

---

# ENTERPRISE INFORMATION INTEGRATION



AN UNSUPERVISED PROPOSAL FOR WEB PAGE  
CLASSIFICATION

---

INMA HERNÁNDEZ

UNIVERSITY OF SEVILLA, SPAIN

DOCTORAL DISSERTATION

SUPERVISED BY DR. DAVID RUIZ AND DR. RAFAEL CORCHUELO



SEPTEMBER, 2012

First published in September 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](mailto:contact@tdg-seville.info)

In keeping with the traditional purpose of furthering science, education and research, it is the policy of the publisher, whenever possible, to permit non-commercial use and redistribution of the information contained in the documents whose copyright they own. You however are *not allowed* to take money for the distribution or use of these results except for a nominal charge for photocopying, sending copies, or whichever means you use redistribute them. The results in this document have been tested carefully, but they are not guaranteed for any particular purpose. The publisher or the holder of the copyright do not offer any warranties or representations, nor do they accept any liabilities with respect to them.

**Classification (ACM 1998):** H.5.1 [Multimedia Information Systems]: Hypertext navigation and maps; H.5.4 [Hypertext/Hypermedia]: Navigation; I.2.6 [Learning] Concept learning, Induction; I.5.2 [Design Methodology]: Classifier design and evaluation; I.5.3 [Clustering]: Algorithms.

**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, and TIN2010-09988-E, TIN2011-15497-E).

# University of Sevilla, Spain

The committee in charge of evaluating the dissertation presented by Inma Hernández 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 \_\_\_\_\_.

---

José Miguel Toro Bonilla  
Catedrático de Universidad  
Univ. de Sevilla

---

Carlos Delgado Kloos  
Catedrático de Universidad  
Univ. Politécnica de Madrid

---

Juan Manuel Corchado Rodríguez  
Catedrático de Universidad  
Univ. de Salamanca

---

Manuel Lama Penín  
Profesor Titular de Universidad  
Univ. de Santiago de Compostela

---

Carlos Alberto Pan Bermúdez  
Profesor Contratado Doctor  
Univ. de A Coruña

To put record where necessary, we sign minutes in \_\_\_\_\_,  
\_\_\_\_\_.





Classification of geometrical shapes into patterns: triangles, diamonds, squares, and circles.

By Laura, aged four.



To Carlos, my sun and stars





---

# Contents

---

Acknowledgements ..... xi

Abstract ..... xiii

Resumen ..... xv

## I Preface

**1 Introduction ..... 3**

1.1 Research context ..... 4

1.2 Research rationale ..... 5

1.2.1 Hypothesis ..... 5

1.2.2 Thesis ..... 6

1.3 Summary of contributions ..... 6

1.4 Collaborations ..... 7

1.5 Structure of this dissertation ..... 7

**2 Motivation ..... 9**

2.1 Introduction ..... 10

2.2 Requirements ..... 10

2.3 Analysis of current solutions ..... 11

2.4 Our proposal ..... 15

2.5 Summary ..... 15

## II Background Information

**3 Enterprise web information integration ..... 19**

3.1	Introduction .....	20
3.2	Mediators .....	21
3.3	Wrappers .....	24
3.4	Summary .....	26
<b>4</b>	<b>Automated web navigation .....</b>	<b>27</b>
4.1	Introduction .....	28
4.2	Taxonomy of web pages .....	29
4.3	Scripting proposals .....	32
4.3.1	Script recorders .....	32
4.3.2	Script learners .....	35
4.4	Crawling proposals .....	36
4.4.1	Blind crawling .....	37
4.4.2	Focused crawling .....	38
4.5	Usage mining proposals .....	40
4.6	Summary .....	41
<b>5</b>	<b>Web page classification .....</b>	<b>43</b>
5.1	Introduction .....	44
5.2	Contents-based proposals .....	45
5.3	Link-based proposals .....	47
5.4	Visual-based proposals .....	48
5.5	URL-based proposals .....	49
5.6	Structure-based proposals .....	51
5.7	Summary .....	53

## III Our Proposal

<b>6</b>	<b>Our crawler .....</b>	<b>57</b>
6.1	Introduction .....	58
6.2	Architecture .....	59
6.3	Algorithm .....	61
6.3.1	Computing keywords .....	64
6.3.2	Discarding empty hubs .....	65
6.3.3	Other ancillary functions .....	66

6.4	Analysis .....	67
6.4.1	Ancillary functions .....	67
6.4.2	Algorithm .....	68
6.5	Summary .....	69
<b>7</b>	<b>Our pattern builder .....</b>	<b>71</b>
7.1	Introduction .....	72
7.2	Architecture .....	72
7.3	Algorithm .....	74
7.3.1	Initialising the prefix set .....	76
7.3.2	Computing siblings .....	77
7.3.3	Computing p-estimators .....	78
7.3.4	Wildcarding prefixes .....	80
7.3.5	Updating the prefix set .....	82
7.4	Analysis .....	82
7.4.1	Ancillary functions .....	82
7.4.2	Algorithm .....	84
7.5	Summary .....	87
<b>8</b>	<b>Evaluation .....</b>	<b>89</b>
8.1	Introduction .....	90
8.2	Experimental evaluation .....	90
8.2.1	Experimentation environment .....	90
8.2.2	Experimental results .....	91
8.3	Statistical analysis .....	97
8.4	Corroboration of conjectures .....	99
8.5	Summary .....	103

## IV Final Remarks

<b>9</b>	<b>Conclusions .....</b>	<b>107</b>
----------	--------------------------	------------

## V Appendices

<b>A</b>	<b>Notation .....</b>	<b>111</b>
----------	-----------------------	------------

<b>B</b>	<b>Detecting outliers</b>	<b>115</b>
<b>C</b>	<b>Datasets</b>	<b>117</b>
<b>D</b>	<b>Web site model discovery using CALA</b>	<b>123</b>
	<b>Bibliography</b>	<b>131</b>

---

## *List of Figures*

---

3.1	Sketch of an enterprise information integration solution. . . . .	20
3.2	Query reformulation using mediators and wrappers. . . . .	22
3.3	Structure of a web application wrapper. . . . .	25
4.1	Sample search form. . . . .	28
4.2	Sample hub page. . . . .	29
4.3	Sample detail page. . . . .	30
4.4	Sample no-results page. . . . .	30
4.5	Sample error page. . . . .	31
4.6	Sample disambiguation page. . . . .	31
4.7	Navigation using crawlers. . . . .	37
4.8	Navigation using focused crawlers. . . . .	38
4.9	Navigation using usage mining techniques. . . . .	40
5.1	Web page classification using template detection techniques. . . . .	51
6.1	Our crawler. . . . .	58
6.2	Class diagram of our crawler. . . . .	59
6.3	Sequence diagram of our crawler. . . . .	60
6.4	Hub page in the running example. . . . .	61
6.5	Search form page in our running example. . . . .	65
6.6	Empty hub in the running example. . . . .	66
7.1	Our pattern builder. . . . .	72
7.2	Class diagram of our pattern builder. . . . .	73
7.3	Sequence diagram of our pattern builder. . . . .	73
7.4	Partial view of the initial prefix set in the running example. . . . .	77
7.5	Wildcarding example. . . . .	80
7.6	Prefixes in the running example after building patterns. . . . .	81

8.1	Performance of CALA with regard to the other techniques. ....	95
8.2	Boxplot of the learning times of CALA, TPM, and SVC. ....	96
8.3	Distribution of p-estimators in our experiments. ....	100
B.1	Sample distribution and its symmetric. ....	116
D.1	XPathTree. ....	125
D.2	XPathTree, after compression. ....	126
D.3	Detail page of class Author. ....	126
D.4	Relationships for MsAcademic. ....	129
D.5	Model for MsAcademic. ....	129

---

## *List of Tables*

---

2.1	Comparison of contents-based proposals. ....	11
2.2	Comparison of link-based proposals. ....	12
2.3	Comparison of visual-based proposals. ....	13
2.4	Comparison of URL-based proposals. ....	13
2.5	Comparison of structure-based proposals. ....	14
7.1	Patterns built for the running example. ....	76
7.2	P-estimators in our running example. ....	79
8.1	Results of the evaluation. ....	92
8.2	Number of classes/clusters created by each technique. ....	97
8.3	Results of our statistical ranking. ....	99
8.4	Number of pages from each web site in the experiment. ....	101
A.1	Some mathematical notation used throughout this dissertation. ...	112
C.1	Datasets description ....	118
D.1	Variability estimator values. ....	125





---

## *List of Programs*

---

6.1	Algorithm gatherHubset. ....	62
7.1	Algorithm buildPatterns. ....	75



---

# Acknowledgements

---

*I would maintain that thanks are the highest form of thought, and that  
gratitude is happiness doubled by wonder.*

*Short History of England, 1917*

*Gilbert K. Chesterton, Writer (1874-1936)*

**I** should begin these lines by thanking the two people that led me to becoming a PHD student: Dr. Carlos Rivero, who first introduced me to the world of academical research, and Dr. Rafael Corchuelo, who gave me the opportunity to become a researcher. I would like to thank him and Dr. David Ruiz, my advisors, who have helped, guided, advised and supported me throughout these years, and who have turned this dissertation into a reality. I would also like to thank my fellow PHD students at Geozoco, for giving me their help and support (and sometimes, a shoulder to cry on, as well), and the members of the TDG in Seville, Huelva and Brazil, who have always been eager to lend me a hand whenever I needed it. All of them have made this work a little less arduous and much more fun.

Thanks to Dr. Paolo Merialdo, who invited me to visit his group at the University Roma Tre, where I had the opportunity to meet interesting people (and thanks to Celine, for guiding me through Rome and making me feel at home).

And last but not least, I would like to thank my whole family, specially my parents, who have dedicated their life to educating me, and making me a better person; my sister Maca, for her unconditional support; my sister Ele, for her deep review of this dissertation, and my grandma, who is not quite sure of what computers are for, but who is proud and happy when I explain her what I do. And finally, I thank Carlos, the love of my life, who makes everything possible.



---

# Abstract

---

'Say on', said Don Quixote, 'and be brief in thy discourse, for there is  
no pleasure in one that is long'.

Don Quixote, 1605

Miguel de Cervantes Saavedra, soldier, novelist, poet, and playwright  
(1547-1616)

**I**ntegrating a web application into an automated business process requires to design wrappers that get user queries as input and map them onto the search forms that the application provides. Such wrappers build, amongst other components, on automatic navigators which are responsible for executing search forms that have been previously filled and navigating to the pages that provide the information required to answer the original user queries; this information is later extracted from those pages by means of an information extractor. A navigator relies on a web page classifier that allows to discern which pages are relevant and which are not.

In this dissertation, we address the problem of designing an unsupervised web page classifier that builds solely on the information provided by the URLs and does not require extensive crawling of the site being analysed. In the literature, there are many proposals to classify web pages. None of them fulfills the requirements for a web page classifier in a navigator context, namely: to avoid a previous extensive crawling, which is costly and unfeasible in some cases, to be unsupervised, which relieves the user from providing training information, or to use features from outside the page to be classified, which avoids having to download it previously.

Our contribution is CALA, a new automated proposal to generate URL-based web page classifiers. CALA builds a number of URL patterns that

represent the different classes of pages in a web site, and further pages can be classified by finding a match between their URLs and one of the patterns. Its salient features are that it fulfills all the previous requirements, it is computationally tractable, and it has been validated by a number of experiments using real-world top-visited web sites. Our validation suggests that CALA is very effective and efficient in practice.

---

## Resumen

---

*'Dilo', dijo Don Quijote, 'y se breve en tus razonamientos, que ninguno  
es gustoso si es largo'.*

*El Ingenioso Hidalgo don Quijote de la Mancha, 1605*

*Miguel de Cervantes Saavedra, soldado, novelista, poeta y dramaturgo  
(1547-1616)*

**L**a integración de aplicaciones web dentro de procesos automatizados de negocio requiere el diseño de wrappers que permitan ejecutar las consultas de un usuario usando los formularios de búsqueda que ofrece cada aplicación. Dichos wrappers se basan, entre otros componentes, en navegadores automáticos que se encargan de enviar los formularios de búsqueda rellenos previamente y navegar hacia las páginas que contienen la información necesaria para responder las consultas del usuario; posteriormente la información se extrae de dichas páginas mediante un extractor de información. Los navegadores hacen uso de clasificadores de páginas web que les permiten distinguir las páginas que son relevantes de las que no.

En esta tesis, tratamos el problema de cómo diseñar un clasificador de páginas web no supervisado que utilice únicamente la información proporcionada por la URL de las páginas y que no requiere un crawling extensivo del sitio analizado. En la bibliografía, existen muchas propuestas de clasificación de páginas web, pero presentan diversos inconvenientes, concretamente: requieren realizar un crawling previo exhaustivo del sitio web, que es costoso e incluso inviable en algunos casos, son supervisados, lo que exige al usuario que proporcione información de entrenamiento, o usan características de dentro de las páginas para clasificarlas, lo que obliga a descargarlas previamente.

Nuestra contribución es CALA, una nueva propuesta automática de generación de clasificadores de páginas web basados en la URL. CALA genera un conjunto de patrones de URL que representan las distintas clases de páginas ofrecidas por un sitio web, de forma que una página puede ser clasificada comparando su URL con los patrones y encontrando aquél con el que coincide. Las principales características de CALA son que no tiene ninguno de los inconvenientes anteriores, que es computacionalmente tratable y que ha sido validada mediante experimentos sobre algunos de los sitios web reales más visitados. Nuestra validación sugiere que CALA es muy eficiente y efectiva en la práctica



---

*Part I*

*Preface*

---



---

# Chapter 1

## Introduction

---

The White Rabbit put on his spectacles. 'Where shall I begin, please your Majesty?' he asked. 'Begin at the beginning,' the King said gravely, 'and go on till you come to the end: then stop.'

Alice in Wonderland, 1865.

Charles Lutwidge Dodgson (Lewis Carroll), writer and mathematician  
(1832-1898)

**T**his 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 the thesis that we prove in this dissertation; Section 1.3 summarises our main contribution; Section 1.4 introduces the collaborations we have conducted throughout the development of this dissertation; finally, we describe the structure of this dissertation in Section 1.5.

## 1.1 Research context

The computing infrastructure of a company that has been running for a few years typically includes several heterogeneous, loosely coupled applications. Most companies have realised that integrating them or the data they manage is very valuable to support business processes. In the beginning, the integration was usually ad-hoc; however, as the number of applications to integrate increased, this soon proved not sustainable, which motivated many researchers to work on principled approaches to engineer integration.

Enterprise information integration aims at providing a unified view over different sources of information, including data stores and applications. Since different data sources usually represent data using various formats and following different schemata, there is a need to integrate them. Enterprise information integration systems provide a unified schema on which users can pose their queries, which are answered using data from the different sources.

Nowadays, one of the main sources for enterprise information integration are web applications, which are software applications that are hosted on a web site [34]. According to some experts [30, 59, 94], web applications that provide a search form constitute the largest source of information in the Web. This information is valuable for companies and individuals alike.

Enterprise information integration systems rely on two main components, namely: mediators [58], which are responsible for reformulating user queries so that they can be executed on the different sources, and wrappers [10, 88], which endow data sources that cannot deal natively with queries with a programmatic interface that transforms them into the actions required to gather the appropriate information to answer them. Web application wrappers are composed of a number of modules that perform the tasks needed to have access to the data behind search forms: first, an enquirer, which is responsible for translating and issuing the queries in the search forms; then, a navigator, which navigates from the page returned from a search form to the pages that provide the information required to answer the user query. Usually, the former page is a hub, i.e., a web page that provides summaries and links to other pages [80]. Some of these links must be followed to reach the pages that contain the relevant information. Then, these pages are passed on to an information extractor, which extracts structured information from them; finally, a verifier is responsible for discarding the information that has been extracted erroneously.

Our focus in this dissertation are navigators. There have been many proposals to solve the problem of web navigation [4, 9, 23, 38, 70, 89, 100,

105, 117, 128]. Furthermore, crawling [3, 5, 25–27, 39, 91, 95, 101, 106, 107, 109] and usage mining techniques [84, 136] can also be applied to solve this problem using a web page classifier to distinguish between the pages that are relevant and irrelevant to the user query.

Web page classification has been extensively researched, and several techniques have been applied with successful experimental results. The features used to classify a web page can be either internal, i.e., extracted from inside web pages, like the page contents [14, 71, 87, 116], the organisation of visual blocks in the page [47, 85, 138], or the disposition of links amongst pages [18, 40, 52, 132, 139]; or external, i.e., computed from outside the page, like the URL [7, 12, 13, 22, 77, 120, 128]. Furthermore, template detection techniques can be applied to the problem of web classification by associating each template to a class of pages [5, 8, 21, 36, 41, 129]. URL-based classifiers are able to classify pages without downloading them, which is specially desirable in enterprise web information integration systems, in which the user is waiting for an answer and response time is an issue [126].

Since our research context is navigators for enterprise web information integration, we focus on URL-based web page classification.

## 1.2 Research rationale

In this section, we present the hypothesis that has motivated our research work in the context of non-supervised URL-based web page classification, and state our thesis, which we prove in the rest of the dissertation.

### 1.2.1 Hypothesis

Nowadays, there is an increasing interest of individuals, organisations and companies in offering their data through web applications that allow users to have access to them using a search form interface [49, 69], and whose pages are identified by URLs with a regular format [22]. According to some experts, these applications are currently the largest source of web information [30, 59, 94]. Therefore, integrating their data is very valuable to support automated business processes. However, these applications are designed to be used by humans, which makes it necessary to design wrappers to integrate them.

