract their attributes. The alignment algorithm is based on global sequence alignment using suffix trees, which is an alignment algorithm similar to the algorithms used in aligning protein sequences in bioinformatics [60].

5.4 WISH

Wrapper Incorporating Set of Heuristic Techniques [72], or WISH for short, is intended to extract data records and their attributes from a web document. It is based on the hypothesis that data records in the input web document are similar, follow a repetitive pattern, contain more than three HTML tags, that the web document contains a unique data region that is usually large, and that the data region contains at least three data records.

Below we describe the WISH algorithm:

i. It takes a web document that contains three or more data records as input, mends its HTML, builds its DOM tree, and tokenises the mended HTML code.

ii. A breadth-first search technique, which is an improved version of MDR [100], searches and groups the nodes in the DOM tree that have the same tag. Groups with at least two nodes are considered candidate data records.

iii. A filtering phase is performed to remove advertisements. This is performed by removing candidate data records that have less than three HTML tags.

iv. The algorithm checks the similarity between the candidate data records inside each group using a so-called dummy tree matching algorithm. This algorithm first counts the number of distinct tags inside each candidate data record subtree and keeps only those that have a similar number of distinct tags in each group. Then, the tags at each level inside the candidate data records subtrees are compared; only the subtrees that have a very similar number of distinct tags at each level are kept inside each group.

v. Groups that contain less than three data records candidates are removed since data regions are supposed to contain at least three data records.

vi. The algorithm filters the remaining groups by applying a scoring function that gives a high score to the groups whose candidate data records have images, a large number of tokens, and separating tags such as <br/> and <hr/>. The group that has the largest score is considered as the main data region in the web document and each candidate data record inside it is considered as a data record.

vii. It aligns the data records inside the data region by building a so-called template tree. This is performed by merging the data records subtrees into a unique tree in which dissimilar nodes are merged under a disjunction node; if they do not occur in every subtree, they are then put under an optional node.

viii. The template tree is used to extract the attributes of the data records. This is performed by putting the text nodes in the template tree that share the same parent node under the same column in a results table.

ix. Adjacent columns in the results table that are only separated by decorative tags are merged together; such tags include <b>, <i>, <font>. A column may be partitioned if every attribute in it shares a common sequence of tokens.

Although this proposal is considered heuristic-based, the authors reported on a simple idea that helps use the template tree as a rule that can be used to extract and align data records from other similar web documents. Unfortunately, this was not evaluated and it is not clear whether the rule learnt from a single document may achieve a good effectiveness.

5.5 DEPTA

Data Extraction based on Partial Tree Alignment [169], or DEPTA for short, is intended to extract data records and their attributes from a web document. It is based on the hypothesis that a web document contains one or more data regions, that each data region contains two or more data records, that the data records inside the same data region are similar, and that the same attributes in different data records share some common words that help aligning them.

The DEPTA algorithm works as follows:

i. It builds the DOM tree of the input web document and computes the rendering information of its nodes.

ii. An improved version of MDR [100], which is referred to as MDR-2, is applied to detect the data regions and the data records embedded inside them. MDR-2 analyses the rendering boxes of the nodes in the DOM tree and builds a so-called containment tree that contains the rendering boxes and the parent-child relations between them.

iii. The algorithm searches for the data regions in the web document by searching for so-called generalised nodes. A generalised node is a combination of adjacent nodes that have more than one level of children and that share the same parent node in the DOM tree. This is performed using an algorithm that searches for combinations of adjacent nodes in the DOM tree. Each collection of adjacent generalised nodes that have the same parent, the same size, and whose tree edit distance is less than a user-defined threshold are extracted as data region candidates.

iv. Visual information is used to remove candidate data regions that are rendered inside another candidate data region.

v. The algorithm now works on extracting the data records from each data region using the following heuristics: i) if the region consists of only one generalised node, it then checks if this node is not a table row, but all of its children are similar; if the condition is met, then the children are returned as independent data records; otherwise the generalised node itself is returned as a data record. ii) If the generalised node contains two or more nodes with the same number of children and these children are similar to each other, then it means that they are non-contiguous data records, i.e., the data region is an HTML table in which each data record is formatted in columns and not in rows; otherwise, the whole generalised node is returned.

vi. It aligns the extracted data records using a partial tree alignment algorithm. First, data records that are composed of more than one subtree are embedded under a ghost node that is used during the alignment process. The algorithm aligns the data records progressively by growing a seed tree. The data record that has the largest number of text nodes is considered as the initial seed tree, then, each node in the remaining trees that has no match in the seed tree is inserted into it. Nodes match when the text they contain is similar.

vii. The seed tree is used to create a table. For each leaf node, a column is created. Each row in the table represents a data record, and nodes that match belong to the same column. Each unmatched node in a data record occupies a single column.

5.6 NET

Nested data Extraction using Tree matching and visual cues [101], or NET for short, is intended to extract data records and their attributes from a web document. It is based on the hypothesis that a web document contains one or more data regions, that each data region contains two or more data records, that the data records are formatted using similar HTML tags, rendered contiguously, share the same parent node in the DOM tree, and that each data record usually contains more than one level of tags. NET was inspired by MDR [100] and DEPTA [169].

The NET algorithm works as follows:

i. It renders the input web document in a browser and creates a DOM tree that takes the rendering boxes of each HTML element into account, i.e., an element is a child of another element in the resulting DOM tree as long as the former is rendered inside the latter.

ii. It traverses the DOM tree using a bottom-up algorithm that compares every pair of nodes using a tree edit distance. It applies a dynamic programming algorithm that computes the tree edit distance between all of the subtrees in the web documents and also uses visual information to compute the similarity between the subtrees. Similar subtrees are considered as data records inside the same data region.

iii. Similar data records are aligned together to create an output table. The nodes that match are inserted into the same columns, whereas nodes that do not match are considered as optional attributes.

iv. The rendering boxes and their relationships are used to detect nested data records and to create their relations in the tables produced by the algorithm.

5.7 ViDE

Vision-based Data Extractor [103], ViDE for short, is intended to extract data records and their attributes from web documents. It is based on the hypothesis that visual features are important for information extraction, that the data region is centrally located on a web document, the size of the data region is usually large compared to the size of the input web document, that the data region contains several data records, these data records are adjacent, aligned to the left, have the same distance from the left, do not overlap, the separating space between them is the same, they are visually similar, use the same fonts, the first attribute in a data record is always mandatory, the presentation of the data attributes follows a fixed order, the attributes of same type in different data records have the same visual presentation, neighbour attributes inside the same data record usually have different fonts, and that the data records usually contain some static text.

The ViDE algorithm works as follows:

i. It builds the visual block tree of the input web document using VIPS [16]. The visual block tree represents a visual segmentation of the input web document in which each block in the tree corresponds to a rectangular region in the web document, i.e., the root block represents the whole web document and leaf blocks are blocks that represent the minimum semantic units that cannot be segmented again, e.g., images or text.

ii. ViDE relies on a component called ViDRE, which searches for the data region and partitions it into data records. ViDRE searches inside the visual block tree for the largest block that is centered horizontally, and extracts this block as the data region. Then, it removes the child blocks that are not aligned to the left since they are supposed to contain irrelevant information.

iii. ViDRE clusters the remaining blocks in the data region according to their visual similarity. The similarity function considers the size of the images in the block and the fonts of the links and text blocks.

iv. Each cluster is now supposed to contain the attributes of same type from every data record. They are now regrouped so that the blocks from each data record are now grouped together. The cluster that has the largest number of blocks is considered as the cluster that contains the first attribute in every data record and it is selected as a seed cluster. For each block in the seed cluster, the algorithm creates a group that contains the blocks from the other clusters that are located between this block and the next block in the seed cluster. Each group is considered as a data record.

v. The second component of ViDE is called ViDIE. It works on aligning the extracted data records to extract their attributes. A matching algorithm is applied: visual blocks inside the data records match together if they have the same fonts and have the same distance from the left (absolute position) or are located between two matching blocks (relative position). Blocks that match are put together in the same column in the results table, whereas optional attributes that do not occur in a data record has empty values in the results table.

Although this proposal is considered heuristic-based, the authors reported on a simple idea that helps use the blocks thus identified and their alignments to create an extraction rule that contains the position information to locate the data region in the visual block tree, and some visual information that helps separate the data records and align them. Unfortunately, this was not evaluated and it is not clear whether the rule learned from a single document may achieve a good effectiveness.

5.8 ListExtract

ListExtract [38] is intended to extract data records and their attributes from HTML lists. It builds on the hypothesis that every element in an HTML list is a data record, that some separators are usually used to separate attributes in each data record, that counting on a corpus helps aligning and extracting data records attributes, and that the same attributes for different data records are similar.

For each HTML list in the input web document, ListExtract works as follows:

i. It separates the HTML list and considers each item as an independent data record.

ii. It computes every possible text fragment for each data record and considers each fragment as an attribute candidate. A scoring function is used to score each attribute candidate. This function evaluates three features, namely: if the type of the attribute candidate usually occurs as an attribute type in the corpus, if the words in the attribute candidate usually occur together in the corpus (cohesive scores), and if the corpus contains an attribute that has the same text as the attribute candidate. The score is then normalised to interval $[0.00, 1.00]$

iii. The attribute candidates are now sorted according to their score in a list on which a greedy algorithm iterates. This algorithm selects the attribute candidate that has the highest score as a data attribute, removes it from the list, and removes the ones that overlap with it.

iv. Now that the data records are fragmented into data attributes, the algorithm works on aligning these attributes to create a table. It creates an initial table that contains k columns, where k is the number of attributes in the majority of data records.

v. The algorithm aligns the data records that contain more or less than k attributes into these columns. Data records that contain more than k attributes are fragmented again to force them to have k attributes by using a modified version of the previous fragmentation algorithm. Data records that contain less than k attributes are aligned by inserting $\mathtt{nil}$ values in some columns. Each attribute in this data record is compared to the information in each column in the initial table, and a matching score is assigned to it. It is inserted to the column that has the highest matching score. A `nil` value is inserted into a column if no attribute matches the information it contains.

vi. The last step refines the initial table by analysing the whole table. For each column in the table, it computes a consistency score and considers the columns with the lowest scores as inconsistent. Inconsistent columns are merged again, fragmented and aligned. The main difference between the algorithms in this step and the previous ones is that they also compare the attribute candidates to the columns in the results table. This step is repeated until every column in the table has a consistency score that is more than a user-defined threshold.

5.9 Summary

In this chapter we have surveyed the heuristic-based information extraction techniques in the literature. Without an exception, these proposals rely on building the DOM tree of the input web document and on a group of similar hypothesis, e.g., that the input web document contains several data records and that these data records are similar. Some of these techniques build only on visual features, others build only on structural features, and others build on both visual and structural features.

Chapter 6

Evaluating information extractors

The only relevant test of the validity of a hypothesis is comparison of prediction with experience.

Milton Friedman, American economist, statistician, and author (1912-2006)

Software engineers and researchers are usually interested in evaluating their new proposals and wish to compare their performance to third-party proposals. In this chapter, we report on how to evaluate information extraction proposals. It is organised as follows: Section §6.1 introduces the chapter and sketches a simple evaluation methodology; Section §6.2 presents a number of repositories to evaluate information extractors; Section §6.3 describes how to partition these repositories during the evaluation; Section §6.4 reports on the standard performance measures to evaluate a proposal; Section §6.5 presents a metrology to rank several techniques statistically; finally, Section §6.6 summarises this chapter.

6.1 Introduction

During the last years, many proposals on information extraction have been introduced. Researchers and software engineers are usually interested in evaluating their new proposals and wish to compare their performance to third-party proposals. (In the sequel, we use term performance to refer to both effectiveness and efficiency.) Performing the side-by-side comparison within a homogeneous evaluation environment is a key requirement for the comparisons to be useful.

The evaluation of information extraction proposals has become a necessity due to the large number of proposals in the literature. The first information extraction evaluation methodologies and repositories were made available at the MUC conferences [69]. However, information extraction has evolved and new methodologies and repositories are now used for this purpose [95].

The following steps sketch how a proposal should be evaluated:

i. The user should choose one or more repositories that contain annotated documents, i.e., documents in which the information to be extracted is made explicit by means of user annotations. Generally speaking, an annotation is a label that endows a piece of text with semantics. Examples of annotations include Book to mean that a piece of text is a book record, Title to mean that it is the title of a book or Subject to mean that it is the subject of a sentence. Note that it is important that public repositories be used, since otherwise other researchers would not be able to compare their results.

ii. If the proposal to be tested is rule-based, then the repository must be partitioned into two parts, namely: a training set that must be used to lean the extraction rules and a testing partition that must be used to evaluate the effectiveness and efficiency of the rules learnt. If the proposal is heuristic-based, then it is not necessary to partition the repository since these proposals do not learn any rules; the whole repository can be used for evaluation purposes.

iii. The proposal must now be executed on the training partitions, if any, and on the testing partitions. The information extracted and performance measures must be collected so that they can be analysed later.

iv. The information collected in the previous step is used to compute averaged performance measures from which intuitive conclusions can be drawn in many cases.

v. Rank the techniques using statistically sound procedures that confirm the conclusions drawn in the previous step.

The previous steps must be applied to every technique to be compared. Once we have gathered performance measures, we must use statistical inference to rank the proposals according to these measures.

6.2 Repositories

The MUC conferences were the first to provide repositories to evaluate free-text information extraction systems [69]. Later, more repositories were made available to the research community. In the following, we briefly describe some of the repositories used to evaluate information extraction proposals.

6.2.1 RISE

Repository of online Information Sources used in information Extraction tasks [117], or RISE for short, provides datasets used to evaluate machine learning techniques and information extraction proposals for both semi-structured and free-text documents. The motivation behind this repository was to provide researchers in the field of information extraction with a repository similar to the well-known UCI repository [160], which is a key reference for researchers in the field of data mining. RISE contains a collection of datasets from 10 web sites, each of which provides from 9 to 255 web documents. Half of these datasets contain free-text web documents, whereas the other half contains semi-structured web documents. This repository is not maintained since 2004 and one of the datasets has four different versions produced by different authors.

6.2.2 TBDW

Test Bed for information extraction from Deep Web [164], or TBDW for short, provides datasets to evaluate information extraction proposals from semi-structured web documents. TBDW contains a collection of datasets from 51 web sites and each dataset provides five documents. TBDW only includes the annotations of the first data records inside each web document. The annotations in TBDW are included in a separate document, but do not have an explicit model. This repository is not maintained since 2006.

6.2.3 TIPSTER

The TIPSTER project [156] was sponsored by the Software and Intelligent Systems Technology Office of the Advanced Research Projects Agency in an effort to advance information extraction from toy to large real-world web documents. The project includes a repository of web documents for evaluation purposes [64]. The documents provided by the repository range from documents from the Wall Street Journal to the USA Federal Register. The annotations in TIPSTER are formatted using SGML-like tags in separated documents. This repository is not maintained since 1993.

6.2.4 The Pascal Challenge

This repository was proposed by Ireson and others [77] in The Pascal Challenge on the Evaluation of Machine Learning for Information Extraction [77]. This repository contains a unique dataset that is composed of a total of 1 100 documents on call for papers, 850 of which are workshop call for papers and 250 of which are conference call for papers. The majority of the call for papers are related to Computer Science. The documents are divided into three parts, namely: a training dataset that contains 400 documents, a test dataset that contains 200 documents, and an enrich dataset that contains 500 documents. The annotations are embedded inside the documents. This repository is not maintained since 2008.

6.2.5 Ad-hoc repositories

Some authors have produced repositories of their own, e.g., Álvarez and others [6], Arasu and Garcia-Molina [7], Crescenzi and Mecca [31], Krishnamurthy and others [88], Suchanek and others [154]. Lavelli and others [95] identified some problems in these repositories, namely: errors in the annotated data, problems in creating versions of these repositories, the annotations that could be embedded or separated, and the heterogeneous format of the annotations. Furthermore, these datasets are usually unavailable and not maintained [142].

6.3 Partitioning repositories

When evaluating rule-based information extractors on a given repository, it is necessary to partition the repository into two parts, namely: training and testing partitions. This can be accomplished as follows:

N **repeated random partitions:** This technique partitions the dataset randomly into training and testing partitions, uses the training partition to learn the extraction rules and test these rules on the testing partition. This procedure is repeated N times using different randomly selected train/test partitions. Each time, performance measures are collected and at the end the weighted arithmetic mean of each computed value is returned. Note that the results computed by this technique are usually different each time this technique is run.

k**-fold cross validation:** The k-fold cross validation technique partitions the dataset into k subsets of web documents with their corresponding annotations and then starts iterating over them. At each iteration, it considers one of these partitions for testing, whereas the remaining partitions are considered as a unique set which is used to learn rules. The rules learnt at each iteration are tested on the selected testing partition, and performance measures are collected. At the end, the weighted arithmetic mean of each computed value is returned. Note that this technique produces the same results on different runs as long as the input documents are ordered according to some criterion, e.g., URL or download time.

Note that partitioning repositories does not make sense in the case of heuristic-based information extraction proposals since they do not have to learn extraction rules. The entire dataset is considered as a test partition in this case.

6.4 Collecting performance measures

The performance measures that are usually reported by the information extraction proposals can be classified into two types, namely: effectiveness measures and efficiency measures. In this section we study them both.

6.4.1 Effectiveness measures

Effectiveness measures aim to characterise how well a proposal works in terms of its ability to extract relevant information and annotate it as expected and/or not to extract irrelevant information. These measures come from the field of classification, where the problem is to classify data instances into one or more classes. In information extraction, the classes are the types of information to be extracted from a document, e.g., attributes like Title, Author, or Price or records like Book or Offer. In other words, we may see an

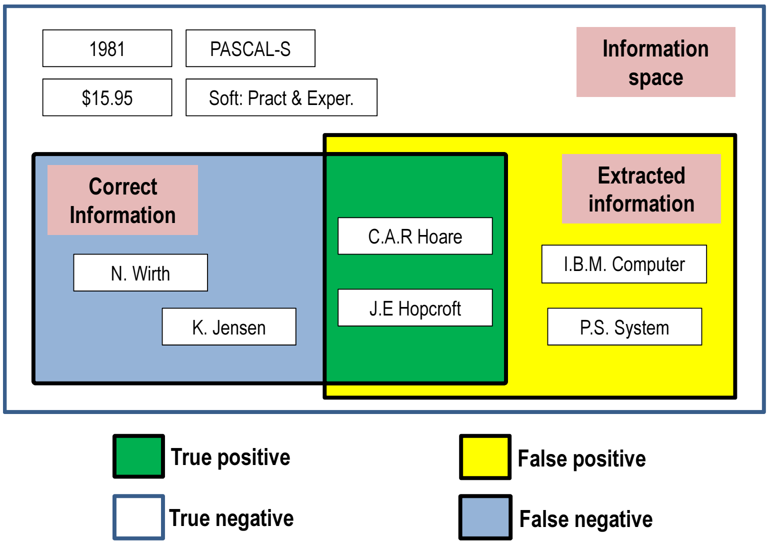


Figure 6.1: *Sample correct versus extracted information.*

information extractor as a text classifier that puts every piece of text in the input document it analyses in a user-defined class; the information that is not extracted is usually classified in a pre-defined class to which we refer to as NA (Not assigned).

Recall that prior to evaluating an information extractor we need a testing set, which is a collection of documents in which the information to be extracted is annotated. We refer to this information as the correct information to emphasise that this is the information that is expected to be extracted. Unfortunately, information extractors are rarely perfect since they make mistakes that are referred to as false positives and false negatives, cf. Figure §6.1. Given a class, the false positives are the pieces of information that an information extractor returns as belonging to that class but are actually of another class; similarly, the false negatives are the pieces of text that an information extractor returns as belonging to another class. For instance, if an information extractor returns "John E. Hopcroft" as a piece of information of class Title, then it is very likely to be wrong since this is quite likely a piece of information of class Author; that is, this information is a false positive for class Title and a false negative for class Author. For the sake of completeness, the pieces of information that an information extractor returns as belonging to the correct class are called true positives and the rest are called true negatives.

Building on the previous concepts, it is common to define the following effectiveness measures regarding a class:

Precision, aka positive predictive value: This measure refers the ratio of true positives of a class to the total amount of information returned by an information extractor as belonging to that class. Intuitively, the higher the precision, the less incorrect information is extracted as belonging to a given class. This measure is formally defined as follows:

$$P = \frac{tp}{tp + fp}$$

Recall, aka sensitivity or true positive rate: This measure refers to the ratio of true positives of a class to the total amount of information that actually belongs to that class. Intuitively, the higher the recall, the more correct information is extracted as belonging to a class. This measure is formally defined as follows:

$$R = \frac{tp}{tp + fn}$$

F_β **measure:** Note that neither a high precision or recall indicates that an information extractor is good. For instance, an information extractor that achieves perfect precision might be the worst information extractor in the world, e.g., an information extractor that does not extract any information at all has perfect precision since it does not make any mistakes; similarly, an extractor that achieves perfect recall might not be useful at all, e.g., an information extractor that returns every piece of information as belonging to a given class has perfect recall with respect to that class. The F_β measure combines both precision and recall in a β-harmonic mean that is close to 1.00 when both precision and recall are high and close to 0.00 when any of them is not good enough. Usually, β is set to 1, which results in the standard harmonic mean of precision and recall. The F_1 measure is indistinctly referred to as F_1 or $F1$. This measure is formally defined as follows:

$$F_\beta = (1 + \beta^2)\frac{P\,R}{(\beta^2\,P) + R}$$

Note that previous measures are defined at the class level. To compute these measures at the information extractor level we must compute the so-called weighted average for each measure, i.e., the sum of the product of each measure times its number of occurrences divided by the total number of occurrences. Assume that we are working with n classes that have

m_i $(1 \leq i \leq n)$ correct instances each, and that P_i and R_i denote, respectively, the precision and recall of an information extractor on those classes. We then define the information-extractor-level precision and recall as follows:

$$P = \frac{\sum_{i=1}^{n} m_i P_i}{\sum_{i=1}^{n} m_i}, R = \frac{\sum_{i=1}^{n} m_i R_i}{\sum_{i=1}^{n} m_i}$$

The information-extractor-level F_β can be computed as usual, using the β-harmonic mean of the information-extractor-level precision and recall.

In the previous paragraphs, we have assumed that the classes that an information extractor returns are meaningful. This is true in the case of supervised proposals since they are trained to extract information of a number of user-defined classes; in the case of unsupervised proposals, the classes returned by the information extractor are not meaningful since they are computer-generated. Typical classes returned by an unsupervised proposal are A, _B_, $C, or D007; it is the responsibility of the user to assign a meaning to them. This makes computing the effectiveness measures of an unsupervised proposal a little more difficult since prior to computing them, we need map each computer-generated class onto a class in the testing set. A simple solution to this problem is to compute the measures on every possible mapping and select the ones with higher F_1 measure.

6.4.2 Efficiency measures

Efficiency measures characterise how well an information extractor works in terms of how much computing resources it requires. Typical efficiency measures include the following:

Heap memory: This measure refers to the amount of heap memory that an algorithm consumes when it is executed. The heap memory is used to store the input, intermediate results, and the output during the execution.

Time: This measure refers to the amount of time an algorithm requires to execute. It is common to distinguish between CPU time, which is the actual time the CPU is allocated to run the algorithm, IO time, which is the time the IO devices are allocated to reading or writing data that belongs to the algorithm, and total time, or simply time, which is the total amount of time that elapses since the algorithm starts running until it finishes (this includes the CPU time, the IO time and the time the algorithm waits for the CPU or the IO devices to be allocated). CPU and IO times are quite stable, i.e., when an algorithm is repeatedly executed on the same input they do not vary largely; contrarily, total times are not so stable because they depend on many other processes that can run concurrently on the same machine. As a conclusion, to measure accurate total times it is a good idea to repeat the experiments a sufficiently large number of times, typically 25 times and to average the results after discarding outliers using, for instance, the well-known Cantelly inequality or other more sophisticated methods [70].

Regarding information extraction, heap memory and timings may be computed regarding a proposal to learn extraction rules or regarding an information extractor (being based on rules or built-in heuristics). Although they provide valuable evaluation information, this information is related to a particular implementation run on a particular computing system. This is the reason why it is also worth analysing the theoretical memory or time complexity of a proposal. Memory complexity refers to the theoretical minimum upper limit to the heap memory an algorithm consumes in terms of the size of the input; similarly, the time complexity refers to the theoretical minimum upper limit to the number of elementary operations an algorithm performs in terms of the size of the input. Some algorithms are far too difficult to analyse since their complexity depends on too many variables or variables that are very difficult to characterise. In such cases, it generally suffices to compute an upper bound that proves that the algorithm is computationally tractable, i.e., is not exponential or worse in the size of the input, neither regarding heap memory nor time.

6.5 Ranking proposals

Intuitively, the techniques we have evaluated can be ranked according to the average values of the performance measures we have gathered before. For instance, assume that the CPU learning times of technique A on a given training set are 4.50, 4.60, 4.80, 4.30, 4.10, and 2.70 seconds, i.e., the mean is 4.17 seconds and the standard deviation is 0.69 seconds; assume now that the CPU learning times of technique B on the same training set are 4.30, 4.20, 4.40, 4.50, 4.30, and 4.10 seconds, i.e., the mean is 4.31 seconds and the standard deviation is 0.12 seconds. As a conclusion, technique A seems to perform better than technique B in the selected training dataset, but the question we really need address when performing an experimental evaluation is whether the difference is actually significant from a statistical point of view. In other words, we need discern if the difference in performance is actually a key difference between these techniques or if they are just a consequence of the randomness factors that underlie the evaluation: the selection of the dataset, how it was partitioned into a training set and a test set, the documents that were selected, and so on. In our previous example, note that the performance of technique A seems worse than the performance of technique B in general; it is the last experiment that makes its average learning time better than the average learning time of the other technique. Our goal is to discern if this difference is significant enough or not from a statistical point of view.

Fortunately, the research on statistics has produced a number of statistical tests that help us rank a number of techniques with regard to a performance variable or a ranking [53, 135]. The tests to be applied depend on whether the experimental data are distributed normally and have equal variances or not; the former are usually referred to as parametric tests and the latter are known as non-parametric tests. Since our experience is that we are very unlikely to gather normally-distributed experimental data and even less likely that two performance variables have the same variance, we restrict our attention to non-parametric tests.

Whatever particular test we use, the goal is to contrast two hypotheses, namely: the null hypothesis, which states that the techniques evaluated behave similarly, and the alternative hypothesis, which states that they do not behave similarly. They rely on computing a so-called statistic, which is a formula that operates on the experimental data and the evaluation data and transforms them into a real value; the distribution of the statistic is obviously known beforehand. The idea is that if the probability of the value of the statistic is high, then the experimental data does not provide enough evidence to reject the null hypothesis; we then have to admit that the differences in performance we have observed in our evaluation are not significant from a statistical point of view. Contrarily, if the value of the probability of the statistic is very low, then we have to admit that the differences in performance are significant, in which case, it proceeds to rank the techniques. The probability value of the statistic is usually referred to as the p-value and to determine if it is high or low, the literature proposes to compare it to a so-called significance level, which is usually denoted as α and set to 0.05.

The ranking of several proposals is usually performed in three steps: i) we compute the average rank of each technique from the evaluation data; ii) we perform a bulk test to determine if the differences in ranks are significant or not; iii) if the differences are significant, we then have to perform a post-hoc test to find out which ranks differ significantly; otherwise, the differences observed are not significant, and we cannot conclude that a technique ranks better than the others. To perform the bulk test, Iman and Davenport [76] developed a proposal that is considered the state of the art in this field [37]. The array of choices is more ample regarding post-hoc tests. According to Derrac and others [37], the most powerful proposal was devised by Bergmann and Hommel [13]. Unfortunately, it is computationally intractable, which does not make it appropriate to compare more than nine techniques; if more techniques need be compared, then we can use the proposal by Shaffer [133].

Once the proposals have been ranked, we might need determine how some input features correlated to their results. This usually helps understand what features of the input have an impact on performance. The most common non-parametric procedure is Kendall's Tau [82].

6.6 Summary

In this chapter, we have reported on the evaluation of information extraction proposals. First, we presented a simple evaluation methodology: we then studied the existing repositories that are used for this purpose; next, we reported on the standard measures that are usually used to evaluate information extraction proposals; finally, we reported on how to rank a number of proposals building on experimental data and statistical procedures.

Part III

Our Proposal

Chapter 7

Devising information extractors with CEDAR

One cannot step twice in the same river.

Heraclitus, Greek philosopher (535-475BC)

In this chapter, we propose a reference architecture to build information extractors for semi-structured web documents and describe its accompanying software framework. It is organised as follows: Section §7.1 introduces this chapter; In Section §7.2, we describe the logical view of the reference architecture; In Section §7.3 we describe the development view; Section §7.4 describes the scenarios view; Section §7.5 reports on our repository of datasets; Section §7.6 reports on the experiments we have conducted to validate our reference architecture; finally, Section §7.7 concludes the chapter.

7.1 Introduction

Several articles in the literature have highlighted the lack of a reference architecture to help software engineers develop extraction rule learners from semi-structured web documents [25, 95, 107]. This is problematic insofar researchers need implement their proposals from scratch in order to validate them, i.e., they need pay attention to a variety of details that are ancillary and common to many other proposals, but do not constitute the core of their research [41]. The lack of a reference architecture has also led to a variety of terminologies, which makes communication amongst software engineers difficult, and experimental results that are not comparable empirically due to differences in the designs and the implementations.

We use the $4 + 1$ architectural view model proposed by Kruchten [89] to describe our reference architecture. Note that since our proposal is not intended to be a functioning system, the process and the physical views, which focus on non-functional requirements like concurrency, distribution, topology or communication, do not actually make sense in this case.

7.2 Logical view

The logical view of an architecture represents the functional requirements a system must provide to its end users. In our case, the end user is a software engineer who aims to devise a new proposal to learn extraction rules, so we describe the subsystems, the services they provide, and the interactions amongst them in this view.

The architecture is divided into the following subsystems, whose relationships are shown in Figure §7.1:

Annotation tool: The reference architecture relies on an annotation tool with which users can download and annotate web documents according to an OWL ontology in which he or she describes classes, properties, and their relationships. Ontology classes are used to represent records of information, object properties represent nested records, and data properties represent attributes.

Dataset: This subsystem provides services that allow end users to work with annotations and persist them. During the annotation process, this subsystem allows users to instantiate ontology classes and properties in

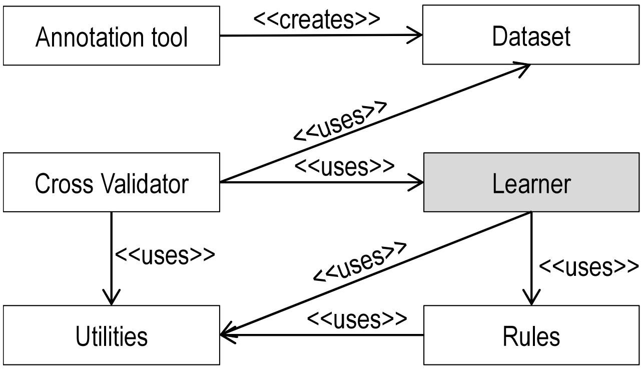


Figure 7.1: *Relationships amongst the subsystems of our architecture.*

addition to their position in the corresponding web document. During the learning process, end users can use a dataset to work with a text view or a tree view of the documents they have annotated, get the annotations sorted according to their position or to their type, obtain separating texts between annotations or work with DOM trees and annotation nodes. During the extraction process, this subsystem allows end users to persist the information that is extracted to OWL documents.

Learner: This subsystem provides end users with services to develop rule learners. For example, there is a service to create the skeleton of a transducer for a given dataset, i.e., its states and transitions, but not the transition conditions. It saves end users from the burden of inferring the structure of a transducer from the annotations in a dataset, since this is common to every learning algorithm. Note that this subsystem is a point of variability where software engineers only have to focus on devising their own learning algorithms to learn extraction rules.

Rules: This subsystem provides a service to construct extraction rules and to execute them on web documents in order to extract information.

Cross Validator: This subsystem provides a tool with which end users can k-cross validate their rule learners. It helps collect precision, recall, specificity, accuracy, and the F1 measure. Thanks to this tool, the results about a given proposal are empirically comparable to other proposals.

Utilities: This subsystem offers some utilities to the rest of subsystems,

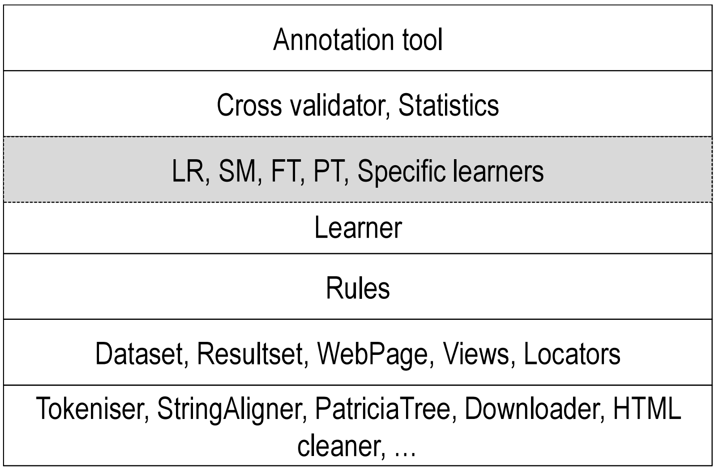

Annotation tool
Cross validator, Statistics
LR, SM, FT, PT, Specific learners
Learner
Rules
Dataset, Resultset, WebPage, Views, Locators
Tokeniser, StringAligner, PatriciaTree, Downloader, HTML cleaner, ...

Figure 7.2: *Layers of our reference architecture.*

namely: a configurable tokeniser, a web documents downloader, preprocessors such as an HTML cleaner, and a few string and tree alignment algorithms.

7.3 Development view

This view shows the system from a developer's perspective by illustrating the component organisation of the system and the class diagrams of each component, which are the basis for assigning work packages to the members of a development team. Our reference architecture is composed of several layers each of which has a well-defined responsibility and provides services to the layers above it. These layers are illustrated in Figure §7.2 and are explained in the following paragraphs.

The annotation tool layer: This is the upper layer in our reference architecture. It provides a tool that end users can use to create Datasets. It uses the lower layers to download, clean HTML, add annotations, and to save Datasets. Ontologies are used to create the annotations of each web documents.

The cross validator layer: Cross validation is used to estimate the performance of a system in practice and to obtain comparable results on a Dataset.

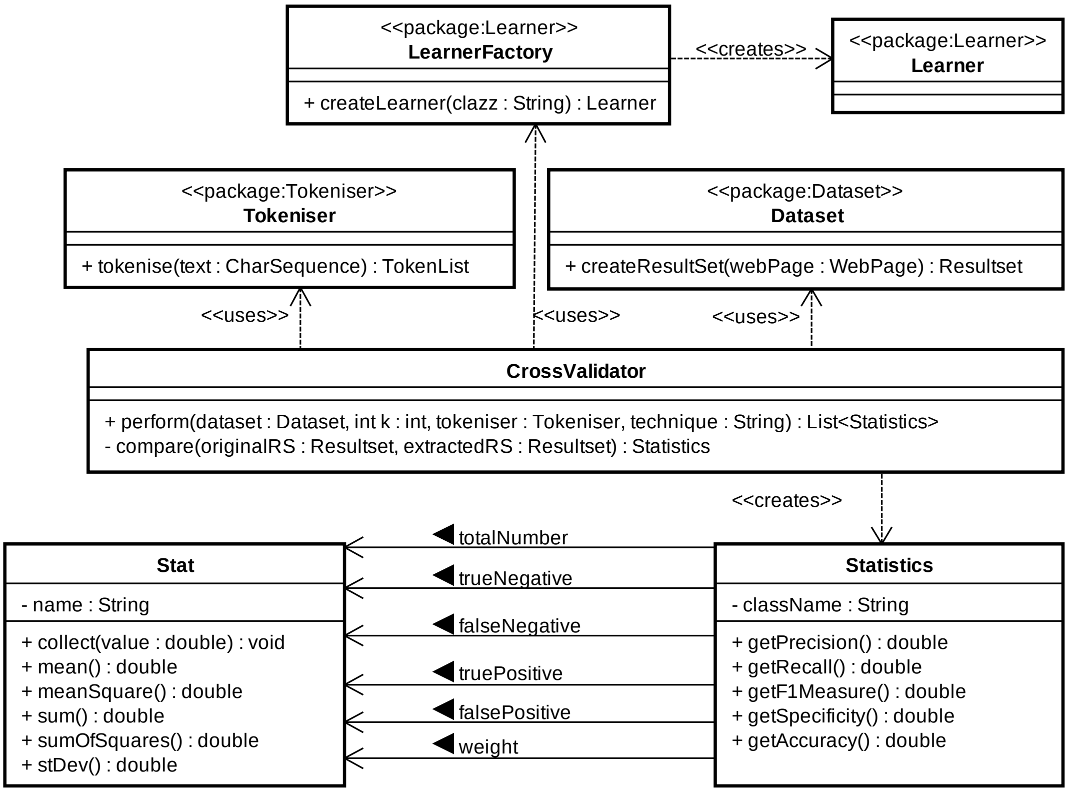

<<package:Learner>>
LearnerFactory
+ createLearner(clazz : String) : Learner
<<creates>>
<<package:Learner>>
Learner
<<package:Tokeniser>>
Tokeniser
+ tokenise(text : CharSequence) : TokenList
<<package:Dataset>>
Dataset
+ createResultSet(webPage : WebPage) : Resultset
<<uses>>
<<uses>>
<<uses>>
CrossValidator
+ perform(dataset : Dataset, int k : int, tokeniser : Tokeniser, technique : String) : List<Statistics>
- compare(originalRS : Resultset, extractedRS : Resultset) : Statistics
<<creates>>
Stat
- name : String
+ collect(value : double) : void
+ mean() : double
+ meanSquare() : double
+ sum() : double
+ sumOfSquares() : double
+ stDev() : double
totalNumber
trueNegative
falseNegative
truePositive
falsePositive
weight
Statistics
- className : String
+ getPrecision() : double
+ getRecall() : double
+ getF1Measure() : double
+ getSpecificity() : double
+ getAccuracy() : double

Figure 7.3: *Class diagram of the cross validation layer.*

Given a Dataset obtained from a web site, the k-fold cross validation technique partitions the Dataset into k subsets of web documents with their corresponding Resultsets and then starts iterating over them. At each iteration, it considers one of these subsets for testing, and the remaining subsets are considered as a unique set which is used to learn rules. The rules learnt at each iteration are tested on the selected subset, and some statistics are collected. At the end, the weighted arithmetic means are returned.

The fifth layer contains the classes of the CrossValidator and the classes to collect results during a k-cross validation, cf. Figure §7.3. These classes are the following:

CrossValidator: This class provides methods to perform a k-cross validation on a given Dataset for a specific Learner. It uses a LearnerFactory to create the Learner to be tested, and during the cross validation process, it compares the Resultsets obtained with the annotated ones to collect the previous Statistics.

Statistics: This class provides methods to collect the following statistics: true positives, false positives, true negatives, false negatives, the total number of annotations at each iteration and the weight of each class and property in the Dataset. The methods in this class to compute effectiveness and efficiency measures use these attributes.

Stat: A statistics class in which measures can be collected. It provides methods to compute statistical measures such as arithmetic mean and standard deviation.

The learners layer: The key of this layer is that it is open, i.e., is intended to provide an extension point that software engineers can use to create their own rule learners, cf. the gray band in Figure §7.2. In our accompanying framework we have implemented a number of extensions, c.f. §7.4. The classes in this layer are the following:

LearnerFactory: This class is intended to provide a method to create a Learner of a given type.

Learner: An interface that should be implemented by the rule learners devised by users.

SkeletonCreator, LearnerInformation, and TransducerRulesLearner: These are classes that model the rule learners that learn transducers. The SkeletonCreator allows to create the skeleton of a transducer, LearnerInformation models information related to each state and to each transition in the transducer, and the TransducerRulesLearner is intended to learn the transition condition using a specific learning technique.

SM, LR, FT, and PT: These classes model specific learners that learn transition conditions using four techniques that were inspired by SoftMealy [73], WIEN [92], FivaTech [81], and IEPAD [21], respectively.

The rules layer: The third layer provides the classes required to implement extraction rules. In our accompanying framework we have implemented rules as transducers, cf. Figure §7.5. The classes in this layer are the following:

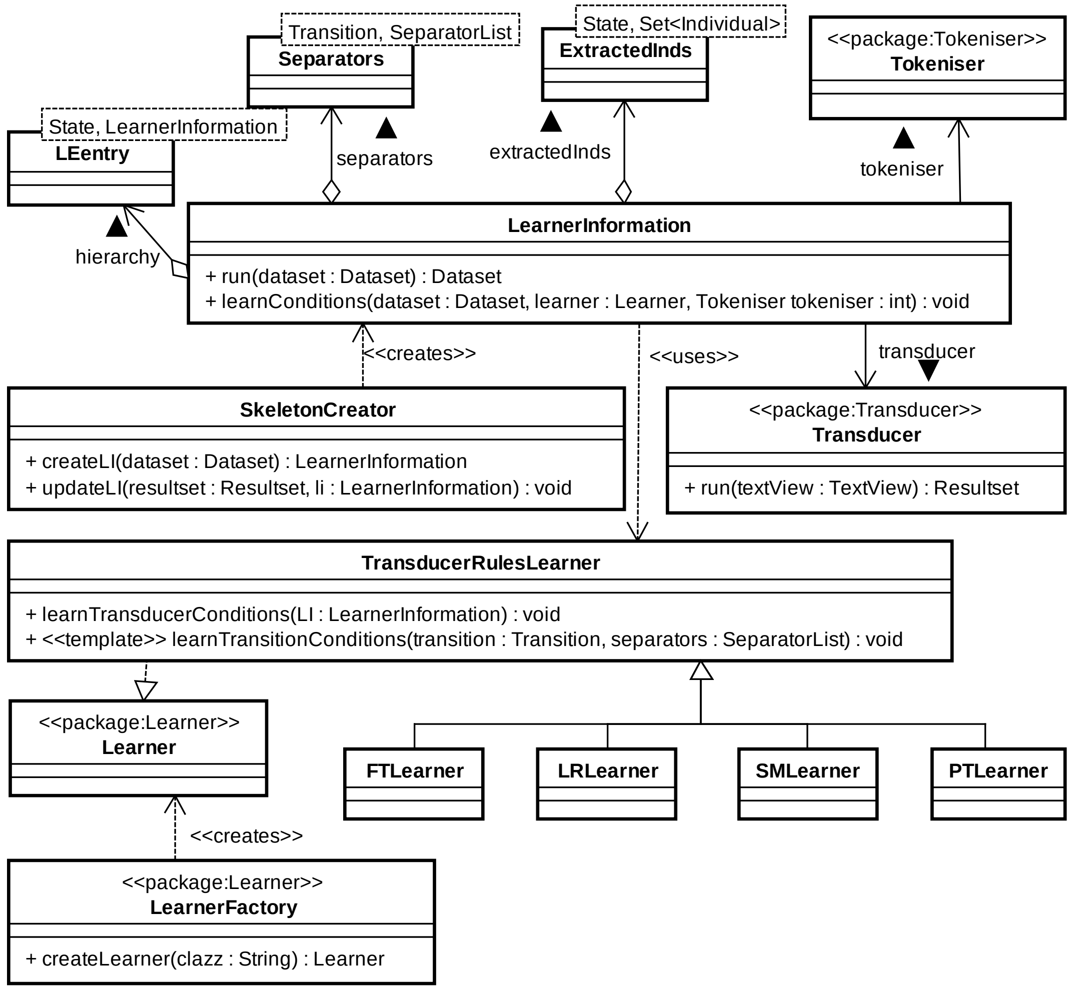

Transition, SeparatorList
Separators
State, Set<Individual>
ExtractedInds
<<package:Tokeniser>>
Tokeniser
State, LearnerInformation
LEentry
separators
extractedInds
tokeniser
hierarchy
LearnerInformation
+ run(dataset : Dataset) : Dataset
+ learnConditions(dataset : Dataset, learner : Learner, Tokeniser tokeniser : int) : void
<<creates>>
<<uses>>
transducer
SkeletonCreator
+ createLI(dataset : Dataset) : LearnerInformation
+ updateLI(resultset : Resultset, li : LearnerInformation) : void
<<package:Transducer>>
Transducer
+ run(textView : TextView) : Resultset
TransducerRulesLearner
+ learnTransducerConditions(LI : LearnerInformation) : void
+ <<template>> learnTransitionConditions(transition : Transition, separators : SeparatorList) : void
<<package:Learner>>
Learner
FTLearner
LRLearner
SMLearner
PTLearner
<<creates>>
<<package:Learner>>
LearnerFactory
+ createLearner(clazz : String) : Learner

Figure 7.4: *Class diagram of the learners layer.*

Rules: This is an interface that should be implemented by the different types of extraction rules learnt by the Learners in the upper layer.

Transducer: This class models a type of rules some information extraction techniques learn. It contains a collection of States, which represent data to be extracted, and Transitions amongst them whose conditions are modeled as regular expressions.

The dataset layer: The second layer contains the dataset classes that help model user annotations and extracted data. The classes in this layer are

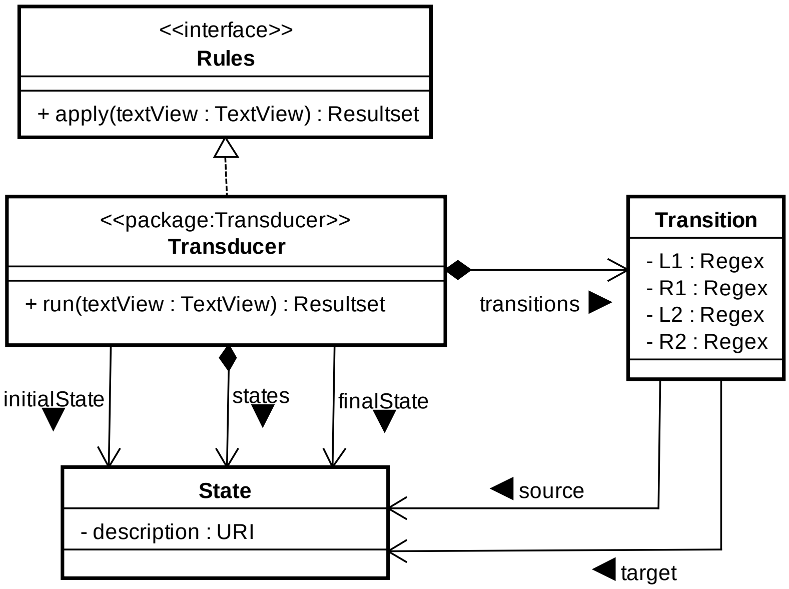


Figure 7.5: *Class diagram of the rules layer.*

shown in Figure §7.6, namely:

Dataset: It models a collection of Resultsets and web documents. They are actually a map from a set of web documents onto their corresponding Resultsets.

Resultset: This class models the annotations on a web documents and allows to save them as an instance of an ontology.

WebPage: This class is used to represent a web document. It keeps a reference to where it is stored in a local cache (cachedURI) and its original location (uri).

TextLocator and TreeLocator: These classes are used to provide locations to the annotations. Each instance of a class and property in the Resultset has two locators. The TextLocator provides information about the offset and length of an annotation, whereas the TreeLocator provides the Path of the annotation in the DOM Tree. Both locators allow to retrieve the text in each annotation.

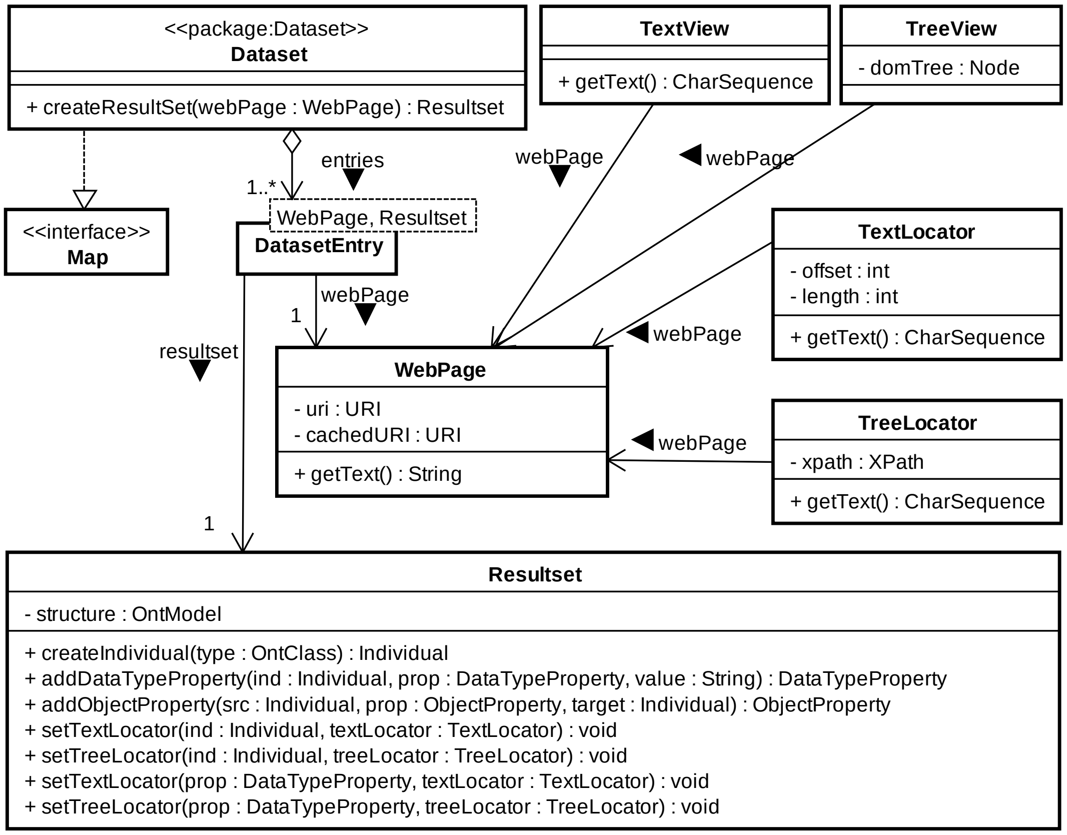

<<package:Dataset>>
Dataset
+ createResultSet(webPage : WebPage) : Resultset
TextView
+ getText() : CharSequence
TreeView
- domTree : Node
<<interface>>
Map
entries
1..*
WebPage, Resultset
DatasetEntry
webPage
webPage
webPage
1
resultset
TextLocator
- offset : int
- length : int
+ getText() : CharSequence
webPage
WebPage
- uri : URI
- cachedURI : URI
+ getText() : String
webPage
TreeLocator
- xpath : XPath
+ getText() : CharSequence
1
Resultset
- structure : OntModel
+ createIndividual(type : OntClass) : Individual
+ addDataTypeProperty(ind : Individual, prop : DataTypeProperty, value : String) : DataTypeProperty
+ addObjectProperty(src : Individual, prop : ObjectProperty, target : Individual) : ObjectProperty
+ setTextLocator(ind : Individual, textLocator : TextLocator) : void
+ setTreeLocator(ind : Individual, treeLocator : TreeLocator) : void
+ setTextLocator(prop : DataTypeProperty, textLocator : TextLocator) : void
+ setTreeLocator(prop : DataTypeProperty, treeLocator : TreeLocator) : void

Figure 7.6: *Class diagram of the dataset layer.*

TextView and TreeView: These classes model the views used in the learning process. The TextView provides a view over the text contents of a WebPage and the TreeView provides a view over the DOM tree.

DatasetPersistence: This is a class used to save and load Datasets.

The utility layer: This layer provides a number of very reusable classes that are used by the upper layers, namely:

Tokeniser: A class to implement a configurable tokeniser, cf. Figure §7.7. The tokeniser is configured by means of an XML document that defines a hierarchy of token classes and their definitions as regular expressions (class TokenClass). When a piece of text is tokenised, the Tokeniser returns a TokenList, which is a Map of Tokens sorted by their offset in the

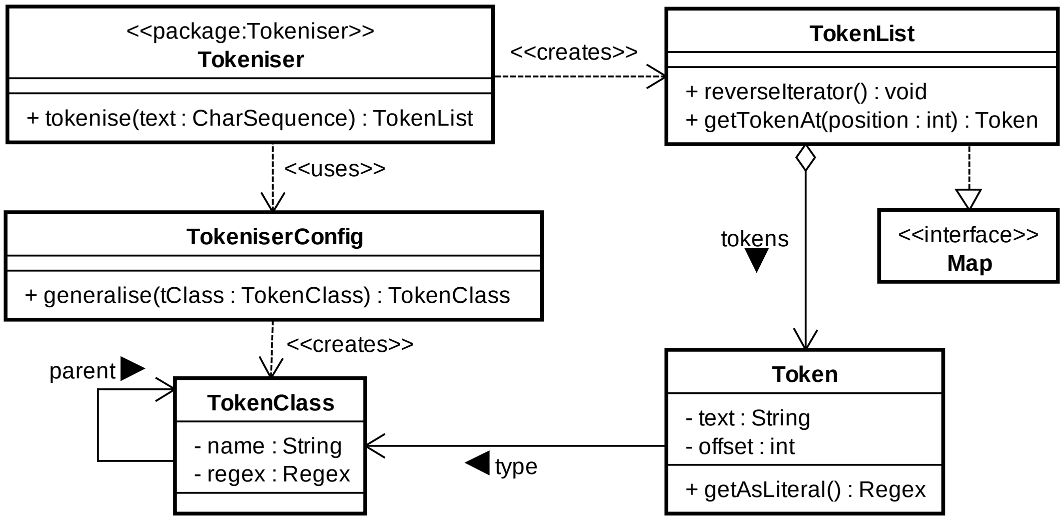


Figure 7.7: *Class diagram of the tokeniser utility.*

tokenised text. The methods generalise in TokeniserConfig and getAsLiteral in class Token allow to generalise and specialise tokens, respectively.

StringAligner: This class provides an implementation of a multiple string alignment algorithm that is similar to the one proposed by Kayed and Chang [81].

PatriciaTree: This class constructs a PatriciaTree starting from a set of token sequences. It provides methods to update them by adding new token sequences and to build the regular expression that corresponds to a tree.

Downloader: This class downloads web documents locally. To ensure the reproducibility of tests, annotated web documents are downloaded and saved locally to avoid that changes in web sites have an impact on user annotations and tests.

Pre-processors: They are usually used before rule learning or information extraction. HTMLCleaner can be used to fix the HTML code of downloaded web documents.

7.4 Scenarios view

According to Kruchten [89], this view shows how an architecture is instantiated in typical use cases. It serves two purposes: as an illustration of how the architecture can be used and as a validation since the scenarios are supposed to be an abstraction of the most important requirements. Below, we describe four non-trivial scenarios.

Developing a new learning technique: This scenario aims to show how a software engineer can devise his or her own technique to learn the extraction rules of type transducers. The steps he or she should perform are as follows:

i. The user imports the rule learning framework.

ii. The user creates a class that extends class TransducerRuleLearner and implements interface Learner.

iii. A Tokeniser should be created by defining an XML document with the tokenisation hierarchy that shall be used by the new technique.

iv. The user defines the template method learnTransitionConditions in the new class created. This method includes the code necessary to learn the transition conditions for a transducer.

Testing a new learning technique: This scenario aims to show how a software engineer can test his or her own technique and obtain comparable results by using a 10-fold cross validation. The steps to validate a learning technique are defined below.

i. The user downloads our testing Datasets.

ii. An instance of class CrossValidator is created and the perform method is invoked. This method receives $k = 10$, the Tokeniser and the name of the learning technique that shall be tested.

iii. The CrossValidator performs a 10-fold cross validation and saves the results to a Statistics object. These Statistics are returned by the CrossValidator and allow to compute measures such as precision and recall for each class and property in the ontologies used.

Learning extraction rules: This scenario describes how a user can learn extraction rules for a given web site and save them for future use. The steps to learn these rules, represented as transducers in this case, are the following:

i. The user annotates a Dataset using web documents from the web site for which he or she wishes to learn extraction rules. Annotations must be handcrafted using the annotation tool provided with our framework.

ii. The user should create a SkeletonLearner and call method create with the Dataset as a parameter. It returns a LearnerInformation object.

iii. The user creates an object of the appropriate class to implement a learner.

iv. The user can now learn the transition conditions by calling method LearnConditions with the Dataset as input.

Applying rules on an input web documents: This scenario describes how users can apply extraction rules to web documents to extract information from them. The user should perform the following steps:

i. The user loads a Transducer using the TransducerPersistence class.

ii. The user should create a Resultset for the web document of interest. The information extracted shall be saved there.

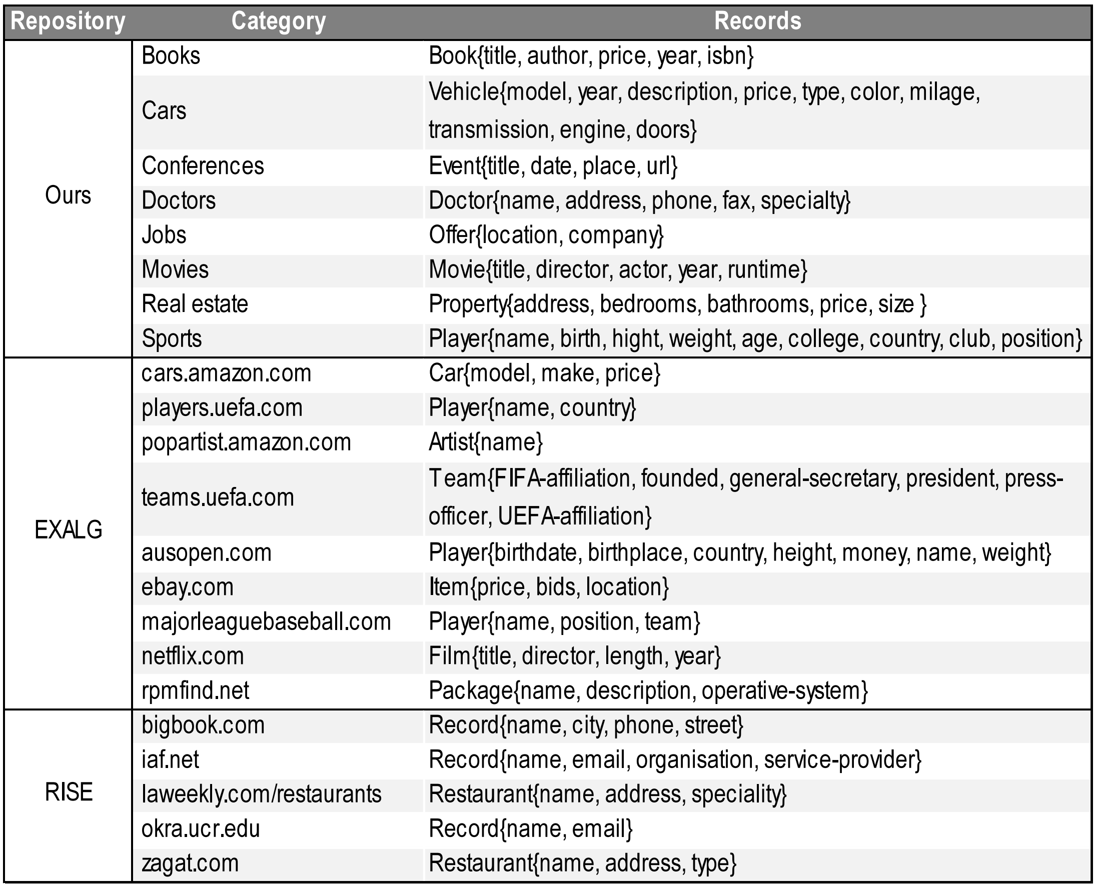
iii. The user creates a TextView over the input web document and calls method apply of the transducer with the TextView, a zero to indicate the starting offset, and the Resultset as parameters.

iv. The transducer runs on this TextView and saves the information extracted into the Resultset. A Dataset with the input WebPage and the final Resultset is created.

v. DatasetPersistence should be used now to save the resulting Dataset.

7.5 Our repository

Our datasets consist of a collection of 55 datasets that contain a total of 2 084 web documents that can be classified into two groups: the first group is composed of datasets gathered from 41 real-world web sites and the second group is composed of 14 datasets downloaded from two public repositories. The first group contains datasets on books, cars, conferences, doctors, jobs, movies, real estates, and sports. These categories were randomly sampled from The Open Directory sub-categories, and the web sites inside each

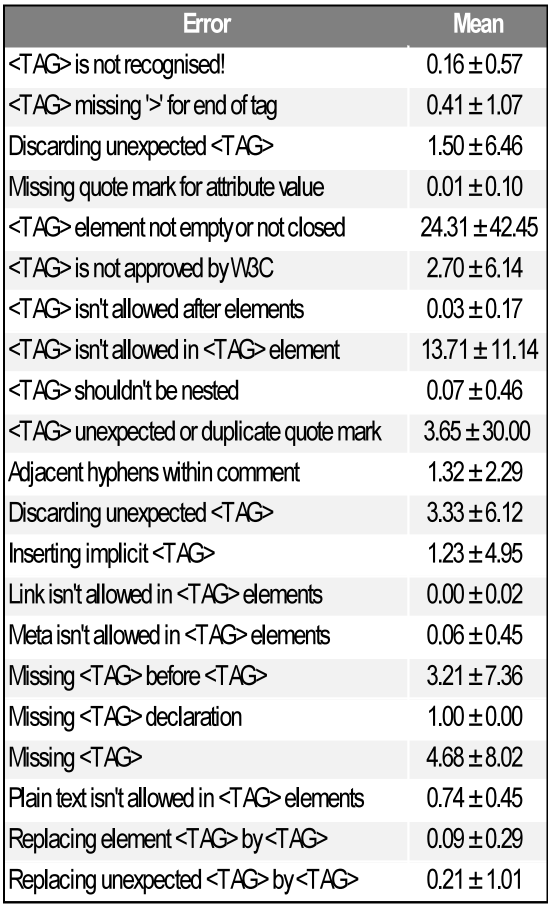


Repository	Category	Records
Ours	Books	Book{title, author, price, year, isbn}
	Cars	Vehicle{model, year, description, price, type, color, milage, transmission, engine, doors}
	Conferences	Event{title, date, place, url}
	Doctors	Doctor{name, address, phone, fax, specialty}
	Jobs	Offer{location, company}
	Movies	Movie{title, director, actor, year, runtime}
	Real estate	Property{address, bedrooms, bathrooms, price, size }
	Sports	Player{name, birth, hight, weight, age, college, country, club, position}
EXALG	cars.amazon.com	Car{model, make, price}
	players.uefa.com	Player{name, country}
	popartist.amazon.com	Artist{name}
	teams.uefa.com	Team{FIFA-affiliation, founded, general-secretary, president, press-officer, UEFA-affiliation}
	ausopen.com	Player{birthdate, birthplace, country, height, money, name, weight}
	ebay.com	Item{price, bids, location}
	majorleaguebaseball.com	Player{name, position, team}
	netflix.com	Film{title, director, length, year}
	rpmfind.net	Package{name, description, operative-system}
RISE	bigbook.com	Record{name, city, phone, street}
	iaf.net	Record{name, email, organisation, service-provider}
	laweekly.com/restaurants	Restaurant{name, address, speciality}
	okra.ucr.edu	Record{name, email}
	zagat.com	Restaurant{name, address, type}

Table 7.1: *Records extracted from our datasets in each category.*

category were randomly selected from the 100 best-ranked web sites between December 2010 and March 2011 according to Google's search engine. We downloaded 30 web documents from each web site and handcrafted a set of annotations with the information that we would like to extract from each site. The second group contains all of the datasets available online at the EXALG repository [7] and the datasets composed of semi-structured web documents available at the RISE repository [117]. Table §7.1 shows the structure of the records of information that we were interested in extracting. Note that the datasets in the EXALG and RISE repositories are not classified by categories, so we list the sites from which they originated.