According to the previous argumentation, we conclude that our hypothesis is the following:

*Most sites that publicly offer interesting information in the Web, do so by means of keyword-based search forms and use regular URLs to identify their pages. These sites are the largest source of web information, therefore there is a need to design wrappers to integrate the information provided by those sites into automated business processes.*

### 1.2.2 Thesis

There are a number of techniques that can be applied to classify web pages and discern those that contain relevant information [5, 7, 8, 12–14, 18, 21, 22, 36, 40, 41, 47, 52, 71, 77, 85, 87, 116, 120, 128, 129, 132, 138, 139]. Unfortunately, they do not fulfill a number of requirements for their use in an enterprise web information integration context, namely: to avoid an extensive crawling of the site, which is costly and in some cases, unfeasible; to be unsupervised, which relieves the user from providing training information; or to use external features to classify a web page, which avoids having to download the pages beforehand. Enterprise web information integration is an on-line process, therefore bandwidth and efficiency are important issues, and minimising the amount of web traffic is mandatory. Furthermore, URLs are good external features for classification, since they are small and every web page possess one [77]. According to the previous argumentation, we conclude that our thesis is the following:

*It is possible to devise a technique that building on a relatively small number of hubs can learn a classifier in a totally unsupervised manner; the resulting classifier builds solely on features of the URLs of the pages to be classified, but can achieve high precision and recall.*

## 1.3 Summary of contributions

To prove our thesis, we have devised CALA, an unsupervised proposal to learn a web page classifier that relies on features extracted from the URLs of a small set of hubs. It is suitable for enterprise web information integration since it does not require a page to be downloaded so that it can be classified.

CALA relies on two modules: a crawler, which gathers a set of hubs from a web site, and a pattern builder, which uses the former set to build a set of patterns that represent the URLs in that site. Then, the user must assign a semantic class to each pattern. Note that the size of the set of patterns is

significantly smaller than the set of pages in a site, so the cost of annotating them is negligible. Finally, the classifier uses the set of annotated patterns to classify new web pages from the same site by finding which pattern, if any, matches its URL. If no match is possible, this means that we have found a page whose URL deviates largely from the URLs from which we learned the patterns, which is likely to be due to a reorganisation of the web site. In such cases, it is necessary to learn the patterns again.

We have performed a number of experiments to validate our contribution. To that purpose, we have compiled a number of datasets composed of annotated web pages, which we have made available.

We have a number of papers describing our contribution in the following conferences: WWW [65], ER [66], IWSSA [64], PAAMS [42], CAEPIA [63], ICAI [67], WISM [68], and ZOCO [62]. Furthermore, we submitted an extended abstract of our contributions to the Knowledge Based Systems journal.

## 1.4 Collaborations

A research visit was paid from June 1 until June 30, 2011 to the research group of Dr. Paolo Atzeni from the Roma Tre University (Italy). The visit was supervised by Dr. Paolo Merialdo, who is an expert in extracting and integrating web data [17, 20, 21, 36], and Dr. Lorenzo Blanco, who is an expert in web page classification [20–22]. We analysed our proposal, and we paid special attention to its evaluation, and other proposals in the area of URL-based web page classification. Furthermore, we studied some problems detected by the members of their group in that area, which were considered during the research work presented in this dissertation.

## 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 current proposals to classify web pages do not fulfil a number of requirements to be used in the context of navigators for enterprise web information integration.

**Part II: Background Information.** It provides information about enterprise web information integration, automated web navigation, and web page

classification. In Chapter 3, we introduce the idea of enterprise information integration, with an emphasis on data sources that are actually web applications that provide a search form. In Chapter 4, we present different proposals to deal with the problem of automated web navigation. In Chapter 5, we describe the problem of web page classification and present several proposals to classify web pages using different types of features.

**Part III: Our Proposal.** It reports on the core contribution we made in this dissertation. In Chapter 6, we present our crawler to automatically gather a set of hub pages without supervision from a web site that provides a search form. In Chapter 7, we describe our pattern builder, which uses the former set of pages to build a set of patterns that can be used for web page classification. In Chapter 8, we present the experimental evaluation of our proposal.

**Part IV: Final Remarks.** It concludes this dissertation and highlights a few future research directions in Chapter 9.

**Appendices.** They provide extended details about some aspects of this dissertation. In Appendix A we describe the datatypes that support our proposal, and the mathematical notation that is used throughout the dissertation. In Appendix B we provide an extended description of the outlier detection technique on which our proposal is based. In Appendix C.1 we describe in detail the datasets used in our experiments. Finally, in Appendix D we provide some insight into web site model discovery, which is one of our future research directions.



---

## Chapter 2

# Motivation

---

*It is proved by surveys that happiness does not come from love, wealth,  
or power but the pursuit of attainable goals.*

*Bridget Jones's Diary, 1996*

*Helen Fielding, writer (1958)*

**O**ur goal in this chapter is to present the requirements for web page classifiers in enterprise web information integration contexts, to detail to which extent current proposals fulfill these requirements, and to motivate the need for a new proposal. The chapter is organised as follows: in Section 2.1, we introduce it; Section 2.2 presents the requirements for web page classifiers in depth; in Section 2.3, we discuss the current proposals and we conclude that none of them fulfills every requirement we have identified; Section 2.4 introduces our contributions and compares them to current proposals; finally, we summarise the chapter in Section 2.5.

## 2.1 Introduction

Nowadays, there is an increasing interest in integrating the information provided by web applications that offer their data through search forms, which are the largest source of web information according to some experts [30, 59, 94]. To integrate this information, wrappers are needed to provide a programmatic interface for executing queries on these applications. The execution of a query implies filling and submitting the form, which returns a web page as a response. The response page may or not contain the information that is relevant to the query. In the latter case, a navigator has to go through the web site following links and looking for pages that contain the relevant information. Therefore, a web page classifier is needed to make the distinction between relevant and irrelevant pages, so that the navigator can avoid downloading the latter.

In the literature, there are different proposals to address the problem of web page classification [5, 7, 8, 12–14, 18, 21, 22, 36, 40, 41, 47, 52, 71, 77, 85, 87, 116, 120, 128, 129, 132, 138, 139]. Unfortunately, none of these proposals fulfill a number of requirements for their use in enterprise web information integration contexts. Consequently, it is still necessary to research on web page classification in the context of integration, which is our purpose in this dissertation.

## 2.2 Requirements

There are many proposals to classify web pages, and they rely on a variety of features. However, there are some requirements that must be considered when using web page classifiers in the context of enterprise web integration information systems. Enterprise web integration information systems are online, i.e., there is a user who must wait for the answer to his or her queries. Therefore, response time is an issue, and navigators should consequently minimise the amount of time that they spend downloading irrelevant pages [126].

In this section, we present the requirements that a web page classifier must fulfill, with an emphasis on the enterprise web information integration context. These requirements are the following:

**(R1) Lightweight crawling:** Some proposals require to perform an exhaustive crawling prior to learning a classifier. This is not usually feasible, chiefly if we take into account that web sites change frequently [82] and

---

Proposals	R1	R2	R3
Beil and others [14]	Yes	Yes	No
Hotho and others [71]	Yes	Yes	No
Hotho and others [72]	Yes	Yes	No
Kwon and Lee [87]	No	No	No
Selamat and Omatu [116]	Yes	Yes	No

---

R1 = Lightweight crawling; R2 = Unsupervised; R3 = Classify without downloading.

---

**Table 2.1:** Comparison of contents-based proposals.

this requires the classifier to be learnt again. Therefore, any crawling that has to be performed on the site to learn the classifier must be lightweight.

**(R2) Unsupervised:** Some proposals require the user to provide some training information to learn a classifier. The user may be required to provide a training set, which is a collection of datasets in which each web pages is assigned a semantic class (this process is commonly referred to as handcrafting annotations for the training set). He or she may be also required to provide a dictionary of words that are specific to an application domain in a given language. In most cases, the task of providing this additional information is cumbersome, prone to errors, and very effort-consuming. It has been noted that supervised methods are not likely to scale well in the Web [95]. Therefore, the classifier must be learn without supervision.

**(R3) Classify without downloading:** In some cases, it is necessary to download a page before classifying it. This is problematic insofar pages that are irrelevant need to be downloaded and discarded, which is inefficient. Therefore, the classifier must use features that allow classifying a page without downloading it previously.

## 2.3 Analysis of current solutions

Table 2.1 summarises the contents-based web page classification proposals in the literature. These proposals represent each page as as a vector whose components are the frequencies of each term in the page; the classifier assigns

---

Proposals	R1	R2	R3
Bhagat and others [18]	No	No	Yes
De Campos and others [40]	No	No	Yes
Getoor and others [52]	No	No	No
Zhu and others [139]	No	No	No
Xie and others [132]	No	No	No

---

R1 = Lightweight crawling; R2 = Unsupervised; R3 = Classify without downloading.

---

**Table 2.2:** Comparison of link-based proposals.

to a new page the class that corresponds to the nearest vector. Regarding R1, Kwon and Lee [87] require to preprocess a web site by assigning each of its pages a weight based on its number of incoming and outgoing links, and use only the term vectors of the pages with a higher weight as a training set. Therefore, they must perform an extensive crawling of the site, whereas the other proposals use a reduced training set. Regarding R2, Kwon and Lee [87] require a training set of annotated term vectors, which makes it supervised, whereas the other proposals are unsupervised. Finally, regarding R3, Beil and others [14], Hotho and others [71, 72], Kwon and Lee [87] and Selamat and Omatu [116] need to analyse the terms in the page that is being classified, which require them to download it before it can be classified.

Table 2.2 summarises the link-based web page classification proposals in the literature. These proposals represent each web site as a graph in which nodes are web pages, and edges are links from one web page to another. They classify each page by assigning it a class that is computed from the classes of its neighbours. Regarding R1, Bhagat and others [18], Getoor and others [52], Xie and others [132], Zhu and others [139] and de Campos and others [40] have to analyse every page in a site to build the graph, which requires a previous extensive crawling. Regarding R2, the former proposals are based on classifying a page using the information of some example pages that are already classified, which makes them supervised. Finally, regarding R3, Getoor and others [52], Zhu and others [139], and Xie and others [132] use a combination of link-based and contents-based features, which means that they have to download a page before classifying it.

Table 2.3 summarises the visual-based web page classification proposals in the literature. These proposals are based on features that can only be com-

---

Proposals	R1	R2	R3
Zhu and others [138]	Yes	No	No
Fersini and others [47]	Yes	No	No
Kovacevic and others [85]	Yes	No	No

R1 = Lightweight crawling; R2 = Unsupervised; R3 = Classify without downloading.

---

**Table 2.3:** Comparison of visual-based proposals.

---

Proposals	R1	R2	R3
Shih and Karger [120]	Yes	No	Yes
Kan and Thi [77]	No	No	Yes
Baykan and others [12]	No	No	Yes
Baykan and others [11]	No	No	Yes
Vidal and others [128]	No	No	Yes
Bar-Yossef and others [7]	No	Yes	Yes
Koppula and others [83]	No	Yes	Yes
Blanco and others [22]	No	Yes	No

R1 = Lightweight crawling; R2 = Unsupervised; R3 = Classify without downloading.

---

**Table 2.4:** Comparison of URL-based proposals.

puted when the web page is rendered by a browser, e.g., the position of an image on the screen, its bounding box, or the distance amongst the different elements in the page. Regarding R1, none of the proposals need to perform an extensive crawling, since they use a small training set of pages. Regarding R2, Fersini and others [47], Zhu and others [138] and Kovacevic and others [85] are based on a training set of annotated pages and they classify a page by assigning it to the class of the most similar page, which makes them supervised. Finally, regarding R3, the former proposals require each page to be rendered to compute some features from it, so it has to be downloaded previously.

Table 2.4 summarises the URL-based web page classification proposals in the literature. These proposals classify a page using features that are computed from its URL. Regarding R1, Bar-Yossef and others [7], Baykan

---

Proposals	R1	R2	R3
Crescenzi and others [36]	Yes	No	No
Bar-Yossef and Rajagopalan [8]	Yes	Yes	No
Arasu and Garcia-Molina [5]	Yes	Yes	No
Blanco and others [21]	Yes	No	No
De Castro Reis and others [41]	Yes	Yes	No
Vieira and others [129]	Yes	Yes	No

---

R1 = Lightweight crawling; R2 = Unsupervised; R3 = Classify without downloading.

---

**Table 2.5:** Comparison of structure-based proposals.

and others [11, 12], Kan and Thi [77], Koppula and others [83] and Blanco and others [22] need a large collection of URLs to learn a classification model for a site so that it can achieve good results, so they need to perform an extensive crawling. Vidal and others [128] need to map the site previously, and then select some of the paths in that map to learn the classifier, which requires an extensive crawling as well. Regarding R2, Vidal and others [128] take a sample annotated web page as input, and build a classification model to discern pages that either are similar to the sample page, or lead to pages that are similar to it by following its links. Baykan and others [12], Kan and Thi [77] and Baykan and others [11] are similar to contents-based proposals, but they represent a web page using the terms in its URL, instead of its contents. They require a dictionary of representative terms from each class that must be provided by the user. Shih and Karger [120] follows a different approach, in which URL tokens are inserted in a tree, and it takes a training set of annotated URLs as input. Bar-Yossef and others [7] and Koppula and others [83] require a training set of both positive and negative samples. Therefore, the previous proposals are supervised. Finally, regarding R3, Blanco and others [22] uses some contents-based features together with the URL-based features, which requires a page to be downloaded before classifying it.

Table 2.5 summarises the structure-based web page classification proposals in the literature. These proposals detect the template that is common to every page in a class, and classify other pages by assigning them the class that is associated to the most similar template. Regarding R1, none of the proposals require an extensive crawling to learn a classifier. Regarding R2, Crescenzi and others [36] and Blanco and others [21] require a train-

ing set of annotated pages of the same class to detect each template, which makes them supervised. Notwithstanding, they require the smallest training sets we have found in the literature. Finally, regarding R3, Arasu and Garcia-Molina [5], Bar-Yossef and Rajagopalan [8], Blanco and others [21], Crescenzi and others [36], Vieira and others [129] and de Castro Reis and others [41] classify pages by comparing their structure to those of the detected templates, so they require to download a page before classifying it.

## 2.4 Our proposal

In this dissertation, we present a proposal called CALA. It helps learn a web page classifier using a technique that fulfills every requirement we have identified previously, cf. Section 2.2: it performs a lightweight crawling, it is not supervised, and it does not require a page to be downloaded so that it can be classified. It does not require an extensive crawling of the site to build the classification model (R1), but only a small set of hub pages (e.g., in our experiments 100 hub pages were enough to achieve a good effectiveness). Furthermore, it is not supervised (R2), since we base on a set of non-annotated pages that are automatically extracted from the web site to be analysed. To fill in the forms, we use keywords that are extracted from the same site automatically, hence no dictionary or user input is needed. Furthermore, it is based exclusively on URL features, so pages do not have to be downloaded to be classified (R3).

## 2.5 Summary

In this chapter, we have motivated the reason for this piece of research work. We have analysed the requirements for web page classifiers in enterprise web information integration contexts and the current proposals in the literature to classify web pages, and we have concluded that none of these proposals fulfills these requirements at a time. This motivated us to work on a proposal to fulfill them and advance the state of the art a step forward.





---

*Part II*

*Background Information*

---



---

## Chapter 3

# Enterprise web information integration

---

Your initial problem will be the breaking tensions arising from the divergent assembly of minutiae/data on specialized subjects. Be warned. Without mental overlay integration, you can immersed in the Babel Problem, which is the label we give to the omnipresent dangers of achieving wrong combinations from accurate information.

Children of Dune, 1976.

Franklin Patrick Herbert, Jr, writer (1920-1986)

**I**n this chapter, we introduce the idea of enterprise information integration, with an emphasis on data sources that are actually web applications that provide a search form. It is organised as follows: in Section 3.1, we introduce it; Section 3.2 provides an overview on mediators, which are responsible for reformulating queries in enterprise integration information systems; in Section 3.3, we describe web wrappers, which provide a programmatic interface for web sites that lack one; finally, we summarise the chapter in Section 3.4.

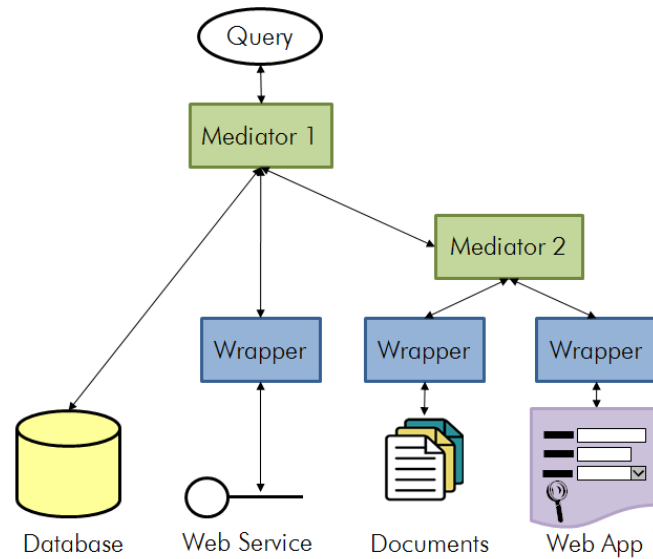


Figure 3.1: Sketch of an enterprise information integration solution.

### 3.1 Introduction

Enterprise information integration has been quite an active research field for years because the costs involved in integration are usually an important component of almost every IT project [131]. The earliest approaches originated in the field of database management systems, and the goal was to integrate several databases so that they looked as if they were a unique database by means of so-called mediators [58] and peer data management systems [44], which in turn relied on so-called mappings amongst the integrated databases [97].

The idea behind enterprise information integration is to build a global schema (aka mediated schema) that integrates the schemata used by the sources of data that have been selected for integration. The global schema allows querying the different independent sources using a high-level query language as if they were a single database [58]. Since the integration is performed on heterogeneous data sources, each data source has its own local schema, with its data types and restrictions. The global schema is a generalisation of the different local schemata, therefore it may include restrictions that do not exist in some of the data sources. Therefore, sometimes it is necessary to filter the data returned by a data source a posteriori to apply those restrictions.

The key in enterprise information integration is the adjective “enterprise”, which conveys the idea that the information sources are actually applications that are running in the software eco-system of a company. This has implications insofar such applications cannot be expected to provide the appropriate programmatic interfaces to facilitate integrating the data they manage but a user-friendly search form. Web applications are particularly important nowadays, since they provide the majority of information that is available in the Web [59]; we refer to the problems that involve such applications as enterprise web information integration.

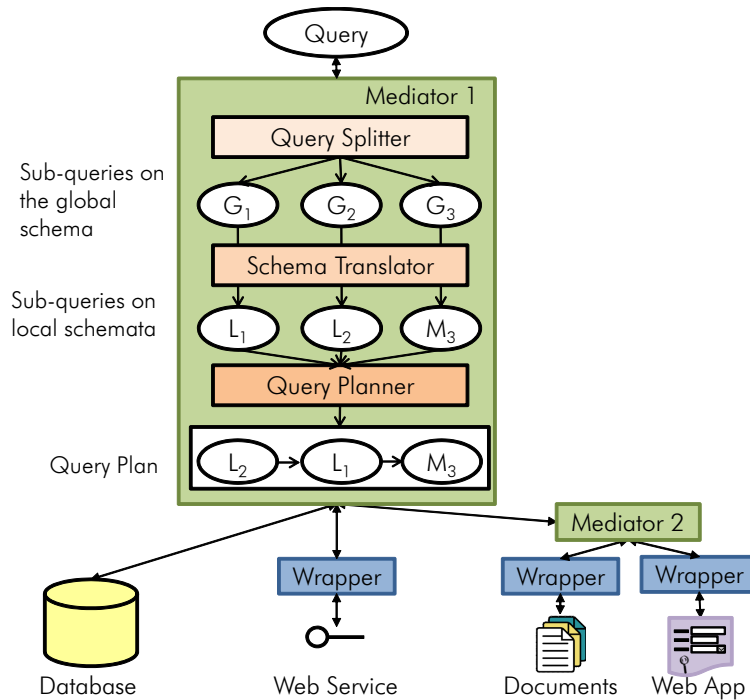
Some authors refer to the portion of the Web that is delivered through search forms as the Deep Web or even the Hidden Web [94]. We, however, prefer not to use these terms since the pages that are delivered through a search form need not necessarily be deep or hidden; for instance, Google reports 45+ million pages from a site like Amazon.com. In this dissertation, we focus on enterprise information integration techniques that integrate data from web applications that offer their data through a search form without providing a programmatic interface.

An example of an enterprise information integration solution is depicted in Figure 3.1. The goal is to integrate data from four heterogeneous sources, namely: a relational database, a web service that offers a programming interface, a corpus of documents, and a web application that offers its data only through search forms. This solution provides a global schema that receives user queries which are answered on-line using data from the different data sources. Therefore, queries must be reformulated in terms of the different local schemata by means of one or more mediators [57]. Then, they must be executed by means of wrappers, which provide a query answering interface for data sources that do not provide one. Once the wrappers return the results, these must be composed into the global schema. So far, we have just gathered the information required to answer the initial user query from the sources; to answer it effectively, it must be executed on this information.

In the following sections we provide an insight into the two key components of every enterprise information integration solution: mediators and wrappers.

## 3.2 Mediators

The goal of a mediator is to reformulate a query posed over a global schema into a number of queries that can be executed on the different



**Figure 3.2:** Query reformulation using mediators and wrappers.

data sources to provide a minimal set of information on which the original query can be executed. A mediator relies on three components: a query splitter, a schema translator, and a query planner, cf. Figure 3.2.

The query splitter needs to identify the data sources that do not provide any relevant information and then splits the original query into a number of sub-queries, each of which involves a unique data source. Then, the schema translator must transform each sub-query so that it can be sent to the corresponding data source, i.e., it must transform references to the entities in the global schema to entities in the local schemata. The query planner is then responsible for producing a plan that allows to orchestrate the sub-queries (possibly in parallel) so that they can be executed as efficiently as possible [74]. Query planners aim to compute a maximally-contained query plan, i.e., a plan that retrieves the data that best answer the query using only the available data sources [2]. Once the execution plan is defined, the sub-queries are sent to the sources. In cases in which they cannot deal natively with a query (e.g., the web application with the search form), a wrapper must be designed.

As an example, in Figure 3.2, the query splitter in Mediator 1 splits the

user query into three sub-queries  $G_1$ ,  $G_2$  and  $G_3$ , which are posed over the global schema. Then, they are transformed into sub-queries  $L_1$ ,  $L_2$ , and  $M_3$ , which are posed on the local schemata of the database and the web service, and the mediated schema provided by Mediator 2, respectively. Note that Mediator 2 is seen by Mediator 1 as a data source with a schema, just like the database or the web service. Then, the query planner in Mediator 1 generates an efficient plan, which consists of executing first  $L_2$ , then  $L_1$ , and then  $M_3$ . Finally, sub-queries  $L_1$  and  $L_2$  are sent to the database and the web service wrapper to be executed, whereas sub-query  $M_3$  is sent to Mediator 2 to be reformulated. This process is repeated in Mediator 2 with sub-query  $M_3$ , which is reformulated into two sub-queries  $L_3$  and  $L_4$  that are posed on the local schemata of the corpus of documents and the web application.

The literature provides many proposals to implement mediators in the context of relational and nested-relational data sources; unfortunately, the work on semantic data is in its earliest stages [113].

The proposals to implement query splitters and schema translators rely on so-called mappings, which are artefacts that identify which entities in a schema correspond to others in another schema [110]. These proposals can be classified into three groups [124], namely:

**Global as View (GaV):** The global schema is defined as a query over the local schemata [31]. The main advantage of GaV is that reformulation of queries is simple. Its main drawback is that adding or removing data sources is complex since it requires to redefine the global schema. Therefore, it is suitable for systems in which data sources do not change frequently.

**Local as View (LaV):** The local schemata are defined as a query over the global schema [90]. The main advantage of LaV is that adding or removing data sources is simple, since the global schema does not change and it only requires to create a new definition for each new source in terms of the global schema. The main drawback of LaV is the complexity of query reformulation, which can be unfeasible in some cases.

**Global and Local as View (GLaV):** Hybrid approaches deal with the drawbacks of the previous ones [46, 133]. In these approaches, global and local schemata are independent from each other and a number of mappings are created to translate from a view of the local schemata to the global schema. Therefore, the addition or removal of new sources does not require redefining the global schema, as in GaV, and query reformulation is not as complex as in LaV. Schema matching techniques

are usually applied to deal with the problem of creating the former mappings automatically.

Regarding query planners, they follow the same approach as relational databases query optimisers, which use information about the database contents (such as table sizes or physical paths to data) to improve query performance. However, integration solutions have access to data from heterogeneous sources, not only databases, and therefore this information is not available in most cases. Therefore, other techniques have been proposed to optimise queries in enterprise information integration contexts. Adaptive query processing [54] refers to techniques that discover characteristics about the data during query execution and exploit this knowledge to improve performance [74]. Some of these techniques work between successive executions of queries [24, 32]; these methods cause the initial queries to show poor performance, whereas subsequent executions of similar queries are progressively more efficient. Other approaches allow queries to be optimised during execution [74].

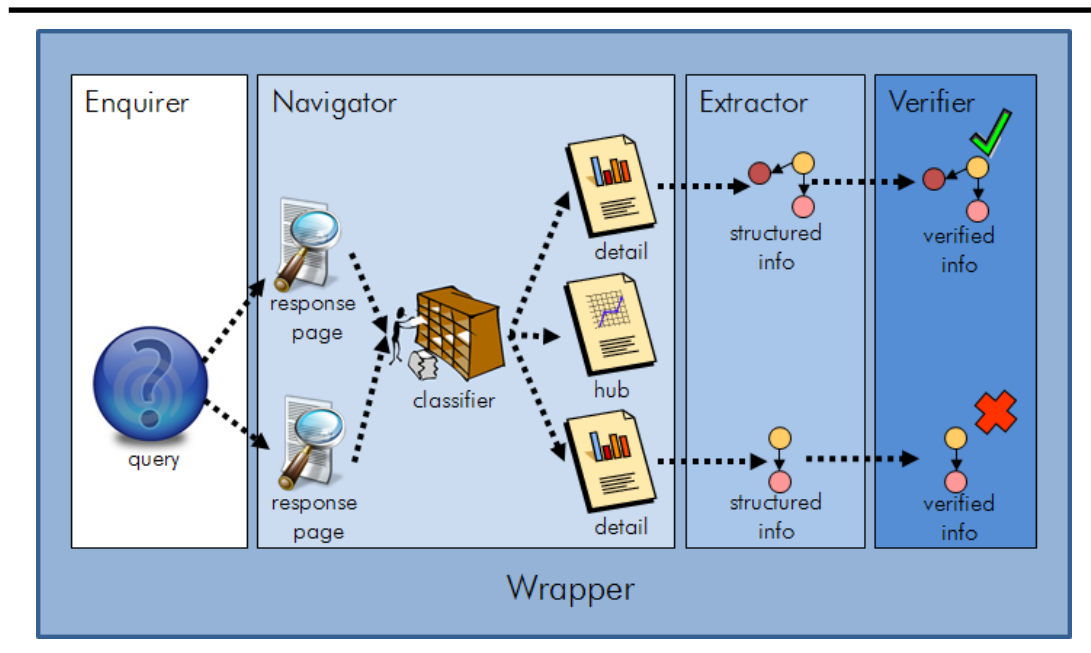
### 3.3 Wrappers

A wrapper helps instruct an application to answer a query in cases in which it does not support this functionality natively or does not deliver it using the appropriate technology. The state of the art provides an array of technologies to implement wrappers to interact with data sources that provide a programmatic interface [48]; the problem becomes more complex when a web application that does not provide a programmatic interface comes on the scene [23].

A typical wrapper for a web application consists of the components presented in Figure 3.3, namely [42]:

**Enquirer:** An enquirer is a module that takes a query as input and maps it onto the appropriate search forms provided by a web application. What a query is may range from a set of field names and values to SQL-like queries. In this dissertation, we are only interested in the latter, since current technologies support the former sufficiently [1]. Current research efforts include a few intelligent techniques to analyse search forms and to extract their search capabilities, i.e., the goal is to have a model that others can use to map high-level queries onto it [60, 137]. Unfortunately, the literature does not provide many other results regarding this topic.





**Figure 3.3:** Structure of a web application wrapper.

**Navigator:** A navigator cares of executing the filled forms provided by an enquirer and navigating through the results to fetch relevant pages. Note that this process may lead to a so-called hub, which is a web page that provides short descriptions of the information in other pages and links to them, to a detail page with information, a no-results page, an error page, or a disambiguation page. Beyond navigators that rely on navigational scripts [4, 9, 23, 38, 70, 89, 100, 105, 117, 128], the literature on crawling [5, 95] provides several techniques that can be applied to solve this problem. Focused crawling improves on traditional crawling in that it tries to avoid crawling pages that do not lead to data pages about a given topic of interest [3, 25–27, 39, 91, 101, 106, 107, 109]. Furthermore, usage mining techniques can also be applied to implement navigators [84, 136].

**Information extractor:** An information extractor is a general algorithm that can be configured by means of rules so that it extracts the information of interest from a web page and returns it according to a structured model. Rules range from regular expressions to context-free grammars or first-order clauses, but they all rely on mark-up tags or natural language properties to find which text corresponds to the data of inter-

est. Beyond hand-crafting information extraction rules, the literature provides a variety of proposals that can be used to learn them automatically, both in cases in which the data of interest is buried into text that is written in natural language [127] and cases in which it is buried into tables, lists and other such layouts [29]. Recently, the problem of identifying the regions of a web page in which the information of interest resides is also attracting an increasing attention [121].

**Information verifier:** An information verifier is an algorithm that analyses the data returned by an information extractor and attempts to find data that deviates largely from data that is known to be correct. They are necessary insofar the previous modules rely on intelligent techniques that may fail if the structure of a site or a web page changes, i.e., if they are confronted with cases that were not seen previously. Information verifiers build on feature-based verification models. The literature provides two probabilistic techniques [86, 97] and a goodness-of-fit technique [81] to build them.

### 3.4 Summary

In this chapter, we have given an overview of the main ideas in the field of enterprise information integration. We have presented an overall idea of what it consists in, and then an insight into the two key components to implement an enterprise information integration solution, namely: mediators and wrappers. Fortunately, the literature provides a number of proposals to implement both mediators and wrappers. Unfortunately, more work seems to be required regarding mediators that have to deal with semantic data and wrappers that have to deal with web applications.

---

# Chapter 4

## Automated web navigation

---

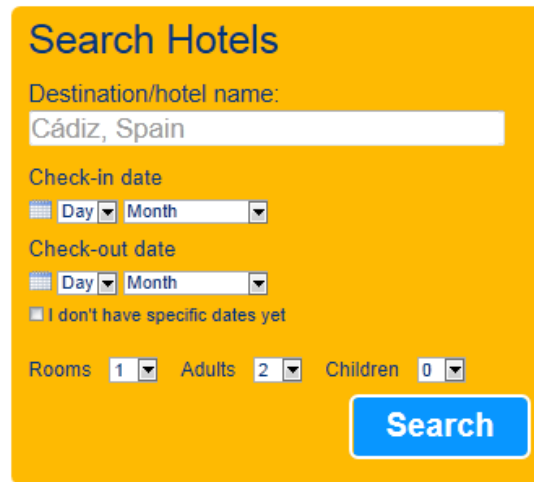
*Read the directions and directly you will be directed in the right  
direction.*

*Alice in Wonderland, 1865.*

*Charles Lutwidge Dodgson (Lewis Carroll), writer and mathematician  
(1832-1898)*

**T**he goal of automated web navigation is to sift through the pages of a web site so as to implement repetitive tasks without user intervention. In this chapter, we present several closely-related concepts and some proposals that address this problem. It is organised as follows: in Section 4.1, we provide an overall picture of web navigation; Section 4.2 describes the different types of web pages with which a web navigator must deal; in Section 4.3, we present the existing proposals to automate web navigation; Section 4.4 reports on some crawling proposals that can be applied to the problem of web navigation; finally, we summarise the chapter in Section 4.6.

---



---

Figure 4.1: Sample search form.

## 4.1 Introduction

Automated web navigation refers to the problem of visiting web pages and interacting with them automatically. This involves following links and executing scripts to, for instance, test web applications, perform maintenance tasks, create shortcuts to pages where repetitive tasks such as filling in and submitting forms are required, or find the pages that are related to a given user query. Since the focus of this dissertation is on enterprise web information integration, we put an emphasis on the last application.

In enterprise web information integration systems, we deal with web applications that are based on search forms that must be filled in with the values provided by a user query. Figure 4.1 reproduces the search form of Booking.com, a hotel booking site that we use as a running example. The goal of automated web navigation in enterprise web information integration systems is to retrieve as many pages that are relevant to the query as possible. An automatic navigator must be able to follow the links on the page returned by a search form in order to find pages that are relevant to the user query.

Automated web navigation has been paid much attention in the literature. Usually, navigators are based on scripts, which can be either handcrafted or learned automatically. These scripts define the sequence of steps the navigator must perform to reach the relevant pages. Furthermore, other proposals from the fields of blind and focused crawling, and web page recommendation can be used to implement automated web navigators.

**Booking.com**

23 results in or near Cádiz, Spain

Sort by: Recommended Stars Distance Price Review score

---

**Hotel Patagonia Sur** ★★★ **Very good, 8.1**  
 Cádiz • [Show map](#) Score from 459 reviews  
 Patagonia Sur's colourful modern rooms include free Wi-Fi and flat-screen cable TV. It is situated 200 metres from Cádiz Cathedral and the Tavira Tower, in the historic Old Town. [More](#)  
 Latest booking: 5 hours ago [Book now](#)

<a href="#">Double Room</a>	<small>Last one!</small> <small>FREE cancellation</small>	<small>Last chance!</small> <small>Only 1 left</small>	€130 € <b>115</b>
<a href="#">Double Room</a>	<small>Last one!</small> <small>FREE cancellation</small>	<small>Last chance!</small> <small>Only 1 left</small>	€130 € <b>100</b>

---

**Senator Cádiz Spa Hotel** ★★★★★ **Good, 7.5**  
 Cádiz • [Show map](#) Score from 684 reviews  
 Senator Cádiz Spa Hotel is located in Cádiz's old town, close to shopping centres. The town's port, railway and bus stations are within 5 minutes' walk of the hotel. *There are 2 people looking at this hotel.* [More](#)  
 Latest booking: 50 minutes ago [Book now](#)

<a href="#">Double Room</a>		Available	€155.00 € <b>113.52</b>
<a href="#">Family Room (2 Adults + 2 Children)</a>		Available	€170.82 € <b>131.12</b>
<a href="#">Double Room</a>	<small>Breakfast included</small>	Available	€170.82 € <b>131.12</b>

3 more room types

Figure 4.2: Sample hub page.