Since the datasets are obtained from real-world web sites, they usually contain errors in their HTML code. Table §7.2 presents the results we

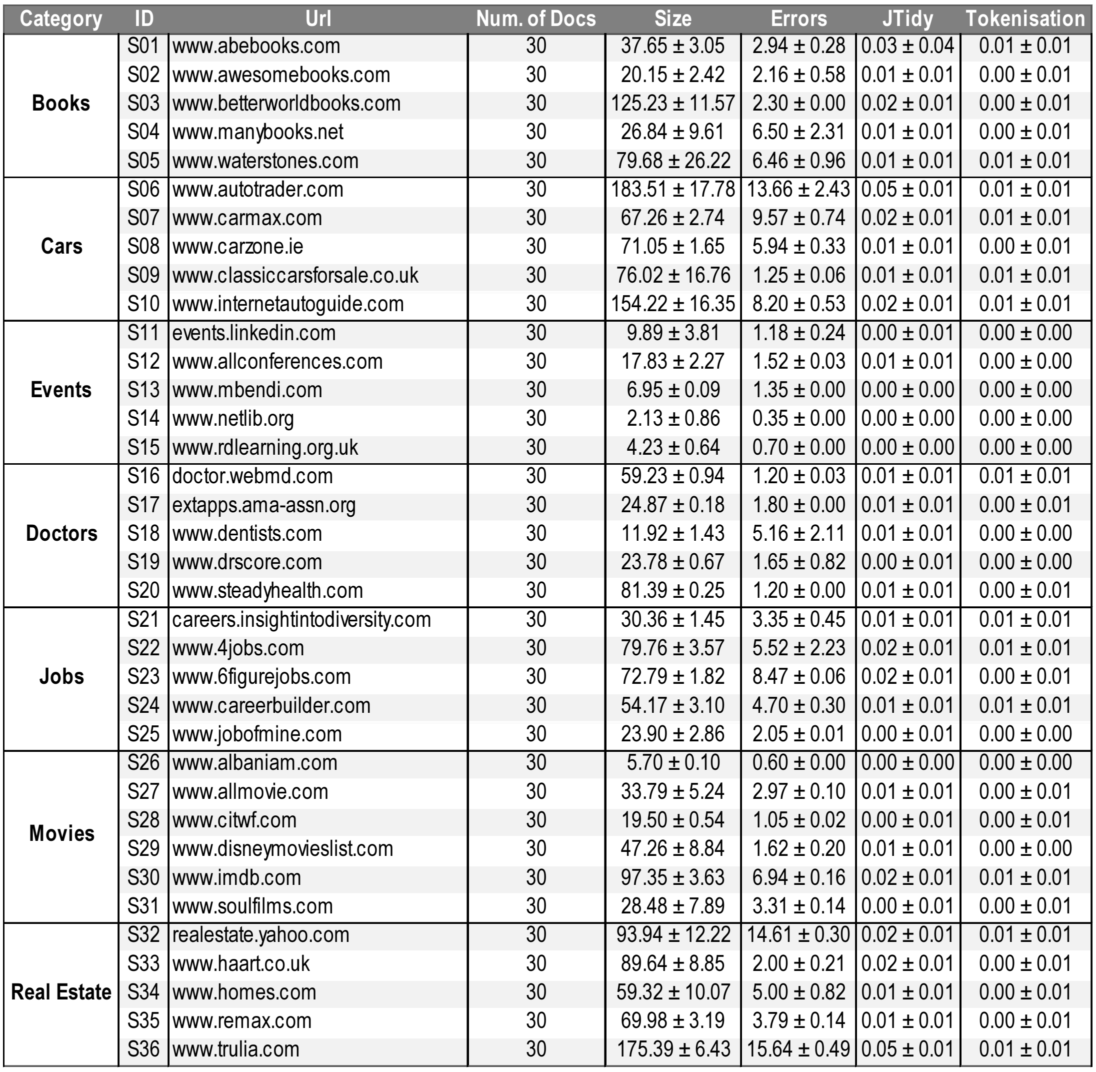


Error	Mean
<TAG> is not recognised!	0.16 ± 0.57
<TAG> missing '>' for end of tag	0.41 ± 1.07
Discarding unexpected <TAG>	1.50 ± 6.46
Missing quote mark for attribute value	0.01 ± 0.10
<TAG> element not empty or not closed	24.31 ± 42.45
<TAG> is not approved by W3C	2.70 ± 6.14
<TAG> isn't allowed after elements	0.03 ± 0.17
<TAG> isn't allowed in <TAG> element	13.71 ± 11.14
<TAG> shouldn't be nested	0.07 ± 0.46
<TAG> unexpected or duplicate quote mark	3.65 ± 30.00
Adjacent hyphens within comment	1.32 ± 2.29
Discarding unexpected <TAG>	3.33 ± 6.12
Inserting implicit <TAG>	1.23 ± 4.95
Link isn't allowed in <TAG> elements	0.00 ± 0.02
Meta isn't allowed in <TAG> elements	0.06 ± 0.45
Missing <TAG> before <TAG>	3.21 ± 7.36
Missing <TAG> declaration	1.00 ± 0.00
Missing <TAG>	4.68 ± 8.02
Plain text isn't allowed in <TAG> elements	0.74 ± 0.45
Replacing element <TAG> by <TAG>	0.09 ± 0.29
Replacing unexpected <TAG> by <TAG>	0.21 ± 1.01

Table 7.2: *Subset of common errors reported by JTidy on our datasets.*

have gathered regarding a subset of common HTML errors that are reported by JTidy; the full report is too large to be reproduced here. Our only purpose was to make it clear that we have dealt with actual documents, and that they usually contain errors that must be fixed heuristically. JTidy is a constituent part of many information extraction proposals that build on DOM trees.

Table §7.3 reports on the datasets. The first column lists the categories of the datasets; the second and the third columns list the web sites and an identifier that we use to refer to them in forth coming result tables; the fourth column presents the number of documents inside each dataset; the fifth column shows the mean size of documents in KiB; the sixth column shows the mean number of errors reported by JTidy inside each dataset; the seventh column lists the mean time in seconds required to clean and fix a web document



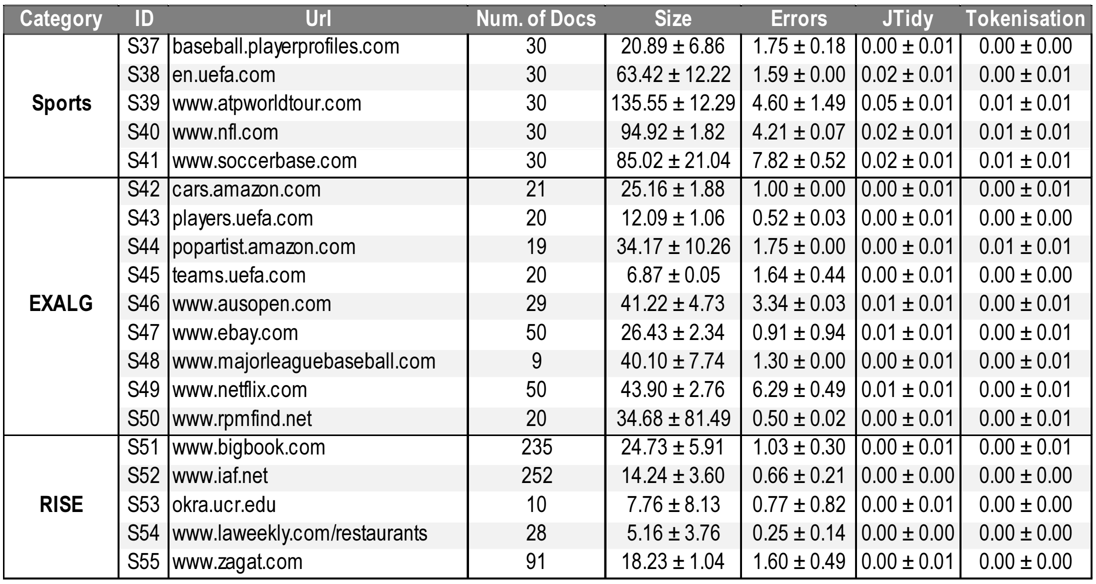
Category	ID	Url	Num. of Docs	Size	Errors	JTidy	Tokenisation
Books	S01	www.abebooks.com	30	37.65 ± 3.05	2.94 ± 0.28	0.03 ± 0.04	0.01 ± 0.01
	S02	www.awesomebooks.com	30	20.15 ± 2.42	2.16 ± 0.58	0.01 ± 0.01	0.00 ± 0.01
	S03	www.betterworldbooks.com	30	125.23 ± 11.57	2.30 ± 0.00	0.02 ± 0.01	0.00 ± 0.01
	S04	www.manybooks.net	30	26.84 ± 9.61	6.50 ± 2.31	0.01 ± 0.01	0.00 ± 0.01
	S05	www.waterstones.com	30	79.68 ± 26.22	6.46 ± 0.96	0.01 ± 0.01	0.01 ± 0.01
Cars	S06	www.autotrader.com	30	183.51 ± 17.78	13.66 ± 2.43	0.05 ± 0.01	0.01 ± 0.01
	S07	www.carmax.com	30	67.26 ± 2.74	9.57 ± 0.74	0.02 ± 0.01	0.01 ± 0.01
	S08	www.carzone.ie	30	71.05 ± 1.65	5.94 ± 0.33	0.01 ± 0.01	0.00 ± 0.01
	S09	www.classiccarsforsale.co.uk	30	76.02 ± 16.76	1.25 ± 0.06	0.01 ± 0.01	0.01 ± 0.01
	S10	www.internetautoguide.com	30	154.22 ± 16.35	8.20 ± 0.53	0.02 ± 0.01	0.01 ± 0.01
Events	S11	events.linkedin.com	30	9.89 ± 3.81	1.18 ± 0.24	0.00 ± 0.01	0.00 ± 0.00
	S12	www.allconferences.com	30	17.83 ± 2.27	1.52 ± 0.03	0.01 ± 0.01	0.00 ± 0.00
	S13	www.mbendi.com	30	6.95 ± 0.09	1.35 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	S14	www.netlib.org	30	2.13 ± 0.86	0.35 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	S15	www.rdlearning.org.uk	30	4.23 ± 0.64	0.70 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Doctors	S16	doctor.webmd.com	30	59.23 ± 0.94	1.20 ± 0.03	0.01 ± 0.01	0.01 ± 0.01
	S17	extapps.ama-assn.org	30	24.87 ± 0.18	1.80 ± 0.00	0.01 ± 0.01	0.00 ± 0.01
	S18	www.dentists.com	30	11.92 ± 1.43	5.16 ± 2.11	0.01 ± 0.01	0.00 ± 0.00
	S19	www.drscore.com	30	23.78 ± 0.67	1.65 ± 0.82	0.00 ± 0.01	0.00 ± 0.00
	S20	www.steadyhealth.com	30	81.39 ± 0.25	1.20 ± 0.00	0.01 ± 0.01	0.00 ± 0.01
Jobs	S21	careers.insightintodiversity.com	30	30.36 ± 1.45	3.35 ± 0.45	0.01 ± 0.01	0.01 ± 0.01
	S22	www.4jobs.com	30	79.76 ± 3.57	5.52 ± 2.23	0.02 ± 0.01	0.01 ± 0.01
	S23	www.6figurejobs.com	30	72.79 ± 1.82	8.47 ± 0.06	0.02 ± 0.01	0.00 ± 0.01
	S24	www.careerbuilder.com	30	54.17 ± 3.10	4.70 ± 0.30	0.01 ± 0.01	0.01 ± 0.01
	S25	www.jobofmine.com	30	23.90 ± 2.86	2.05 ± 0.01	0.00 ± 0.01	0.00 ± 0.00
Movies	S26	www.albaniam.com	30	5.70 ± 0.10	0.60 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	S27	www.allmovie.com	30	33.79 ± 5.24	2.97 ± 0.10	0.01 ± 0.01	0.00 ± 0.01
	S28	www.citwf.com	30	19.50 ± 0.54	1.05 ± 0.02	0.00 ± 0.01	0.00 ± 0.01
	S29	www.disneymovieslist.com	30	47.26 ± 8.84	1.62 ± 0.20	0.01 ± 0.01	0.00 ± 0.00
	S30	www.imdb.com	30	97.35 ± 3.63	6.94 ± 0.16	0.02 ± 0.01	0.01 ± 0.01
	S31	www.soulfilms.com	30	28.48 ± 7.89	3.31 ± 0.14	0.00 ± 0.01	0.00 ± 0.01
Real Estate	S32	realestate.yahoo.com	30	93.94 ± 12.22	14.61 ± 0.30	0.02 ± 0.01	0.01 ± 0.01
	S33	www.haart.co.uk	30	89.64 ± 8.85	2.00 ± 0.21	0.02 ± 0.01	0.00 ± 0.01
	S34	www.homes.com	30	59.32 ± 10.07	5.00 ± 0.82	0.01 ± 0.01	0.00 ± 0.01
	S35	www.remax.com	30	69.98 ± 3.19	3.79 ± 0.14	0.01 ± 0.01	0.00 ± 0.01
	S36	www.trulia.com	30	175.39 ± 6.43	15.64 ± 0.49	0.05 ± 0.01	0.01 ± 0.01

Table 7.3: *Properties of our datasets.*

using JTidy; finally, the last column lists the mean tokenisation time.

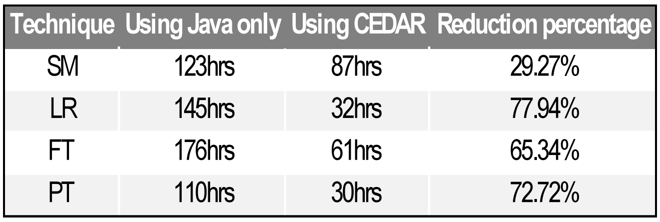
7.6 Experimentation

We have developed a framework to check the viability of our reference architecture. This framework was used to develop some of the most cited proposals in the literature that are based on transducers or that can be adapted to be used with transducers. We have implemented SM, LR, FT, and



Category	ID	Url	Num. of Docs	Size	Errors	JTidy	Tokenisation
Sports	S37	baseball.playerprofiles.com	30	20.89 ± 6.86	1.75 ± 0.18	0.00 ± 0.01	0.00 ± 0.00
	S38	en.uefa.com	30	63.42 ± 12.22	1.59 ± 0.00	0.02 ± 0.01	0.00 ± 0.01
	S39	www.atpworldtour.com	30	135.55 ± 12.29	4.60 ± 1.49	0.05 ± 0.01	0.01 ± 0.01
	S40	www.nfl.com	30	94.92 ± 1.82	4.21 ± 0.07	0.02 ± 0.01	0.01 ± 0.01
	S41	www.soccerbase.com	30	85.02 ± 21.04	7.82 ± 0.52	0.02 ± 0.01	0.01 ± 0.01
EXALG	S42	cars.amazon.com	21	25.16 ± 1.88	1.00 ± 0.00	0.00 ± 0.01	0.00 ± 0.01
	S43	players.uefa.com	20	12.09 ± 1.06	0.52 ± 0.03	0.00 ± 0.01	0.00 ± 0.00
	S44	popartist.amazon.com	19	34.17 ± 10.26	1.75 ± 0.00	0.00 ± 0.01	0.01 ± 0.01
	S45	teams.uefa.com	20	6.87 ± 0.05	1.64 ± 0.44	0.00 ± 0.01	0.00 ± 0.00
	S46	www.ausopen.com	29	41.22 ± 4.73	3.34 ± 0.03	0.01 ± 0.01	0.00 ± 0.01
	S47	www.ebay.com	50	26.43 ± 2.34	0.91 ± 0.94	0.01 ± 0.01	0.00 ± 0.01
	S48	www.majorleaguebaseball.com	9	40.10 ± 7.74	1.30 ± 0.00	0.00 ± 0.01	0.00 ± 0.01
	S49	www.netflix.com	50	43.90 ± 2.76	6.29 ± 0.49	0.01 ± 0.01	0.00 ± 0.01
	S50	www.rpmfind.net	20	34.68 ± 81.49	0.50 ± 0.02	0.00 ± 0.01	0.00 ± 0.01
RISE	S51	www.bigbook.com	235	24.73 ± 5.91	1.03 ± 0.30	0.00 ± 0.01	0.00 ± 0.01
	S52	www.iaf.net	252	14.24 ± 3.60	0.66 ± 0.21	0.00 ± 0.00	0.00 ± 0.00
	S53	okra.ucr.edu	10	7.76 ± 8.13	0.77 ± 0.82	0.00 ± 0.01	0.00 ± 0.00
	S54	www.laweekly.com/restaurants	28	5.16 ± 3.76	0.25 ± 0.14	0.00 ± 0.00	0.00 ± 0.00
	S55	www.zagat.com	91	18.23 ± 1.04	1.60 ± 0.49	0.00 ± 0.01	0.00 ± 0.00

Table 7.3: *Properties of our datasets. (Cont'd)*



Technique	Using Java only	Using CEDAR	Reduction percentage
SM	123hrs	87hrs	29.27%
LR	145hrs	32hrs	77.94%
FT	176hrs	61hrs	65.34%
PT	110hrs	30hrs	72.72%

Table 7.4: *Comparing implementation times for SM, LR, FT, and PT.*

PT with the help of Master Degree students who had been working in the industry for at least one year.

The aim of our reference architecture is to reduce the costs of devising rule learners and to allow to compare learners with each other. To validate it, we have conducted an experiment following the guidelines reported in [85]. This experiment was conducted to check if relying on our reference architecture and an accompanying framework, development costs were reduced remarkably. For this purpose, we requested four postgraduate students with a degree in Software Engineering to study the previous proposals. Then, they were requested to implement them using Java 1.6. The time to study the requirements, design, develop, and test these proposals was measured in hours. Since their development processes were totally independent, each of the participants had to create their datasets for testing.

New postgraduate students were requested to develop the same techniques, but this time using our framework. First, they went through to a training period that was added to the total time that was necessary to study the requirements, design, develop and test the developed techniques. In this case, datasets were reused between the different participants to compare the techniques side by side too.

Table §7.4 shows the time in hours that was necessary to develop and test the techniques on which we report in the first column. The second and the third columns show the time that was necessary to develop these techniques using only Java libraries and using our framework, respectively. The fourth column shows the time reduction for each technique. The costs reduction is clear since the framework allowed to reuse components during the development phase and reusing datasets in the testing phase. The last column shows the reduced time percentage; the arithmetic mean of the reduction percentage is $61.31 \pm 21.98\%$.

7.7 Summary

We have presented a reference architecture to help software engineers devise new learning techniques in the domain of information extraction from semi-structured web documents. This is the first reference architecture in the literature which provides an abstract, reusable, easy-to-maintain, and easy-to-adapt design that should allow software engineers and researchers to face the development of a new rule learning technique, for information extraction from semi-structured web documents, without incurring the high costs of developing it from scratch. The reference architecture was validated by an accompanying framework, that was used to implement four transducer-based information extractors.

Chapter 8

Extracting attributes with TEX

I paint objects as I think of them, not as I see them.

Pablo Picasso, Spanish painter (1881–1973)

TEX is an information extractor that focuses on extracting individual attributes from a number of similar documents. The chapter is organised as follows: Section §8.1 introduces the chapter; Section §8.2 describes the algorithms on which TEX relies; Section §8.3 reports on their complexity; Section §8.4 reports on the experimental analysis we have conducted to evaluate TEX empirically; Section §8.5 reports on the statistical analysis conducted to rank TEX; finally, Section §8.6 summarises this chapter.

8.1 Introduction

In this chapter, we introduce TEX, which is a heuristic-based information extractor that focuses on extracting attributes. Contrarily to other heuristic-based proposals, it does not require the input web documents to be translated into DOM trees, i.e., it can work on malformed web documents without correcting them, and does not require the relevant information to be formatted using repetitive patterns inside a web document.

It works on two or more web documents and compares them in an attempt to discover shared patterns (token strings) that are not likely to provide any relevant information, but parts of the template used to generate the web documents. TEX relies on quite a simple multi-string alignment algorithm that has proven to be very effective and efficient in practice. We have computed an upper limit to the worst-case space and time complexities of our algorithm and we have proved that it is computationally tractable (note that there are very few complexity results in this field); furthermore, we have conducted a series of experiments on real-world web sites and our results confirm that our proposal can achieve a mean precision as high as 96%, a mean recall as high as 95%, with a mean execution time of 0.81 seconds. We conducted the same experiments using other well-known techniques in the literature, and our conclusion is that our proposal outperforms them.

8.2 Algorithms

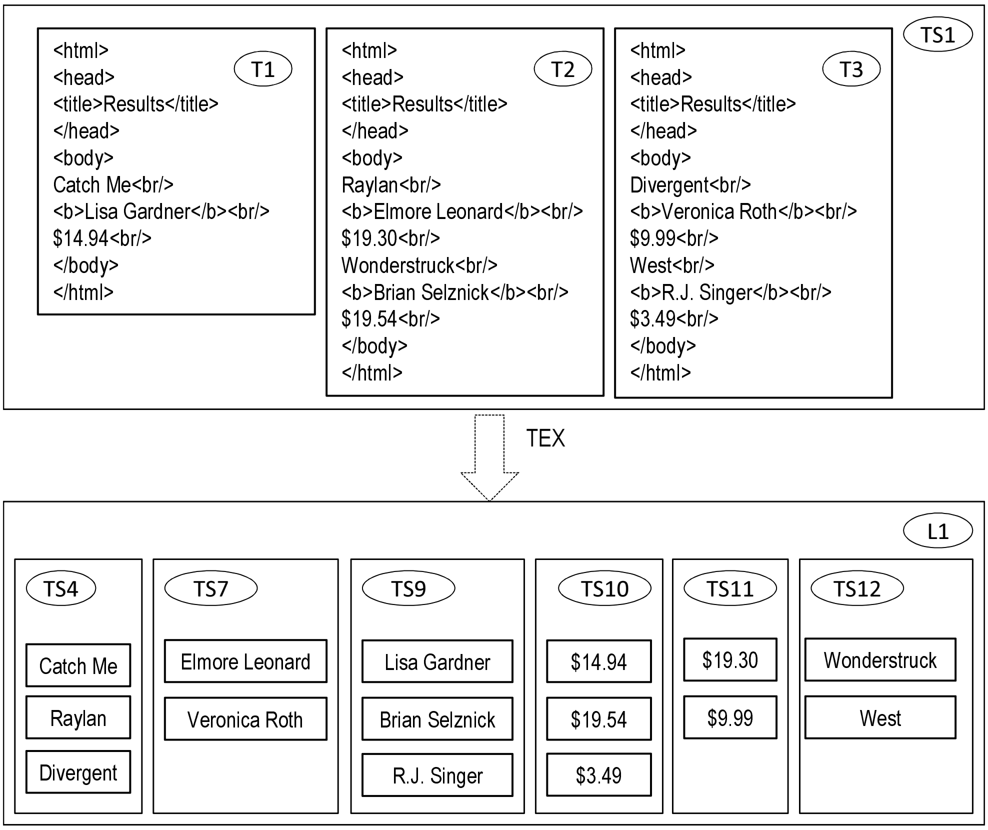
In this section, we describe the algorithms that lie at the heart of TEX. The main extraction algorithm searches for shared patterns in the input web documents and fragments them, and then is applied recursively to each fragment until no more shared patterns are found. In the following, we first describe the structures used by TEX, and then we explain the main and ancillary algorithms.

8.2.1 Structures

TEX works on a collection of web documents, which we denote as TextSet, and a range of integers, which can introduce a bias to our search procedure. Intuitively, a TextSet is a set of Texts, which are sequences of Tokens. TEX is not bound with a particular tokenisation schema; our implementation and

```
TEX($ts$: TextSet; $min$, $max$: int): List⟨TextSet⟩
    $l$ = extract($ts$, $min$, $max$)
    $result$ = filter($l$)
return $result$
```

Program 8.1: *Algorithm TEX.*



```
<html>
<head>
<title>Results</title>
</head>
<body>
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49<br/>
</body>
</html>
```

Figure 8.1: *A running example: input and output of TEX.*

our experiments were carried out using a simple tokenisation schema according to which tokens represent either script blocks, style blocks, HTML tags, or #PCDATA. Note that we use Text as a data type that allows to represent both web documents and fragments of web documents, as well as the information that is extracted from them.

We present the algorithm that lies at the heart of TEX in Program §8.1.

```
extract($ts$: TextSet; $min$, $max$: int): List⟨TextSet⟩
  $result$ = ⟨ts⟩
  for $size = max$ down to $min$ do
    $buffer$ = ⟨⟩
    while $result \neq$ ⟨⟩ do
      $ts$ = dequeue($result$)
      $expansion$ = expand($ts$, $size$)
      if $expansion$ = ⟨⟩ then
        enqueue($buffer$, $ts$)
      else
        enqueue($result$, $expansion$)
      end
    end
    $result$ = $buffer$
  end
return $result$
```

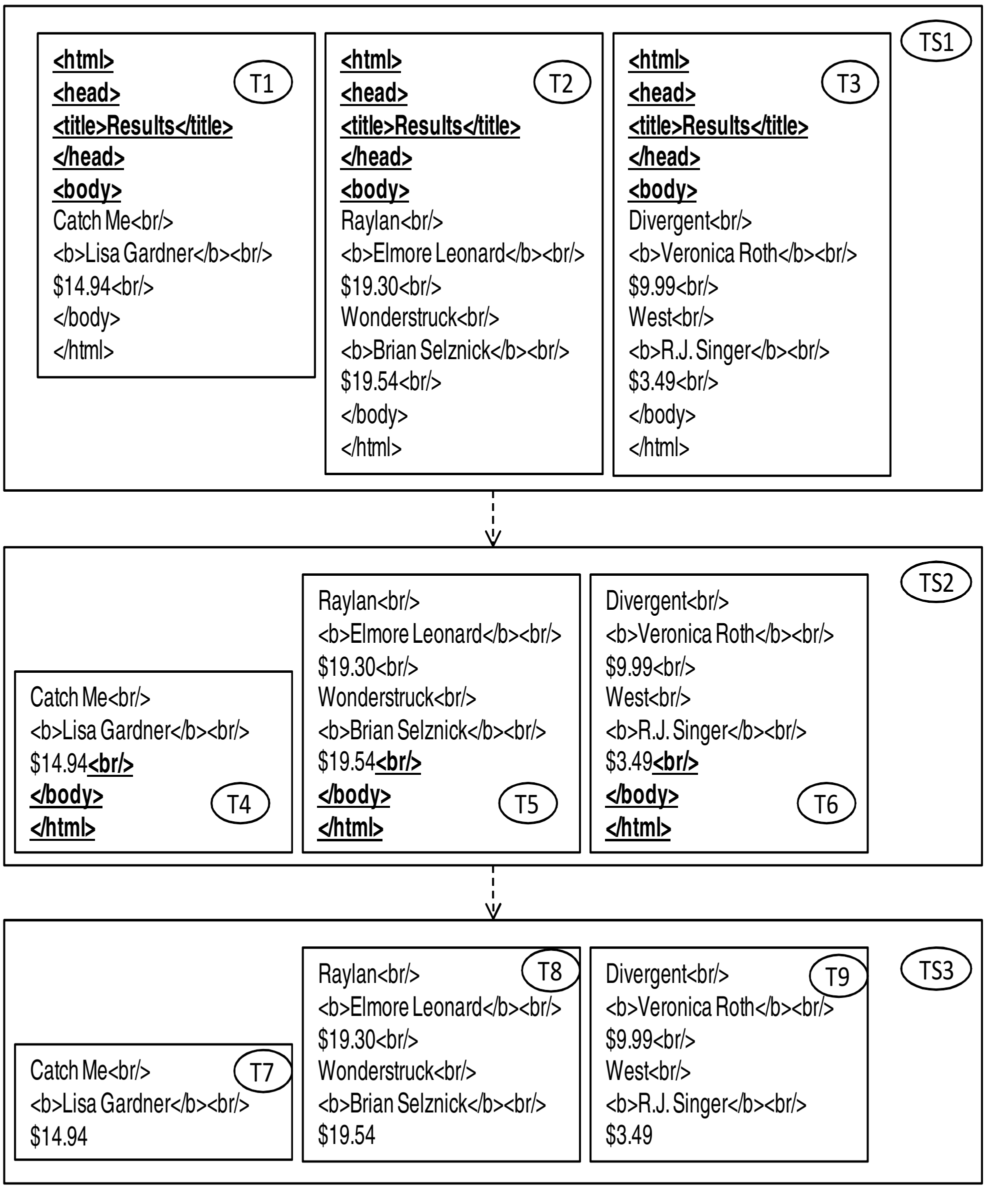
Program 8.2: *Algorithm extract.*

The algorithm works in two steps: at line §2, we invoke Algorithm extract, which makes an attempt to extract the information that varies from document to document; in other words, it attempts to discard information that is likely to belong to the template used to generate the input web documents. Algorithm extract works on the collection of input web documents and searches for shared patterns of size $max, max - 1, \ldots, min$. If $min > 1$ or max is less than the size of the shortest input document, then the search has a bias that may lead to situations in which Algorithm extract returns information that actually belongs to the template, which is the reason why we invoke a filtering algorithm at line §3.

Figure §8.1 presents a running example. We assume that the algorithm is executed on TextSet TS1, which is composed of documents T1, T2, and T3; the result is the list of TextSets L1, which contains the extracted TextSets TS4, TS7, TS11, TS12, TS9, and TS10.

8.2.2 Algorithm extract

Algorithm extract searches for shared patterns of size max down to min in a TextSet. For instance, assume that it is invoked on the TextSet denoted as TS1 in Figure §8.2 and that it has to search for shared patterns



```
<html>
<head>
<title>Results</title>
</head>
<body>
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49<br/>
</body>
</html>
```

```
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94<br/>
</body>
</html>
```

```
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54<br/>
</body>
</html>
```

```
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49<br/>
</body>
</html>
```

```
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94
```

```
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54
```

```
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49
```

Figure 8.2: *Expansion of a TextSet during extraction.*

whose size is in the range 10 down to 1. Note that there are neither shared patterns of size 10, 9, nor 8; the longest shared pattern is <html><head><title>Results</title></head><body>, whose size is 7 tokens. The algorithm then attempts to expand TextSet TS1 into three additional TextSets that contain the prefixes, the separators, and the suffixes into which the

```
expand($ts$: TextSet; $s$: int): List⟨TextSet⟩
  $result$ = ⟨⟩
  $shortest$ = find the shortest text in $ts$
  if $shortest \neq \langle\rangle$ and size($shortest$) >= $s$ then
    $shared$ = findPattern($ts$, $shortest$, $s$)
    if $shared \neq \{\}$ then
      $result$ = createExpansion($ts$, $shared$)
    end
  end
return $result$
```

Program 8.3: *Algorithm expand.*

shared pattern partitions the Texts in TS1. In this example, there are neither prefixes nor separators, since the shared pattern is found at the beginning of the Texts in TS1; there are, however, three suffixes that are stored in TextSet TS2. The algorithm then discards TextSet TS1 and proceeds with the new TextSet TS2. The longest shared pattern that is discovered in TS2 is <br/></body></html>, which results in a new TextSet that is denoted as TS3. The same procedure is applied as many times as necessary until no more shared patterns are discovered.

We present Algorithm extract in Program §8.2. It works on a TextSet ts, a minimum pattern size min and a maximum pattern size max; it returns a list of TextSets that should contain as much prospective information as possible. The main loop at lines §3–§15 iterates over all possible sizes from max down to min; for each size, the inner loop at lines §5–§13 searches for a shared pattern of that size. Note that variable $result$ acts as a queue in which we initially put the TextSet on which the algorithm has to work, and then the new TextSets into which it is expanded. In each iteration of the inner loop, a TextSet is removed from $result$ and expanded at line §7. Algorithm expand, which is presented in the following section, searches for shared patterns of a given size in a TextSet; if one such pattern is found, then it is used to expand the current TextSet into new TextSets with prefixes, separators, and suffixes, which are added to $result$ so that they can be analysed later in the inner loop; if no shared pattern is found, then the original TextSet is added to a buffer. Once the inner loop finishes, the buffer contains all of the new TextSets that have been produced, and it is transferred to the $result$ variable so that the algorithm can search for new shared patterns of a smaller size, if possible.

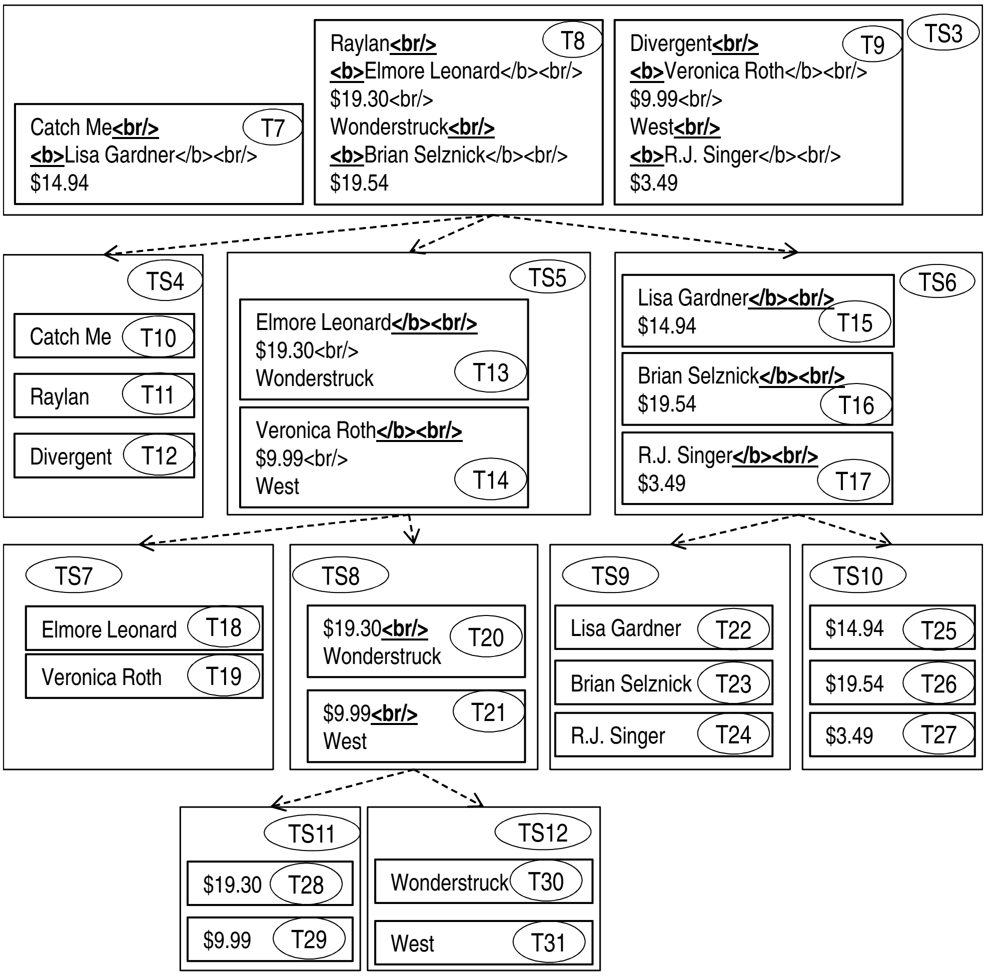


Figure 8.3: *Expansion of a sample TextSet.*

Algorithm expand: This algorithm searches for a shared pattern of a given size inside a given TextSet; if such a pattern is found, it then expands the TextSet into a collection of new TextSets with prefixes, separators, and suffixes. We have already illustrated how Algorithm expand works on two simple cases in which the expansion led to prefixes or suffixes only, cf. Figure §8.2. Assume now that it is invoked on TextSet TS3 in Figure §8.3 to search for a shared pattern of size two tokens. The algorithm can easily detect that the first 2-token shared pattern is <br/><b> and expands TextSet TS3 into the following new TextSets: i) TS4, which contains the prefixes of the Texts in TS3 up to the first occurrence of the shared pattern; ii) TS5, which contains

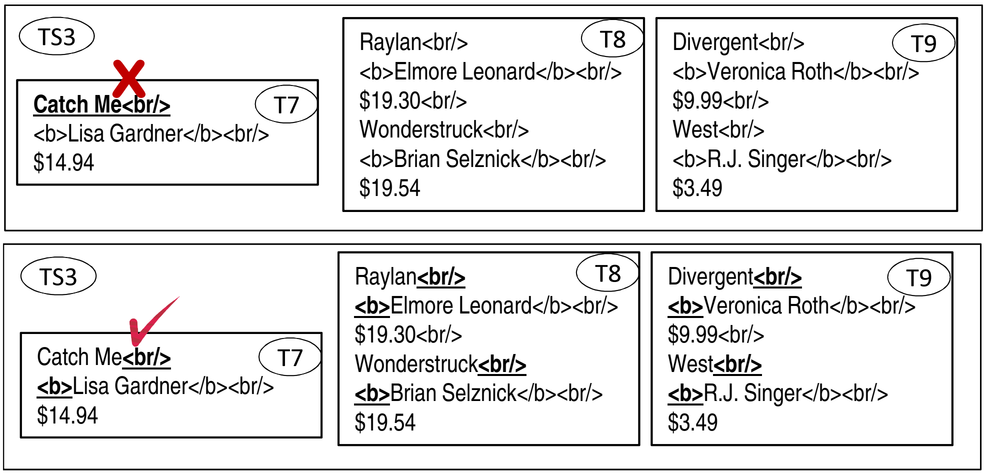


Figure 8.4: *Searching for a pattern in a base Text.*

the separators, i.e., the Texts in TS3 in between successive occurrences of the shared pattern; iii) and TS6, which contains the suffixes of the Texts in TS3 regarding the last occurrence of the shared pattern. When expand is invoked on TextSet TS5 to find a shared pattern of size 2 tokens, it finds </b><br/>, and creates TS7 and TS8, which contain the prefixes and suffixes of the shared pattern respectively. The same happens when expand is invoked on the TextSet TS6 to search for a shared pattern of size 2 tokens, and creates TS9 and TS10. If we invoke expand on TextSet TS8 to search for a shared pattern of one token, then, it finds pattern <br/> and creates TextSets TS11 and TS12.