## 4.2 Taxonomy of web pages

The response to a user query depends on the ability of a web application to answer it, that is, on the availability of relevant information in its back-end databases. We can distinguish between different types of responses, being the most common the following:

**Hub:** Hubs are web pages that provide summaries and links to other pages [80]. Hub pages usually provide more URLs than other pages in a web site since their goal is to offer the users as many results that are related to their queries as possible. Figure 4.2 depicts a sample hub page returned when searching for 'Cádiz, Spain'.

**Detail:** Detail pages provide complete details about an item of information offered by a web site. Figure 4.3 depicts a detail page that was returned in response for query 'Senator Spa Cádiz'. It shows the detailed information about a hotel in Cádiz, including its address, photos, interesting features, and the description and prices of the available rooms.

**Booking.com**

**Senator Cádiz Spa Hotel** ★★★★★

Rubio y Díaz, 1, 11004 Cádiz ([Show map](#))

[Book now](#)

[See all reviews](#)

**Good, 7.5**  
Score from 684 reviews

*"Great location with helpful staff, we hope to return someday."*

Henry, Dublin   
February 5, 2012

Senator Cádiz Spa Hotel is located in Cádiz's old town, close to shopping centres. The town's port, railway and bus stations are within 5 minutes' walk of the hotel.

Aquaplaya is the Senator Cádiz's superb and exclusive urban spa (fees apply) with a variety of treatments to enjoy. It is open from Tuesday until Sunday. During the summer months, you can make the most of the outdoor pool.

*Hotel Rooms: 91, Hotel Chain: Playa Senator.*

Figure 4.3: Sample detail page.

**Search Hotels**

**Sorry, we don't recognize that name.**

Destination/hotel name:

Check-in date

Check-out date

I don't have specific dates yet

Rooms  Adults  Children

[Search](#)

Figure 4.4: Sample no-results page.

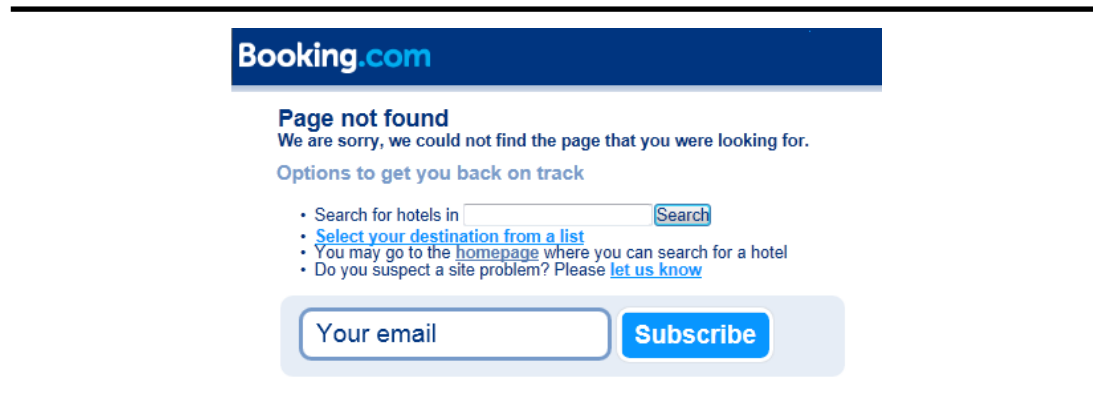


Figure 4.5: Sample error page.

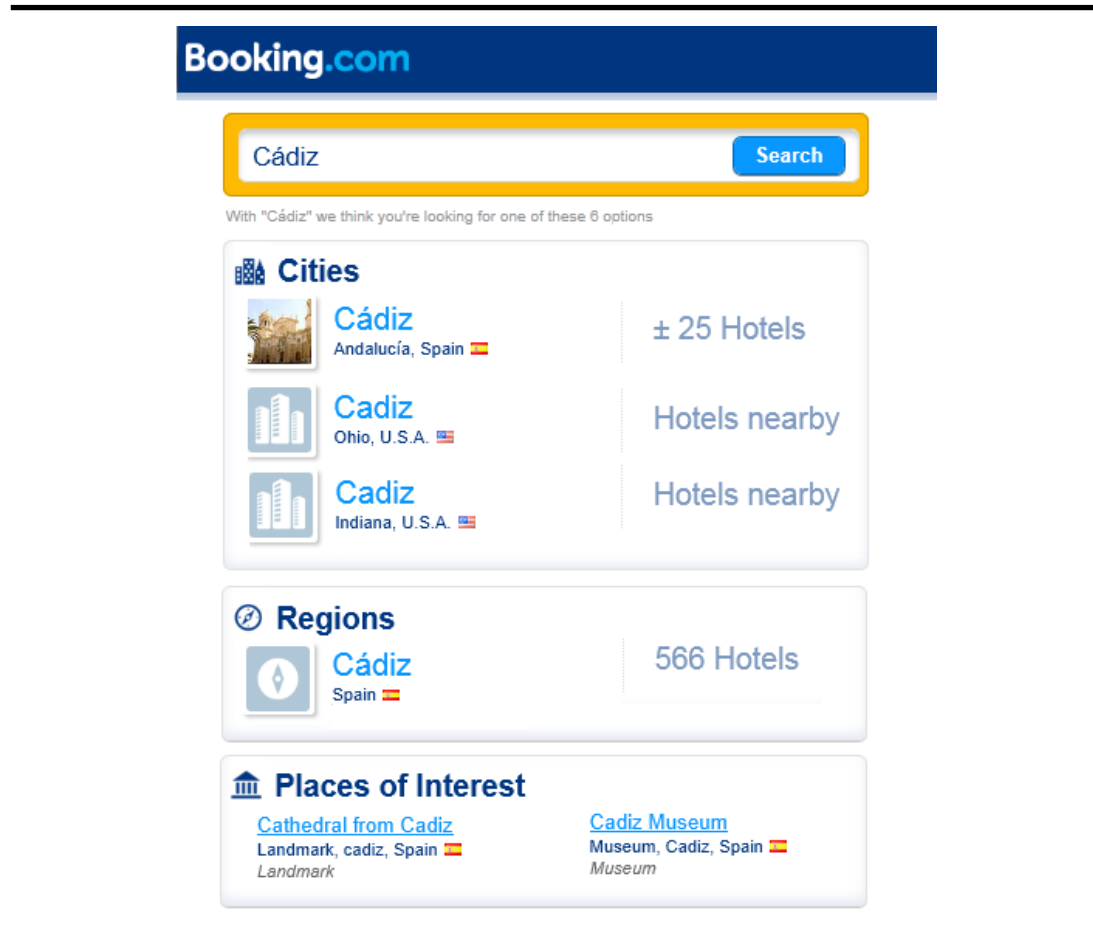


Figure 4.6: Sample disambiguation page.

**No result:** When a web application does not have any information that is relevant to a query, it usually returns a page with an informative message, so that the user may change his or her query. Figure 4.4 depicts a sample no-result page that results from query "...".

**Error:** When an error occurs, web applications usually return a page with an informative message. Figure 4.5 depicts a sample error page.

**Disambiguation:** In some cases, a user query can be ambiguous, e.g., it may include polysemic or misspelled words. In such cases, some web applications return a page that presents some disambiguations so that the user can refine his or her query. Figure 4.6 depicts a sample disambiguation page that was returned in response for query 'Cádiz'. It requires disambiguation since there are three cities with that name located in Spain, Ohio (USA), and Indiana (USA), as well as a province with the same name in Spain; the page also reports on two places of interest that refer to that name.

The relevancy of a page depends exclusively on the user query, i.e., in some cases a hub page might be relevant if it provides enough information to answer the query, whereas in other cases it is necessary to reach the detail pages. No-result, error, or disambiguation pages are generally of little interest, but must be dealt with appropriately by automatic web navigators.

## 4.3 Scripting proposals

Scripting proposals build on navigation patterns, aka navigation maps or navigation sequences, which help navigate from a search form to relevant pages. Navigation patterns define a sequence of pages that must be visited to reach relevant pages, as well as the interactions needed in each page (e.g., the forms to be filled, the user events to simulate, or the links to be followed). These proposals only download the pages that are defined in the scripts, which either are relevant or lead to relevant pages. Therefore, they minimise the number of irrelevant pages downloaded.

### 4.3.1 Script recorders

Some scripting proposals provide visual support for the user to define the scripts. The simplest proposals offer the user a recorder-like interface, and they require the user to perform the navigation steps, such as selecting a search form, providing the keywords to fill it in, selecting the links that



must be followed from the resulting hubs, and so on. These steps compose the scripts, which are recorded and later repeated step by step.

Davulcu and others [38] proposed a technique to extract information from the Web, which included a navigator. In this technique, scripts are expressed using two declarative languages: F-logic to model objects, such as web pages, forms and links, and transactional logic to model sequences of actions, such as submitting forms or following links. Their navigator includes a Prolog-based module that learns the scripts from user examples, i.e., as the user navigates the site, the system models the objects in the pages he or she visits, and the actions he or she performs. In some cases, the user has to provide some additional information, like which fields are mandatory in each form. Then, the scripts are executed by a transaction F-logic interpreter module. This system is able to automatically update the scripts when minor changes are made to web sites, such as the addition of new options in a select field of a search form.

Anupam and others [4] proposed WebVCR, a recorder with a VCR-like interface to record and replay a user's navigation steps through a web site. Scripts are XML-like documents in which the steps can be either following a link or submitting a form with user-provided values. For each link to be followed, the recorder stores some of its DOM attributes (name, href, target, and locator). For each form to be submitted, the recorder stores some of its DOM attributes (name, method, action), as well as the list of form fields and values that must be used to fill in the form. The recorder can be implemented as a client applet to be loaded in a web browser, or as a web server that acts as a proxy for the user requests. In the server version, the replay of a script is invisible to the user, who only receives the final page. The authors propose some heuristics to update the script automatically after minor changes in web pages occur, and to optimise the replay of scripts by not loading the images in the intermediate pages.

Pan and others [105] introduced the WARGO system to generate web wrappers. WARGO uses a browser-based recorder that captures the user actions and creates a script that is written in a declarative language called Navigation Sequence Language (NSEQL). NSEQL is based on the Web-Browser Control, which is a module in Microsoft Internet Explorer that supplies the functionalities associated to navigation. Therefore, NSEQL is composed of commands that allow, for example, loading a web page, finding a form in a web page by name, filling form fields, or clicking on links and buttons. Often, it is necessary to leave some commands undefined until runtime, e.g., the number of clicks on a "next" link, which depend on the results returned.

Baumgartner and others [9] proposed Lixto Visual Wrapper, a web wrapper generation tool. It provides an Eclipse-based interface which embeds a Mozilla browser that captures user actions on a web page, stores them in an XML-based script, and replays them when necessary. Scripts include actions that involve mouse or keyboard user events, such as following a link, filling a text field in a form, or selecting an option in a select field. The HTML elements that are involved in each action are identified by means of XPath expressions. More complex actions can be also modelled using Elog, a logic-based programming language. Their proposal includes some handlers to deal with unexpected events, such as popup dialogues that did not pop while the script was being recorded.

Montoto and others [100] proposed a recorder that captures user actions building on DOM events (e.g., clicking on a button, or filling a text field). In contrast to the previous proposals, they consider every possible DOM event that involves any HTML element (e.g., moving the mouse over an element or entering a character in a text field), and they deal with AJAX requests. Another difference is that the user has to explicitly select each element and choose on a menu which of the events available he or she wishes to execute; this helps discern irrelevant events easily. The recorder implementation is based on the Internet Explorer programming interface to record user actions, but the resulting scripts can be executed using either Internet Explorer or Firefox. The authors proposed an algorithm to build robust XPath expressions to identify HTML elements in the script. Actually, their technique is an extension of the proposal by Pan and others [105].

Selenium [70] is a widely-used open-source project for testing automation. It provides Selenium IDE, a recorder that can be integrated in web browsers to capture user actions and translate them into a script that can be later executed by this tool. Scripts can also be handcrafted building on an programming interface of predefined commands that is provided by the recorder. These commands implement usual interactions with web pages, such as opening a web page, typing text in a form field, or clicking on buttons and links. Scripts created in the recorder are stored in HTML format. Furthermore, Selenium provides libraries for different programming languages such as Perl, PHP, C#, Java, Python, or Ruby, which allow defining scripts that take advantage of the functionalities offered by those languages. These scripts are executed on the Selenium server (Selenium Remote Control Server), which is responsible for launching and shutting down browsers, and acts as a proxy for the requests made to the browsers.

### 4.3.2 Script learners

Script learners analyse a web site to learn navigation scripts for that site. They can be supervised, which requires the user to provide sample navigation sequences from which a script can be learned, or unsupervised, which provide techniques to analyse a web site and learn navigation sequences with no user intervention.

Blythe and others [23] proposed a supervised technique called EzBuilder. To create a navigation script for a web site the user has to provide some examples, submitting forms, and navigating to several relevant pages. The number of examples depends on the regularity of the site. User actions, such as clicking buttons or filling forms, are captured by the system. Then, EzBuilder generalises the user's behaviour to create a script, i.e., the script is not a repetition of the user steps, but a general sequence of steps that must be performed to reach pages that are similar to the ones that were provided as examples. The resulting scripts may require the user to provide additional data to fill in forms; the information extracted from the relevant pages is returned in XML format.

Lage and others [89] proposed an unsupervised technique to learn navigation scripts in web sites that follow a common script template, e.g., scripts start by filling a search form, which after submission returns a hub page, which contains links to detail pages. This technique is supported by a repository of sample relevant data, which consists of objects specified by a set of attribute-value pairs. To create a script, their technique first performs a blind search to find a search form; then, it tries to find a correspondence between the form labels and the attributes in the repository to fill in and submit the form. The response page is analysed: if it contains any of the sample data, it is a detail page; otherwise, if it contains links with anchors similar to "More" or "Next", or links with non-alphabetical anchor text (e.g., "»"), it is a hub, its links are followed and the target pages are characterised similarly. Finally, the previous steps are encoded as a Java program.

Vidal and others [128] proposed an additional unsupervised technique that receives a sample page as input, which represents the class of pages that are considered relevant, and it returns a navigation pattern, which is composed of the sequence of regular expressions that represent the URLs that lead to relevant pages in a web site. In this context, relevant pages are pages that are structurally similar to the sample page, and the similarity is measured in terms of a tree-edit distance on DOM trees. Their proposal involves two steps: site mapping and pattern generation. Site mapping consists in building a map of the web site, which requires to crawl the entire site.

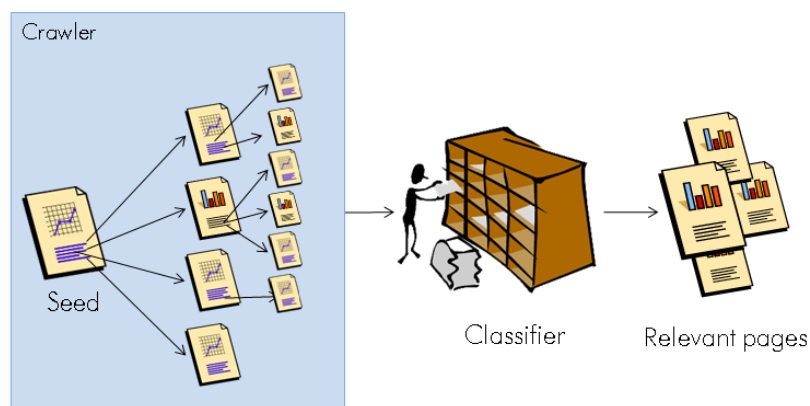
They keep record of the paths in the map that lead (directly or indirectly) to pages that are similar to the sample page (target pages). Then, pattern generation consists in generalising the URLs of the pages in the former paths using regular expressions, and then selecting the best path, i.e., the one that leads to the largest number of target pages.

Senkul and Salin [117] proposed a page recommendation system that uses semantic information to improve the recommendations. They analyse data from web application logs, which provide sequences of pages requested by users during a single navigation session of a maximum of 20 minutes. Their system uses a mining technique that is based on SPADE, a sequential association rule mining algorithm, to generate navigation sequences. Regarding semantic information, they use a handcrafted ontology to model the information provided by each site, and they handcraft annotations that help establish a correspondence between every web page and the ontology concepts that describe the information it provides. When users start navigating a site, the system shows them the ontology that models the information in that site, and they choose the concepts that they think are relevant. Then, the system generates navigation sequences using exclusively the information about web pages that correspond to the relevant concepts selected by the users.

## 4.4 Crawling proposals

The goal of a web crawler is to download a subset of pages from a web site. They can be either blind or focused crawlers. The former start with a so-called seed page and follows the links it provides transitively until every page that is reachable from the seed has been downloaded; the latter make an attempt to download as few irrelevant pages as possible, which implies that they do not follow every link, but only those that are likely to lead to relevant pages. To implement an automatic web navigator based on a crawler, an a posteriori classifier has to be used to discern which of the pages that were retrieved are actually relevant to the user query.