We present Algorithm expand in Program §8.3. Line §3 searches for the shortest Text in ts, which is used at line §5 as a basis to search for shared patterns by means of Algorithm findPattern, which is described in the following section. If this algorithm can find a shared pattern, then line §7 expands ts using that shared pattern and updates variable result; if not, result remains an empty list, which indicates that TextSet ts cannot be expanded.

Algorithm findPattern: This algorithm works on a TextSet ts, a Text base, which is assumed to be the shortest non-empty Text in ts, and a size s. Its goal is to find a pattern inside base that occurs in every Text in ts. For instance, assume that the algorithm is invoked on TextSet TS3 in Figure §8.4 to search for a pattern of size 2; we implicitly assume that base is the shortest Text in TS3, i.e., base = Catch Me<br/><b>Lisa Gardner</b><br/>$14.94 in this example. The algorithm first searches for Catch Me<br/> in every Text in TS3, but does not

```
 1: findPattern(ts: TextSet; base: Text; s: int): Map<Text, List⟨int⟩>
 2:    found = false
 3:    for i = 0 until size(base) - s while not found do
 4:       result = {}
 5:       found = true
 6:       foreach text in ts while found do
 7:          matches = findMatches(text, base, i, s)
 8:          found = size(matches) > 0
 9:          result = result ∪ {text ↦ matches}
10:       end
11:    end
12: return result
```

Program 8.4: *Algorithm findPattern.*

find it; then it searches for <br/><b>, which is found in every Text in TS3. As a conclusion <br/><b> is a shared pattern that can be used to expand TextSet TS3. Note that Algorithm findPattern returns a map from Text onto lists of integers; the map represents the positions where the search pattern is found. In our example, this map is $\{T7 \mapsto \langle 1 \rangle, T8 \mapsto \langle 1, 9 \rangle, T9 \mapsto \langle 1, 9 \rangle\}$.

We present Algorithm findPattern in Program §8.4. The main loop at lines §3–§11 allows to implement a sliding window over $base$: index i iterates from 0 until $\text{size}(base) - s$ as long as no shared pattern is found, i.e., it searches for all patterns of size s in $base$. The actual search is performed in the inner loop at lines §6–§10: in this loop, the algorithm iterates over every Text in the input TextSet and finds all of the matches of the subsequence of $base$ that starts at position i and has size s. We do not provide any additional details on Algorithm findMatches since it is implemented using the well-known Knuth-Pratt-Morris pattern search algorithm [86]. This algorithm returns a list of integers that indicate the non-overlapping positions at which the previous subsequence of $base$ matches $text$; if there is at least one match, we record it in variable $result$ and go ahead to examine the next Text in ts; otherwise, the inner loop finishes and the outer loop slides the window on $base$ and resets $result$, if possible. If the algorithm returns an empty map, this means that no shared pattern has been found.

Algorithm createExpansion: When a shared pattern is found in a TextSet, the TextSet is expanded to three new TextSets, namely: prefixes, separators, and suffixes. We present Algorithm createExpansion in Program §8.5. This al-

```
1: createExpansion($ts$: TextSet; $r$: Map$\langle$Text, List$\langle$int$\rangle\rangle$; $s$: int): List$\langle$TextSet$\rangle$
2:    $result$ = $\langle\rangle$
3:    $ts_1$ = new TextSet()
4:    $ts_2$ = new TextSet()
5:    $ts_3$ = new TextSet()
6:    foreach $text$ in $ts$ do
7:       $matches$ = get($r$, $text$)
8:       if prefix($matches$, $text$) $\neq \langle\rangle$ then
9:          add prefix($matches$, $text$) to $ts_1$
10:      end
11:      add non-empty separators($matches$, $text$) to $ts_2$
12:      if suffix($matches$, $text$) $\neq \langle\rangle$ then
13:         add suffix($matches$, $text$) to $ts_3$
14:      end
15:   end
16:   if $ts_1$ is not empty then
17:      add $ts_1$ to $result$
18:   end
19:   if $ts_2$ is not empty then
20:      add $ts_2$ to $result$
21:   end
22:   if $ts_3$ is not empty then
23:      add $ts_3$ to $result$
24:   end
25: return result
```

Program 8.5: *Algorithm createExpansion.*

gorithm works on a TextSet called ts and a map r that contains the indexes of the shared pattern inside each Text of ts. The loop at lines §6–§15 iterates over all of the Texts in ts and adds the prefixes, separators, and suffixes to variables ts_1, ts_2, and ts_3, respectively. Later, we add these intermediate TextSets to the $result$ variable as long as they are not empty.

8.2.3 Algorithm filter

Algorithm extract returns a list of TextSets that are expected to have the variable information in the initial TextSet. Note, however, that min and max introduce a bias to the search algorithm if min is greater than one or max is less than the size of the shortest Text in the initial TextSet. Our experimental results prove that this bias helps effectively reduce the amount of effort

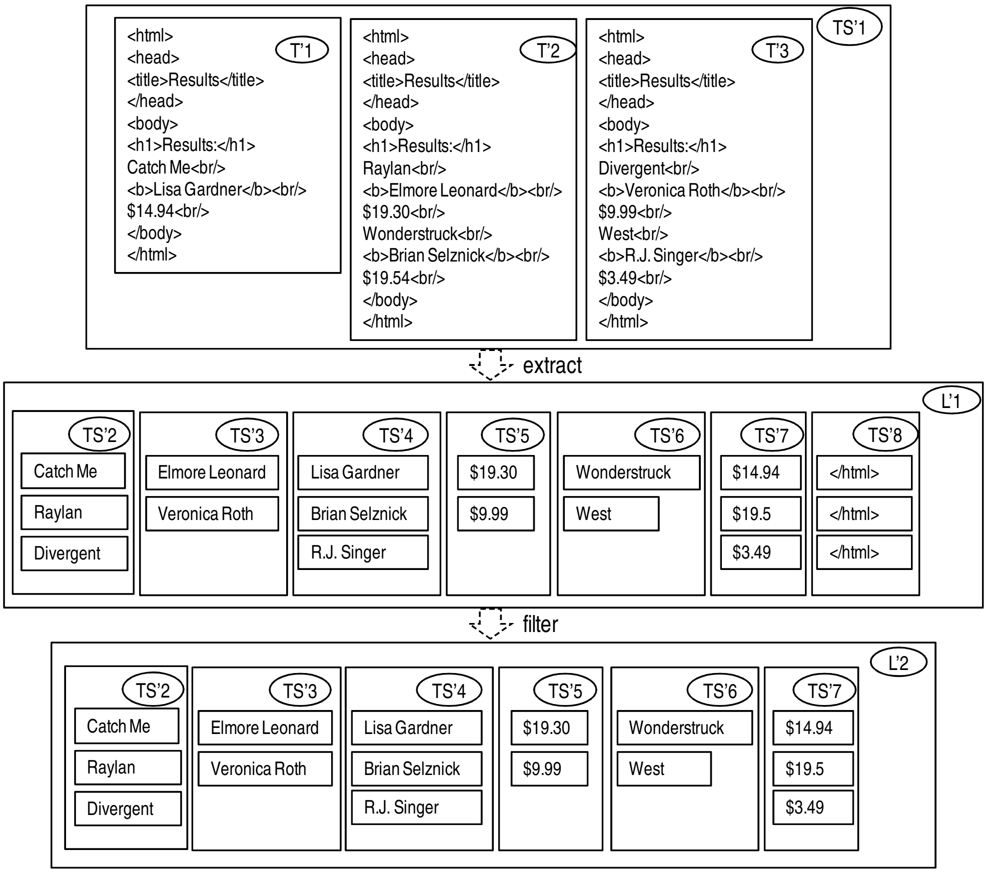


Figure 8.5: *A case in which filtering TextSets is required.*

```
filter(tss: List⟨TextSet⟩): List⟨TextSet⟩
    result = ⟨⟩
    foreach ts in tss do
        if ts has variability then
            add ts to result
        end
    end
return result
```

Program 8.6: *Algorithm filter.*

required to extract information from typical web documents, without sacrificing effectiveness. There are, however, cases in which setting min to a value greater than one and setting max to a small value, may prevent Algorithm extract from finding small shared patterns. For instance, assume that we set $min = 2$ and $max = 2$ and that Algorithm extract returns the list of TextSets $L'1$ in Figure §8.5: if the algorithm was allowed to search for patterns of size one, then it would discover that <html> is a shared pattern and would discard TextSet $TS'8$; however, min was set to 2, which prevents the algorithm from finding this pattern.

We present Algorithm filter in Program §8.6. The main loop at lines §3–§7 iterates over the list of input TextSets and simply removes those without variability from the result, i.e., those TextSets in which all of the Texts are the same.

8.2.4 Limitations

A limitation of TEX is that it does not extract data records, but only their attributes. A possible solution for this limitation is to apply a post-processing phase to reconstruct the data records starting from the attributes extracted by TEX. Another limitation of TEX is in extracting information from multi-record web documents. When the input is a collection of list web documents, TEX groups the attributes from the first data record and from the last data records together, whereas the attributes of the remaining data records are grouped correctly. For instance, the TextSets TS4, TS9, and TS10 in Figure §8.1, which contain the titles, authors, and prices of the first and last data records, are grouped together, but not with the other titles in TS12, authors in TS7, and prices in TS11. The main reason of this limitation is that TEX searches for the largest shared pattern amongst the web documents, and usually the header of the data records is shared, but it includes some part of the data record tags. Once the previous tokens of the first data records and the posterior tokens of the last data records are removed since they are shared amongst the web documents, it is not possible to align the data records attributes with the attributes of the remaining data records in the web documents.

8.3 Complexity analysis

In this section, we provide an upper limit to the worst-case space and time complexities of Algorithm TEX. Note that it is not common to find a complexity analysis in the literature regarding information extraction, but we think that it is important to make sure that the proposal is computationally tractable.

We have characterised an upper bound to the worst-case time complexity building on two sensible assumptions: i) simple instructions like adding an item to a set, comparing two tokens, or constructing a tuple can be implemented in $O(1)$ time with regard to the other algorithms in our proposal; ii) the number of input documents is generally very small as compared with the number of tokens of the longest document to be analysed.

In the following subsections, we first report on the space requirements, then on time requirements, and conclude with the theorems that prove that TEX is computationally tractable. In the sequel, we use variable n to denote the number of documents that TEX has to analyse and m to denote the size of the longest such document.

8.3.1 Space requirements

Proposition 8.1 (Maximum size of a TextSet) *Assume that we are extracting information from a TextSet denoted as ts using Algorithm TEX. $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to maximum size of a TextSet generated by TEX.*

Proof Algorithm expand is the unique algorithm that generates new TextSets, which happens when a shared pattern p is found in a given TextSet. The new TextSets correspond to the prefixes, separators, and suffixes to which p leads. Regarding the prefix and suffix TextSets, note that there cannot be more than n such prefixes or suffixes; that is, n is an upper bound to the maximum size of a TextSet that contains prefixes or suffixes. Regarding separator TextSets the worst case happens when p is a one-token pattern that occurs every two tokens; that is, $\lfloor \frac{m}{2} \rfloor$ is an upper bound to the number of separators in this case, or, otherwise, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a TextSet that contains separators. As a conclusion, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a TextSet generated by TEX. □

Proposition 8.2 (Maximum number of TextSets) *$3\,m$ is an upper bound to the number of TextSets created by TEX.*

Proof Algorithm expand creates three new TextSets when a shared pattern p is found in a given TextSet. The new TextSets correspond to the prefixes, separators, and suffixes to which p leads. We know that m is an upper bound to the number of partitions of a Text of size m, which means that m is an upper bound to the number partitions in the worst-case. Since partition has three TextSets, then $3\,m$ is an upper bound to the number of TextSet created by TEX. □

8.3.2 Time requirements

Proposition 8.3 (Algorithm createExpansion) *Let* ts *be a Textset,* r *a map from the Texts in* ts *to lists of indices that denote where a shared pattern occurs, and* s *the size of the shared pattern.* $O(n\,m^2)$ *is an upper bound to its worst-case time required to execute createExpansion*(ts, r, s).

Proof The algorithm iterates through every Text in ts. According to Proposition §8.1, $n\,\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a TextSet, which means that $n\,m$ is an upper bound to the number of iterations of this loop. Inside this loop, accessing the map and calculating the prefix and suffix of the shared pattern can be performed in $O(1)$ time, whereas computing the separators requires variable time. According to the proof of Proposition §8.1, the maximum number of separators in a given Text is $\lfloor\frac{m}{2}\rfloor$, which is less than m. Then, $O(m)$ is an upper bound to the time required to compute the separators. As a conclusion, $O(n\,m\,m) = O(n\,m^2)$ is an upper bound to the worst-case time required to execute createExpansion(ts, r, s). □

Proposition 8.4 (Algorithm findPattern) *Let* ts *be a TextSet,* base *the shortest Text in* ts*, and* s *the size of the pattern for which the algorithm searches.* $O(n\,m^3)$ *is an upper bound to the worst-case time required to execute findPattern*$(\mathrm{ts}, \mathrm{base}, s)$.

Proof The main loop iterates through base until finding a pattern of s tokens that occurs in every other Text in ts. In the worst case, base has the maximum size m and the shared pattern is found at the end of base, which means that the main loop iterates $m - s$ times, i.e., $O(m)$ times. In each iteration of the main loop, the inner loop iterates through the Texts in ts. According to Proposition §8.1, $n\,\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a TextSet, which implies that the inner loop does not iterate more than $n\,\lfloor\frac{m}{2}\rfloor$ times. In each iteration, it invokes Algorithm findMatches, whose worst-time complexity is $O(k)$, where k denotes the size of the text in which a pattern is searched [86]. This implies that $O(m)$ is an upper bound to the worst-case time complexity of the instructions inside the inner loop. As a conclusion, $O(m\,n\,\lfloor\frac{m}{2}\rfloor\,m) \subseteq O(n\,m^3)$ is an upper bound to the worst-case time required to execute findPattern$(\mathrm{ts}, \mathrm{base}, s)$. □

Proposition 8.5 (Algorithm expand) *Let* ts *be a TextSet, and* s *be a pattern size.* $O(n\,m^3)$ *is an upper bound to the worst-case time required to execute expand*(ts, s).

Proof The algorithm first searches for the shortest Text in ts. According to Proposition §8.1, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a TextSet, which implies that $n \lfloor \frac{m}{2} \rfloor$ is also an upper bound to the maximum time required to find the shortest Text in a TextSet. In the worst case, the invocation to Algorithm expand requires to invoke Algorithms findPattern and createExpansion in sequence, which according to Propositions §8.4 and §8.3 require no more than $O(n\, m^3)$ and $O(n\, m^2)$ time in the worst case. As a conclusion, $O(n \lfloor \frac{m}{2} \rfloor + n\, m^3 + n\, m^2) \subseteq O(n\, m^3)$ is an upper bound to the worst-case time required to execute expand(ts, s). □

Proposition 8.6 (Algorithm extract) *Let ts be a TextSet, min and max be the minimum and maximum sizes of the shared patterns for which the algorithm searches, respectively. $O(n\, m^5)$ is an upper bound to the worst-case time required to execute extract(ts, min, max).*

Proof The algorithm first iterates through all possible sizes between min and max, which amounts to m times in the worst case. In each iteration, the algorithm executes an inner loop that iterates through successive expansions of ts. Note that m puts an upper bound to the number of times that a TextSet can be expanded, which implies that m is also an upper bound to the number of iterations of the inner loop. Within this loop, the only significant instruction regarding our complexity analysis is the invocation of Algorithm expand, which according to Proposition §8.5 requires no more than $O(n\, m^3)$ time. As a conclusion, $O(m\, m\, n\, m^3) = O(n\, m^5)$ is an upper bound to the worst-case time required to execute extract(ts, min, max). □

Proposition 8.7 (Algorithm filter) *Let L be a list of TextSets of size k. $O(k\, n\, m)$ is an upper bound to the worst-case time required to execute filter(L).*

Proof The algorithm iterates through every Text in L. If we denote the size of L as k, then this loop iterates k times. In each iteration, the algorithm checks the variability of the current TextSet, which requires to compare the first Text to every other in order to determine whether the TextSet has variability or not. According to Proposition §8.1, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the number of Texts in a TextSet, which implies that $O(k\, n \lfloor \frac{m}{2} \rfloor) \subseteq O(k\, n\, m)$ is an upper bound to the worst-case time required to execute filter(L). □

8.3.3 Computational tractability

Theorem 8.1 (Space requirements of Algorithm TEX) *Let ts be a TextSet and min and max be the minimum and maximum sizes of the shared patterns for which TEX searches. $O(n\, m^2)$ is an upper bound to the space required to execute TEX(ts, min, max) in the worst case.*

Proof The proof follows straightforwardly from Propositions §8.1 and §8.2. If $O(n\lfloor\frac{m}{2}\rfloor)$ is an upper bound to the maximum size of a TextSet and $O(3\,m)$ is an upper bound to the maximum number of TextSets the algorithm creates, then $O(n\lfloor\frac{m}{2}\rfloor 3\,m) \subseteq O(n\,m^2)$ is an upper bound to the space required to execute $\mathsf{TEX}(ts, min, max)$ in the worst case. □

Theorem 8.2 (Time requirements of Algorithm TEX) *Let ts be a TextSet and min and max be the minimum and maximum sizes of the shared patterns for which TEX searches. $O(n\,m^5)$ is an upper bound to the worst-case time required to execute TEX(ts, min, max).*

Proof The proof follows straightforwardly from the previous propositions. Note that Algorithm TEX invokes Algorithms extract and filter in sequence, which, according to Propositions §8.6 and §8.7 require no more than $O(n\,m^5)$ and $O(k\,n\,m)$ time to complete, where k denotes the size of the list of TextSets returned by Algorithm extract. Note that m puts an upper bound to the size of this list, which implies that $O(n\,m^5 + n\,m^2) \subseteq O(n\,m^5)$ is an upper bound to the worst-case time required to execute $\mathsf{TEX}(ts, min, max)$. □

Corollary 8.1 *If we assume that $n \ll m$, then $(n\,m^2)$ is an upper bound to the space required to execute Algorithm TEX and $O(m^5)$ is an upper bound to the worst-case time complexity. Since both results are polynomial, we can conclude that Algorithm TEX is computationally tractable.*

8.4 Experimental analysis

In this section, we present the results of the experiments we have carried out to compare our proposal to other techniques in the literature from an empirical point of view. We ran RoadRunner, FiVaTech, SM, and LR on CEDAR repository in order to learn extraction rules and then applied these rules and compared the results with TEX. We measured the standard effectiveness measures (precision, recall, and the $F1$ measure) and two efficiency measures (learning and extraction time). We did not measure memory since the heap size was set to the default value and it was never exceeded.

8.4.1 Experimentation environment

We have developed a Java 1.7 prototype of TEX using our framework. We performed a series of experiments on a cloud machine that was equipped with a four-threaded Intel Core i7 processor that ran at 2.93 GHz, had 4 GiB of RAM, Windows 7 Pro 64-bit, Oracle's Java Development Kit 1.7.0_02, and GNU Regex 1.1.4. The configuration parameters of the Java Virtual Machine were set to their default values.

8.4.2 Effectiveness analysis

In the case of SM and LR, it was easy to compute the precision and recall for each attribute since both techniques are supervised. Contrarily, TEX, RoadRunner, and FiVaTech are unsupervised, so we used a technique similar to the one used to evaluate RoadRunner, FiVaTech, and EXALG, i.e., given an annotation, we can consider that the precision and recall to extract it correspond to the extracted TextSet with the highest $F1$ measure.

Recall that TEX can introduce a bias to the algorithm that searches for shared patterns. In our experiments, we used the following heuristics to find the most appropriate bias: we first set min to one and max to the size of the longest input document, let it be m, and we measured precision and recall; note that this does not introduce any bias, since these values allow TEX to find every possible shared pattern. We then set $max = \lfloor \frac{m}{2} \rfloor$ and measured precision and recall again; if there were no changes, we then set $max = \lfloor \frac{m}{4} \rfloor$ and repeated the procedure until precision or recall was affected. Similarly, we explored changes to min. We experimentally found that setting $min = 2$ and $max = \lfloor 0.05m \rfloor$ was the maximum allowable bias. This resulted in a significant reduction of CPU time without having an impact on neither precision nor recall.

Table §8.1 shows our results regarding effectiveness. The first few rows provide a summary in terms of mean and standard deviation of precision (P), recall (R), and $F1$ measure. In average, TEX seems to outperform the other techniques in both precision and recall. The remaining rows provide the results computed for each web site. Note that some cells contain a dash, which indicates that the corresponding technique was not able to learn an extraction rule in 15 CPU minutes.

According to Table §8.1, TEX outperforms the other techniques regarding effectiveness. Figure §8.6 illustrates this conclusion since the majority of points that correspond to TEX are very close to the upper right corner, whereas the points that correspond to the other techniques are more scattered. Intuitively, the closer the points to $(1.00, 1.00)$ the higher the $F1$ measure; similarly, the closer to $(0.00, 0.00)$ the lower the $F1$ measure.

To discern if errors in the input documents have an impact from a statistical point of view on the effectiveness of the proposals, we need compute the correlation from the number of errors to the $F1$ measure using non-parametric Kendall's Tau procedure. Table §8.2 presents the results of this

		TEX			RoadRunner			FivaTech			SM			LR		
	Summary	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
	Mean	0.96	0.95	0.95	0.36	0.36	0.36	0.80	0.87	0.81	0.84	0.61	0.66	0.72	0.61	0.64
	StDev	0.07	0.11	0.09	0.45	0.45	0.45	0.20	0.17	0.17	0.15	0.32	0.30	0.24	0.31	0.29
	Site	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Books	S01	1.00	1.00	1.00	-	-	-	0.92	0.99	0.95	0.87	0.58	0.70	0.52	0.16	0.24
	S02	1.00	0.87	0.93	1.00	1.00	1.00	0.85	1.00	0.92	1.00	0.39	0.56	0.77	0.26	0.39
	S03	0.99	1.00	0.99	0.00	0.00	0.00	0.99	0.96	0.97	0.98	0.99	0.98	0.43	0.35	0.39
	S04	0.99	0.99	0.99	-	-	-	0.77	0.97	0.86	0.99	0.99	0.99	0.25	0.23	0.24
	S05	0.96	1.00	0.98	1.00	0.89	0.94	1.00	0.94	0.97	1.00	1.00	1.00	0.71	0.67	0.69
Cars	S06	0.99	1.00	1.00	0.00	0.00	0.00	-	-	-	0.89	0.87	0.88	0.89	0.00	0.00
	S07	1.00	1.00	1.00	0.00	0.00	0.00	0.45	0.89	0.60	0.89	0.89	0.89	0.88	0.88	0.88
	S08	0.98	1.00	0.99	0.00	0.00	0.00	0.92	1.00	0.96	0.92	0.02	0.05	0.82	0.83	0.83
	S09	0.86	0.90	0.88	0.00	0.00	0.00	-	-	-	0.90	0.90	0.90	0.17	0.13	0.15
	S10	1.00	1.00	1.00	0.00	0.00	0.00	0.97	0.94	0.96	-	-	-	0.11	0.11	0.11
Events	S11	0.96	0.96	0.96	0.74	0.74	0.74	-	-	-	0.87	0.55	0.68	0.57	0.20	0.30
	S12	0.98	0.99	0.99	-	-	-	0.84	0.90	0.87	0.99	0.25	0.40	0.80	0.40	0.53
	S13	1.00	1.00	1.00	0.90	1.00	0.95	0.90	1.00	0.95	0.60	0.60	0.60	0.80	0.40	0.53
	S14	0.96	0.98	0.97	0.00	0.00	0.00	0.39	0.50	0.44	0.87	0.44	0.59	0.40	0.40	0.40
	S15	0.99	0.99	0.99	0.00	0.00	0.00	0.99	0.79	0.88	0.52	0.39	0.45	0.62	0.23	0.34
Doctors	S16	1.00	1.00	1.00	0.00	0.00	0.00	0.77	1.00	0.87	0.86	0.45	0.59	0.60	0.60	0.60
	S17	0.98	1.00	0.99	-	-	-	-	-	-	0.79	0.39	0.53	0.60	0.60	0.60
	S18	0.92	1.00	0.96	1.00	1.00	1.00	0.56	0.99	0.72	0.61	0.61	0.61	1.00	1.00	1.00
	S19	1.00	1.00	1.00	1.00	1.00	1.00	0.78	1.00	0.88	0.91	0.87	0.89	0.78	0.80	0.79
	S20	1.00	1.00	1.00	0.00	0.00	0.00	0.83	0.83	0.83	0.75	0.25	0.38	0.75	0.75	0.75
Jobs	S21	0.83	0.83	0.83	0.70	0.70	0.70	1.00	0.74	0.85	0.57	0.47	0.52	1.00	1.00	1.00
	S22	0.92	0.98	0.95	0.00	0.00	0.00	-	-	-	0.45	0.25	0.32	0.94	0.94	0.94
	S23	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.99	0.75	0.00	0.00	0.25	0.25	0.25
	S24	1.00	1.00	1.00	0.00	0.00	0.00	0.80	0.83	0.82	0.75	0.00	0.00	0.75	0.75	0.75
	S25	0.86	1.00	0.93	0.86	1.00	0.93	-	-	-	0.75	0.03	0.06	0.50	0.50	0.50
Movies	S26	0.95	0.98	0.96	0.81	1.00	0.89	0.82	0.81	0.81	0.85	0.40	0.54	0.87	0.24	0.38
	S27	0.97	0.96	0.96	0.27	0.30	0.28	0.79	0.74	0.77	0.93	0.29	0.44	0.13	0.07	0.09
	S28	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.92	0.72	0.81	0.39	0.30	0.34
	S29	1.00	1.00	1.00	0.00	0.00	0.00	0.71	0.67	0.69	0.97	0.97	0.97	0.72	0.72	0.72
	S30	0.93	0.86	0.89	0.00	0.00	0.00	-	-	-	0.88	0.85	0.86	0.38	0.38	0.38
	S31	0.99	0.92	0.95	0.00	0.00	0.00	0.59	1.00	0.74	0.99	0.95	0.97	0.91	0.81	0.86
Real Estate	S32	1.00	1.00	1.00	0.00	0.00	0.00	0.77	0.97	0.86	1.00	1.00	1.00	0.83	0.83	0.83
	S33	1.00	1.00	1.00	0.00	0.00	0.00	0.94	1.00	0.97	1.00	1.00	1.00	0.78	0.75	0.76
	S34	0.99	0.99	0.99	0.00	0.00	0.00	-	-	-	0.80	0.79	0.79	0.92	0.92	0.92
	S35	0.70	0.98	0.82	-	-	-	0.77	0.99	0.87	0.84	0.84	0.84	1.00	1.00	1.00
	S36	0.63	1.00	0.77	0.00	0.00	0.00	-	-	-	0.88	0.92	0.90	1.00	0.89	0.94

Table 8.1: *Comparison of TEX's effectiveness to other techniques.*

		TEX			RoadRunner			FivaTech			SM			LR		
	Site	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Sports	S37	1.00	1.00	1.00	0.00	0.00	0.00	0.36	0.99	0.52	0.83	0.13	0.23	1.00	1.00	1.00
	S38	1.00	1.00	1.00	0.00	0.00	0.00	-	-	-	1.00	1.00	1.00	1.00	1.00	1.00
	S39	0.97	0.99	0.98	0.00	0.00	0.00	0.99	0.88	0.93	0.94	0.94	0.94	0.71	0.71	0.71
	S40	1.00	1.00	1.00	0.93	1.00	0.97	0.53	0.81	0.64	0.71	0.71	0.71	0.86	0.86	0.86
	S41	0.97	1.00	0.98	0.00	0.00	0.00	-	-	-	0.89	0.89	0.89	0.89	0.89	0.89
EXALG	S42	0.93	0.73	0.82	0.27	0.33	0.30	0.60	0.67	0.63	0.98	1.00	0.99	0.97	1.00	0.98
	S43	1.00	0.90	0.95	0.92	0.92	0.92	0.91	0.94	0.92	0.92	0.90	0.91	0.92	0.51	0.66
	S44	1.00	0.98	0.99	0.99	0.99	0.99	1.00	1.00	1.00	0.94	0.72	0.82	0.92	0.58	0.71
	S45	0.99	0.99	0.99	0.90	0.92	0.91	0.97	0.99	0.98	0.81	0.89	0.85	0.64	0.75	0.69
	S46	1.00	1.00	1.00	0.37	0.39	0.38	0.24	0.82	0.37	0.65	0.28	0.39	0.67	0.32	0.43
	S47	0.97	1.00	0.98	0.00	0.00	0.00	0.83	1.00	0.91	0.70	0.12	0.20	0.70	0.12	0.20
	S48	0.98	0.55	0.70	0.00	0.00	0.00	0.99	1.00	0.99	0.99	0.46	0.63	0.65	0.33	0.44
	S49	0.99	0.99	0.99	-	-	-	0.82	0.80	0.81	0.94	0.82	0.88	0.99	0.99	0.99
	S50	0.95	0.97	0.96	0.98	0.99	0.98	0.99	0.41	0.58	0.72	0.03	0.06	0.99	0.99	0.99
RISE	S51	0.95	0.94	0.94	1.00	1.00	1.00	-	-	-	0.81	0.77	0.79	0.68	0.98	0.80
	S52	0.84	0.38	0.52	0.00	0.00	0.00	0.53	0.69	0.60	0.37	0.43	0.40	1.00	1.00	1.00
	S53	1.00	0.82	0.90	0.96	0.56	0.71	0.49	0.34	0.40	0.83	0.82	0.82	0.60	0.67	0.63
	S54	0.97	0.92	0.94	0.00	0.00	0.00	0.83	0.57	0.68	0.87	0.49	0.63	0.90	0.80	0.85
	S55	1.00	0.86	0.92	0.00	0.00	0.00	1.00	0.98	0.99	0.81	0.63	0.71	0.81	0.72	0.76

Table 8.1: *Comparison of TEX's effectiveness to other techniques. (Cont'd)*

procedure and Figure §8.7 illustrates the F1 measures we gathered for every pair of datasets and techniques and the number of errors in a radial chart. Note that the p-value of the correlation coefficients is smaller than the standard significance level $\alpha = 0.05$ except for the case of RoadRunner and FiVaTech; in these cases the correlation coefficient is negative, which means that the effectiveness of these techniques is expected to decrease as the number of errors in the input documents increases. As a conclusion, our experiments do not show any statistical evidence that the effectiveness of our technique is sensible to malformed input documents.