Web page classifiers assign each page to one or more classes after analysing some of its features. Regarding the location, the features are usually internal [5, 8, 14, 21, 36, 41, 47, 52, 71, 85, 116, 129, 132, 138, 139]; or external [7, 12, 13, 18, 22, 40, 77, 87, 120, 128]. Regarding the types of features, we distinguish between contents-based classifiers [14, 71, 87, 116], link-based classifiers [18, 40, 52, 132, 139], visual-based classifiers [47, 85, 138], URL-based classifiers [7, 12, 13, 22, 77, 120, 128] and structure-based classifiers [5, 8, 21, 36, 41, 129]



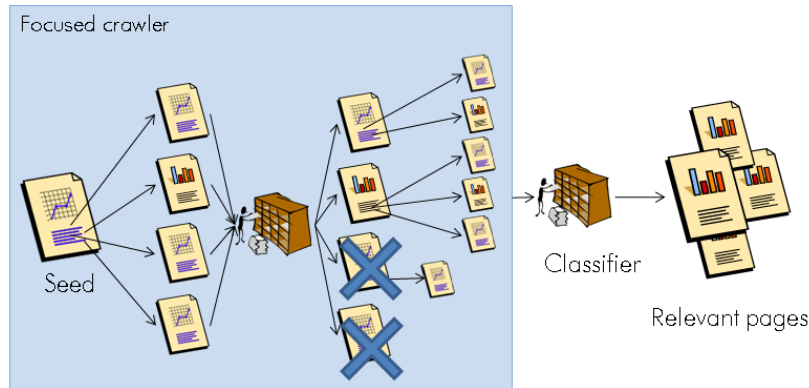
**Figure 4.7:** Navigation using crawlers.

#### 4.4.1 Blind crawling

Blind crawlers aim to collect as many pages as possible from a web site. If we consider a web site as a graph in which its pages are the nodes and the links between pairs of pages are the edges, blind crawling a site amounts to traversing every node in the graph that is connected to a previously traversed node, starting from a number of seeds, cf. Figure 4.7.

Initially, blind crawlers were designed to retrieve static pages from web sites. Later, web applications that offered their data only through search forms progressively became the main source of data in the Web. Therefore, an effort was made to develop crawlers that were able to retrieve pages behind web forms. To achieve this, many form filling techniques were proposed to analyse forms and determine a subset of form fields and a set of values to fill them [92, 95].

Raghavan and Garcia-Molina [112] proposed HiWE, a blind crawler to extract information from pages behind web forms. HiWE behaves as a regular blind crawler; it iterates through a list of URLs and downloads every page that can be reached transitively from these seeds. Whenever a downloaded page contains a form, it is analysed and filled automatically to reach the pages that are behind it. Every form is composed of a set of fields, some of which have a descriptive label. Form submission is supported by a repository of user-provided sample labelled values. HiWE tries to find correspondences between the form field labels and the labels in the repository, using a matching algorithm that is based on minimising string-edit distances. When a



**Figure 4.8:** Navigation using focused crawlers.

match is found between a form label and a repository label, the value associated to the latter is used to fill in the corresponding form fields.

Madhavan and others [95] proposed a blind crawler to retrieve pages behind web forms, so that they could be indexed by Google's search engine. The goal is not to retrieve as many hubs as possible, but to retrieve a set of hubs from which the majority of pages in a web site are reachable. They proposed a technique to discover informative query templates, i.e., combinations of input fields such that filling them results in sufficiently distinct response pages. Each possible query template in a form is evaluated in terms of the number of dissimilar web pages that result from filling its fields and submitting the form. The similarity between two web pages is measured using a contents-based web page classifier, cf. Section 5.2. Query templates that return a number of dissimilar pages higher than a threshold are considered informative. Then, informative query templates are filled in and the resulting hubs are used as seeds by a blind crawler to retrieve the rest of the pages from the site.

#### 4.4.2 Focused crawling

Focused crawlers are similar to blind crawlers, but they classify the pages retrieved into a number of topics. If a page belongs to a given topic, then its links are used as seeds to crawl new pages; otherwise, if a page is out of topic, it is classified as irrelevant and its links are not followed, cf., Figure 4.8. The process is repeated until no more links are available or a given number of pages has been retrieved. Recent research in this field fo-



cuses on improving the efficiency of the crawling process by minimising the number of irrelevant pages that are downloaded [3, 91, 101, 107].

Chakrabarti and others [26] proposed one of the earliest focused crawlers. It uses a initial set of seeds gathered from the results of a keyword-based search using a set of user-provided words that define some topics. Then, the pages that are linked from pages in the initial set, or that link to them, are added to the set. The pages in the resulting set are assigned a score building on an iterative procedure that is similar to Kleinberg's HITS algorithm [80], which is based on the hub and authority concepts, i.e., a hub is a web page that contains many links to other pages and an authority is a web page that is linked by many other pages. Recall that a hub page is usually returned as a response to a query on a search form. Basically, a hub score is computed as the weighted sum of the authority scores of the pages to which it links, which are in turn computed from the scores of the hubs from which they are linked. Weights are computed considering the quantity of topic-related text that surrounds each link. After some iterations, the pages with the highest hub and authority scores are selected as relevant.

Chakrabarti [25], Chakrabarti and others [27], de Assis and others [39], Pant and Srinivasan [106] and Partalas and others [109] proposed several focused crawling techniques that measure the relevance of a page as it is retrieved by a crawler. When the relevance of a downloaded page is low, it is considered out of topic and its links are discarded; otherwise, they are added to the seed set. These techniques build on contents-based classifiers that are based on internal features of the pages being crawled. Contrarily, the proposals by Li and others [91], Aggarwal and others [3] and Pant and Srinivasan [107] explore the idea of using external features, i.e., features computed on the pages from which they are linked. This is appealing insofar it helps to classify a web page without downloading it. The external features are based on words in the text surrounding a link, the anchor text, or even in the URL address.

Mukherjea [101] proposed two heuristics to improve the efficiency of focused crawlers. Their proposal relies on the idea that web applications organise their web pages internally as a tree-like hierarchical directory. The heuristics are based on the following observations: first, pages that are close in the hierarchy are more likely to provide similar contents, so a link should not be followed if it refers to a page that is not close; second, pages in the same directory are likely to provide similar contents; therefore, if a significant number of out-of-topic pages are found in a directory, we may safely discard every page in that directory.

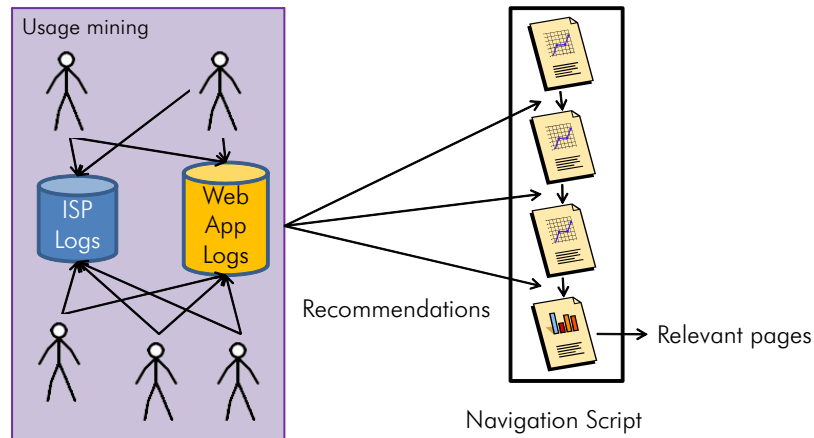


Figure 4.9: Navigation using usage mining techniques.

## 4.5 Usage mining proposals

Usage mining techniques are based on mining real-world web application logs or Internet Service Providers (ISP) records to learn navigation patterns that are common to a number of users. Usually, these techniques are used to generate page recommendations that provide orientation for users when navigating a web site. These techniques can be used to learn, for example, which is the usual page users start navigating or which the most frequent action in each page is. As a conclusion, these techniques help implement navigators that find what the crowd think are the more relevant pages in a web site, cf. Figure 4.9.

Yao and others [136] proposed PagePrompter, a system to generate intelligent agents for page recommendation. PagePrompter collects information from different sources: web application logs (e.g., visitor statistics), the web site itself (e.g., page updates or changes in the web site layout), and user feedback (e.g., explicit recommendations). PagePrompter is based on association rule mining to compute associations of web pages that are frequently accessed together, and clustering algorithms to refine the former associations. Web pages are clustered according to their contents and the log information about users visiting them, to create clusters of pages that are semantically related and visited together. Both association rules and clusterings are considered when making recommendations. PagePrompter offers the visitor of a web page recommendations such as: the most visited pages in the site, the most recently created or updated pages, and pages that are commonly visited after/together with the current page.



Korfiatis and Paliouras [84] proposed CANUMGI, a technique to model user navigation for page recommendation. It performs an offline analysis of data from different web sites across the Web, using data from Internet Service Providers logs. These logs offer information about commonly visited sequences of pages. CANUMGI performs usage mining on these data and learns a probabilistic regular grammar in which web pages are the symbols and page sequences are the strings. This grammar is used to create page recommendations for users, i.e., when the user is visiting a certain page, CANUMGI recommends him or her the pages that have a higher probability of being reached from the current page according to the grammar. To create more useful recommendations, the probability between two pages in the grammar is computed considering the similarity between their contents as well, using a vector of keywords to represent each page.

## 4.6 Summary

In this chapter, we have presented an overview of automated web navigation proposals. First, we have described the different types of pages that must be dealt with by a navigator, and then we have presented current proposals that implement automated navigation in different contexts. We also provide an insight into how recommender systems and crawlers can be used to implement automatic web navigators.



---

## Chapter 5

# Web page classification

---

*It is clear that there is no classification of the Universe that is not arbitrary and full of conjectures. The reason for this is very simple: we do not know what kind of thing the universe is.*  
*The Analytical Language of John Wilkins, 1942.*

*Jorge Francisco Isidoro Luis Borges, writer (1899-1986)*

**W**eb page classification refers to the problem of automatically assigning a web page to one or more classes after analysing its features. In this chapter, we present an overview of web page classification proposals and describe the different techniques that have been proposed to classify web pages in the literature. The chapter is organised as follows: in Section 5.1, we introduce the concept of web page classification; Section 5.2 reports on the existing proposals that classify web pages according to their contents; Section 5.3 reports on the proposals that classify web pages according to the links between them and other pages; Section 5.4 reports on the proposals that classify web pages according to visual features; Section 5.5 reports on the proposals that classify web pages according to their URL address; Section 5.4 reports on some techniques that detect the template behind a web page, which can be applied to the problem of web page classification; finally, we summarise the chapter in Section 5.7.

## 5.1 Introduction

Web page classification has been extensively researched and several techniques have been applied with successful experimental results. They usually have leveraged previous results in the field of text classification. However, web pages present some peculiarities that require a more specific processing, e.g., the existence of links amongst pages.

In general, we distinguish between binary classifiers, which are able to determine if a page belongs to a certain class or not, and multiclass classifiers, which can classify a page in one (or more) classes out of a set of classes.

Classifiers are learned from a training set, which is a dataset of selected web pages. Depending on whether the pages in the training set have a predefined class or not, the techniques to learn classifiers are catalogued as supervised or unsupervised. Note that it is very common to use the expression “a supervised classifier” or an “unsupervised classifier” to refer to classifiers that were learned using supervised or unsupervised techniques, respectively. In this dissertation we use these expressions since they are so common that they cannot induce any confusion.

Supervised classifiers require the user to annotate the training set, i.e., they require the user to analyse every page in the training set and assign it to one or more classes; the goal of the technique used to learn the classifier is to infer the intuition behind this process. This is usually considered one of the main problems with supervised classifiers, since analysing the training set is usually tedious, time-consuming, and error-prone.

Unsupervised classifiers, aka clustering classifiers, work on a training set in which the web pages have not been pre-classified by the users [75]. This problem is far more difficult to solve since there is no information about the classes, aka clusters in this field. These techniques are based on the concept of distance between the elements to be classified [43]; in general, they try to find a set of classes such that the pages that belong to a class are as close as possible to each other, but as distant as possible from the pages in other classes [134]. Unsupervised classifiers are appealing insofar they relieve the user from the burden of analysing the training set, but require him or her to analyse the resulting classes and assigning a meaning to them (which hopefully requires much less effort than annotating a training set).

The most usual clustering algorithms are either partitional or hierarchical. Partitional algorithms such as k-means [93] or k-medoid [78, 108] consider mutually disjoint clusters of elements, whereas hierarchical algorithms

such as Single-Link [122] or Average-Link [115] consider a hierarchy of clusters, in which elements in a parent cluster are the union of the elements in its child clusters [78]. Hierarchical algorithms can take a bottom-up or top-down approach. Bottom-up algorithms are agglomerative, i.e., they start with an initial clustering in which each element is in a different cluster, and they recursively merge pairs of clusters to create a parent cluster. On the other hand, top-down algorithms start with every element in the same cluster, and they recursively split each parent cluster into child clusters. There are also hybrid approaches, e.g., Scatter/Gather, which first creates a hierarchical clustering and then refines it using the k-means algorithm [37].

An additional dimension to catalogue web page classifiers is regarding the location and the type of the features on which they rely.

Regarding the location, the features are usually internal, i.e., they are computed from the contents of the page to be classified, including the frequency of words [71], tag paths [21], or sets of words that frequently appear together [14]; this, obviously, requires the pages to be downloaded before they can be classified. Some classifiers use external features, i.e., features that are computed from neighbouring pages, which includes the words in the anchors, the words in the paragraph that surrounds the anchors [33], or a combination of them [50]. If the link is surrounded by a descriptive paragraph or the link itself contains descriptive words, it is possible to decide the topic of the page before downloading it.

Regarding the types of features, we distinguish contents-based, link-based, visual, and URL-based [111]. Furthermore, there are some techniques to detect the template behind a web page that can be used for classification purposes, as well.

## 5.2 Contents-based proposals

Contents-based classifiers categorise a web page according to the words it contains. The proposals in this area do not actually work with the words as they are presented in a web page, but with their stems in order to reduce the number of features to be analysed; similarly, stop words like articles, prepositions, or conjunctions are usually discarded. The resulting strings are usually referred to as the page terms.

The authors in this area do not usually refer to words, but to terms, which are basically words that have been preprocessed, e.g., applying stemming and/or discarding stop words. Some of these proposals are based on

the well-known bag of terms approach that is frequently used in text processing. According to it, a page is represented as a map from terms onto their frequency; later, a proposal to learn a classifier from these frequencies can be used, such as neural networks [135], k-nearest neighbours [87], or Support Vector Machines [45].

However, most proposals apply the vector space model to represent pages as vectors. In this model, we keep a list of every possible term in the set of documents and each page is represented as the vector of term frequencies. Then, they measure the distance between two pages as the cosine of the angle between their corresponding vectors [96]. Since each page usually contains a large number of terms, the vector space has a high dimensionality, but the majority of components are zeroes. Some authors have noted that contents-based classification can be improved by using a brief summary of the page instead of the whole document [118].

Beil and others [14] proposed a technique to cluster web pages that is based in the frequent terms set concept. A frequent term set is a set of terms that frequently co-occur in every input page more than a pre-defined number of times. Frequent term sets are used as descriptions for candidate clusters, i.e., each term set represents a cluster, and pages that contain every term in that term set belong to the corresponding cluster. Since this definition produces overlapping clusters that diminish clustering precision, they propose two greedy algorithms to compute a clustering with a minimal overlapping: FTC, which computes a partitional clustering, and HTC, which computes a hierarchical clustering.

Hotho and others [71] proposed COSA, a technique to represent web pages as a vector of concepts. Their technique takes a set of input web pages that are represented as vectors of terms, and it is supported by a handcrafted hierarchical ontology that models the concepts. Then, they use a text processor to automatically map the terms in each vector onto concepts in the ontology, thus creating vectors of concepts. These vectors are then used as input to a k-means clustering algorithm. Since several terms refer to the same concept, the former transformation reduces the dimensionality of the vector space. They apply some heuristics based on the support of a concept, which is computed as the frequency of the terms mapped onto that concept that appear in pages in the input set. To reduce further the space dimensionality, they discard concepts that have the smallest support and split concepts with the largest support into their child concepts. The authors extended this proposal using WordNet [72].

Kwon and Lee [87] proposed a supervised technique to classify web pages based on the k-nearest neighbour algorithm [135]. Their technique

takes the URL of a web site as input, and it assigns each page in the site a weight based on its number of incoming and outgoing links, and the number of intermediate links that must be followed to reach it from the home page. Pages with a high weight are more representative, so the user is required to assign a class to their term vectors them so that they can be used as a training set. Term vectors are reduced by discarding terms that are not representative of any classes. To classify a test page, the algorithm estimates the average class of its  $k$ -nearest training pages. The similarity between two pages is measured as the standard cosine similarity multiplied by a weight that depends on the number of terms co-occurring in both pages.

Selamat and Omatu [116] proposed WPCM, another supervised web page classification method based on neural networks that uses the term vector representation. It takes a set of pre-classified terms vectors as input, which are preprocessed using two feature selection algorithms: first, the well-known Principal Component Analysis algorithm [76] is used to discern the principal terms, i.e., the terms that are most useful for classification. Then, the resulting vectors are augmented with the  $N$  terms with the highest entropy in the input set. Then, these vectors are used as the input to train a neural network using the well-known back-propagation method [61].

### 5.3 Link-based proposals

Some authors have proposed techniques to classify web pages that not only consider the web page themselves, but the links to and from other web pages. Usually, these techniques represent the Web as a graph in which nodes are web pages, and edges are links from one web page to another.