8.4.3 Efficiency analysis

Table §8.3 shows our results regarding efficiency of TEX and the other techniques. The table shows the learning time LT and the extraction time ET for each technique. (Note that TEX does not learn extraction rules, so we use NA in the corresponding column to mean not applicable. All timings are expressed in seconds.) The first few rows provide a summary in terms of mean

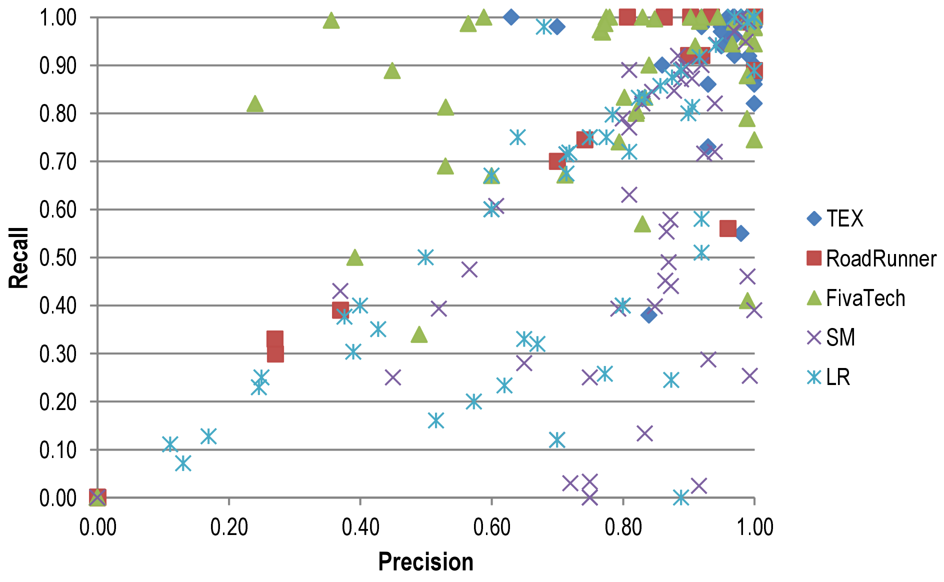
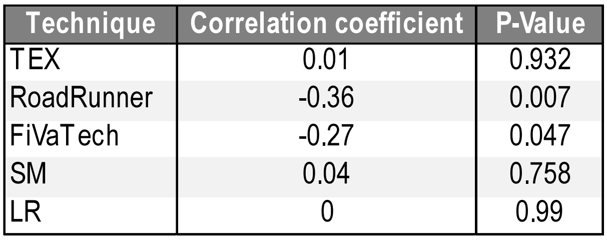


Figure 8.6: *Precision versus recall in our experiments regarding TEX.*



Technique	Correlation coefficient	P-Value
TEX	0.01	0.932
RoadRunner	-0.36	0.007
FiVaTech	-0.27	0.047
SM	0.04	0.758
LR	0	0.99

Table 8.2: *Impact of errors on TEX effectiveness.*

and standard deviation for each of the variables. The learning time and the extraction time do not account for the time consumed by JTidy in the case of RoadRunner and FiVaTech, neither account them for the tokenisation time in the case of TEX, SM, and LR. The reason is that the time to clean or tokenise a document is not actually an intrinsic feature of the proposals being analysed.

The results in Table §8.3 support the idea that TEX is more effective regarding the extraction time than the other techniques, but has a similar extraction time to RoadRunner. (In the table, 0.00 ± 0.00 means that the time was less than one millisecond, but the resolution of our timer was not enough

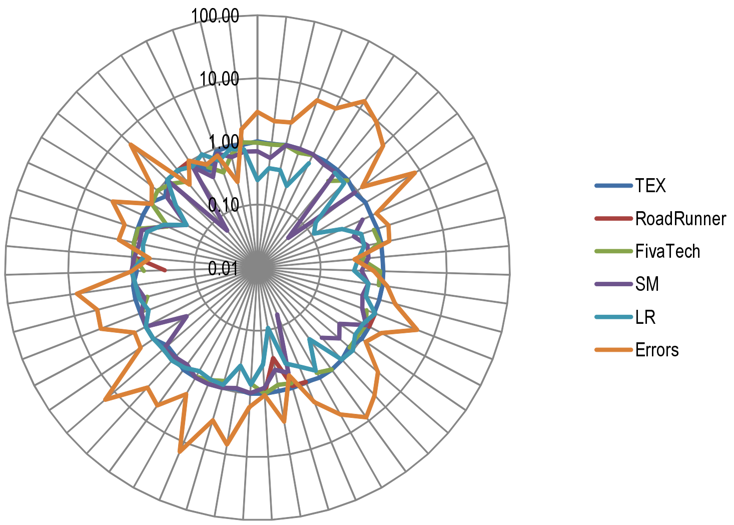
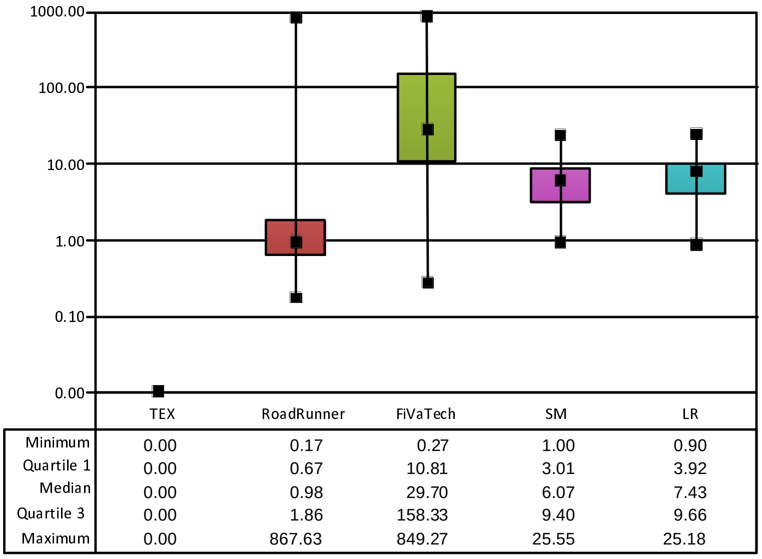
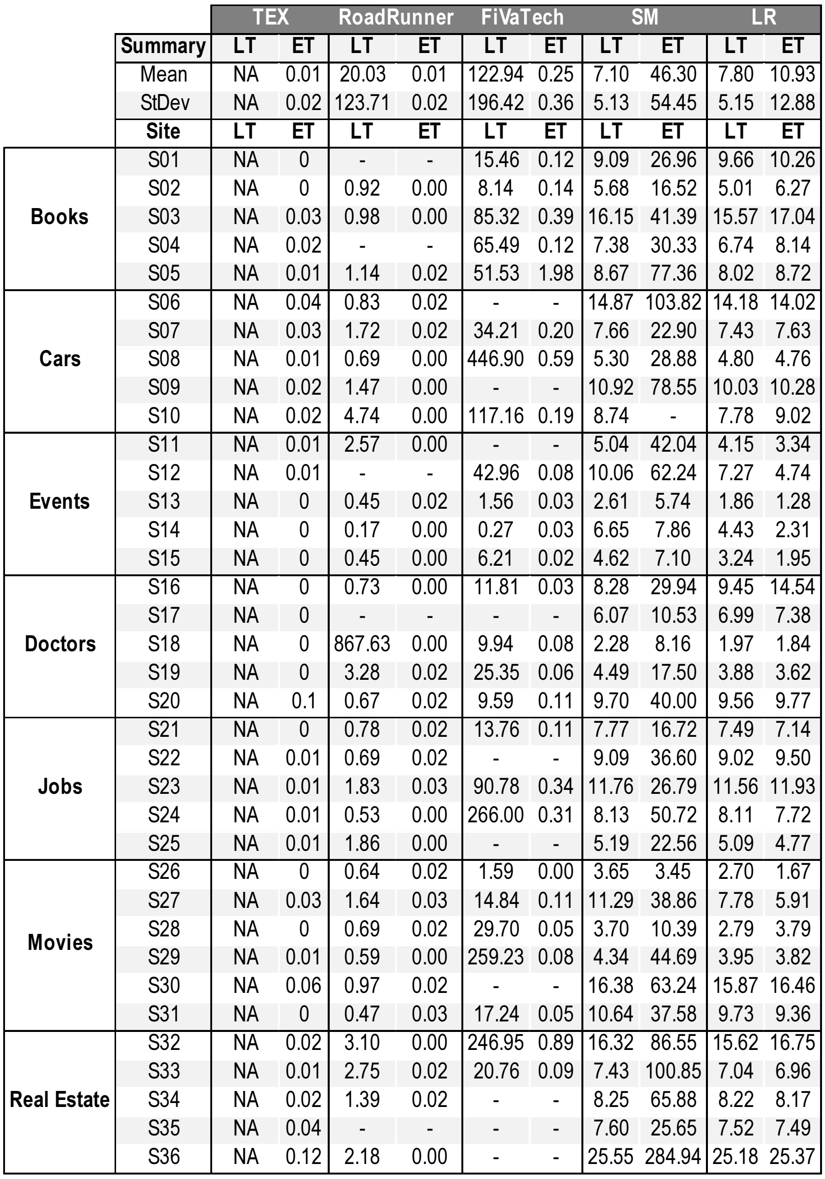


Figure 8.7: *Correlation from number of errors to the* F1 *measure in TEX.*



	TEX	RoadRunner	FiVaTech	SM	LR
Minimum	0.00	0.17	0.27	1.00	0.90
Quartile 1	0.00	0.67	10.81	3.01	3.92
Median	0.00	0.98	29.70	6.07	7.43
Quartile 3	0.00	1.86	158.33	9.40	9.66
Maximum	0.00	867.63	849.27	25.55	25.18

Figure 8.8: *Comparison of learning times regarding TEX.*



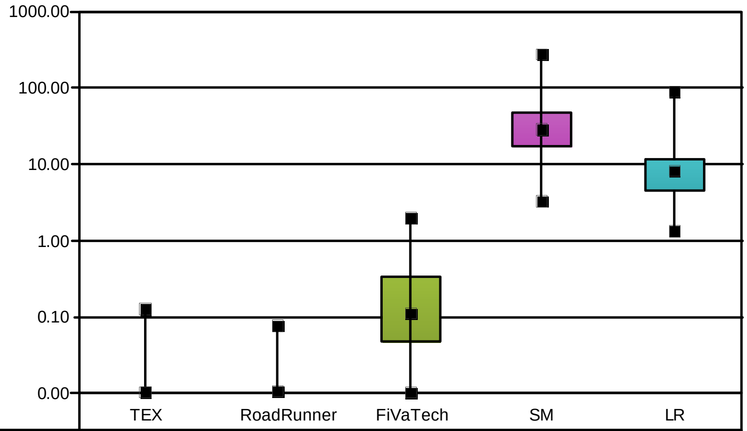
		TEX		RoadRunner		FiVaTech		SM		LR	
	Summary	LT	ET	LT	ET	LT	ET	LT	ET	LT	ET
	Mean	NA	0.01	20.03	0.01	122.94	0.25	7.10	46.30	7.80	10.93
	StDev	NA	0.02	123.71	0.02	196.42	0.36	5.13	54.45	5.15	12.88
	Site	LT	ET	LT	ET	LT	ET	LT	ET	LT	ET
Books	S01	NA	0	-	-	15.46	0.12	9.09	26.96	9.66	10.26
	S02	NA	0	0.92	0.00	8.14	0.14	5.68	16.52	5.01	6.27
	S03	NA	0.03	0.98	0.00	85.32	0.39	16.15	41.39	15.57	17.04
	S04	NA	0.02	-	-	65.49	0.12	7.38	30.33	6.74	8.14
	S05	NA	0.01	1.14	0.02	51.53	1.98	8.67	77.36	8.02	8.72
Cars	S06	NA	0.04	0.83	0.02	-	-	14.87	103.82	14.18	14.02
	S07	NA	0.03	1.72	0.02	34.21	0.20	7.66	22.90	7.43	7.63
	S08	NA	0.01	0.69	0.00	446.90	0.59	5.30	28.88	4.80	4.76
	S09	NA	0.02	1.47	0.00	-	-	10.92	78.55	10.03	10.28
	S10	NA	0.02	4.74	0.00	117.16	0.19	8.74	-	7.78	9.02
Events	S11	NA	0.01	2.57	0.00	-	-	5.04	42.04	4.15	3.34
	S12	NA	0.01	-	-	42.96	0.08	10.06	62.24	7.27	4.74
	S13	NA	0	0.45	0.02	1.56	0.03	2.61	5.74	1.86	1.28
	S14	NA	0	0.17	0.00	0.27	0.03	6.65	7.86	4.43	2.31
	S15	NA	0	0.45	0.00	6.21	0.02	4.62	7.10	3.24	1.95
Doctors	S16	NA	0	0.73	0.00	11.81	0.03	8.28	29.94	9.45	14.54
	S17	NA	0	-	-	-	-	6.07	10.53	6.99	7.38
	S18	NA	0	867.63	0.00	9.94	0.08	2.28	8.16	1.97	1.84
	S19	NA	0	3.28	0.02	25.35	0.06	4.49	17.50	3.88	3.62
	S20	NA	0.1	0.67	0.02	9.59	0.11	9.70	40.00	9.56	9.77
Jobs	S21	NA	0	0.78	0.02	13.76	0.11	7.77	16.72	7.49	7.14
	S22	NA	0.01	0.69	0.02	-	-	9.09	36.60	9.02	9.50
	S23	NA	0.01	1.83	0.03	90.78	0.34	11.76	26.79	11.56	11.93
	S24	NA	0.01	0.53	0.00	266.00	0.31	8.13	50.72	8.11	7.72
	S25	NA	0.01	1.86	0.00	-	-	5.19	22.56	5.09	4.77
Movies	S26	NA	0	0.64	0.02	1.59	0.00	3.65	3.45	2.70	1.67
	S27	NA	0.03	1.64	0.03	14.84	0.11	11.29	38.86	7.78	5.91
	S28	NA	0	0.69	0.02	29.70	0.05	3.70	10.39	2.79	3.79
	S29	NA	0.01	0.59	0.00	259.23	0.08	4.34	44.69	3.95	3.82
	S30	NA	0.06	0.97	0.02	-	-	16.38	63.24	15.87	16.46
	S31	NA	0	0.47	0.03	17.24	0.05	10.64	37.58	9.73	9.36
Real Estate	S32	NA	0.02	3.10	0.00	246.95	0.89	16.32	86.55	15.62	16.75
	S33	NA	0.01	2.75	0.02	20.76	0.09	7.43	100.85	7.04	6.96
	S34	NA	0.02	1.39	0.02	-	-	8.25	65.88	8.22	8.17
	S35	NA	0.04	-	-	-	-	7.60	25.65	7.52	7.49
	S36	NA	0.12	2.18	0.00	-	-	25.55	284.94	25.18	25.37

Table 8.3: *Comparison of TEX's efficiency to other techniques.*

to measure such small amounts of time.) Figure §8.8 illustrates this idea: note that the learning time of TEX is set to zero since it does not learn extraction rules. The extraction time is similar to RoadRunner's, but clearly smaller than the other techniques'; the dispersion from the minimum values to the first quartile and from the third quartile to the maximum are also a clear indication

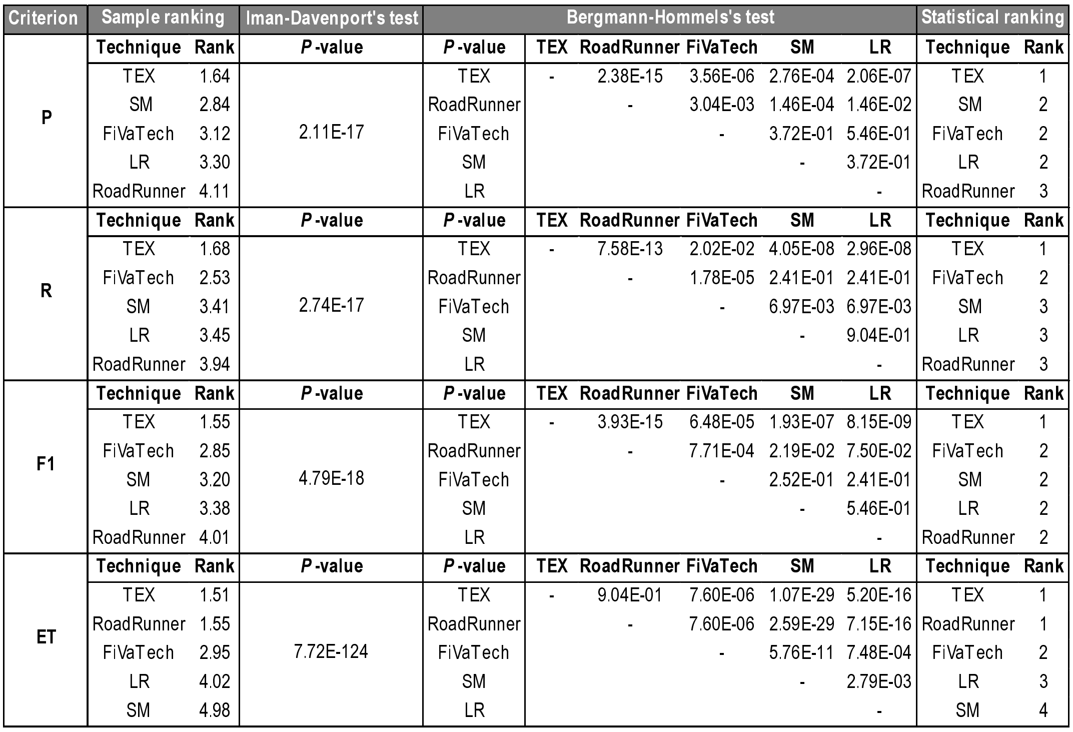
		TEX		RoadRunner		FiVaTech		SM		LR	
	Site	LT	ET	LT	ET	LT	ET	LT	ET	LT	ET
Sports	S37	NA	0.02	1.37	0.02	14.68	0.06	5.51	17.43	5.34	4.96
	S38	NA	0.01	0.64	0.00	-	-	5.99	33.23	5.83	5.71
	S39	NA	0.06	1.11	0.02	111.03	0.83	19.14	182.38	18.75	18.33
	S40	NA	0.02	1.65	0.00	159.39	0.39	10.14	39.91	9.64	9.56
	S41	NA	0.04	2.01	0.02	-	-	11.54	47.80	11.04	10.87
EXALG	S42	NA	0	0.55	0.02	2.29	0.05	1.06	10.09	19.47	8.10
	S43	NA	0.01	0.51	0.02	10.81	0.05	1.34	11.64	3.53	3.70
	S44	NA	0	0.98	0.03	246.97	0.11	3.42	23.35	15.66	10.22
	S45	NA	0.01	0.28	0.02	0.89	0.02	1.00	3.32	1.79	2.28
	S46	NA	0.01	1.15	0.02	132.24	0.27	1.98	22.06	9.67	16.96
	S47	NA	0.04	2.51	0.02	577.97	0.48	2.23	16.04	5.44	19.55
	S48	NA	0.11	0.95	0.02	158.33	0.06	1.29	19.61	3.60	6.13
	S49	NA	0.01	-	-	706.64	0.76	3.68	34.15	7.47	31.54
	S50	NA	0	1.31	0.03	-	0.56	1.40	26.22	1.29	10.42
RISE	S51	NA	0	12.04	0.08	-	-	1.50	166.23	2.68	89.14
	S52	NA	0	0.89	0.02	14.41	0.06	1.39	13.34	2.00	2.40
	S53	NA	0.02	11.23	0.08	849.27	0.37	1.73	216.36	1.64	35.08
	S54	NA	0	0.37	0.00	4.24	0.05	2.18	23.45	0.90	2.73
	S55	NA	0	33.60	0.02	158.48	0.06	2.57	20.39	13.60	19.75

Table 8.3: *Comparison of TEX's efficiency to other techniques. (Cont'd)*



	TEX	RoadRunner	FiVaTech	SM	LR
Minimum	0.00	0.00	0.00	3.32	1.28
Quartile 1	0.00	0.00	0.05	16.90	4.75
Median	0.01	0.02	0.11	27.92	8.10
Quartile 3	0.02	0.02	0.31	47.02	11.40
Maximum	0.12	0.08	1.98	284.94	89.14

Figure 8.9: *Comparison of extraction times regarding TEX.*



Criterion	Sample ranking		Iman-Davenport's test	Bergmann-Hommels's test						Statistical ranking	
P	Technique	Rank	*P*-value	*P*-value	TEX	RoadRunner	FiVaTech	SM	LR	Technique	Rank
	TEX	1.64	2.11E-17	TEX	-	2.38E-15	3.56E-06	2.76E-04	2.06E-07	TEX	1
	SM	2.84		RoadRunner		-	3.04E-03	1.46E-04	1.46E-02	SM	2
	FiVaTech	3.12		FiVaTech			-	3.72E-01	5.46E-01	FiVaTech	2
	LR	3.30		SM				-	3.72E-01	LR	2
	RoadRunner	4.11		LR					-	RoadRunner	3
R	Technique	Rank	*P*-value	*P*-value	TEX	RoadRunner	FiVaTech	SM	LR	Technique	Rank
	TEX	1.68	2.74E-17	TEX	-	7.58E-13	2.02E-02	4.05E-08	2.96E-08	TEX	1
	FiVaTech	2.53		RoadRunner		-	1.78E-05	2.41E-01	2.41E-01	FiVaTech	2
	SM	3.41		FiVaTech			-	6.97E-03	6.97E-03	SM	3
	LR	3.45		SM				-	9.04E-01	LR	3
	RoadRunner	3.94		LR					-	RoadRunner	3
F1	Technique	Rank	*P*-value	*P*-value	TEX	RoadRunner	FiVaTech	SM	LR	Technique	Rank
	TEX	1.55	4.79E-18	TEX	-	3.93E-15	6.48E-05	1.93E-07	8.15E-09	TEX	1
	FiVaTech	2.85		RoadRunner		-	7.71E-04	2.19E-02	7.50E-02	FiVaTech	2
	SM	3.20		FiVaTech			-	2.52E-01	2.41E-01	SM	2
	LR	3.38		SM				-	5.46E-01	LR	2
	RoadRunner	4.01		LR					-	RoadRunner	2
ET	Technique	Rank	*P*-value	*P*-value	TEX	RoadRunner	FiVaTech	SM	LR	Technique	Rank
	TEX	1.51	7.72E-124	TEX	-	9.04E-01	7.60E-06	1.07E-29	5.20E-16	TEX	1
	RoadRunner	1.55		RoadRunner		-	7.60E-06	2.59E-29	7.15E-16	RoadRunner	1
	FiVaTech	2.95		FiVaTech			-	5.76E-11	7.48E-04	FiVaTech	2
	LR	4.02		SM				-	2.79E-03	LR	3
	SM	4.98		LR					-	SM	4

Table 8.4: *Results of ranking TEX statistically.*

that our technique performs more homogeneously than the others regarding extraction time, with the only exception of RoadRunner, c.f. Figure §8.9.

8.5 Statistical analysis

Once we have evaluated the proposals on datasets, it is necessary to perform a statistical ranking to check whether the conclusions we have drawn are statistically significant or not, cf. Section §6.5. We need perform a statistical ranking regarding our performance measures and determine if there is a significant correlation from the number of errors in the web documents to the effectiveness of the techniques we have evaluated.

We have conducted a Shapiro-Wilk test at the standard significance level $\alpha = 0.05$ on every measure and we have found out that the majority of them do not behave normally. For instance, Shapiro-Wilk's statistic regarding the normality of TEX's precision is $W(55) = 0.59$, whose p-value is 0.00, which is a strong indication that the data is not distributed normally. This is not surprising at all; a quick look at the scatter plot in

Figure §8.6 makes it clear that these cloud of points are far from a Gaussian circle. As a conclusion, we have performed a non-parametric analysis whose results are presented in Table §8.4. Note that the p-value of Iman-Devenport's statistic is nearly zero in every case, which is a strong indication that there are statistically significant differences in the ranks we have computed from our experiments. There is also a strong statistical evidence that TEX outperforms RoadRunner, FiVaTech, SM, and LR in terms of precision, recall, and, consequently, F1 measure. For the sake of readability, we also provide an explicit ranking in the last column. Regarding extraction time TEX has a performance similar to that one achieved by RoadRunner, but outperforms the other techniques. Note that regarding F1 measure, Bergmann-Hommel's test found that there is a strong statistical indication that TEX outperforms the other techniques, but that there is not enough evidence to confirm that the difference between the F1 measure of the other techniques is significant when these techniques are compared to each other.

8.6 Summary

In this chapter, we have presented an information extraction proposal called TEX that focuses on extracting attributes. It is based on the idea that web documents that are generated by the same server side template share tokens, and that these tokens contain irrelevant information since they are parts of the server-side template that was used to generate them.

TEX is a completely unsupervised information extractor that saves end users from the burden of annotating training examples to learn extraction rules, and from maintaining extraction rules. It allows to work on malformed web documents since it does not require converting HTML code into XHTML or to build DOM trees, which reduces its extraction time. Furthermore, it does not need the information in the input web documents to be formatted using repetitive patterns.

We have studied the complexity of TEX and demonstrated that it is computationally tractable. Our empirical analysis of TEX on a collection of real-world datasets has proven that our technique achieves a very high precision and recall, which are very close to 100%. Our comparison and statistical analysis has shown that our technique performs better than other techniques in the literature.

Chapter 9

Extracting information records with Trinity

Simplicity is the ultimate sophistication.

Leonardo da Vinci, Italian Renaissance polymath: painter, sculptor, architect, musician, scientist, mathematician, engineer, inventor, anatomist, geologist, cartographer, botanist, and writer (1452-1519)

Trinity is an information extractor that learns a regular expression that represents the template used to generate a number of similar web documents. The chapter is organised as follows: Section §9.1 introduces the chapter; Section §9.2 describes the algorithms on which Trinity relies; Section §9.3 reports on their complexity; Section §9.4 reports on our experimental analysis we have conducted to evaluate Trinity empirically; Section §9.5 reports on the statistical analysis conducted to rank Trinity; finally, Section §9.6 summarises this chapter.

9.1 Introduction

Software engineers use scripts to retrieve information from server-side databases and fill in templates that present the information retrieved in human-friendly formats. These templates are based on HTML, and they usually introduce irrelevant information, e.g., formatting tags, headers, or footers. This makes it difficult to extract relevant information from web documents.

In this chapter, we introduce a technique called Trinity; it allows to learn a regular expression that describes the template used to generate a number of similar web documents and the information in these documents. It works on two or more documents and compares them in order to discover shared patterns that are common to all of the input documents and, thus, are not likely to contain any relevant information.

Our proposal relies on a multi-string alignment algorithm that has proven to be very effective and efficient in practice. Contrarily to the other proposals, Trinity does not require the input web documents to be translated into DOM trees and thus does not require the input documents to be corrected so that they are well-formed HTML. We have conducted a series of experiments with 2 084 web documents from 55 real-world web sites and our results confirm that our proposal can achieve a mean precision as high as 96%, a mean recall as high as 95%, with a mean learning time of 0.13 seconds and a mean extraction time of 0.02 seconds using a commodity computer. We conducted the same experiments using other well-known techniques in the literature, and our conclusion is that our proposal outperforms them.

9.2 Algorithms

In this section, we describe the rule learning algorithm that lies at the heart of our proposal. Trinity searches for shared patterns in the input web documents and builds a structure called trinary tree in which it tries to align the prefixes, suffixes, and the separators between successive occurrences of the shared pattern, if any. This structure is then used to learn a regular expression that models the template that was used to generate the input web documents and the schema of the information inside these web documents. Figure §9.1 illustrates a running example, in which we assume that our algorithm is executed on the root node N1, which contains three input web

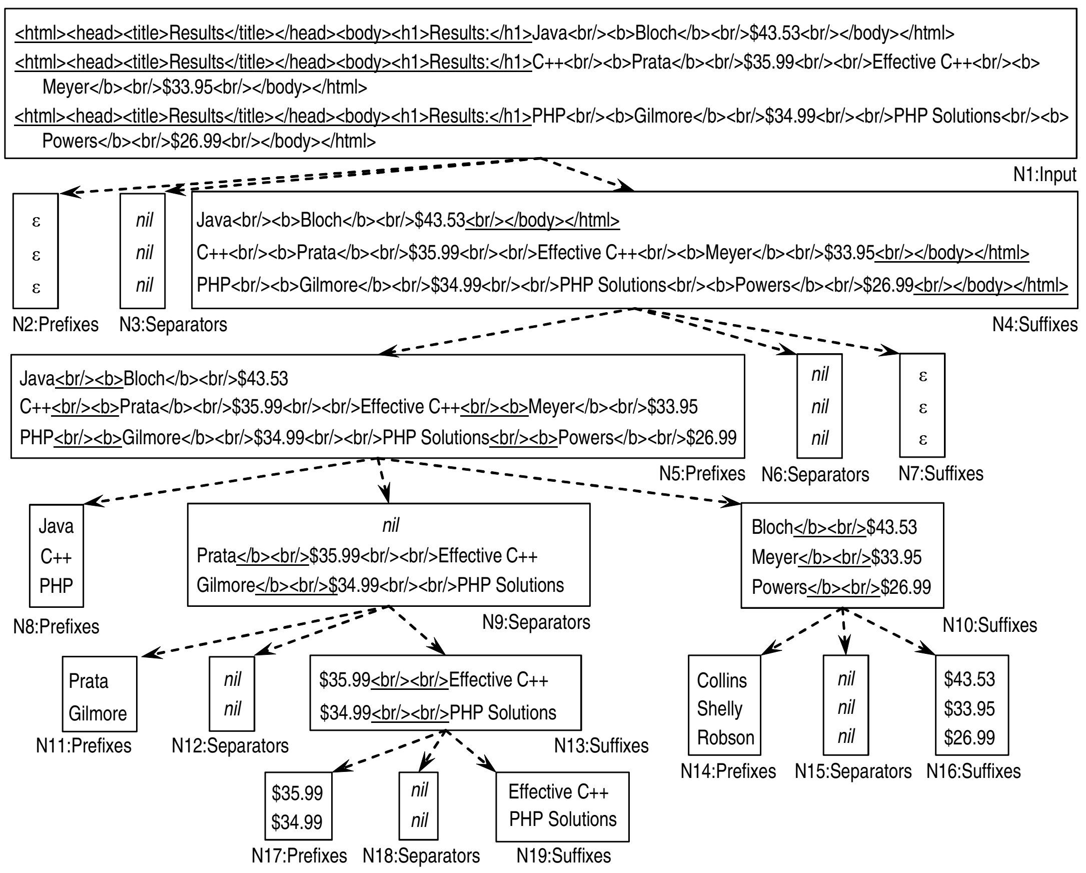

<html><head><title>Results</title></head><body><h1>Results:</h1>Java<br/><b>Bloch</b><br/>$43.53<br/></body></html>
<html><head><title>Results</title></head><body><h1>Results:</h1>C++<br/><b>Prata</b><br/>$35.99<br/><br/>Effective C++<br/><b>Meyer</b><br/>$33.95<br/></body></html>
<html><head><title>Results</title></head><body><h1>Results:</h1>PHP<br/><b>Gilmore</b><br/>$34.99<br/><br/>PHP Solutions<br/><b>Powers</b><br/>$26.99<br/></body></html>
N1:Input
ε ε ε
N2:Prefixes
nil nil nil
N3:Separators
Java<br/><b>Bloch</b><br/>$43.53<br/></body></html>
C++<br/><b>Prata</b><br/>$35.99<br/><br/>Effective C++<br/><b>Meyer</b><br/>$33.95<br/></body></html>
PHP<br/><b>Gilmore</b><br/>$34.99<br/><br/>PHP Solutions<br/><b>Powers</b><br/>$26.99<br/></body></html>
N4:Suffixes
Java<br/><b>Bloch</b><br/>$43.53
C++<br/><b>Prata</b><br/>$35.99<br/><br/>Effective C++<br/><b>Meyer</b><br/>$33.95
PHP<br/><b>Gilmore</b><br/>$34.99<br/><br/>PHP Solutions<br/><b>Powers</b><br/>$26.99
N5:Prefixes
nil nil nil
N6:Separators
ε ε ε
N7:Suffixes
Java C++ PHP
N8:Prefixes
nil
Prata</b><br/>$35.99<br/><br/>Effective C++
Gilmore</b><br/>$34.99<br/><br/>PHP Solutions
N9:Separators
Bloch</b><br/>$43.53
Meyer</b><br/>$33.95
Powers</b><br/>$26.99
N10:Suffixes
Prata Gilmore
N11:Prefixes
nil nil
N12:Separators
$35.99<br/><br/>Effective C++
$34.99<br/><br/>PHP Solutions
N13:Suffixes
Collins Shelly Robson
N14:Prefixes
nil nil nil
N15:Separators
$43.53 $33.95 $26.99
N16:Suffixes
$35.99 $34.99
N17:Prefixes
nil nil
N18:Separators
Effective C++ PHP Solutions
N19:Suffixes

Figure 9.1: *A sample trinary tree. (Shared patterns are underlined.)*

documents. Note that these are intentionally simple HTML documents that are used for illustration purpose only. In the following subsections, we first introduce the structures used by the algorithm and then describe it.