Following this approach, Bhagat and others [18] proposed a semi-supervised link-based classifier that focuses on blogs, i.e., web pages in which users publish personal information about their lives and interests. It is based on the idea that people usually include in their blogs links to the blogs of other people with whom they share some common interests or demographical attributes (e.g., age, location, or gender). Their technique takes a web page graph in which some nodes are pre-classified as input, and they propose two algorithms to use the information provided by classified nodes to predict the classes of the other nodes. These algorithms are based on relational learning [102] and the  $k$ -nearest neighbour technique [135]. De Campos and others [40] explored the same idea to classify web pages (not only blogs) using a Bayesian network.

Finally, some proposals do not use link-based features in isolation, but together with other types of features to improve the classification. As an example, Getoor and others [52], Zhu and others [139] and Xie and others [132] included link-based features into a contents-based web page classifier, which helped them achieve significant improvements regarding the precision of their classifier.

## 5.4 Visual-based proposals

Some authors have worked on techniques to classify web pages using visual features, i.e., features that can only be computed when a web page is rendered by a browser, e.g., the position of an image on the screen, its bounding box, the distance to the other elements in the page, and so on.

Visual features are used to distinguish between different areas or vision blocks in a web page, which are sections of the page that represent a single unit with a certain functionality or topic. Then, the classification of web pages is based on the idea that pages that belong to the same class usually organise their vision blocks similarly. Therefore, to classify a web page, it is compared against other annotated pages, and assigned the class of the page that distributes its vision blocks more similarly.

Zhu and others [138] proposed a technique to classify links according to their functionality; it relies on a predefined taxonomy of functionalities that includes navigational, indexing and directories, recommendations and expanding information, or advertising. Their classifier was based on two types of features: structural, such as the position of the link in the DOM tree, visual, such as the position of the link in the page rendered by the browser, and the position and size of the vision block that contains the link. They used these features to build two classifiers (a support vector machine and a decision tree).

Fersini and others [47] proposed a technique to classify web pages that assigns a measure of importance to the elements of the page. Their measure is based on the idea that images are used in web pages to attract the user's attention, and therefore the text in the same vision block as an image usually has more importance than the text in other sections of the page. An analysis of visual features is performed to detect which sections of text are included in the same vision block as an image.

Kovacevic and others [85] proposed a technique to distinguish between different areas of a web page. They analyse the screen position of each element in a web page, where the screen is assumed to be a maximised browser



window on a standard monitor with a resolution of  $1024 \times 768$  pixels. Then, their technique uses some heuristics to identify standard screen areas, such as header, footer, left menu, right menu or centre (e.g., the left menu is identified as the area in the leftmost part of the screen, which occupies the 30% of the total screen width).

## 5.5 URL-based proposals

Classifying a web page building on features of its URL is appealing insofar it can be classified without actually downloading it, which has a positive impact on performance [12]. The classification features in this case are the segments of which it is composed.

To classify a page based on its URL, it is not necessary to analyse the page itself, but only pages that link to it. Classification features are extracted from those links, such as the link URL [22, 128] or the URL parts [12, 13, 77, 120]. There are some proposals to classify web pages according to their URL.

Shih and Karger [120] proposed a supervised web page classification technique that is based on the idea that two visually nearby elements probably belong to the same class and, likewise, similar URLs probably have similar pages as target. The technique tokenises a set of training URLs using the characters '/', '&', and '?' as separators, and inserts the tokens in a tree structure. The root of the tree contains the first token (e.g., http:), and the other tokens are progressively inserted in order in the tree, each of them as a child of the previous tokens. Then, the technique builds a Bayesian network from the tree as follows: they initially assign a class to each node of the tree so that the probability of a token belonging to the same class as the parent token is maximised; to prevent overfitting, they introduce a mutation probability that allows a child token to change its class. Then, some leaves in the tree are assigned the class they have in a trained set of annotated URLs, and the classes of other nodes in the tree are updated according to their mutation probability. Finally, each URL is assigned the class of its associated leaf node.

Kan and Thi [77] proposed a supervised web page classifier for pages in different web sites that is based exclusively on features computed from their URLs. The URLs are tokenised using the standard RFC 3986 format for URIs, and their tokens are used as features; then, more features are computed, such as the position of each token in the URL, the length of the URL, or the lexical kind of token (e.g., if it represents a number, a word, or a non-alphabetical symbol). These features are used as input to an entropy maximization algorithm, a well-known machine learning approach usually applied to text

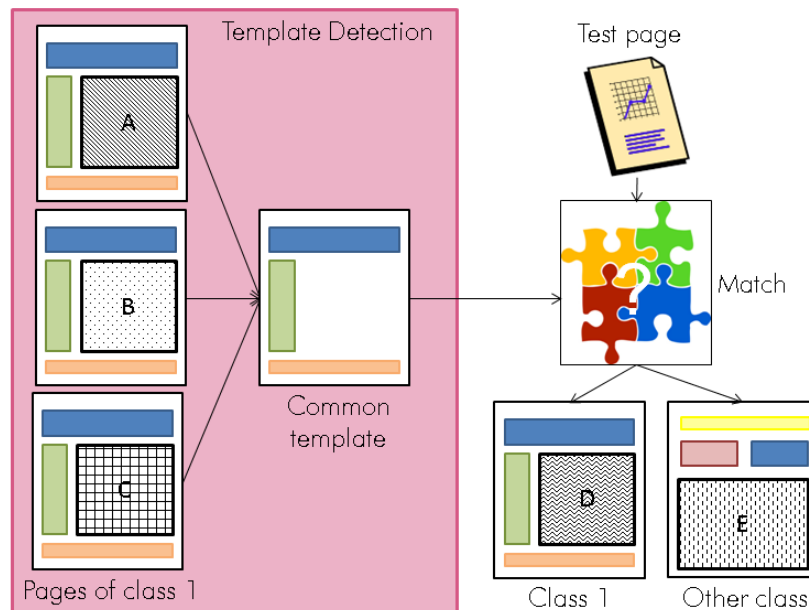
classification [16, 103]. To build the classifier, they use large training sets of URLs to achieve good precision and recall, which requires a previous extensive crawling of the sites that are being analysed.

Baykan and others [11, 12] presented a supervised web page classification proposal that builds exclusively on URLs. They create feature vectors by tokenising URLs and then use those features to build both a support vector machine and a naïve bayes classifier. In their experiments, they use large training sets of URLs, and they require the user to provide a list of words and URLs that are representative of every class; furthermore, they also require the user to provide sample URLs that are not representative of each class.

Vidal and others [128] proposed a supervised technique to classify web pages building on their URL. Their proposal takes a sample page as input, and returns a set of URL patterns that match the URLs of pages that are structurally similar to the sample page. It is based on two steps: site mapping and pattern generation. Site mapping consists in building a map of the web site, which requires to crawl the entire site starting from its home page and following every possible path. They keep a record of the paths in the map that lead (directly or indirectly) to pages that are similar to the sample page. The similarity is measured using a tree-edit distance between the DOM trees underlying the pages. Then, pattern generation consists of generalising the URLs of the pages in the former paths using regular expressions, and then selecting the path that leads to the largest number of target pages.

Bar-Yossef and others [7] proposed a supervised technique to detect web pages with different URLs that have the same contents, which has a negative impact on crawling efficiency. To solve this problem, they classify URLs according to the contents of their target, and they build regular expressions to define each class of URLs. Then, some rules are generated to normalise those URLs, using a rule mining algorithm. They need to have a large collection of URLs to achieve good results, which means that a previous extensive crawling of the web site must be performed to gather them. A similar proposal was presented by Koppula and others [83].

Blanco and others [22] proposed an unsupervised algorithm to classify web pages that combines both internal and external features. Their proposal is based on the idea that every web site is created by populating a number of HTML templates with data from a database, and that the URLs of those pages are created by populating a URL template with data from the same database. Therefore, pages created from the same HTML template have similar contents and URLs generated from the same URL template link to pages



**Figure 5.1:** Web page classification using template detection techniques.

with similar contents. They proposed an algorithm that combines web page contents and its URL as features to cluster web pages so that each cluster contains pages that were created using a certain template. Their algorithm is based on the well-known minimum description length method [55]. They require a large training set, so they crawl the entire site in their experiments. Note that to improve the classification efficiency, internal features are used, which means that in some cases the page must be downloaded previously.

## 5.6 Structure-based proposals

Structure-based classifiers build on the idea of template. Web pages are usually generated by means of a server-side templates that provide the structure of the pages and have placeholders that must be filled in with data by means of server-side scripts [8, 53]. As a consequence, web pages that are generated by the same template are likely to belong to the same class. This implies that the proposals in the literature to learn templates can be easily leveraged to learn web page classifiers: first a template for each class is learned; when a web page needs to be classified it is compared to the different templates and assigned to the class whose template is more similar, cf. Figure 5.1.

Crescenzi and others [36] proposed RoadRunner, a technique to automatically generate wrappers to extract structured information from web pages. RoadRunner includes a supervised template detection technique. It starts using one of the pages in the training set as an initial template and then tries to match the other pages of the same class to the template. Whenever a mismatch is found, they generalise the template so that it can account for it. The generalisation process consists in changing some literal parts of the template into regular expressions that account for the variability found in the pages in the training set. This process is not always possible, which happens when a page is not actually generated using the same template as the others. As a conclusion, the structural feature is a regular expression that characterises the template used to generate the pages that belong to a given class.

Bar-Yossef and Rajagopalan [8] proposed an unsupervised template detection technique that is based on pagelets, i.e., non-overlapping sections of a web page that provide information about a single topic. Their technique detects pagelets as nodes in the DOM tree of a web page such that none of its children has more than  $k$  links (an element with less than  $k$  links is probably not an independent pagelet, but part of a parent pagelet), and that none of their ancestors is another pagelet. A template is defined as a collection of pagelets with the same contents that are shared by several pages in the collection. Therefore, to find the common template of a set of pages, these are split into pagelets, which are grouped by their contents.

Arasu and Garcia-Molina [5] proposed ExAlg, an algorithm to detect the common template behind a set of pages. ExAlg takes a set of pages as input, which are tokenised into words and HTML tags. First, the algorithm computes equivalence classes of tokens, i.e., sets of tokens that occur with the same frequency in every web page in the input set. ExAlg keeps classes that have a sufficiently large number of tokens whose frequency is high and discards the rest. Then, each equivalence class is split according to the context in which the tokens appear, e.g., the path from the root of the DOM tree to the token. The template is characterised by the equivalence classes that result from this process.

Blanco and others [21] proposed a technique to detect page templates that are expressed as sets of paths in the DOM tree of the input pages. Their technique takes a set of pages of a given class as input. They group their DOM nodes according to their frequency and select the ones that belong to the template using four strategies: Template Page Model (TPM), which selects the leaves with a higher frequency; Advanced TPM, which excludes leaves that are shared by every page in the input set; Link Page Model (LPM),

which considers only leaves that represent links; and Advanced LPM, which excludes the link nodes that are shared by every page in the input set.

De Castro Reis and others [41] devised a technique that builds on RTDM, a tree-edit distance that is limited to edits on the leaves. It takes a set of pages as input and represents them using their DOM trees. First, the trees are clustered using a hierarchical clustering algorithm that uses RTDM as the similarity measure. Each resulting cluster is generalised to a template by creating a tree that contains the nodes that are common to every tree in the cluster; some nodes are wildcarded to abstract away from the differences in the trees. Vieira and others [129] proposed a variation in which nodes that are identical in every DOM tree in the training set are immediately considered a part of the template.

## 5.7 Summary

In this chapter, we have presented an overview of the many proposals in the literature regarding web page classification. We have grouped them into several categories according to the kinds of features on which they rely, namely: contents-based, link-based, visual-based, and URL-based. Furthermore, we have reported on some template-detection techniques and we have provided some clues on how they can be applied to the problem of web page classification. Most of the techniques presented are based on internal features, i.e., features that are computed from the page to be classified, which requires the page to be downloaded beforehand. On the contrary, URL-based techniques are based exclusively on page URLs, which allows to classify them without downloading them.



---

*Part III*

*Our Proposal*

---





---

## Chapter 6

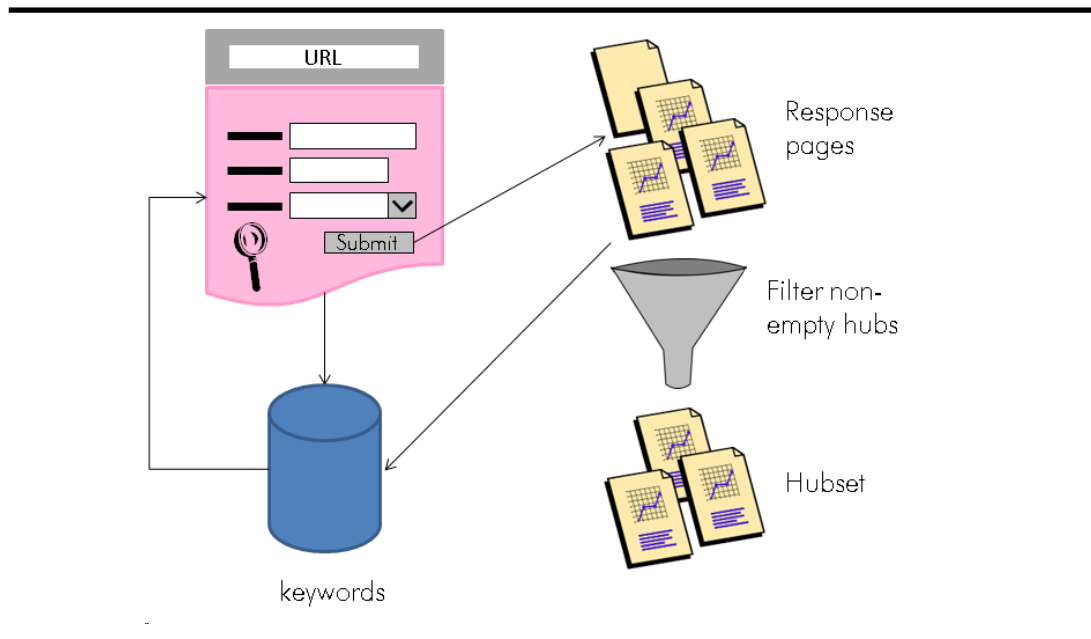
# Our crawler

---

*Different roads sometimes lead to the same castle*  
*A Game of Thrones, 1996.*

*George Raymond Richard Martin, writer (1948)*

**I**n this chapter, we describe the architecture of our crawler and report on the algorithms that implement it. The chapter is organised as follows: in Section 6.1, we introduce it; Section 6.2 describes the architecture of the crawler; Section 6.3 presents the main algorithm that orchestrates the crawler and the ancillary functions that support it; Section 6.4 reports on the complexity analysis of the previous algorithm and functions; finally, we summarise the chapter in Section 6.5.



**Figure 6.1:** *Our crawler.*

## 6.1 Introduction

In this chapter, we describe our proposal to crawl a site and automatically gather a set of hubs from it. Our crawler takes the URL of a web page with a keyword-based search form and is able to fill in the forms using words from the site itself, discern which of the response pages are not empty (i.e., they contain some information related to the query), and gather them. First, our crawler downloads the form page and analyses it to compute some keywords. Then, these keywords are used to fill in the form and gather the response hubs. The hubs are similarly analysed to gather more keywords, which in turn are used to gather more hubs. This process is repeated until a number of hubs have been gathered, cf. Figure 6.1.

In addition, we prove that our crawler is computationally tractable, since  $O(n \log n)$  is an upper bound to its worst-case time complexity, where  $n$  denotes the size of the largest web page in the web site being analysed, which is measured in words.

Throughout the rest of this chapter, we use Microsoft Academic Search as a running example. It is an scholarly web site that offers information about items that include papers, authors, citations, and publishing hosts, i.e., journals and conferences. We also use a number of data types (URL, Token,

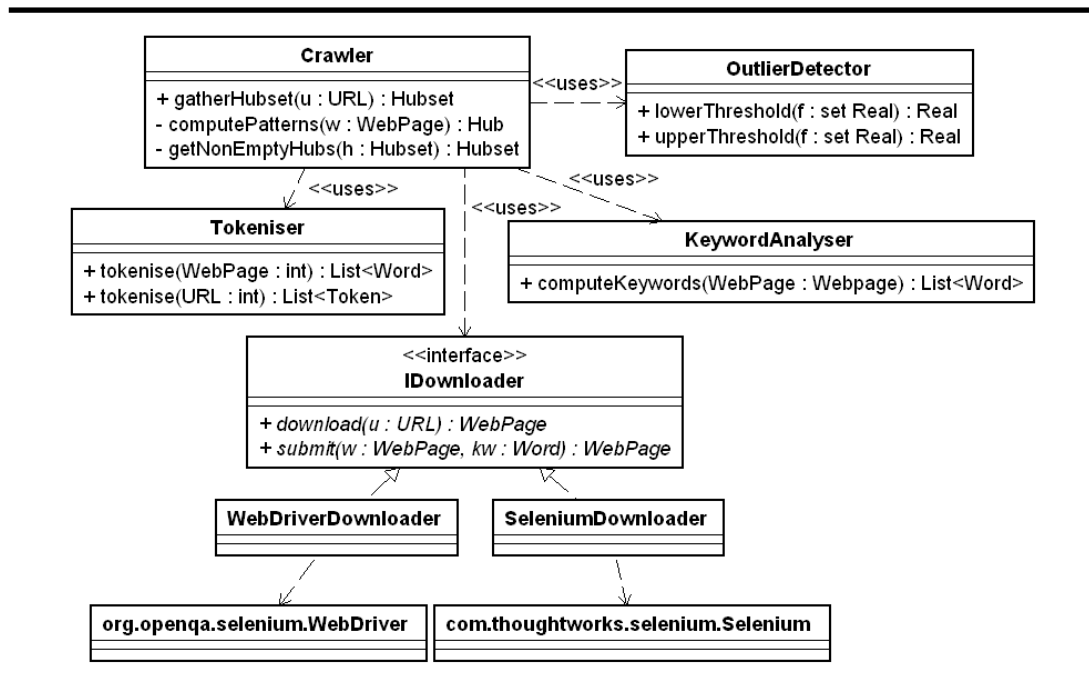


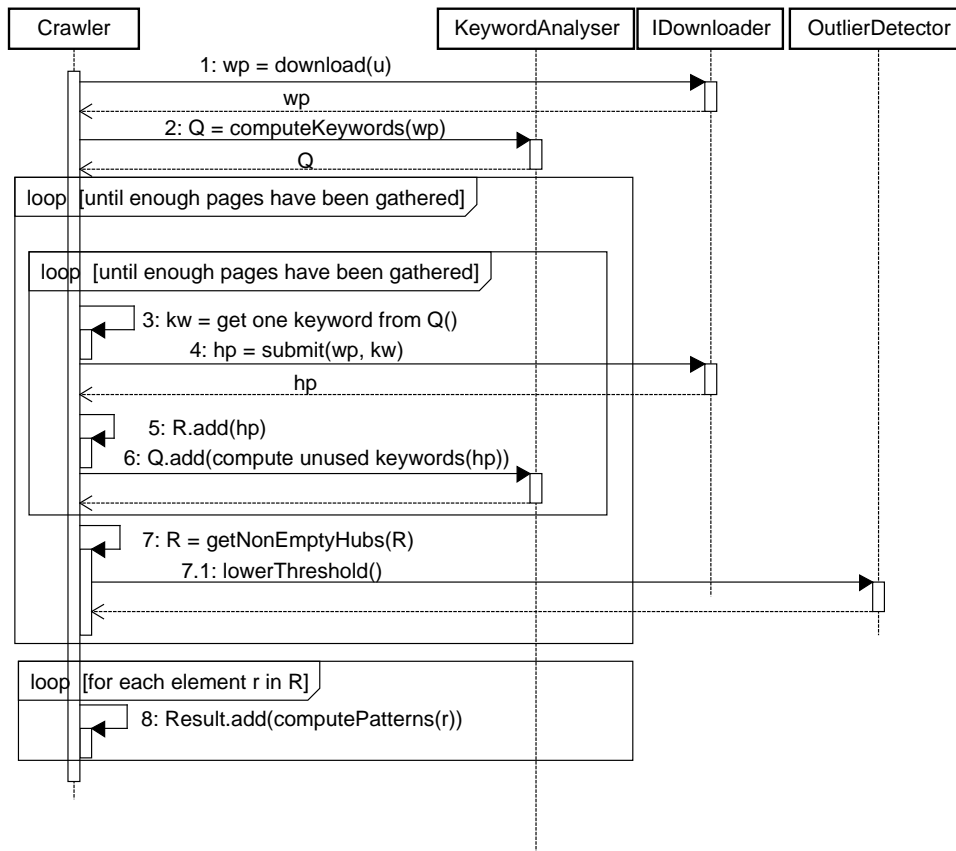
Figure 6.2: Class diagram of our crawler.

Separator, Word, WebPage, Pattern, Hub, and Hubset) and a number of simple functions to work with sets, bags, and sequences. They are formally defined in Appendix A.

## 6.2 Architecture

The architecture of the crawler is presented in Figure 6.2. Class `Crawler`, which provides function `gatherHubset` to gather a set of hubs, has to interact with a keyword-based search form. Therefore, it needs a number of keywords, which are provided by class `KeywordAnalyser`. This class provides a method to analyse the pages of a web site and computes their keywords. The tokenisation is provided by class `Tokeniser`. The interaction with a web site is performed by interface `IDownloader`, which is responsible for handling the HTML requests. Finally, an outlier detection technique is needed to discard empty hubs, i.e., hubs that do not contain any result that is relevant to the query, which is provided by class `OutlierDetector`. This outlier detector is based on the Cantelli inequality, which is described in depth in Appendix B.

We tested two different implementations for interface `IDownloader` using Selenium and WebDriver. `WebDriver` [1] belongs to the Selenium project [70],



**Figure 6.3:** Sequence diagram of our crawler.

but instead of injecting Javascript functions into the browsers to execute the scripts, it calls their native programming interface. An additional difference is that WebDriver executes the scripts on a built-in non-graphical user interface.

Function `gatherHubset` orchestrates the other elements of the architecture. We illustrate how it works using a sequence diagram, cf. Figure 6.3. It takes the URL of a page from a web site that provides a keyword-based search form with one text field as input (usually, the web site home page fulfills this requirement). First, the downloader downloads the page and the keyword analyser computes the words with a lower frequency in the page as keywords that can be used to issue queries to the search form. Then, the downloader finds a text field in the form, fills it with the keywords and submits the form, which yields some response pages. These pages are processed to compute more keywords. This process is repeated until enough pages (po-

Microsoft Academic Search

authors All Domains Advanced Search

Academic > Results for "authors" in All Domains

Computer Science (238570)	<a href="#">Proxy-Based Authorization and Accounting for Distributed Systems</a>
Medicine (159555)	B. Clifford Neuman
Engineering (97096)	...discussion of distributed mechanisms to support <b>authorization</b> and accounting by c
Biology (52747)	tosupport restricted proxies, both <b>authorization</b> and accounting can be easily support
Mathematics (51937)	model for <b>authorization</b> and shows how the model can...
Physics (32791)	Conference: International Conference on Distributed Computing Systems - ICDCS, pp.
Social Science (28559)	<a href="#">A Community Authorization Service for Group Collaboration</a> (Citation
Economics & Business (28263)	Laura Pearlman, Von Welch, Ian T. Foster, Carl Kesselman, Steven Tuecke
Geosciences (18810)	...to delegate some of the <b>authority</b> for maintaining fine-grained access...
Chemistry (12174)	Conference: Policies for Distributed Systems and Networks, pp. 50-59, 2002
Arts & Humanities (7257)	<a href="#">Kerberos Authentication and Authorization System</a> (Citations: 150)
Environmental Sciences (7188)	S. P. Miller, B. C. Neuman, J. I. Schiller, J. H. Saltzer
Material Science (5494)	This document describes the assumptions, short and long term goals, and system mo
	system, named Kerberos, for the Athena environment. An appendix specifies the detai
	these goals, and a set of

Figure 6.4: Hub page in the running example.

tential hubs) have been retrieved. Then, the crawler applies our outlier-based technique to discard empty hubs. Since this may reduce the number of hubs gathered, the whole process must be repeated again until enough not-empty potential hubs have been retrieved. Finally, the potential hubs are processed to create hubs by transforming each of their URLs into a pattern. The result is a hubset that contains a representative collection of URLs from the web site.

As an example, Figure 6.4 depicts one of the hubs that can be gathered from our running example web site.

## 6.3 Algorithm

The crawler is responsible for retrieving a set of hubs from a web site, using the URL of a page with a search form as input. Our focus are search forms that provide a unique text box since they are very common in the Web nowadays. Typically, the response to such a query is a hub page, i.e., a web page

---

```

1: algorithm gatherHubset
2: input    u : URL
3: output   Result : Hubset
4: variables wp, hp : WebPage; P, Q : seq Word; R : seq WebPage, n : ℕ
5: constants M, T : ℕ
6:
7: n := 0
8: P := ∅
9: R := ∅
10: Result := ∅
11: – Step 1: Download initial page
12: wp := download(u)
13: – Step 2: Compute keywords from the page contents
14: Q := computeKeywords(wp, P)
15: while n ≤ T ∧ Q ≠ ∅ ∧ #Result < M do
16:   while Q ≠ ∅ ∧ #Result < M do
17:     – Step 3: Submit the form in wp using a keyword from Q
18:     kw := get one keyword from Q
19:     P := P ∪ {kw}
20:     Q := Q \ {kw}
21:     hp := submit(wp, kw)
22:     – Step 4: Add new page to set
23:     R := R ∪ {hp}
24:     – Step 5: Update the set of keywords
25:     Q := Q ∪ computeKeywords(hp, P)
26:   end while
27:   – Step 6: Keep only non-empty hubs
28:   R := getNonEmptyHubs(R)
29:   n := n + 1
30: end while
31: for each r ∈ R do
32:   – Step 7: Create a new hub using the patterns in r
33:   Result := Result ∪ {computePatterns(r)}
34: end for

```

---

**Program 6.1:** *Algorithm gatherHubset.*

that provides summaries and links to other pages [80]. Hub pages usually contain a larger number of URLs than other pages in a web site since their goal is to offer the users as many results related to their queries as possible. Therefore, the probability that they contain a sufficiently representative set of URLs is higher than for other pages. Recall from Section 4.2 that the submission of search form may also result in a detail page, a no-results page, an error page, or a disambiguation page. A detail page or a disambiguation page are not likely to provide as many links as a hub page, but they are, in general, expected to be relevant. Contrarily, no-results pages and error pages are not likely to provide many relevant links, which implies that the crawler must be able to discard them.

The crawler is based on the algorithm in Program 6.1. It takes the URL of a page with a keyword-based search form as input and outputs a hub-set. We assume that the user has set the following constants before executing this algorithm:  $M$ , which refers to the number of hubs the algorithm is expected to return,  $T$ , which refers to the maximum number of attempts that the algorithm is allowed to make in order to gather  $M$  hubs,  $L$ , which refers to the minimum size that a word must have to be chosen as a keyword, and  $N$ , which refers to the number of keywords that we select from each page. We formally define these constants as follows:

$$\mid M, T, L, N : \mathbb{N}$$

We make the following conjecture regarding these constants (These conjectures and others that we establish throughout this dissertation are corroborated by our experimental results in Section 8.4):

**Conjecture 6.1 (Value of  $M$ )** *A relatively small number of hubs suffices to achieve a high precision and recall in our proposal.*

**Conjecture 6.2 (Value of  $T$ )** *A relatively small number of attempts suffices to gather  $M$  hubs.*

**Conjecture 6.3 (Value of  $L$ )** *Discarding words with a size lower than  $L$  suffices to gather enough non-stop words to be used as keywords.*

**Conjecture 6.4 (Value of  $N$ )** *A relatively small number of keywords from each page suffices to gather  $M$  hubs.*

The algorithm first downloads the page with the search form at line 12, and then computes an initial set of keywords at line 14. These keywords are

the  $N$  less frequent words in the search page whose size is greater than  $L$ ; this very often prevents us from returning typical stop words without committing to a language-specific dictionary. The loop at lines 15–30 iterates as long as the maximum number of attempts has not been reached, the set of keywords is not empty, and we have not collected the desired number of hubs.

The inner loop at lines 16–26 selects a keyword from the set of keywords we have computed previously, and uses it to fill in and submit the search form, which results in a web page, which is added to the set of potential hubs  $R$ . Next, we update the set of keywords at line 25 by analysing the result page using the same procedure that we described previously. Note that we keep set  $P$  updated with the keywords that are used to fill in the search form, so that we avoid using the same keyword more than once.

The inner loop finishes either because no more keywords are available or set  $R$  has the desired number of potential hubs. There can be keywords that return a so-called empty hub, which is a term that we use to designate no-results or error pages. Since we wish our technique to be independent from the web site to which it is applied, we use a simple technique that has proven to work well in practice [128]: we discard the pages in  $R$  whose number of links can be considered a lower outlier, which we implement at line 28. As a result, the collection of pages collected in the inner loop might not contain as many useful hubs as expected, which is the reason why we need to re-start the outer loop in order to make a new attempt to find new potential hubs. Note that this loop must eventually terminate, since we define a limit  $T$  to the maximum number of attempts that we make to gather the desired number of pages. If we cannot collect enough pages after  $T$  attempts, then we would have to discard the web site being analysed.

Finally, the algorithm computes a hub from each page in  $R$  at line 33, which consists of tokenising the URLs found in the links from the page and transforming them into patterns.

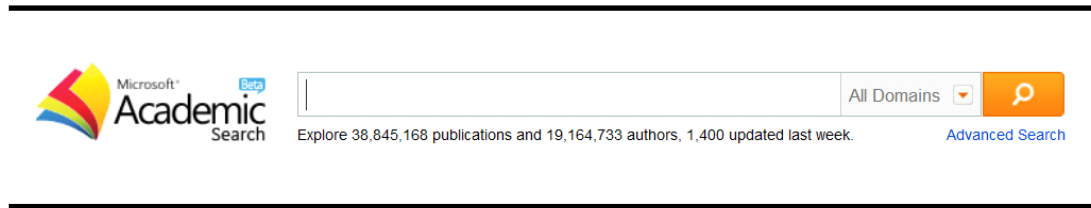
In our running example, we set  $M = 100$  and gathered 100 hubs in 5 attempts, one of which is illustrated in Figure 6.4.

In the following subsections, we describe the ancillary functions on which the algorithm relies.

### 6.3.1 Computing keywords

Function `computeKeywords` takes a web page and a set of words as input, and returns a set with the  $N$  least frequent words in the page that are not





**Figure 6.5:** Search form page in our running example.

included in the set and whose size is greater than  $L$ . We formally define this function as follows:

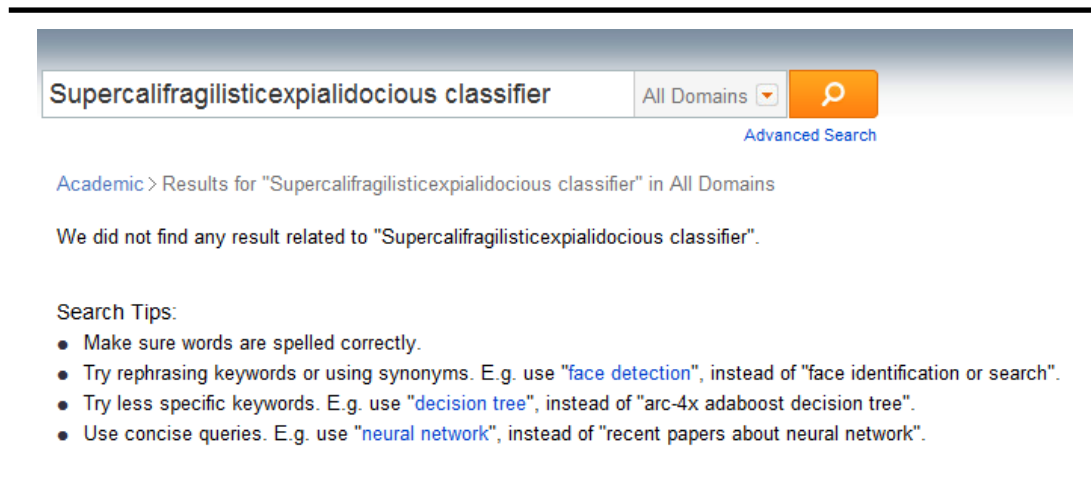
$$\begin{array}{|l} \text{computeKeywords} : \text{WebPage} \times \text{seq Word} \rightarrow \text{seq Word} \\ \hline \forall \text{wp} : \text{WebPage}, P : \text{set Word} \bullet \\ \quad \text{let } K == \text{subseq}((\text{sortBag } \text{computeWords}(\text{wp}) \setminus P), 1, N) \bullet \\ \quad \text{computeKeywords}(\text{wp}, P) = \{k : \text{Word} \mid k \in K \wedge \#k \geq L\} \end{array}$$

To compute the frequencies of each word in  $\text{wp}$ , we compute the bag of words using function `computeWords`, and then we sort them according to their frequency using function `sortBag`. Note that  $P$  denotes the set of processed keywords in Algorithm `gatherHubset` and that it is passed on to this function as a parameter to check if any of the  $N$  least frequent keywords has already been used. This way, only the keywords that have not been considered before and have a lower frequency are considered.

In our running example, we set  $L = 3$  and gathered the following keywords from the form page, cf. Figure 6.5: `authors`, `Advanced`, `Search`, `publications`, `last`, `updated`, `Explore`, `week`, and `and`, all of which appeared once in the page. Then, when we used `authors` as a keyword, and we gathered the following keywords from the resulting hub, cf. Figure 6.4: `Welch`, `Environment`, `authority`, `Group`, `Miller`, `Kesselman`, `Tuecke`, `appendix`, `assumptions`, and `Pearlman`, all of which appear once in the page, whereas the other words have a higher frequency, such as `authorization` (5), `Computer` (2), `system` (6), or `Citations` (10), to mention a few examples.