9.2.1 Structures

Our proposal requires to tokenise the input web documents, but it is not bound with a particular tokenisation schema. Our implementation and our experiments were carried out using the same tokenisation schema as in TEX. A sequence of tokens is called Text. We define a trinary tree as a collection of Nodes, each of which is a tuple of the form (T, a, p, e, s), where T is a collection of Text, a is of type Text and contains a shared pattern in T, p is a Node called prefixes, e is a Node called separators, and s is

```
1: Trinity($root$: Node; $min$, $max$: int): Regex, Schema
2:    createTrinaryTree($root$, $min$, $max$)
3:    $template$ = "^" + learnTemplate($root$, "") + "$"
4:    $schema$ = "<?xml version=\"1.0\"> <schema>" + learnSchema($root$, "") + "</schema>"
5: return $template$ , $schema$
```

Program 9.1: *Algorithm Trinity.*

a Node called suffixes. The root node of a trinary tree contains a collection of Text, such that each Text is the tokenisation of an input document; leaf nodes are of the form $(T, \epsilon, nil, nil, nil)$, where ϵ denotes an empty sequence of Texts and nil denotes either a missing Node or Text .

In Figure §9.1, N1 is the root node, and it contains three Texts that represent three input documents; N2, N3, and N4 are the child nodes of the root Node, namely: prefixes, separators, and suffixes.

9.2.2 Main algorithm

We present the main algorithm in Program §9.1. It works on a Node called $root$, and a range of integers called min and max, which limit the search for shared patterns to those of size max down to min. The algorithm works in three steps: line §2 expands the input root Node to create a full trinary tree, line §3 uses the previous tree to learn a regular expression that models the template used to generate the input documents, and line §4 uses it to learn the information schema. In the following subsections, we provide additional details on the ancillary algorithms.

9.2.3 Creating a trinary tree

Algorithm createTrinaryTree searches for shared patterns of size max down to min in a Node. If a shared pattern of size s is found, then the Texts inside the node are partitioned to create three children for the input node, and the algorithm is executed recursively on them to search for patterns of size s down to min. For instance, assume that it is invoked on Node N1 in Figure §9.1, and that it has to search for a shared pattern whose size is in the range 12 down to one. The algorithm searches for a shared pattern of size 12 tokens, 11 tokens, until it finds the following 10-token pattern: <html><head><title>Results</title></head><body><h1>

```
 1: createTrinaryTree($node$: Node; $min$, $max$: int)
 2:   $expanded$ = false
 3:   $size$ = $max$
 4:   while $size \geq min$ and not $expanded$ do
 5:     $expanded$ = expand($node$, $size$)
 6:     $size$ = $size - 1$
 7:   end
 8:   if $expanded$ then
 9:     $leaves$ = getLeaves($node$)
10:     foreach $leaf$ in $leaves$ do
11:       createTrinaryTree($leaf$, $min$, $size + 1$)
12:     end
13:   end
14: end
```

Program 9.2: *Algorithm createTrinaryTree.*

Results:</h1>. The algorithm tries to expand Node N1 into three additional nodes, namely: N2, N3, and N4. Since the shared pattern is found at the beginning of the Texts in N1 and it is not repeated in any of them, then Node N2 contains three empty Texts that we denote as ϵ. Node N3, which contains the separators between the occurrences of the shared pattern in each Text, only contains three nil Texts that indicate that there are not actually any separators; contrarily, if there are three suffixes that are stored in Node N4. Then, the algorithm is applied recursively to N2, N3, and N4 to search for shared patterns of size 10 down to 1. N2 and N3 are not processed again since they only contain empty or nil Texts; only Node N4, whose Texts share the 3-token pattern <br/></body></html>, is expanded again to create Nodes N5, N6, and N7. The same procedure is applied as many times as necessary until no more shared patterns are discovered.

We present Algorithm createTrinaryTree in Program §9.2. The loop at lines §4-§7 iterates over every possible size from max down to min until Algorithm expand finds a shared pattern of the current size. If such a pattern is found, then expand creates the child nodes of the input Node and returns $true$, which breaks the loop after decreasing $size$; if no shared pattern is found, then the loop decreases $size$, and invokes expand again with the new size. If a shared pattern is found and the loop is broken, the leaves of the input node are obtained at line §9, and the loop at lines §10-§12 iterates over them and executes the same algorihm recursively. The algorithm is executed recursively

```
1: expand(node: Node; s: int): boolean
2:     result = false
3:     if size(node) > 1 then
4:         map, pattern = findPattern(node, s)
5:         if $map \neq \{\}$ then
6:             result = true
7:             createChildren(node, map, pattern)
8:         end
9:     end
10: return result
```

Program 9.3: *Algorithm expand.*

at line §11 taking $size + 1$ as max parameter since the loop at lines §4-§7 has decreased it after invoking expand at line §5. Note that min and max introduce a bias to the search algorithm if min is greater than one or max is less than the size of the shortest input document. Our experimental results prove that this bias helps effectively reduce the amount of effort required to extract information from typical web documents, without sacrificing effectiveness.

Algorithm expand: This algorithm searches for a shared pattern of a given size in a Node; if such a pattern is found, it then expands this Node by creating its child nodes (prefixes, separators, and suffixes). For instance, assume that Algorithm expand is invoked on Node N5 in Figure §9.1 to search for a shared pattern of size two tokens. The algorithm can easily find that pattern <br/><b> is the first 2-token pattern shared amongst the Texts in Node N5. It expands Node N5 by creating its three children, namely: N8, N9, and N10. N8 contains the prefixes of each Text in N5, i.e., the Text fragments between the beginning of each Text and the first occurrence of the shared pattern; N9 contains the separators of each Text in N5, i.e., the Text fragments between successive occurrences of the shared pattern in each Text in N5, or nil if only one occurrence is found; N10 contains the suffixes of each Text in N5, i.e., the Text fragments between the last occurrence of the shared pattern and the end of each Text.

We present Algorithm expand in Program §9.3. Line §3 checks if $node$ contains more than one Text; if so, it searches for a shared pattern by invoking Algorithm findPattern at line §4, which is described in the next section; the algorithm checks if findPattern has found a shared pattern at line §5, and invokes Algorithm createChildren at line §7. Algorithm createChildren, which is

```
 1: findPattern($node$: Node; $s$: int): Map<Text, List⟨int⟩>, Text
 2:     $found$ = false
 3:     $base$ = findShortestText($node$)
 4:     $pattern$ = $\langle\rangle$
 5:     for $i$ = $0$ until size($base$) - $s$ while not $found$ do
 6:         $map$ = {}
 7:         $found$ = true
 8:         foreach non-empty $text$ in $node$ while $found$ do
 9:             $matches$ = findMatches($text$, $base$, $i$, $s$)
10:             $found$ = size($matches$) > 0
11:             $map = map \cup \{text \mapsto matches\}$
12:         end
13:         if $found$ then
14:             $pattern$ = subsequence($base$, $i$, $s$)
15:         end
16:     end
17: return $map$, $pattern$
```

Program 9.4: *Algorithm findPattern.*

described below, partitions the Text collection inside the input Node, and creates its child nodes; if no shared pattern is found, then the input Node remains unchanged.

Algorithm findPattern: Algorithm findPattern works on a Node $node$ and a pattern size s. Its goal is to find a pattern of size s that has at least one occurrence in each non-empty Text in $node$. For instance, assume that the algorithm is invoked on Node N5 in Figure §9.1 to search for a shared pattern of size two. The shortest Text in N5 is the first one (Java<br/><b>Bloch</b><br/>$43.53), which makes it the basis to search for a shared pattern. The algorithm first searches for Java<br/> in every Text in N5, but does not find it; then, the search starts from the second token, and the algorithm searches for <br/><b>, which is shared amongst all of the Texts in N5. The algorithm returns a map from Text onto lists of integers and a pattern; the map represents the positions where the search pattern is found in each Text in the input Node. In our example, this map is $\{t_1 \mapsto \langle 1\rangle, t_2 \mapsto \langle 1, 10\rangle, t_3 \mapsto \langle 1, 10\rangle\}$, where t_1, t_2, and t_3 denote the first, the second, and the third Text in N5 respectively.

We present Algorithm findPattern in Program §9.4. First it searches for the shortest non-empty Text inside $node$ at line §3 and stores it in $base$. The main

```
createChildren(node: Node; map: Map⟨Text, List⟨int⟩⟩; pattern: Text)
    prefixes = new Node()
    separators = new Node()
    suffixes = new Node()
    setPattern(node, pattern)
    foreach text in node do
        matches = getValue(map, text)
        add(prefixes, computePrefix(matches, text))
        add(separators, computeSeparators(matches, text))
        add(suffixes, computeSuffix(matches, text))
    end
    setPrefix(node, prefixes)
    setSeparators(node, separators)
    setSuffix(node, suffixes)
end
```

Program 9.5: *Algorithm createChildren.*

loop at lines §5-§16 allows to implement a sliding window over $base$: index i iterates from 0 until size$(base) - s$ as long as no shared pattern is found, i.e., it searches for all patterns of size s in $base$. The actual search is performed in the inner loop at lines §8-§12: in this loop, the algorithm iterates over every Text in the input Node, and finds all of the matches of the subsequence of $base$ that starts at position i and has size s. Algorithm findMatches is implemented using the well-known Knuth-Pratt-Morris pattern search algorithm [86]. This algorithm returns a list of integers that indicate the non-overlapping positions at which the previous subsequence of $base$ matches $text$; if there is at least one match, we record it in variable $result$ and go ahead to examine the next Text in $node$; otherwise, the inner loop finishes and the outer loop slides the window on $base$ and resets map, if possible. If the algorithm returns an empty map, this means that no shared pattern has been found.

Algorithm createChildren: When a shared pattern is found in a Node, Algorithm createChildren creates three child nodes, namely: prefixes, separators, and suffixes, and adds them to their parent Node. For instance, assume that the algorithm is invoked on Node N5 in Figure §9.1 to create its children. The algorithm creates the prefixes Node N8, which contains the fragments from the beginning of each Text in N5 until the first occurrence of the shared pattern; it creates the separators Node N9, which contains the fragments of each Text in N5 between consecutive occurrences of the shared pattern, if any; finally, it creates the suffixes Node N10, which contains the fragments from the last occurrence of the shared pattern until the end of each Text in N5.

We present Algorithm createChildren in Program §9.5. This algorithm works on a Node, a map, and a pattern. Lines §2-§4 create three empty Nodes, namely *prefixes*, *separators*, and *suffixes*. Line §5 sets the node's shared pattern to *pattern*. Then, the loop at lines §6-§11 iterates over the Texts in map: for each Text *text* in the map, lines §8-§10 get the matches where the shared pattern occurs, computes the prefix, separators, and suffix of the pattern in *text*, and adds them to the *prefixes*, *separators*, and *suffixes* Nodes respectively. If *pattern* is found at the beginning of *text*, the prefix is then an empty Text; if *pattern* is found at the end of *text*, the suffix is then an empty Text; if two occurrences of *pattern* are consecutive in *text*, their separator is an empty Text, but if *text* contains only one occurrence of *pattern*, we add the special value *nil* to *separators*. We do not provide a pseudocode to the algorithms to compute prefixes, separators, and suffixes since they are quite straightforward.

9.2.4 Algorithm learnTemplate

Algorithm learnTemplate works on a trinary tree *node*, constructs a regular expression that represents a template, and returns it, cf. Program §9.6. It relies on two ancillary algorithms, namely: isOptional and isRepeatable. A Node is optional if one or more of its Texts, but not all, are empty. A Node is repeatable if one or more of its non-empty Texts have more than one occurrence of the shared pattern, which implies that the size of its child Node separators is greater than the size of this Node. The core of the algorithm is the if-then-else sentence at lines §5-§23, which distinguishes between leaf and non-leaf nodes.

If *node* is a leaf and not all of its Texts are empty, this means that it contains variable information. In such cases, line §7 assigns a new label that represents the piece of text that has to be extracted by invoking freshLabel. Note that we enclose such label in curly brackets to denote that the pieces of text that match it have to be extracted, aka information groups. The exact notation depends on the regular expression engine used to implement the algorithm.

If the node being processed is not a leaf, then line §10 builds the regular expression that corresponds to the prefixes, which is performed recursively. The shared pattern is added to *result* at line §11 and the regular expression that corresponds to the separators is built at lines §12-§20: if the node is repeatable, then the regular expression of the separators Node is built at line §14, the shared pattern is added to *result* at line §15, and the plus or star

```
 1: learnTemplate(node: Node; result: Regex): Regex
 2:   if isOptional(node) then
 3:     result += "("
 4:   end
 5:   if isLeaf(node) then
 6:     if not allTextsEmpty(node) then
 7:       result += "{" + freshLabel() + "}"
 8:     end
 9:   else
10:     result += learnTemplate(getPrefix(node), result)
11:     result += getPattern(node)
12:     if isRepeatable(node, getSeparators(node)) then
13:       result += "("
14:       result += learnTemplate(getSeparators(node), result)
15:       result += getPattern(node)
16:       if contains(getSeparators(node), nil) then
17:         result += ")*"
18:       else
19:         result += ")+"
20:       end
21:     end
22:     result += learnTemplate(getSuffix(node), result)
23:   end
24:   if isOptional(node) then
25:     result += ")?"
26:   end
27: return result
```

Program 9.6: *Algorithm learnTemplate.*

closures are added at lines §16-§20. If the separators node contains the special value nil, this means that the shared pattern has occurred only once in at least one of the Texts in node, thus a star closure must be used; contrarily, the shared pattern has two or more occurrences in each Text in node, and a plus closure must be added. The algorithm builds now the regular expression that corresponds to the suffixes at line §22, which is performed recursively.

Lines §2-§4 and §24-§26 check if the node being processed is optional, in which case parenthesis and an optional operator are added to the resulting regular expression.

In our running example, the template learnt for the input Node in

```
(<html><head><title>Results</title></head><body>
<h1>Results:</h1>) {_A_} (<br/><b>) ( ({_B_}
(</b><br/>) {_C_} (<br/><br/>) {_D_} )? (<br/><b>) )*
{_E_} (</b><br/>) {_F_} (<br/></body></html>)
```

Figure 9.2: *The regular expression learnt for our running example.*

Figure §9.1 is illustrated in Figure §9.2.

9.2.5 Algorithm learnSchema

Algorithm learnSchema works on a trinary tree node, constructs an XML document that represents the schema of the relevant information. (A formal definition of the schema can be found elsewhere [7].) We present Algorithm learnSchema in Program §9.7. The core is the if-then-else sentence at lines §5-§25, which distinguishes between leaf and non-leaf nodes.

If node is a leaf and not all of its Texts are empty, this means that it contains information of basic type (a sequence of tokens). In such cases, line §7 assigns an ID to the information represented by this node and considers that it is of a basic type. If the node being processed is not a leaf, then line §10 assigns tuple type to the information represented by this node, i.e., the information in is prefixes, separators, and suffixes. Lines §12-§22 check if the separators node is repeatable, then line §13 assigns the type set to the information represented by this node. Line §23 builds the schema for the suffixes Node recursively. Lines §2-§4 and §24-§26 check if the node being processed is optional, in which case the appropriate XML tags are added to the schema.

In our running example, the schema learnt for the input Node in Figure §9.1 is illustrated in Figure §9.3.

9.2.6 Limitations

When the input is a collection of multi-record web documents, Trinity groups the attributes from the first data record and from the last data records together, whereas the attributes of the remaining data records are extracted correctly. This limitation is shown in Figure §9.3: the basic types "_A_", "_E_", and "_F_", which are the titles of the first data records, the authors of the first data records, and the prices of the last data records

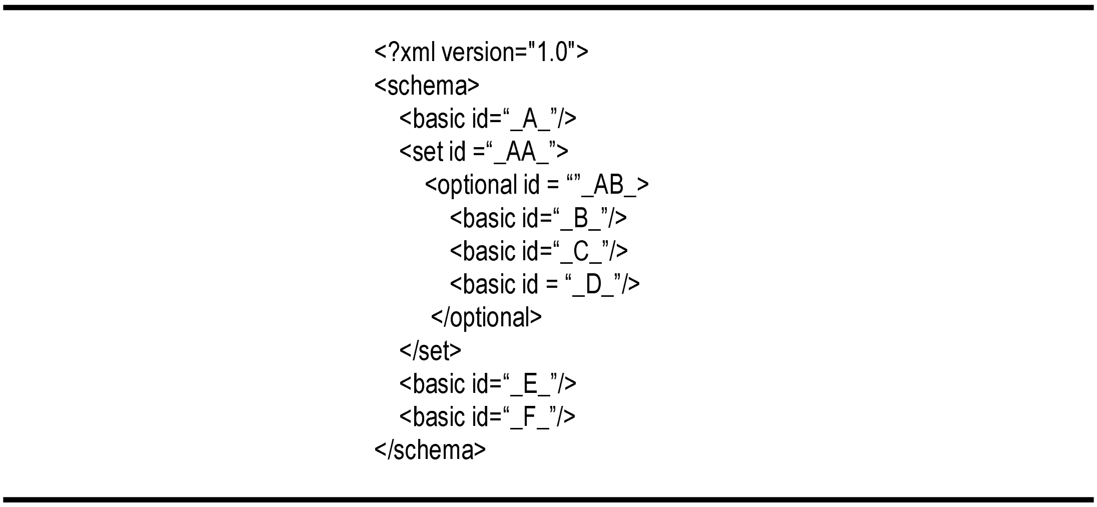
```
learnSchema(node: Node; String result): String
    if isOptional(node) then
     result += "<optional>"
    end
    if isLeaf(node) then
        if not allTextsEmpty(node) then
            result += "<basic id=\"" + freshLabel() +"\"/>"
        end
    else
        result += "<tuple id=\"" + freshLabel() +"\"/>"
        result += learnSchema(getPrefix(node), result)
        if isRepeatable(node, getSeparators(node)) then
            result += "<set id=\"" + freshLabel() +"\"/>"
            if contains(getSeparators(node), nil) then
                result += "<optional>"
            end
            result += learnSchema(getSeparators(node), result)
            if contains(getSeparators(node), nil) then
                result += "</optional>"
            end
            result += "</set>"
        end
        result += learnSchema(getSuffix(node), result)
        result += "</tuple>"
    end
    if isOptional(node) then
        result += "</optional>"
    end
return result
```

Program 9.7: *Algorithm learnSchema.*

respectively, are kept outside the list of data records, which are represented by the set type "_AA_". The problem of mapping the information from this schema onto structured records was dealt with elsewhere [8].

9.3 Complexity analysis

In this section, we provide an upper limit to the worst-case space and time complexities of Algorithm Trinity. Note that it is not common to find a complexity analysis in the literature regarding information extraction, but we



```
<?xml version="1.0">
<schema>
   <basic id="_A_"/>
   <set id ="_AA_">
      <optional id = “”_AB_>
         <basic id="_B_"/>
         <basic id="_C_"/>
         <basic id = “_D_"/>
      </optional>
   </set>
   <basic id="_E_"/>
   <basic id="_F_"/>
</schema>
```

Figure 9.3: *The schema learnt for our running example.*

think that it is important to make sure that the proposal is computationally tractable.

We have characterised an upper bound to the worst-case time complexity building on two sensible assumptions: i) simple instructions like adding an item to a set, comparing two tokens, or constructing a tuple can be implemented in $O(1)$ time with regard to the other algorithms in our proposal; ii) the number of input documents is generally very small with regard to the number of tokens of the longest document to be analysed.

In the following subsections, we first report on the space requirements, then on time requirements, and conclude with the theorems that prove that Trinity is computationally tractable. In the sequel, we use variable n to denote the number of documents that Trinity has to analyse and m to denote the size of the longest such document.

9.3.1 Space requirements

Proposition 9.1 (Maximum size of a Node) $n \lfloor \frac{m}{2} \rfloor$ *is an upper bound to maximum size of a Node created by Trinity.*

Proof Algorithm expand is the unique algorithm that creates new Nodes, which happens when a shared pattern p is found in a given Node. The new Nodes correspond to the prefixes, separators, and suffixes to which p leads. Regarding the prefix and suffix Nodes, note that there cannot be more than n

such prefixes or suffixes since a document may not have more than one prefix or one suffix; that is, n is an upper bound to the maximum size of a Node that contains prefixes or suffixes. Regarding separators Nodes the worst-case happens when p is a one-token pattern that occurs every two tokens; that is, $\lfloor \frac{m}{2} \rfloor$ is an upper bound to the number of separators in this case, or, otherwise, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a Node that contains separators. As a conclusion, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a Node created by Trinity. □

Proposition 9.2 (Maximum number of Nodes) $3\,m$ *is an upper bound to the number of Nodes created by Trinity.*

Proof Algorithm expand creates three new Nodes when a shared pattern p is found in a given Node. The new Nodes correspond to the prefixes, separators, and suffixes to which p leads. We know that m is an upper bound to the number of partitions of a Text of size m, which means that m is an upper bound to the number of levels of the trinary tree in the worst-case. Since each level has three nodes, then $3\,m$ is an upper bound to the number of Nodes inside the trinary tree. As a conclusion, $3\,m$ is an upper bound to the number of Nodes created by Trinity. □

9.3.2 Time requirements

Proposition 9.3 (Algorithm createChildren) *Let* nd *be a Node,* r *a map from the Texts in* nd *onto lists of indices that denote where a shared pattern occurs, and* p *the shared pattern.* $O(n\,m^2)$ *is an upper bound to the worst-case time required to execute createChildren*(nd, r, p)*.*

Proof The algorithm iterates through every Text in nd. According to Proposition §9.1, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a Node, which means that the $n\,m$ is an upper bound to the number of iterations of this loop. Inside this loop, accessing the map and calculating the prefix and suffix of the shared pattern can be performed in $O(1)$ time, whereas computing the separators requires variable time. According to the proof of Proposition §9.1, the maximum number of separators in a given Text is $\lfloor \frac{m}{2} \rfloor$, which is less than m. Then, $O(m)$ is an upper bound to the time required to compute the separators. The instructions to create new nodes at lines §2-§4 and the instructions to link them to the input Node at lines §12-§14 require $O(1)$ time. As a conclusion, $O(n\,m\,m) = O(n\,m^2)$ is an upper bound to the worst-case time required to execute createChildren(nd, r, p). □

Proposition 9.4 (Algorithm findPattern) *Let nd be a Node and s the size of the pattern for which the algorithm searches. $O(n\,m^3)$ is an upper bound to the worst-case time required to execute findPattern(nd, s).*

Proof The algorithm first searches for the shortest Text in nd. According to Proposition §9.1, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a Node, which implies that $O(n \lfloor \frac{m}{2} \rfloor)$ is an upper bound to the maximum time required to find the shortest Text in a Node. The main loop iterates through $base$ until finding a pattern of s tokens that occurs in every other Text in nd. In the worst-case, $base$ has the maximum size m and the shared pattern is found at the end of $base$, which means that the main loop iterates $m - s$ times, i.e., $O(m)$ times. In each iteration of the main loop, the inner loop iterates through the Texts in nd. According to Proposition §9.1, $n \lfloor \frac{m}{2} \rfloor$ is an upper bound to the maximum size of a Node, which implies that the inner loop does not iterate more than $n \lfloor \frac{m}{2} \rfloor$ times. In each iteration, it invokes Algorithm findMatches, whose worst-time complexity is $O(k)$, where k denotes the size of the text in which a pattern is searched [86]. This implies that $O(m)$ is an upper bound to the worst-case time complexity of the instructions inside the inner loop. As a conclusion, $O(n \lfloor \frac{m}{2} \rfloor + m\,n \lfloor \frac{m}{2} \rfloor\, m) \subseteq O(n\,m + m\,n\,m\,m) \subseteq O(n\,m^3)$ is an upper bound to the worst-case time required to execute findPattern(nd, s). □

Proposition 9.5 (Algorithm expand) *Let nd be a Node, and s be a pattern size. $O(n\,m^3)$ is an upper bound to the worst-case time required to execute expand(nd, s).*

Proof In the worst-case, the invocation to Algorithm expand requires to invoke Algorithms findPattern and createChildren in sequence, which according to Propositions §9.4 and §9.3 require no more than $O(n\,m^3)$ and $O(n\,m^2)$ time in the worst-case. As a conclusion, $O(n\,m^3 + n\,m^2) \subseteq O(n\,m^3)$ is an upper bound to the worst-case time required to execute expand(nd, s). □

Proposition 9.6 (Algorithm createTrinaryTree) *Let nd be a Node, min and max be the minimum and maximum sizes of the shared patterns for which the algorithm searches, respectively. $O(n\,m^5)$ is an upper bound to the worst-case time required to execute createTrinaryTree(nd, min, max).*

Proof The algorithm first iterates through every possible size between min and max, which amounts to m times in the worst-case. In each iteration, the algorithm executes Algorithm expand, which according to Lemma §9.5 requires no more than $O(n\,m^3)$ time. Then, $O(n\,m^4)$ is an upper bound to the first loop. If nd is expanded, then another loop iterates through its leaves and executes Algorithm createTrinaryTree recursively. According to Proposition §9.2, $3\,m$ is an upper bound to the number of Nodes created by Trinity. As a conclusion, $O(n\, m^4\, 3\, m) \subseteq O(n\, m^5)$ is an upper bound to the worst-case time required to execute createTrinaryTree(nd, min, max). □

Proposition 9.7 (Algorithm learnTemplate) $O(n\, m^2)$ *is an upper limit to the worst-case time required to execute learnTemplate*$(nd, “”)$*., where* nd *denotes the root node of a trinary tree.*

Proof The algorithm works on every node in the input trinary tree, which in the worst-case are $3\, m$ nodes according to Proposition §9.2. If the Node is a leaf, the algorithm iterates on its Texts no more than $n \lfloor \frac{m}{2} \rfloor$ times according to Proposition §9.1. Otherwise, the algorithm is invoked recursively on the children of this node, namely: prefixes, separators, and suffixes. In the case of the separators, whose number is limited by the upper bound $n \lfloor \frac{m}{2} \rfloor$, the algorithm iterates on them twice to check for repeatability and searching for nil values. As a conclusion, $O(3\, m\, 2\, n \lfloor \frac{m}{2} \rfloor) \subseteq O(n\, m^2)$ is an upper bound to the worst-case time required to execute learnTemplate$(nd, “”)$. □

Proposition 9.8 (Algorithm learnSchema) $O(n\, m^2)$ *is an upper limit to the worst-case time required to execute learnSchema*$(nd, “”)$*., where* nd *denotes the root node of a trinary tree.*

Proof The algorithm works on every node in the input trinary tree, which in the worst-case is $3\, m$ according to Proposition §9.2. If the Node is a leaf, the algorithm iterates on its Texts, no more than $\frac{m}{2}$ times according to Proposition §9.1. Otherwise, the algorithm is invoked recursively on the children of this node, namely: prefixes, separators, and suffixes. The number of separators is limited by the upper bound $n \lfloor \frac{m}{2} \rfloor$, and the algorithm iterates on them twice to check for repeatability and searching for nil values. As a conclusion, $O(3\, m\, 2\, n \lfloor \frac{m}{2} \rfloor) \subseteq O(n\, m^2)$ is an upper bound to the worst-case time required to execute learnSchema$(nd, “”)$. □

9.3.3 Computational tractability

Theorem 9.1 (Space requirements of Algorithm Trinity) *Let* nd *be a Node and* min *and* max *be the minimum and maximum sizes of the shared patterns for which Trinity searches.* $O(n\, m^2)$ *is an upper bound to the space required to execute Trinity*(nd, min, max) *in the worst case.*

Proof The proof follows straightforwardly from Propositions §9.1 and §9.2. If $O(n \lfloor \frac{m}{2} \rfloor)$ is an upper bound to the maximum size of a node and $O(3\, m)$ is an upper bound to the maximum number of nodes the algorithm creates, then $O(n \lfloor \frac{m}{2} \rfloor\, 3\, m) \subseteq O(n\, m^2)$ is an upper bound to the space required to execute Trinity(nd, min, max) in the worst case. □

Theorem 9.2 (Time requirements of Algorithm Trinity) *Let* nd *be a Node and* min *and* max *be the minimum and maximum sizes of the shared patterns for which Trinity searches.* $O(n\,m^5)$ *is an upper bound to the worst-case time required to execute Trinity*$(\mathrm{nd}, \mathrm{min}, \mathrm{max})$.

Proof The proof follows straightforwardly from the previous propositions. Note that Algorithm Trinity invokes Algorithms createTrinaryTree, learnTemplate, and buildTemplate in sequence, which, according to propositions §9.6, §9.7, and §9.8 require no more than $O(n\,m^5)$, $O(n\,m^2)$, and $O(n\,m^2)$ time to complete, respectively. This implies that $O(n\,m^5 + n\,m^2 + n\,m^2) \subseteq O(n\,m^5)$ is an upper bound to the worst-case time required to execute Trinity$(\mathrm{nd}, \mathrm{min}, \mathrm{max})$. □

Corollary 9.1 *If we assume that* $n \ll m$*, then* $(n\,m^2)$ *is an upper bound to its space required to execute Algorithm Trinity and* $O(m^5)$ *is an upper bound to the worst-case time complexity. Since both results are polynomial, we can conclude that Algorithm Trinity is computationally tractable.*

9.4 Experimental analysis

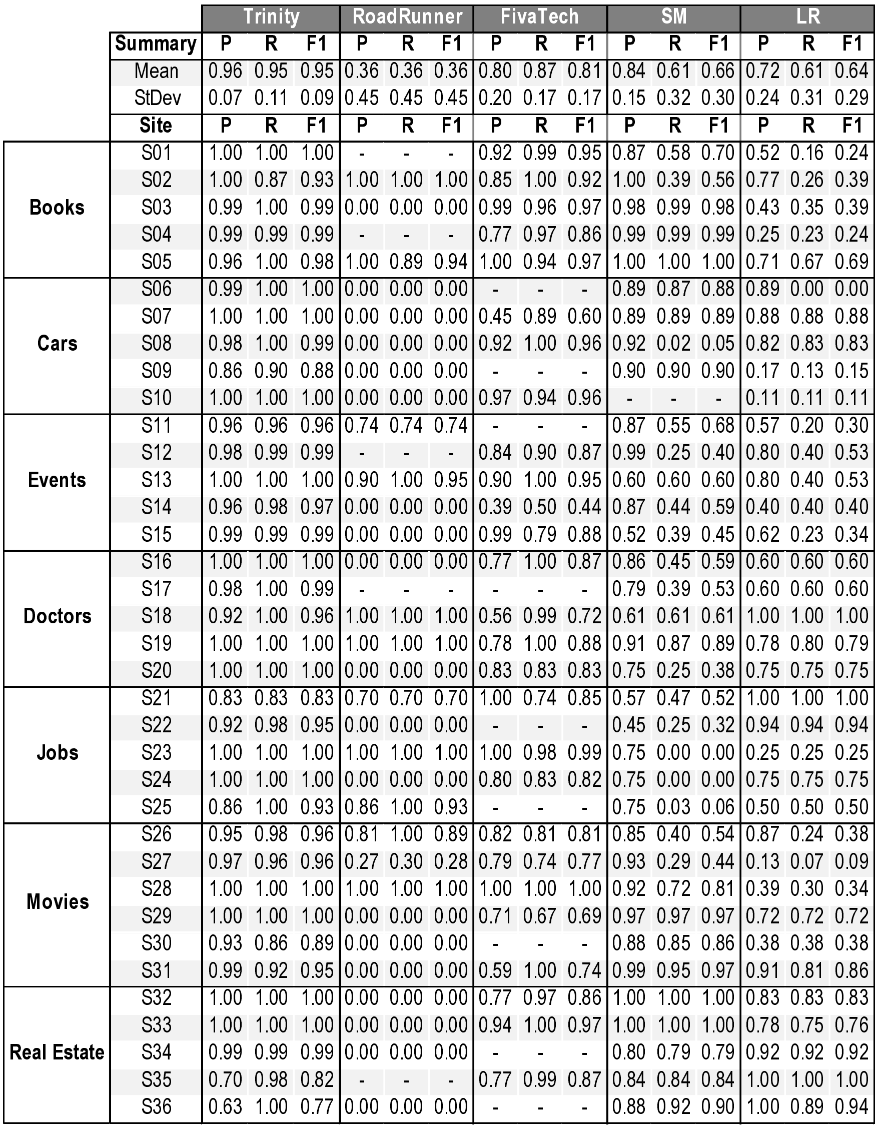
In this section, we present the results of the experiments we have carried out to compare our proposal to other techniques in the literature from an empirical point of view. We ran Trinity, RoadRunner, FiVaTech, SM, and LR on CEDAR repository in order to learn extraction rules. We measured the standard effectiveness measures (precision, recall, and the $F1$ measure) and two efficiency measures (learning and extraction time). We did not measured memory since the heap size was set to the default value and it was never exceeded.

9.4.1 Experimentation environment

We have developed a Java 1.7 prototype of Trinity using the our framework. We performed a series of experiments on a cloud machine that was equipped with a four-threaded Intel Core i7 processor that ran at 2.93 GHz, had 4 GiB of RAM, Windows 7 Pro 64-bit, Oracle's Java Development Kit 1.7.0_02, and GNU Regex 1.1.4. The configuration parameters of the Java Virtual Machine were set to their default values.

9.4.2 Effectiveness analysis

In the case of SM and LR, it was easy to compute the precision and recall since both techniques are supervised. Contrarily, Trinity, RoadRunner,

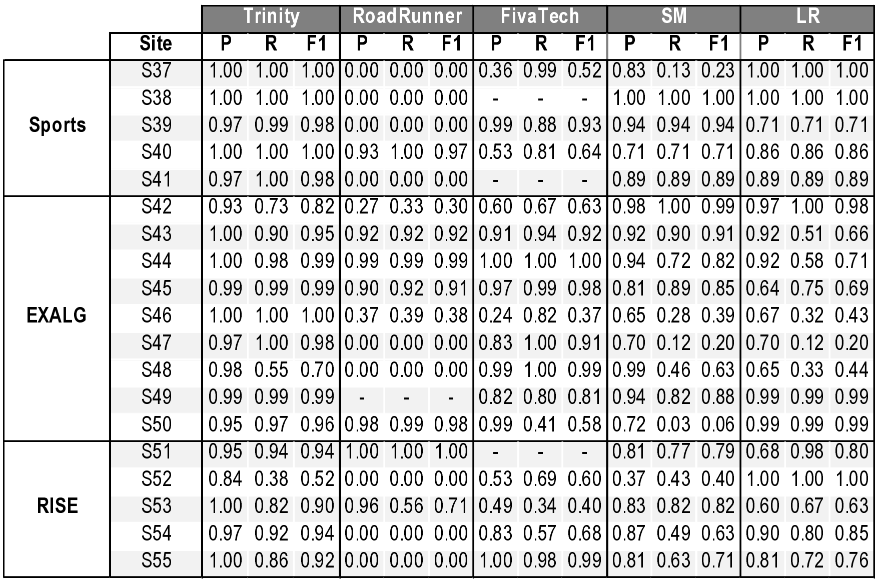


		Trinity			RoadRunner			FivaTech			SM			LR		
	Summary	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
	Mean	0.96	0.95	0.95	0.36	0.36	0.36	0.80	0.87	0.81	0.84	0.61	0.66	0.72	0.61	0.64
	StDev	0.07	0.11	0.09	0.45	0.45	0.45	0.20	0.17	0.17	0.15	0.32	0.30	0.24	0.31	0.29
	Site	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Books	S01	1.00	1.00	1.00	-	-	-	0.92	0.99	0.95	0.87	0.58	0.70	0.52	0.16	0.24
	S02	1.00	0.87	0.93	1.00	1.00	1.00	0.85	1.00	0.92	1.00	0.39	0.56	0.77	0.26	0.39
	S03	0.99	1.00	0.99	0.00	0.00	0.00	0.99	0.96	0.97	0.98	0.99	0.98	0.43	0.35	0.39
	S04	0.99	0.99	0.99	-	-	-	0.77	0.97	0.86	0.99	0.99	0.99	0.25	0.23	0.24
	S05	0.96	1.00	0.98	1.00	0.89	0.94	1.00	0.94	0.97	1.00	1.00	1.00	0.71	0.67	0.69
Cars	S06	0.99	1.00	1.00	0.00	0.00	0.00	-	-	-	0.89	0.87	0.88	0.89	0.00	0.00
	S07	1.00	1.00	1.00	0.00	0.00	0.00	0.45	0.89	0.60	0.89	0.89	0.89	0.88	0.88	0.88
	S08	0.98	1.00	0.99	0.00	0.00	0.00	0.92	1.00	0.96	0.92	0.02	0.05	0.82	0.83	0.83
	S09	0.86	0.90	0.88	0.00	0.00	0.00	-	-	-	0.90	0.90	0.90	0.17	0.13	0.15
	S10	1.00	1.00	1.00	0.00	0.00	0.00	0.97	0.94	0.96	-	-	-	0.11	0.11	0.11
Events	S11	0.96	0.96	0.96	0.74	0.74	0.74	-	-	-	0.87	0.55	0.68	0.57	0.20	0.30
	S12	0.98	0.99	0.99	-	-	-	0.84	0.90	0.87	0.99	0.25	0.40	0.80	0.40	0.53
	S13	1.00	1.00	1.00	0.90	1.00	0.95	0.90	1.00	0.95	0.60	0.60	0.60	0.80	0.40	0.53
	S14	0.96	0.98	0.97	0.00	0.00	0.00	0.39	0.50	0.44	0.87	0.44	0.59	0.40	0.40	0.40
	S15	0.99	0.99	0.99	0.00	0.00	0.00	0.99	0.79	0.88	0.52	0.39	0.45	0.62	0.23	0.34
Doctors	S16	1.00	1.00	1.00	0.00	0.00	0.00	0.77	1.00	0.87	0.86	0.45	0.59	0.60	0.60	0.60
	S17	0.98	1.00	0.99	-	-	-	-	-	-	0.79	0.39	0.53	0.60	0.60	0.60
	S18	0.92	1.00	0.96	1.00	1.00	1.00	0.56	0.99	0.72	0.61	0.61	0.61	1.00	1.00	1.00
	S19	1.00	1.00	1.00	1.00	1.00	1.00	0.78	1.00	0.88	0.91	0.87	0.89	0.78	0.80	0.79
	S20	1.00	1.00	1.00	0.00	0.00	0.00	0.83	0.83	0.83	0.75	0.25	0.38	0.75	0.75	0.75
Jobs	S21	0.83	0.83	0.83	0.70	0.70	0.70	1.00	0.74	0.85	0.57	0.47	0.52	1.00	1.00	1.00
	S22	0.92	0.98	0.95	0.00	0.00	0.00	-	-	-	0.45	0.25	0.32	0.94	0.94	0.94
	S23	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.99	0.75	0.00	0.00	0.25	0.25	0.25
	S24	1.00	1.00	1.00	0.00	0.00	0.00	0.80	0.83	0.82	0.75	0.00	0.00	0.75	0.75	0.75
	S25	0.86	1.00	0.93	0.86	1.00	0.93	-	-	-	0.75	0.03	0.06	0.50	0.50	0.50
Movies	S26	0.95	0.98	0.96	0.81	1.00	0.89	0.82	0.81	0.81	0.85	0.40	0.54	0.87	0.24	0.38
	S27	0.97	0.96	0.96	0.27	0.30	0.28	0.79	0.74	0.77	0.93	0.29	0.44	0.13	0.07	0.09
	S28	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.92	0.72	0.81	0.39	0.30	0.34
	S29	1.00	1.00	1.00	0.00	0.00	0.00	0.71	0.67	0.69	0.97	0.97	0.97	0.72	0.72	0.72
	S30	0.93	0.86	0.89	0.00	0.00	0.00	-	-	-	0.88	0.85	0.86	0.38	0.38	0.38
	S31	0.99	0.92	0.95	0.00	0.00	0.00	0.59	1.00	0.74	0.99	0.95	0.97	0.91	0.81	0.86
Real Estate	S32	1.00	1.00	1.00	0.00	0.00	0.00	0.77	0.97	0.86	1.00	1.00	1.00	0.83	0.83	0.83
	S33	1.00	1.00	1.00	0.00	0.00	0.00	0.94	1.00	0.97	1.00	1.00	1.00	0.78	0.75	0.76
	S34	0.99	0.99	0.99	0.00	0.00	0.00	-	-	-	0.80	0.79	0.79	0.92	0.92	0.92
	S35	0.70	0.98	0.82	-	-	-	0.77	0.99	0.87	0.84	0.84	0.84	1.00	1.00	1.00
	S36	0.63	1.00	0.77	0.00	0.00	0.00	-	-	-	0.88	0.92	0.90	1.00	0.89	0.94

Table 9.1: *Comparison of Trinity's effectiveness to other techniques.*

and FiVaTech are unsupervised, i.e., they learn an extraction rule that extracts as much information as possible, give each group of information a computer-generated label, and it is the responsibility of the user to assign a meaning to these labels. We then used the method described in Section §6.5 to compute their effectiveness measures.

Table §9.1 reports on the results of our experiments. The columns report



		Trinity			RoadRunner			FivaTech			SM			LR		
	Site	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Sports	S37	1.00	1.00	1.00	0.00	0.00	0.00	0.36	0.99	0.52	0.83	0.13	0.23	1.00	1.00	1.00
	S38	1.00	1.00	1.00	0.00	0.00	0.00	-	-	-	1.00	1.00	1.00	1.00	1.00	1.00
	S39	0.97	0.99	0.98	0.00	0.00	0.00	0.99	0.88	0.93	0.94	0.94	0.94	0.71	0.71	0.71
	S40	1.00	1.00	1.00	0.93	1.00	0.97	0.53	0.81	0.64	0.71	0.71	0.71	0.86	0.86	0.86
	S41	0.97	1.00	0.98	0.00	0.00	0.00	-	-	-	0.89	0.89	0.89	0.89	0.89	0.89
EXALG	S42	0.93	0.73	0.82	0.27	0.33	0.30	0.60	0.67	0.63	0.98	1.00	0.99	0.97	1.00	0.98
	S43	1.00	0.90	0.95	0.92	0.92	0.92	0.91	0.94	0.92	0.92	0.90	0.91	0.92	0.51	0.66
	S44	1.00	0.98	0.99	0.99	0.99	0.99	1.00	1.00	1.00	0.94	0.72	0.82	0.92	0.58	0.71
	S45	0.99	0.99	0.99	0.90	0.92	0.91	0.97	0.99	0.98	0.81	0.89	0.85	0.64	0.75	0.69
	S46	1.00	1.00	1.00	0.37	0.39	0.38	0.24	0.82	0.37	0.65	0.28	0.39	0.67	0.32	0.43
	S47	0.97	1.00	0.98	0.00	0.00	0.00	0.83	1.00	0.91	0.70	0.12	0.20	0.70	0.12	0.20
	S48	0.98	0.55	0.70	0.00	0.00	0.00	0.99	1.00	0.99	0.99	0.46	0.63	0.65	0.33	0.44
	S49	0.99	0.99	0.99	-	-	-	0.82	0.80	0.81	0.94	0.82	0.88	0.99	0.99	0.99
	S50	0.95	0.97	0.96	0.98	0.99	0.98	0.99	0.41	0.58	0.72	0.03	0.06	0.99	0.99	0.99
RISE	S51	0.95	0.94	0.94	1.00	1.00	1.00	-	-	-	0.81	0.77	0.79	0.68	0.98	0.80
	S52	0.84	0.38	0.52	0.00	0.00	0.00	0.53	0.69	0.60	0.37	0.43	0.40	1.00	1.00	1.00
	S53	1.00	0.82	0.90	0.96	0.56	0.71	0.49	0.34	0.40	0.83	0.82	0.82	0.60	0.67	0.63
	S54	0.97	0.92	0.94	0.00	0.00	0.00	0.83	0.57	0.68	0.87	0.49	0.63	0.90	0.80	0.85
	S55	1.00	0.86	0.92	0.00	0.00	0.00	1.00	0.98	0.99	0.81	0.63	0.71	0.81	0.72	0.76

Table 9.1: *Comparison of Trinity's effectiveness to other techniques. (Cont'd)*

on the precision (P), recall (R), the F1 measure (F1). The first few rows provide a summary in terms of mean and standard deviations of the previous measures. The remaining rows provide the results we computed for each web site. Note that some cells contain a dash, which indicates that the corresponding technique was not able to learn an extraction rule in 15 CPU minutes.

According to Table §9.1, Trinity outperforms the other techniques regarding effectiveness. Figure §9.4 illustrates this conclusion since the majority of points that correspond to Trinity are very close the upper right corner, whereas the points that correspond to the other techniques are more scattered. Intuitively, the closer the points to $(1.00, 1.00)$ the higher the F1 measure; similarly, the closer to $(0.00, 0.00)$ the lower the F1 measure.

To discern if errors in the input documents have an impact from a statistical point of view on the effectiveness of the proposals, we need compute the correlation from the number of errors to the F1 measure using non-parametric Kendall's Tau procedure. Table §9.2 presents the results of this procedure and Figure §9.5 illustrates the F1 measures we gathered for every pair of datasets and techniques and the number of errors in a radial chart. Note that the p-value of the correlation coefficients is smaller than the standard significance level $\alpha = 0.05$ except for the case of RoadRunner and FiVaTech; in these cases the correlation coefficient is negative, which means

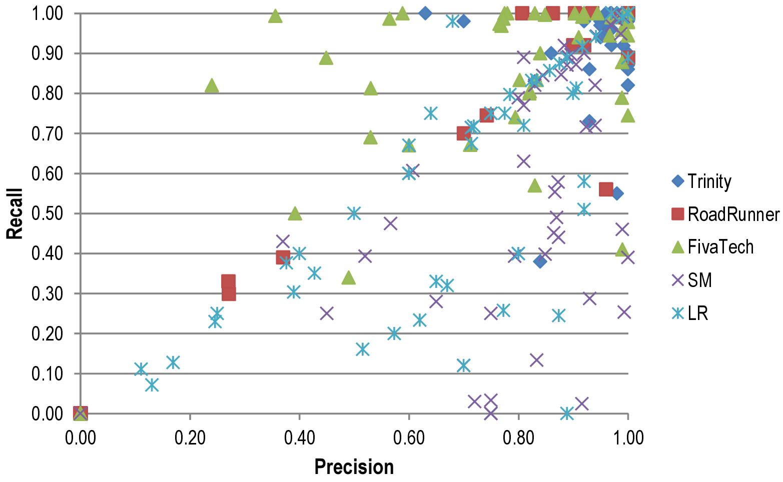
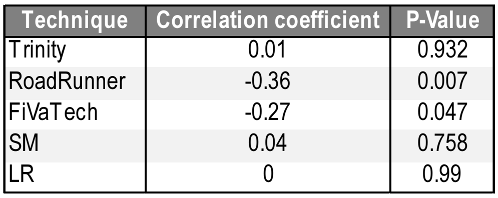


Figure 9.4: *Precision versus recall in our experiments regarding Trinity.*



Technique	Correlation coefficient	P-Value
Trinity	0.01	0.932
RoadRunner	-0.36	0.007
FiVaTech	-0.27	0.047
SM	0.04	0.758
LR	0	0.99

Table 9.2: *Impact of errors on Trinity effectiveness.*

that the effectiveness of these techniques is expected to decrease as the number of errors in the input documents increases. As a conclusion, our experiments do not show any statistical evidence that the effectiveness of our technique is sensible to malformed input documents.

9.4.3 Efficiency analysis

Table §9.3 shows our results regarding efficiency of Trinity and the other techniques. The columns report on the learning time in CPU seconds (LT) and the extraction time in CPU seconds (ET). The first two rows provide a summary of these measures in terms of mean values and standard deviations. A dash in a cell means that the corresponding technique was not able to learn an extraction rule in 15 CPU minutes. The learning time and the ex-

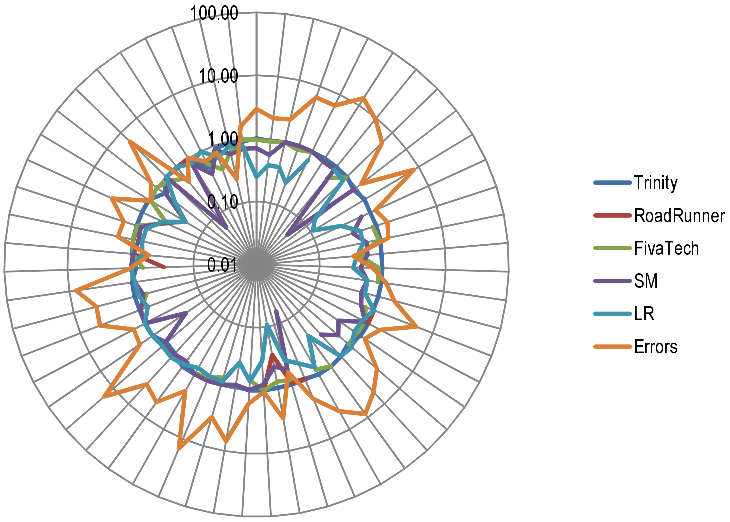


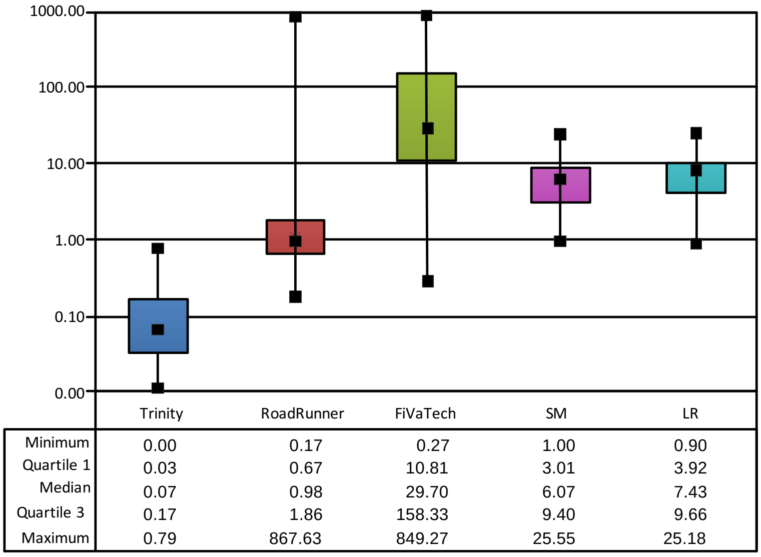
Figure 9.5: *Correlation from number of errors to the* F1 *measure in Trinity.*

traction time do not account for the time consumed by JTidy in the case of RoadRunner and FiVaTech, neither they include the tokenisation time in the case of Trinity, SM, and LR. The reason is that the time to clean or tokenise a document is not actually an intrinsic feature of the proposals being analysed.

The results in Table §9.3 support the idea that Trinity is more effective regarding learning time than the other techniques, and that it is comparable to RoadRunner regarding extraction time and clearly better than the other techniques. Figure §9.6 and §9.6 illustrate this idea: note that the range from the first to the third quartile is smaller for Trinity than for the other techniques regarding learning time and similar to RoadRunner's regarding the extraction time, but clearly smaller than for the other techniques; the dispersion from the minimum values to the first quartile and from the third quartile to the maximum are also a clear indication that our technique performs more homogeneously than the others regarding learning and extraction time, with the only exception of RoadRunner regarding the extraction time.

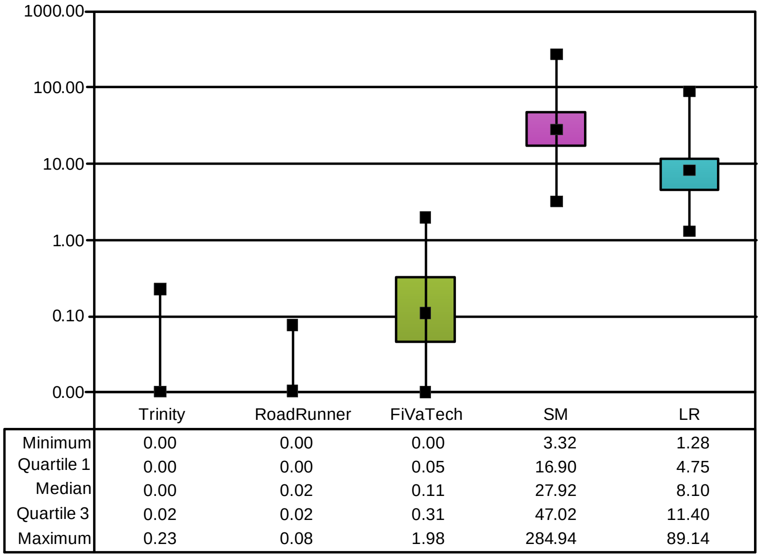
9.5 Statistical analysis

To confirm that the conclusions we have drawn from our empirical evaluation are valid, we need perform a statistical analysis, which consists of



	Trinity	RoadRunner	FiVaTech	SM	LR
Minimum	0.00	0.17	0.27	1.00	0.90
Quartile 1	0.03	0.67	10.81	3.01	3.92
Median	0.07	0.98	29.70	6.07	7.43
Quartile 3	0.17	1.86	158.33	9.40	9.66
Maximum	0.79	867.63	849.27	25.55	25.18

Figure 9.6: *Comparison of learning times regarding Trinity.*



	Trinity	RoadRunner	FiVaTech	SM	LR
Minimum	0.00	0.00	0.00	3.32	1.28
Quartile 1	0.00	0.00	0.05	16.90	4.75
Median	0.00	0.02	0.11	27.92	8.10
Quartile 3	0.02	0.02	0.31	47.02	11.40
Maximum	0.23	0.08	1.98	284.94	89.14

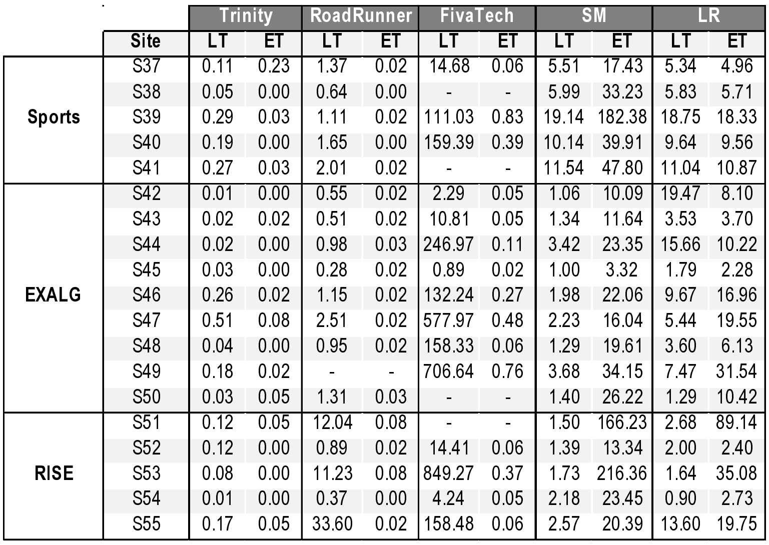
Figure 9.7: *Comparison of extraction times regarding Trinity.*

		Trinity		RoadRunner		FivaTech		SM		LR	
	Summary	**LT**	**ET**	**LT**	**ET**	**LT**	**ET**	**LT**	**ET**	**LT**	**ET**
	Mean	0.13	0.02	20.03	0.01	122.94	0.24	7.10	46.30	7.80	10.93
	StDev	0.16	0.03	123.71	0.02	196.42	0.36	5.13	54.45	5.15	12.88
	Site	**LT**	**ET**	**LT**	**ET**	**LT**	**ET**	**LT**	**ET**	**LT**	**ET**
Books	S01	0.03	0.00	-	-	15.46	0.12	9.09	26.96	9.66	10.26
	S02	0.03	0.00	0.92	0.00	8.14	0.14	5.68	16.52	5.01	6.27
	S03	0.17	0.00	0.98	0.00	85.32	0.39	16.15	41.39	15.57	17.04
	S04	0.11	0.02	-	-	65.49	0.12	7.38	30.33	6.74	8.14
	S05	0.07	0.00	1.14	0.02	51.53	1.98	8.67	77.36	8.02	8.72
Cars	S06	0.28	0.03	0.83	0.02	-	-	14.87	103.82	14.18	14.02
	S07	0.21	0.02	1.72	0.02	34.21	0.20	7.66	22.90	7.43	7.63
	S08	0.04	0.00	0.69	0.00	446.90	0.59	5.30	28.88	4.80	4.76
	S09	0.11	0.02	1.47	0.00	-	-	10.92	78.55	10.03	10.28
	S10	0.12	0.05	4.74	0.00	117.16	0.19	8.74	-	7.78	9.02
Events	S11	0.04	0.02	2.57	0.00	-	-	5.04	42.04	4.15	3.34
	S12	0.11	0.00	-	-	42.96	0.08	10.06	62.24	7.27	4.74
	S13	0.01	0.00	0.45	0.02	1.56	0.03	2.61	5.74	1.86	1.28
	S14	0.00	0.00	0.17	0.00	0.27	0.03	6.65	7.86	4.43	2.31
	S15	0.02	0.00	0.45	0.00	6.21	0.02	4.62	7.10	3.24	1.95
Doctors	S16	0.02	0.00	0.73	0.00	11.81	0.03	8.28	29.94	9.45	14.54
	S17	0.02	0.02	-	-	-	-	6.07	10.53	6.99	7.38
	S18	0.01	0.00	867.63	0.00	9.94	0.08	2.28	8.16	1.97	1.84
	S19	0.03	0.00	3.28	0.02	25.35	0.06	4.49	17.50	3.88	3.62
	S20	0.67	0.02	0.67	0.02	9.59	0.11	9.70	40.00	9.56	9.77
Jobs	S21	0.04	0.00	0.78	0.02	13.76	0.11	7.77	16.72	7.49	7.14
	S22	0.09	0.02	0.69	0.02	-	-	9.09	36.60	9.02	9.50
	S23	0.04	0.02	1.83	0.03	90.78	0.34	11.76	26.79	11.56	11.93
	S24	0.05	0.00	0.53	0.00	266.00	0.31	8.13	50.72	8.11	7.72
	S25	0.06	0.00	1.86	0.00	-	-	5.19	22.56	5.09	4.77
Movies	S26	0.02	0.00	0.64	0.02	1.59	0.00	3.65	3.45	2.70	1.67
	S27	0.21	0.00	1.64	0.03	14.84	0.11	11.29	38.86	7.78	5.91
	S28	0.02	0.00	0.69	0.02	29.70	0.05	3.70	10.39	2.79	3.79
	S29	0.07	0.00	0.59	0.00	259.23	0.08	4.34	44.69	3.95	3.82
	S30	0.45	0.03	0.97	0.02	-	-	16.38	63.24	15.87	16.46
	S31	0.03	0.00	0.47	0.03	17.24	0.05	10.64	37.58	9.73	9.36
Real Estate	S32	0.15	0.02	3.10	0.00	246.95	0.89	16.32	86.55	15.62	16.75
	S33	0.05	0.00	2.75	0.02	20.76	0.09	7.43	100.85	7.04	6.96
	S34	0.10	0.00	1.39	0.02	-	-	8.25	65.88	8.22	8.17
	S35	0.20	0.02	-	-	-	-	7.60	25.65	7.52	7.49
	S36	0.79	0.05	2.18	0.00	-	-	25.55	284.94	25.18	25.37

Table 9.3: *Comparison of Trinity's efficiency to other techniques.*

performing a statistical ranking regarding our performance measures and determining if there is a significant correlation from the number of errors to the effectiveness of the techniques we have evaluated, cf. Section §6.5.

We have conducted a Shapiro-Wilk test at the standard significance level $\alpha = 0.05$ on every measure and we have found out that none of them behaves normally. For instance, Shapiro-Wilk's statistic regarding the normality of Trinity's precision is $W(55) = 0.59$, whose p-value is 0.00; this is a strong indication that the data is not distributed normally. This is not surprising at all; a



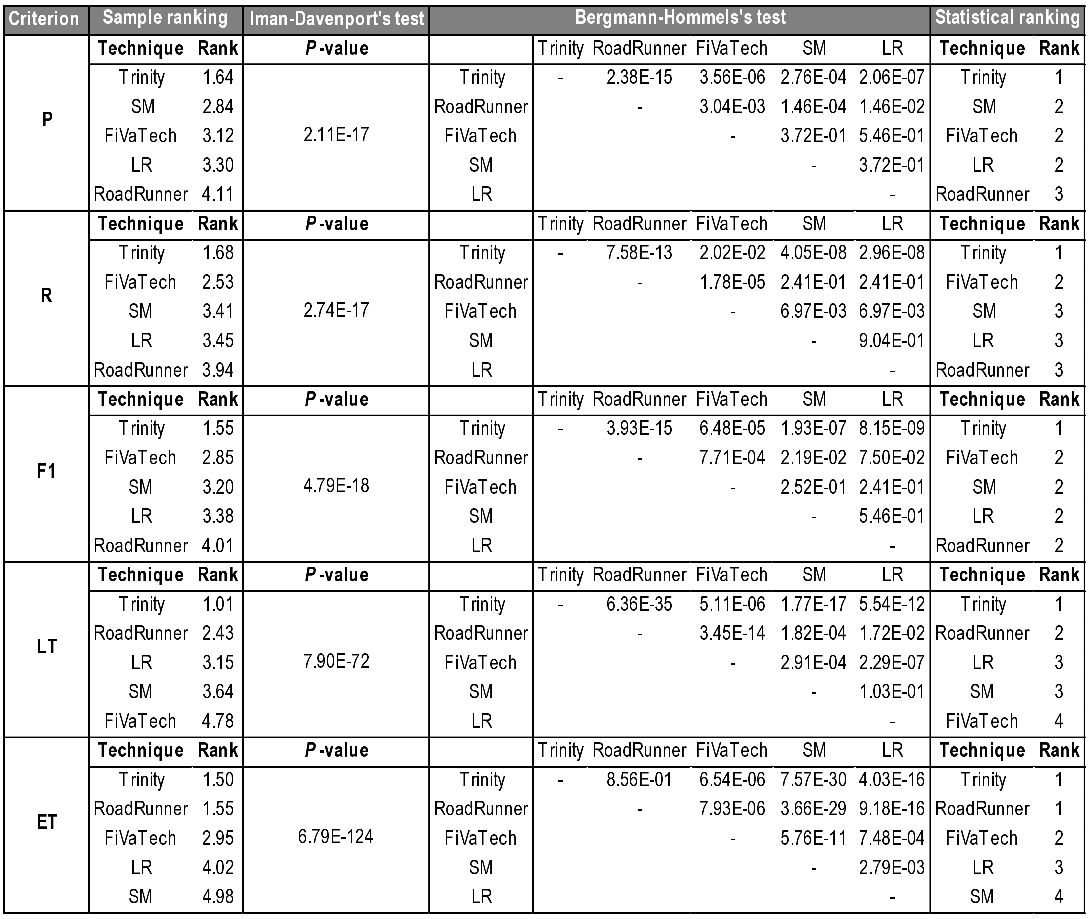
		Trinity		RoadRunner		FivaTech		SM		LR	
	Site	LT	ET	LT	ET	LT	ET	LT	ET	LT	ET
Sports	S37	0.11	0.23	1.37	0.02	14.68	0.06	5.51	17.43	5.34	4.96
	S38	0.05	0.00	0.64	0.00	-	-	5.99	33.23	5.83	5.71
	S39	0.29	0.03	1.11	0.02	111.03	0.83	19.14	182.38	18.75	18.33
	S40	0.19	0.00	1.65	0.00	159.39	0.39	10.14	39.91	9.64	9.56
	S41	0.27	0.03	2.01	0.02	-	-	11.54	47.80	11.04	10.87
EXALG	S42	0.01	0.00	0.55	0.02	2.29	0.05	1.06	10.09	19.47	8.10
	S43	0.02	0.02	0.51	0.02	10.81	0.05	1.34	11.64	3.53	3.70
	S44	0.02	0.00	0.98	0.03	246.97	0.11	3.42	23.35	15.66	10.22
	S45	0.03	0.00	0.28	0.02	0.89	0.02	1.00	3.32	1.79	2.28
	S46	0.26	0.02	1.15	0.02	132.24	0.27	1.98	22.06	9.67	16.96
	S47	0.51	0.08	2.51	0.02	577.97	0.48	2.23	16.04	5.44	19.55
	S48	0.04	0.00	0.95	0.02	158.33	0.06	1.29	19.61	3.60	6.13
	S49	0.18	0.02	-	-	706.64	0.76	3.68	34.15	7.47	31.54
	S50	0.03	0.05	1.31	0.03	-	-	1.40	26.22	1.29	10.42
RISE	S51	0.12	0.05	12.04	0.08	-	-	1.50	166.23	2.68	89.14
	S52	0.12	0.00	0.89	0.02	14.41	0.06	1.39	13.34	2.00	2.40
	S53	0.08	0.00	11.23	0.08	849.27	0.37	1.73	216.36	1.64	35.08
	S54	0.01	0.00	0.37	0.00	4.24	0.05	2.18	23.45	0.90	2.73
	S55	0.17	0.05	33.60	0.02	158.48	0.06	2.57	20.39	13.60	19.75

Table 9.3: *Comparison of Trinity's efficiency to other techniques. (Cont'd)*

quick look at the scatter plot in Figure §9.4 makes it clear that these cloud of points are far from a Gaussian circle. As a conclusion, we have performed a non-parametric analysis whose results are presented in Table §9.4. Note that the P-value of Iman-Davenport's statistic is nearly zero in every case, which is a strong indication that there are statistically significant differences in the ranks we have computed from our experiments. It then proceeds to rank the techniques pairwise using Bergmann-Hommel's test. For the sake of readability, we also provide an explicit ranking in the last column. Note that our proposal ranks the first regarding every effectiveness and efficiency measure; the only tie is regarding extraction time, in which case the difference with respect to RoadRunner does not seem to be statistically significant. As a conclusion, our experiments prove that there is enough statistical evidence to conclude that our proposal outperform the others.

9.6 Summary

In this chapter, we have proposed a new effective and efficient unsupervised information extractor called Trinity. It is based on the hypothesis that web documents generated by the same server-side template share patterns that do not provide any relevant information, but help delimit it. The rule learning algorithm searches for these patterns and creates a trinary tree,



Criterion	Sample ranking		Iman-Davenport's test	Bergmann-Hommels's test						Statistical ranking	
	Technique	Rank	*P*-value		Trinity	RoadRunner	FiVaTech	SM	LR	Technique	Rank
P	Trinity	1.64	2.11E-17	Trinity	-	2.38E-15	3.56E-06	2.76E-04	2.06E-07	Trinity	1
	SM	2.84		RoadRunner		-	3.04E-03	1.46E-04	1.46E-02	SM	2
	FiVaTech	3.12		FiVaTech			-	3.72E-01	5.46E-01	FiVaTech	2
	LR	3.30		SM				-	3.72E-01	LR	2
	RoadRunner	4.11		LR					-	RoadRunner	3
	Technique	Rank	*P*-value		Trinity	RoadRunner	FiVaTech	SM	LR	Technique	Rank
R	Trinity	1.68	2.74E-17	Trinity	-	7.58E-13	2.02E-02	4.05E-08	2.96E-08	Trinity	1
	FiVaTech	2.53		RoadRunner		-	1.78E-05	2.41E-01	2.41E-01	FiVaTech	2
	SM	3.41		FiVaTech			-	6.97E-03	6.97E-03	SM	3
	LR	3.45		SM				-	9.04E-01	LR	3
	RoadRunner	3.94		LR					-	RoadRunner	3
	Technique	Rank	*P*-value		Trinity	RoadRunner	FiVaTech	SM	LR	Technique	Rank
F1	Trinity	1.55	4.79E-18	Trinity	-	3.93E-15	6.48E-05	1.93E-07	8.15E-09	Trinity	1
	FiVaTech	2.85		RoadRunner		-	7.71E-04	2.19E-02	7.50E-02	FiVaTech	2
	SM	3.20		FiVaTech			-	2.52E-01	2.41E-01	SM	2
	LR	3.38		SM				-	5.46E-01	LR	2
	RoadRunner	4.01		LR					-	RoadRunner	2
	Technique	Rank	*P*-value		Trinity	RoadRunner	FiVaTech	SM	LR	Technique	Rank
LT	Trinity	1.01	7.90E-72	Trinity	-	6.36E-35	5.11E-06	1.77E-17	5.54E-12	Trinity	1
	RoadRunner	2.43		RoadRunner		-	3.45E-14	1.82E-04	1.72E-02	RoadRunner	2
	LR	3.15		FiVaTech			-	2.91E-04	2.29E-07	LR	3
	SM	3.64		SM				-	1.03E-01	SM	3
	FiVaTech	4.78		LR					-	FiVaTech	4
	Technique	Rank	*P*-value		Trinity	RoadRunner	FiVaTech	SM	LR	Technique	Rank
ET	Trinity	1.50	6.79E-124	Trinity	-	8.56E-01	6.54E-06	7.57E-30	4.03E-16	Trinity	1
	RoadRunner	1.55		RoadRunner		-	7.93E-06	3.66E-29	9.18E-16	RoadRunner	1
	FiVaTech	2.95		FiVaTech			-	5.76E-11	7.48E-04	FiVaTech	2
	LR	4.02		SM				-	2.79E-03	LR	3
	SM	4.98		LR					-	SM	4

Table 9.4: *Results of ranking Trinity statistically.*

which is then used to learn a regular expression that represents the template that was used to generate input web documents and the schema of the extracted information. Our experiments on 2 084 real-world web documents proved that our technique achieves very high precision and recall, which are very close to 100%; furthermore, the rule learning time and the extraction time are very small.

Part IV

Final Remarks

Chapter 10

Conclusions

> Even when we are unable to see the light at the end of the tunnel, we ought to believe that there is light, and that one day, undoubtedly, we will see it.
>
> Amin Maalouf, Lebanese author

In this dissertation, we have presented CEDAR, a reference architecture that is accompanied with a software framework to help software engineers devise new learning techniques in the domain of information extraction from semi-structured web documents. It provides an abstract and reusable design that should allow software engineers and researchers to face the development of a new information extraction techniques without incurring the high costs of developing it from scratch. The reference architecture was validated by implementing four techniques that got inspiration from classical techniques in the literature, with an overall time reduction that goes beyond 60%. CEDAR was used to devise and implement two new information extraction techniques that were also presented in this dissertation, namely: TEX and Trinity.

TEX is a completely unsupervised information extractor that focuses on extracting attributes. It saves end users from the burden of annotating training examples to learn extraction rules, and from maintaining extraction rules. Trinity is an effective and efficient unsupervised information extraction technique that learns a regular expression that represents the template that was used to generate the input web document. They both build on the hypothesis that web documents generated by the same server-side template share patterns that do not provide any relevant information, but help delimit it. Both TEX and Trinity achieve very good performance, which suggest that our proposals seem promising enough to extract information from real-world web documents.

In addition to enterprise information integration, the rules and the schema learnt by Trinity have more potential uses in other fields, and it is our plan to research on them as future work: they can be used in the context of web site model discovery and on mapping semi-structured web sites onto ontological models to populate them [66, 147]. The idea of using hybrid machine learning techniques in this field remains unexplored and shall be paid much attention in future research activities.

Bibliography

[1] B. Adelberg. *NoDoSE: a tool for semi-automatically extracting semi-structured data from text documents*. In *SIGMOD Conference*, pages 283–294, 1998.

[2] E. Agichtein and L. Gravano. *Snowball: extracting relations from large plain-text collections*. In *ICDL*, pages 85–94, 2000.

[3] M. Agosti, F. Crivellari, and M. Melucci. *Evaluation methods to improve information content extraction from the web*. In *Workshop on Web Information and Data Management*, pages 25–28, 1998.

[4] H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis, and N. Shadbolt. *Automatic ontology-based knowledge extraction from web documents*. *IEEE Intelligent Systems*, 18(1):14–21, 2003.

[5] R. Albert, H. Jeong, and A.-L. Barabási. *The diameter of the world wide web*. *Nature*, 401:130–131, 1999.

[6] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda. *Extracting lists of data records from semi-structured web pages*. *Data Knowl. Eng.*, 64(2):491–509, 2008.

[7] A. Arasu and H. Garcia-Molina. *Extracting structured data from web pages*. In *SIGMOD Conference*, pages 337–348, 2003.

[8] J. L. Arjona, R. Corchuelo, D. Ruiz, and M. Toro. *From wrapping to knowledge*. *IEEE Trans. Knowl. Data Eng.*, 19(2):310–323, 2007.

[9] G. O. Arocena and A. O. Mendelzon. *WebOQL: restructuring documents, databases, and webs*. *TAPOS*, 5(3):127–141, 1999.

[10] F. Ashraf, T. Özyer, and R. Alhajj. *Employing clustering techniques for automatic information extraction from HTML documents*. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(5):660–673, 2008.

[11] R. Basili, M. T. Pazienza, and M. Vindigni. *Corpus-driven learning of event recognition rules*. In *ECAI Workshop on Machine Learning for Information Extraction*, pages 1–7, 2000.

[12] R. Baumgartner, S. Flesca, and G. Gottlob. *Visual web information extraction with Lixto*. In *VLDB*, pages 119–128, 2001.

[13] B. Bergmann and G. Hommel. *Improvements of general multiple test procedures for redundant systems of hypotheses*. In *Multiple Hypothesis Testing Symposium*, pages 100–115, 1988.

[14] L. Bing, W. Lam, and Y. Gu. *Towards a unified solution: data record region detection and segmentation*. In *CIKM*, pages 1265–1274, 2011.

[15] D. Buttler, L. Liu, and C. Pu. *A fully automated object extraction system for the World Wide Web*. In *ICDCS*, pages 361–370, 2001.

[16] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. *Extracting content structure for web pages based on visual representation*. In *APWeb*, pages 406–417, 2003.

[17] M. E. Califf and R. J. Mooney. *Bottom-up relational learning of pattern matching rules for information extraction*. *Journal of Machine Learning Research*, 4:177–210, 2003.

[18] A. Carlson and C. Schafer. *Bootstrapping information extraction from semi-structured web pages*. In *ECML/PKDD (1)*, pages 195–210, 2008.

[19] J. Y. Chai, A. W. Biermann, and C. I. Guinn. *Two dimensional generalization in information extraction*. In *AAAI*, pages 431–438, 1999.

[20] C.-H. Chang and S.-C. Kuo. *OLERA: semisupervised web-data extraction with visual support*. *IEEE Intelligent Systems*, 19(6):56–64, 2004.

[21] C.-H. Chang and S.-C. Lui. *IEPAD: information extraction based on pattern discovery*. In *WWW*, pages 681–688, 2001.

[22] B. Chidlovskii, B. Roustant, and M. Brette. *Documentum ECI self-repairing wrappers: performance analysis*. In *SIGMOD Conference*, pages 708–717, 2006.

[23] H. L. Chieu and H. T. Ng. *A maximum entropy approach to information extraction from semi-structured and free text*. In *AAAI*, pages 786–791, 2002.

[24] H. L. Chieu, H. T. Ng, and Y. K. Lee. *Closing the gap: learning-based information extraction rivaling knowledge-engineering methods*. In *ACL*, pages 216–223, 2003.

[25] L. Chiticariu, Y. Li, S. Raghavan, and F. Reiss. *Enterprise information extraction: recent developments and open challenges*. In *SIGMOD Conference*, pages 1257–1258, 2010.

[26] P. Cimiano and J. Völker. *Text2Onto: A framework for ontology learning and data-driven change discovery*. In *Applications of Natural Language to Databases*, pages 227–238, 2005.

[27] F. Ciravegna, A. Dingli, Y. Wilks, and D. Petrelli. *Amilcare: adaptive information extraction for document annotation*. In *Research and Development in Information Retrieval*, pages 367–368, 2002.

[28] W. W. Cohen, M. Hurst, and L. S. Jensen. *A flexible learning system for wrapping tables and lists in HTML documents*. In *WWW*, pages 232–241, 2002.

[29] C. Cox, J. Nicolson, J. R. Finkel, C. Manning, and P. Langley. *Template sampling for leveraging domain knowledge in information extraction*. In *PASCAL Challenges Workshop*, 2005.

[30] V. Crescenzi and G. Mecca. *Grammars have exceptions*. *Inf. Syst.*, 23(8): 539–565, 1998.

[31] V. Crescenzi and G. Mecca. *Automatic information extraction from large websites*. *J. ACM*, 51(5):731–779, 2004.

[32] V. Crescenzi, G. Mecca, and P. Merialdo. *RoadRunner: towards automatic data extraction from large web sites*. In *VLDB*, pages 109–118, 2001.

[33] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. *GATE: a framework and graphical development environment for robust NLP tools and applications*. In *Meeting of the Association for Computational Linguistics*, pages 1–8, 2002.

[34] N. N. Dalvi, A. Machanavajjhala, and B. Pang. *An analysis of structured data on the Web*. *PVLDB*, 5(7):680–691, 2012.

[35] I. F. de Viana, I. Hernández, P. Jiménez, C. R. Rivero, and H. A. Sleiman. *Integrating deep-web information sources*. In *PAAMS (Special Sessions and Workshops)*, pages 311–320, 2010.

[36] *Denodo*, September 2012. URL: http://www.denodo.com.

[37] J. Derrac, S. García, D. Molina, and F. Herrera. *A practical tutorial on the use of non-parametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms*. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[38] H. Elmeleegy, J. Madhavan, and A. Y. Halevy. *Harvesting relational tables from lists on the Web*. *PVLDB*, 2(1):1078–1089, 2009.

[39] D. W. Embley, Y. S. Jiang, and Y.-K. Ng. *Record-boundary discovery in web documents*. In *SIGMOD Conference*, pages 467–478, 1999.

[40] A. Esuli and F. Sebastiani. *Evaluating information extraction*. In *CLEF*, pages 100–111, 2010.

[41] M. E. Fayad, D. C. Schmidt, and R. E. Johnson. *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, 1999.

[42] G. Fernández, H. A. Sleiman, and R. Corchuelo. *An annotation tool for semantic web*. In *CAEPIA*, pages 1–8, 2011.

[43] G. Fernández, H. A. Sleiman, and R. Corchuelo. *An experiment on using datamining techniques to extract information from the web*. *PAAMS (Workshops)*, pages 169–176, 2011.

[44] G. Fernández, H. A. Sleiman, R. Corchuelo, and R. Z. Frantz. *On mining dom trees to build information extractors*. In *ICOMP*, pages 363–367, 2011.

[45] D. Ferrucci and A. Lally. *UIMA: an architectural approach to unstructured information processing in the corporate research environment*. *Nat. Lang. Eng.*, 10(3-4):327–348, 2004.

[46] *Fetch technologies*, January 2012. URL: http://www.fetch.com.

[47] A. Finn and N. Kushmerick. *Information extraction by convergent boundary classification*. In *AAAI Workshop on Adaptive Text Extraction And Mining*, 2004.

[48] D. Freitag. *Information extraction from HTML: application of a general machine learning approach*. In *AAAI/IAAI*, pages 517–523, 1998.

[49] D. Freitag. *Toward general-purpose learning for information extraction*. In *COLING-ACL*, pages 404–408, 1998.

[50] D. Freitag and A. McCallum. *Information extraction with HMM structures learned by stochastic optimization*. In *AAAI*, pages 584–589, 2000.

[51] D. Freitag and A. K. Mccallum. *Information extraction with HMMs and shrinkage*. In *AAAI Workshop on Machine Learning for Information Extraction*, pages 31–36, 1999.

[52] R. J. Gaizauskas, K. Humphreys, H. Cunningham, and Y. Wilks. *University of Sheffield: description of the LaSIE system as used for MUC-6*. In *MUC*, pages 207–220, 1995.

[53] S. García, A. Fernández, J. Luengo, and F. Herrera. *Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power*. *Inf. Sci.*, 180(10):2044–2064, 2010.

[54] G. H. Gonnet, R. A. Baeza-Yates, and T. Snider. *New indices for text: PAT trees and PAT arrays*. In W. B. Frakes and R. Baeza-Yates, editors, *Information retrieval*, pages 66–82. Prentice-Hall, 1992.

[55] D. G. Gregg and S. Walczak. *Exploiting the Information Web*. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 37(1):109–125, 2007.

[56] P. Gulhane, A. Madaan, R. R. Mehta, J. Ramamirtham, R. Rastogi, S. Satpal, S. H. Sengamedu, A. Tengli, and C. Tiwari. *Web-scale information extraction with Vertex*. In *ICDE*, pages 1209–1220, 2011.

[57] P. Gulhane, R. Rastogi, S. H. Sengamedu, and A. Tengli. *Exploiting content redundancy for web information extraction*. In *WWW*, pages 1105–1106, 2010.

[58] A. Gulli and A. Signorini. *The indexable Web is more than 11.5 billion pages*. In *WWW (Special interest tracks and posters)*, pages 902–903, 2005.

[59] R. Gupta and S. Sarawagi. *Answering table augmentation queries from unstructured lists on the Web*. *PVLDB*, 2(1):289–300, 2009.

[60] D. Gusfield. *Algorithms on strings, trees, and sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[61] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. *Data integration: the teenage years*. In *VLDB*, pages 9–16, 2006.

[62] J. Hammer, J. McHugh, and H. Garcia-Molina. *Semistructured data: the Tsimmis experience*. In *ADBIS*, pages 1–8, 1997.

[63] W. Han, D. Buttler, and C. Pu. *Wrapping web data into XML*. *SIGMOD Record*, 30(3):33–38, 2001.

[64] D. Harman and M. Liberman. *TIPSTER complete*, 1993. URL: http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC93T3A.

[65] M. Hepple. *Independence and commitment: assumptions for rapid training and execution of rule-based POS taggers*. In *ACL*, 2000.

[66] I. Hernández, C. R. Rivero, D. Ruiz, and R. Corchuelo. *Towards discovering conceptual models behind web sites*. In *ER*, 2012.

[67] I. Hernández, H. A. Sleiman, D. Ruiz, and R. Corchuelo. *A conceptual framework for efficient web crawling in virtual integration contexts*. In *WISM (2)*, pages 282–291, 2011.

[68] I. Hernández, H. A. Sleiman, D. Ruiz, and R. Corchuelo. *A tool for web links prototyping*. In *ICAI*, pages 1–7, 2011.

[69] L. Hirschman. *The evolution of evaluation: lessons from the Message Understanding Conferences*. *Computer Speech & Language*, 12 (4):281–305, 1998.

[70] V. J. Hodge and J. Austin. *A survey of outlier detection methodologies*. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[71] A. W. Hogue and D. R. Karger. *Thresher: automating the unwrapping of semantic content from the World Wide Web*. In *WWW*, pages 86–95, 2005.

[72] J. L. Hong, E.-G. Siew, and S. Egerton. *Information extraction for search engines using fast heuristic techniques*. *Data Knowl. Eng.*, 69(2): 169–196, 2010.

[73] C.-N. Hsu and M.-T. Dung. *Generating finite-state transducers for semi-structured data extraction from the Web*. *Inf. Syst.*, 23(8):521–538, 1998.

[74] S. B. Huffman. *Learning information extraction patterns from examples*. In *Learning for Natural Language Processing*, pages 246–260, 1995.

[75] *Unstructured information: the knowledge rush*, 2011. URL: http://tinyurl.com/UIMA-KR.

[76] R. L. Iman and J. M. Davenport. *Approximations to the critical region of the Friedman statistic*. *Comm. Stat.: Theor. Meth*, A9(6):571–595, 1980.

[77] N. Ireson, F. Ciravegna, M. E. Califf, D. Freitag, N. Kushmerick, and A. Lavelli. *Evaluating machine learning for information extraction*. In *International Conference on Machine Learning*, pages 345–352, 2005.

[78] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. *Avatar information extraction system*. *IEEE Data Eng. Bull.*, 29 (1):40–48, 2006.

[79] N. Kambhatla. *Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations*. In *ACL (Interactive poster & demonstration sessions)*, pages 1–4, 2004.

[80] J. Kang and J. Choi. *Recognising informative web page blocks using visual segmentation for efficient information extraction*. *J. UCS*, 14(11): 1893–1910, 2008.

[81] M. Kayed and C.-H. Chang. *FiVaTech: page-level web data extraction from template pages*. *IEEE Trans. Knowl. Data Eng.*, 22(2):249–263, 2010.

[82] M. G. Kendall. *A new measure of rank correlation*. *Biometrika*, 30(1-2): 81–89, 1983.

[83] J.-T. Kim and D. I. Moldovan. *Acquisition of linguistic patterns for knowledge-based information extraction*. *IEEE Trans. Knowl. Data Eng.*, 7(5):713–724, 1995.

[84] Y.-J. Kim. *Emerging trends: 2010 through 2015*, 2005. URL: http://www.kait.or.kr/filedb/051207-it839/KAIT-1.pdf.

[85] B. Kitchenham, S. L. Pfleeger, L. Pickard, P. Jones, D. C. Hoaglin, J. Rosenberg, and K. E. Emam. *Preliminary guidelines for empirical research in software engineering*. *IEEE Trans. Software Eng.*, 28(8): 721–734, 2002.

[86] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt. *Fast pattern matching in strings*. *SIAM J. Comput.*, 6(2):323–350, 1977.

[87] R. Kosala, H. Blockeel, M. Bruynooghe, and J. V. den Bussche. *Information extraction from structured documents using* k*-testable tree automaton inference*. *Data Knowl. Eng.*, 58(2):129–158, 2006.

[88] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. *SystemT: a system for declarative information extraction*. *SIGMOD Record*, 37(4):7–13, 2008.

[89] P. Kruchten. *The 4+1 view model of architecture*. *IEEE Software*, 12(6): 42–50, 1995.

[90] N. Kumar, J. D. Beer, J. Vanthienen, and M.-F. Moens. *Evaluation of information retrieval and text mining tools on automatic named entity extraction*. In *Internationales Symposium für Informationswissenschaft*, pages 666–667, 2006.

[91] N. Kushmerick. *Regression testing for wrapper maintenance*. In *AAAI/IAAI*, pages 74–79, 1999.

[92] N. Kushmerick. *Wrapper induction: efficiency and expressiveness*. *Artif. Intell.*, 118(1-2):15–68, 2000.

[93] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. *Wrapper induction for information extraction*. In *IJCAI (1)*, pages 729–737, 1997.

[94] A. H. F. Laender, B. A. Ribeiro-Neto, and A. S. da Silva. *DEByE: data extraction by example*. *Data Knowl. Eng.*, 40(2):121–154, 2002.

[95] A. Lavelli, M. E. Califf, F. Ciravegna, D. Freitag, N. Kushmerick, C. Giuliano, L. Romano, and N. Ireson. *Evaluation of machine learning-based information extraction algorithms: Criticisms and recommendations*. *Language Resources and Evaluation*, 42(4):361–393, 2008.

[96] K. Lerman, S. Minton, and C. A. Knoblock. *Wrapper maintenance: a machine learning approach*. *J. Artif. Intell. Res.*, 18:149–181, 2003.

[97] L. Li, Y. Liu, A. Obregon, and M. Weatherston. *Visual segmentation-based data record extraction from web documents*. In *IRI*, pages 502–507, 2007.

[98] Q. Li, Y. Ding, A. Feng, and Y. Dong. *A novel method for extracting information from web pages with multiple presentation templates*. *JSW*, 5(5):506–513, 2010.

[99] Y. Li, L. Chiticariu, H. Yang, F. Reiss, and A. Carreno-fuentes. *WizIE: a best practices guided development environment for information extraction*. In *ACL*, pages 109–114, July 2012.

[100] B. Liu, R. L. Grossman, and Y. Zhai. *Mining web pages for data records*. *IEEE Intelligent Systems*, 19(6):49–55, 2004.

[101] B. Liu and Y. Zhai. *NET: a system for extracting web data from flat and nested data records*. In *WISE*, pages 487–495, 2005.

[102] L. Liu, C. Pu, and W. Han. *XWRAP: an XML-enabled wrapper construction system for web information sources*. In *ICDE*, pages 611–621, 2000.

[103] W. Liu, X. Meng, and W. Meng. *ViDE: a vision-based approach for deep web data extraction*. *IEEE Trans. Knowl. Data Eng.*, 22(3):447–460, 2010.

[104] W. Liu, D. Shen, and T. Nie. *An effective method supporting data extraction and schema recognition on the Deep Web*. In *APWeb*, pages 419–431, 2008.

[105] *Lixto*, September 2012. URL: http://www.lixto.com.

[106] A. Machanavajjhala, A. S. Iyer, P. Bohannon, and S. Merugu. *Collective extraction from heterogeneous web lists*. In *WSDM*, pages 445–454, 2011.

[107] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. *Web-scale data integration: you can afford to pay as you go*. In *Conference on Innovative Data Systems Research*, pages 342–350, 2007.

[108] D. Maynard and H. Cunningham. *Multilingual adaptations of ANNIE, a reusable information extraction tool*. In *ACL*, volume 2, pages 219–222, 2003.

[109] D. Maynard, H. Cunningham, K. Bontcheva, R. Catizone, G. Demetriou, R. Gaizauskas, O. Hamza, M. Hepple, and P. Herring. *A survey of uses of GATE*. Technical report, University of Sheffield, 2000.

[110] X. Meng, D. Hu, and C. Li. *Schema-guided wrapper maintenance for web-data extraction*. In *WIDM*, pages 1–8, 2003.

[111] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. *Extracting data records from the Web using tag path clustering*. In *WWW*, pages 981–990, 2009.

[112] R. C. Miller and B. A. Myers. *Lightweight structured text processing*. In *USENIX Annual Technical Conference, General Track*, pages 131–144, 1999.

[113] S. Miller, H. Fox, L. A. Ramshaw, and R. M. Weischedel. *A novel use of statistical parsing to extract information from text*. In *ANLP*, pages 226–233, 2000.

[114] S. Minton, S. I. Ticrea, and J. Beach. *Trainability: developing a responsive learning system*. In *IIWeb*, pages 27–32, 2003.

[115] P. Montoto, A. Pan, J. Raposo, J. Losada, F. Bellas, and V. Carneiro. *A workflow language for web automation*. *J. UCS*, 14(11):1838–1856, 2008.

[116] D. R. Morrison. *Patricia - practical algorithm to retrieve information coded in alphanumeric*. *J. ACM*, 15(4):514–534, 1968.

[117] I. Muslea. *RISE: repository of online information sources used in information extraction*, 1998. URL: http://www.isi.edu/info-agents/RISE.

[118] I. Muslea, S. Minton, and C. A. Knoblock. *Hierarchical wrapper induction for semistructured information sources*. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[119] N. Papadakis, D. Skoutas, K. Raftopoulos, and T. A. Varvarigou. *STAVIES: a system for information extraction from unknown web data sources through automatic web wrapper generation using clustering techniques*. *IEEE Trans. Knowl. Data Eng.*, 17(12):1638–1652, 2005.

[120] J. Park and D. Barbosa. *Adaptive record extraction from web pages*. In *WWW*, pages 1335–1336, 2007.

[121] L. Peshkin and A. Pfeffer. *Bayesian information extraction network*. In *IJCAI*, pages 421–426, 2003.

[122] B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, and M. Goranov. *KIM: semantic annotation platform*. In *International Semantic Web Conference*, pages 834–849, 2003.

[123] J. Raposo, A. Pan, M. Álvarez, and J. Hidalgo. *Automatically generating labeled examples for web wrapper maintenance*. In *Web Intelligence*, pages 250–256, 2005.

[124] J. Raposo, A. Pan, M. Álvarez, J. Hidalgo, and Á. Viña. *The Wargo System: semi-automatic wrapper generation in presence of complex data access modes*. In *DEXA Workshops*, pages 313–320, 2002.

[125] J. Raposo, A. Pan, M. Álvarez, and Á. Viña. *Automatic wrapper maintenance for semi-structured web sources using results from previous queries*. In *SAC*, pages 654–659, 2005.

[126] L. F. Rau. *Information extraction and evaluation*. In *MUC*, page 349, 1993.

[127] E. Riloff. *Automatically generating extraction patterns from untagged text*. In *AAAI/IAAI*, volume 2, pages 1044–1049, 1996.

[128] D. Roth and W.-T. Yih. *Relational learning via propositional algorithms: an information extraction case study*. In *IJCAI*, pages 1257–1263, 2001.

[129] N. Sager. *Medical language processing: computer management of narrative data*. Addison-Wesley, 1987.

[130] A. Sahuguet and F. Azavant. *Building intelligent web applications using lightweight wrappers*. *Data Knowl. Eng.*, 36(3):283–316, 2001.

[131] S. Sarawagi. *Information extraction*. *Foundations and Trends in Databases*, 1(3):261–377, 2007.

[132] K. Seymore, A. McCallum, and R. Rosenfeld. *Learning hidden Markov model structure for information extraction*. In *AAAI*, pages 37–42, 1999.

[133] J. P. Shaffer. *Modified sequentially rejective multiple test procedures*. *Journal of the American Statistical Association*, 81(395):826–831, 1986.

[134] Y. K. Shen and D. R. Karger. *U-REST: an unsupervised record extraction system*. In *WWW*, pages 1347–1348, 2007.

[135] D. J. Sheskin. *Handbook of parametric and non-parametric statistical procedures*. Chapman and Hall/CRC, edition 5, 2011.

[136] K. Simon and G. Lausen. *ViPER: augmenting automatic information extraction with visual perceptions*. In *CIKM*, pages 381–388, 2005.

[137] M. Skounakis, M. Craven, and S. Ray. *Hierarchical hidden Markov models for information extraction*. In *IJCAI*, pages 427–433, 2003.

[138] H. A. Sleiman. *Information extraction from the World Wide Web*. In *TJISBD*, volume 3, pages 18–29, 2009.

[139] H. A. Sleiman and R. Corchuelo. *An architecture for web information agents*. In *ISDA*, pages 18–23, 2011.

[140] H. A. Sleiman and R. Corchuelo. *Information extraction framework*. In *PAAMS (Workshops)*, pages 149–156, 2012.

[141] H. A. Sleiman and R. Corchuelo. *A reference architecture to devise web information extractors*. In *CAiSE Workshops*, pages 235–248, 2012.

[142] H. A. Sleiman and R. Corchuelo. *A survey on region extractors from web documents*. *IEEE Trans. Knowl. Data Eng.*, 99(PrePrints), 2012.

[143] H. A. Sleiman and R. Corchuelo. *TEX: An efficient and effective unsupervised web information extractor*. *Knowledge-Based Systems*, 2012. (To be published).

[144] H. A. Sleiman and R. Corchuelo. *Towards a method for unsupervised web information extraction*. In *ICWE*, pages 427–430, 2012.

[145] H. A. Sleiman and R. Corchuelo. *An unsupervised technique to extract information from semi-structured web pages*. In *WISE*, 2012. (To be published).

[146] H. A. Sleiman, I. Hernández, G. Fernández, and R. Corchuelo. *A transducer model for web information extraction*. In *ICAI*, pages 1–8, 2011.

[147] H. A. Sleiman and I. Hernández. *A framework for populating ontological models from semi-structured web documents*. In *ER*, pages 1–6, 2012.

[148] H. A. Sleiman, C. R. Rivero, and R. Corchuelo. *On a proposal to integrate web sources using semantic-web technologies*. In *NWeSP*, pages 326–331, 2011.

[149] H. A. Sleiman, A. W. Sultán, R. Z. Frantz, and R. Corchuelo. *Towards automatic code generation for eai solutions using dsl tools*. In *JISBD*, pages 134–145, 2009.

[150] S. Soderland. *Learning to extract text-based information from the World Wide Web*. In *KDD*, pages 251–254, 1997.

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

[152] W. Su, J. Wang, and F. H. Lochovsky. *ODE: ontology-assisted data extraction*. *ACM Trans. Database Syst.*, 34(2), 2009.

[153] L. V. Subramaniam, S. Mukherjea, P. Kankar, B. Srivastava, V. S. Batra, R. Kothari, and P. V. Kamesam. *Information extraction from biomedical literature: methodology, evaluation and an application*. In *International Conference on Information and Knowledge Management*, pages 410–417, 2003.

[154] F. M. Suchanek, M. Sozio, and G. Weikum. *SOFIE: a self-organizing framework for information extraction*. In *WWW*, pages 631–640, 2009.

[155] A. Sun, M.-M. Naing, E.-P. Lim, and W. Lam. *Using support vector machines for terrorism information extraction*. In *ISI*, pages 1–12, 2003.

[156] B. Sundheim. *TIPSTER/MUC-5: information extraction system evaluation*. In *MUC*, pages 27–44, 1993.

[157] C. Tao and D. W. Embley. *Automatic hidden-web table interpretation, conceptualization, and semantic annotation*. *Data Knowl. Eng.*, 68 (7):683–703, 2009.

[158] J. Turmo, A. Ageno, and N. Català. *Adaptive information extraction*. *ACM Comput. Surv.*, 38(2), 2006.

[159] J. Turmo and H. Rodríguez. *Learning rules for information extraction*. *Nat. Lang. Eng.*, 8:167–191, 2002.

[160] *The UCI repository*, September 2012. URL: http://archive.ics.uci.edu/ml.

[161] J. Wang and F. H. Lochovsky. *Data-rich section extraction from HTML pages*. In *WISE*, pages 313–322, 2002.

[162] J. Wang and F. H. Lochovsky. *Data extraction and label assignment for web databases*. In *WWW*, pages 187–196, 2003.

[163] L. Wei, X. Meng, and W. Meng. *Vision-based web data records extraction*. In *WebDB*, 2006.

[164] Y. Yamada, N. Craswell, T. Nakatoh, and S. Hirokawa. *Testbed for information extraction from the Deep Web*. In *WWW (Alternate Track Papers & Posters)*, pages 346–347, 2004.

[165] W. Yang. *Identifying syntactic differences between two programs*. *Softw., Pract. Exper.*, 21(7):739–755, 1991.

[166] R. Yangarber. *Counter-training in discovery of semantic patterns*. In *ACL*, pages 343–350, 2003.

[167] L. Yi, B. Liu, and X. Li. *Eliminating noisy information in web pages for data mining*. In *KDD*, pages 296–305, 2003.

[168] D. Zelenko, C. Aone, and A. Richardella. *Kernel methods for relation extraction*. *Journal of Machine Learning Research*, 3:1083–1106, 2003.

[169] Y. Zhai and B. Liu. *Structured data extraction from the Web based on partial tree alignment*. *IEEE Trans. Knowl. Data Eng.*, 18(12):1614–1628, 2006.

[170] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. T. Yu. *Fully automatic wrapper generation for search engines*. In *WWW*, pages 66–75, 2005.

[171] H. Zhao, W. Meng, and C. T. Yu. *Automatic extraction of dynamic record sections from search engine result pages*. In *VLDB*, pages 989–1000, 2006.

[172] S. Zhao and R. Grishman. *Extracting relations with integrated information using kernel methods*. In *ACL*, 2005.

[173] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. *Simultaneous record detection and attribute labeling in web data extraction*. In *KDD*, pages 494–503, 2006.

This document was typeset on November 15, 2013 at 12:54 using class RC-BOOK α2.14 for LaTeX2ε. As of the time of writing this document, this class is not publicly available since it is in alpha version. Only members of The Distributed Group are using it to typeset their documents. Should you be interested in giving forthcoming public versions a try, please, do contact us at contact@tdg-seville.info. Thanks!