### 6.3.2 Discarding empty hubs

Empty hubs are usually very similar: they display an image and/or a message to inform the user that his or her query did not produce any results, that it was incorrect, or that an unrecoverable error happened, and, optionally, a few links to recommended items. This implies that their size, in terms



**Figure 6.6:** *Empty hub in the running example.*

of number of patterns, tends to be smaller than usual and that they can be discarded by looking for lower outliers. We formally define this function as follows:

$$\begin{array}{|l} \text{getNonEmptyHubs} : \text{Hubset} \rightarrow \text{Hubset} \\ \hline \forall \text{hs} : \text{Hubset} \mid \text{hs} \neq \emptyset \bullet \\ \quad \text{let } t == \text{lowerThreshold}(\{\text{h} : \text{Hub} \mid \text{h} \in \text{hs} \bullet \#\text{h}\}) \bullet \\ \quad \text{getNonEmptyHubs}(\text{hs}) = \{\text{h} : \text{Hub} \mid \text{h} \in \text{hs} \wedge \#\text{h} \geq t\} \end{array}$$

In our running example, non-empty hubs have sizes of around 130 patterns, whereas empty hubs have sizes around 19 patterns. If we have a mean size of 130.43 and a standard deviation of 25.43, choosing  $\alpha = 0.05$  we get a lower threshold of 19.58, which leaves out every empty hub. Figure 6.6 illustrates an empty hub from our running example that is returned as a response to query “supercalifragilisticexpialidocious classifier”. It contains 5 patterns that result from transforming the links with anchor texts Advanced Search, Academic, face detection, decision tree, and neural network.

### 6.3.3 Other ancillary functions

Our crawler relies on a few more functions that are straightforward, but difficult to present within a formal framework. We describe them below:

`download` : URL  $\rightarrow$  WebPage. This function takes a URL as input and returns a copy of the document therein located. A typical web page

usually requires a number of resources so that it can be rendered, e.g., images, style sheets, or scripts. These resources are not necessary at all in our proposal, so this function can discard them safely and return the HTML text only.

`computeWords` : `WebPage`  $\rightarrow$  `bag Word`. This function takes a web page as input and returns a bag with the words in that page, excluding tags, scripts, and in-line style definitions. In order to keep our proposal language-independent, we used the following unicode regular expression to implement this function:  $((\backslash\{L\}\backslash\{N\}+)\backslash\{N\}+\backslash\{L\})\backslash\{L\}+$ , where  $\backslash\{L\}$  is a regular expression that represents unicode characters that are letters and  $\backslash\{N\}$  unicode characters that are numbers.

`submit` : `WebPage`  $\times$  `Word`  $\rightarrow$  `WebPage`. This function takes a web page with a keyword-based search form, and a word as input. It is responsible for finding the keyword-based search form, filling its unique text field using the word, submitting it, and returning the response hub page.

`computePatterns` : `WebPage`  $\rightarrow$  `Hub`. This function takes a web page as input and returns a hub with the patterns in that page, which are extracted from its links. Pattern extraction is supported by a tokeniser based on RFC 3986 recommendation for URIs

## 6.4 Analysis

In the following subsections, we analyse the upper bounds to the worst-case complexity of the previous ancillary functions and the algorithm `gatherHubset`, to prove that they are computationally tractable. In our proofs, we assume that simple arithmetical and set operations can be implemented in  $O(1)$  time with respect to the other operations. Furthermore, we assume that  $O(n \log n)$  is an upper bound to the worst case complexity of the algorithm used to sort a set or a bag, e.g., using Merge Sort. Finally, we assume that network operations such as downloading a page or submitting a form can be implemented in  $O(1)$  time since the operating system sets a timeout to the maximum time a user can wait on these operations.

### 6.4.1 Ancillary functions

**Proposition 6.1** (`computeKeywords`) *Let  $w_p$  be a web page and  $P$  be a set of words. Function `computeKeywords`( $w_p, P$ ) terminates in  $O(k \log k)$  in the*

worst case, where  $k$  denotes the size of the largest web page in a web site. The size of a web page is measured in words.

**Proof** Function `computeKeywords` has to chunk the web page into a bag of words  $Q$  using function `computeWords`. We assume that delimiting the words in a web page can be implemented in  $O(k)$  time, where  $k$  denotes the size of a page in tokens. Then, it has to sort the words in  $Q$  according to their frequency, which can be accomplished in  $O(k \log k)$ . Then, it must compute the difference between the former ordered bag and the input set  $P$ , which can be accomplished in  $O(1)$  time. Then, it must compute the subsequence of the first  $N$  keywords, which is a basic operation that terminates in  $O(1)$  time. As a conclusion, `computeKeywords(wp, P)` terminates in  $O(k \log k)$  in the worst case.  $\square$

**Proposition 6.2** (`computePatterns`) *Let  $hp$  be a web page. Function `computePatterns(hp)` terminates in  $O(k)$  time in the worst case, where  $k$  denotes the size of the largest web page in a web site. The size of a web page is measured in words.*

**Proof** Function `computePatterns` scans  $hp$  to find HTML anchors, and retrieve their URLs. Therefore, its execution time depends on the size of  $hp$ . As a conclusion, `computePatterns(hp)` terminates in  $O(k)$  in the worst case.  $\square$

**Proposition 6.3** (`getNonEmptyHubs`) *Function `getNonEmptyHubs(hs)` terminates in  $O(k)$  in the worst case, where  $hs$  denotes a hubset and  $k$  denotes the size of  $hs$ .*

**Proof** Function `getNonEmptyHubs` has to compute the lower outliers from a set of values that represent the sizes of the hubs in  $hs$ , for which it calls function `lowerThreshold`. This function calculates the mean and standard deviation of a distribution of values, both of which operations terminate in  $O(k)$ , being  $k$  the size of the distribution. Then, the function has to iterate through  $hs$  to discard the hubsets whose size  $s$  smaller than the threshold, which requires  $O(k)$  time, as well. As a conclusion, `getNonEmptyHubs(hs)` terminates in  $O(k)$  time in the worst case.  $\square$

## 6.4.2 Algorithm

**Theorem 6.1** (`gatherHubset`) *Let  $u$  be a URL that references a web page with a keyword-based search form.  $O(n \log n)$  is an upper bound to the worst-case time complexity of Algorithm `gatherHubset(u)`, where  $n$  denotes the size of the largest web page in the web site being analysed. The size of a page is measured in words.*

**Proof** Algorithm `gatherHubset` first downloads a web page, which requires  $O(1)$  time. It then computes keywords from the web page contents. According to Proposition 6.1,  $O(n \log n)$  is an upper bound to the time required to extract the keywords, where  $n$  denotes the size of the largest page in a site (measured in words). Then, it has to iterate through lines 15–30 a maximum of  $T$  times. In each iteration, the inner loop at lines 16–26 iterates until there are not any keywords in set  $Q$  or we have gathered the maximum number of hubs, which means  $M$  iterations in the worst case. Inside the inner loop, the algorithm performs different set operations and a form is submitted to the server, which terminates in  $O(1)$  time. Afterwards, new keywords are computed from the contents of the page and added to the keywords set, which terminates in  $O(n \log n)$  time according to Proposition 6.1. Therefore, the execution time for the inner loop is  $O(M(1 + n \log n))$ . After the inner loop, the algorithm has to discard empty hubs, which terminates in  $O(k)$  time, according to Proposition 6.3, where  $k$  is the size of the input set. In the worst case,  $hs$  has the maximum size  $M$ , so it terminates in  $O(M)$  time. Therefore,  $O(TM(n \log n + 1))$  is an upper bound to the worst-case time the outer loop requires to execute. Finally, for each web page in the potential hubs set a new hub is created by computing its patterns, which terminates in  $O(n)$  time according to Proposition 6.2. Since the potential hubs set size is  $M$  in the worst case, the execution time for the last loop is  $O(nM)$ .

Therefore,  $O(n \log n + T(M(n \log n + 1) + M) + nM)$  is an upper bound to the time required by `gatherHubset(u)` to terminate. Since  $T$  and  $M$  denote constants, we can simplify this upper limit to  $O(n \log n)$ , which concludes the proof.  $\square$

## 6.5 Summary

In this chapter, we have described our crawler, which is responsible for gathering a set of hubs from a web site, by filling in forms using keywords computed from the site itself. This hubset is then passed onto the pattern builder to build the patterns for URL classification. We have proved that our proposal is computationally tractable.



---

## Chapter 7

# Our pattern builder

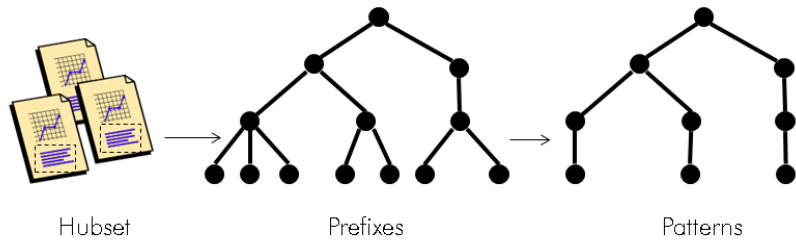
---

There is in all things a pattern that is part of our universe. It has symmetry, elegance, and grace - those qualities you find always in that which the true artist captures. You can find it in the turning of the seasons, in the way sand trails along a ridge, in the branch clusters of the creosote bush or the pattern of its leaves. We try to copy these patterns in our lives and our society, seeking the rhythms, the dances, the forms that comfort.

Dune, 1965.

Frank Patrick Herbert, writer (1920-1986)

**I**n this chapter, we describe the architecture of our pattern builder and provide the algorithms and functions that implement it. The chapter is organised as follows: in Section 7.1, we introduce it; Section 6.2 describes the architecture of the crawler; Section 7.3 defines the main algorithm that orchestrates the pattern builder, and the ancillary functions that support the algorithm; Section 7.4 reports on the complexity analysis of the former algorithm and functions; finally, we summarise the chapter in Section 7.5.



**Figure 7.1:** *Our pattern builder.*

## 7.1 Introduction

In this chapter, we describe our proposal to automatically build a set of patterns that represent the URLs of the different classes of pages of a site. Our pattern builder takes a set of hubs that contain a set of initial patterns as input and uses a statistical criteria to discern which parts of the patterns should be abstracted to learn more general patterns. First, our pattern builder creates a set with every possible prefix of every pattern in the input hubset. Then, it iterates the prefixes in that set in increasing order of length, and compares each prefix with its siblings to detect which siblings do not have a significantly high frequency and should be abstracted. Then, the siblings that are not significant are wildcarded, i.e., their last element is replaced with a wildcard. The result is a set of patterns in which some of their elements are wildcards.

We use a tree notation that is based on PATRICIA trees to represent sets of prefixes and patterns; it allows to represent a large collection of patterns and prefixes compactly, cf. Figure 7.1.

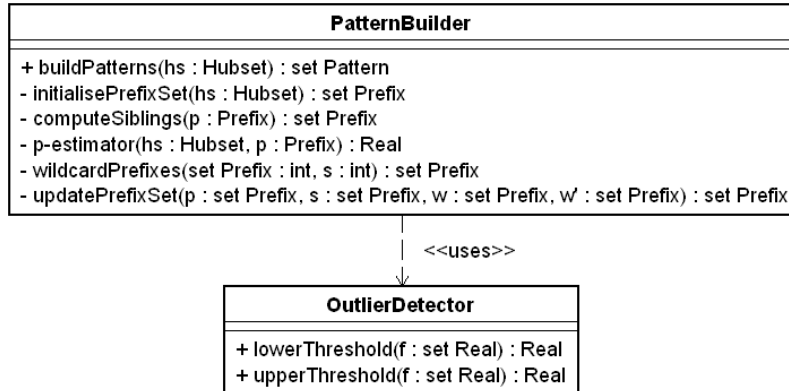
In addition, we prove that our crawler is computationally tractable, since  $O(n^3)$  is an upper bound to its worst-case time complexity, where  $n$  denotes the number of patterns in the input hubset.

Throughout this chapter, we use the same running example as in the previous one. The data types and simple functions to work with bags, sets, or sequences are formally defined in Appendix A.

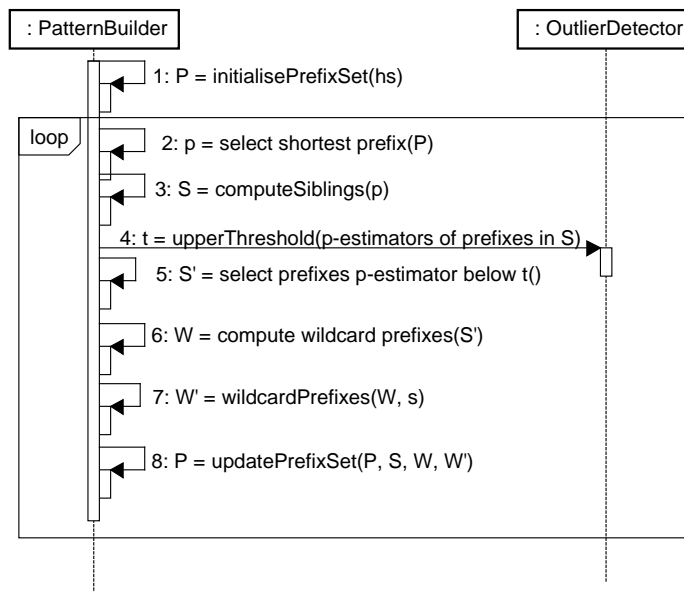
## 7.2 Architecture

The architecture of the pattern builder is presented in Figure 7.2. Class `PatternBuilder` provides the functions needed to build a set of patterns. We





**Figure 7.2:** Class diagram of our pattern builder.



**Figure 7.3:** Sequence diagram of our pattern builder.

apply an outlier-based statistical technique to build the patterns. Therefore, an outlier detection method is needed, which is provided by class `OutlierDetector`.

We illustrate how function `buildPatterns` works using a sequence dia-

gram, cf. Figure 7.3. It takes the hubset gathered by the crawler as input. First, it extracts the set of prefixes for every pattern in every hub from the hubset. Then, it iterates on the set of prefixes and computes the set of siblings of each prefix, i.e., prefixes that share a common prefix with it. Then, it computes their probability estimators building on their frequencies. Finally, prefixes with a probability estimator smaller than a threshold are wild-carded; the threshold is computed by the outlier detector. The result is a set of URL patterns that represent the URLs of the different classes of pages in the web site.

As an example, our running example provides pages with information about authors. Some examples of the URLs of this pages are `http://academic.research.microsoft.com/Author/10851937/edwards-reynolds`, `http://academic.research.microsoft.com/Author/516258/peter-j-burt`, or `http://academic.research.microsoft.com/Author/1697139/edward-h-adelson`. It is not difficult to see that all these URLs follow a common pattern, which consists of the prefix `http://academic.research.microsoft.com/Author/` followed by a number that identifies the author in the internal database of the site, and then by the author's name. Since the URLs of authors share the same prefix, its frequency is significantly high, in comparison to other prefixes. Algorithm `buildPatterns` is able to discern the prefixes whose frequency is not significant and abstract them to learn the pattern  $\langle \wedge, \text{http}, \text{://}, \text{academic.research.microsoft.com}, \text{/}, \text{Author}, \text{/}, *, \text{/}, *, \$ \rangle$ . Finally, any web page can be classified as an author page by finding a match between its URL and this pattern; otherwise, if this match cannot be found, the page is probably not an author page.

### 7.3 Algorithm

The pattern builder relies on the algorithm in Program 7.1. It takes a hubset `hs` as input and outputs a set of patterns. The algorithm starts by creating the set of prefixes `P` at line 8; this set contains every prefix in `hs` whose size is greater than one in `hs`, ordered by size. Then, the loop at lines 9–32 iterates until `P` is empty. In each iteration, the algorithm selects the shortest prefix in `P` at line 10. If the selected prefix ends with `$`, which marks the end of a pattern, it means that every prefix in the pattern has already been processed, and it can then be added to the output set of patterns at line 13. Otherwise, the prefix has to be processed along with its siblings at line 16; the siblings are the prefixes that share a common prefix with `p`, excluding its last element. Then, the algorithm computes the probability estimators of the siblings and calculates the threshold above which such estimator is considered an upper outlier at

---

```

1: algorithm buildPatterns
2: input    hs : Hubset
3: output   Result : seq Pattern
4: variables P, S, S', W, W' : seq Prefix; p, q : Prefix; pe, t : ℝ
5:
6: Result := ∅
7: – Step 1: Initialise prefix set
8: P := initialisePrefixSet(hs)
9: while P ≠ ∅ do
10:  p := shortest prefix in P
11:  if last p = $ then
12:    P := P \ {p}
13:    Result := Result ∪ {p}
14:  else
15:    – Step 2: Compute siblings
16:    S := computeSiblings(hs, p)
17:    – Step 3: Compute upper-outlying siblings
18:    t := upperThreshold({q : Prefix | q ∈ S • p-estimator(hs, q)})
19:    S' := ∅
20:    for each q ∈ S do
21:      pe := p-estimator(hs, q)
22:      if pe ≤ t ∧ pe ≈ 1.00 then
23:        S' := S' ∪ {q}
24:      end if
25:    end for
26:    – Step 4: Wildcard prefixes
27:    W := {v, w : Prefix | v ∈ S' ∧ w ∈ P ∧ v prefix w • w}
28:    W' := wildcardPrefixes(W, #p)
29:    – Step 5: Update prefix set
30:    P := updatePrefixSet(P, S, W, W')
31:  end if
32: end while

```

---

**Program 7.1:** *Algorithm* buildPatterns.

line 18. We refer to probability estimators as p-estimators for short, and they refer to the probability of finding a URL that begins with that prefix in hubs from hs. The goal of the loop at lines 20–25 is to select a subset of siblings that are not frequent enough to be wildcarded. We only select siblings that can be considered lower outliers as long as their p-estimators are not close to 1.00.

**Conjecture 7.1 (Wildcarding criterion)** *Wildcarding prefixes with a p-*

Pattern	Class
$\langle \wedge, \text{http}://, \langle \text{MSAS} \rangle, /, \text{Publication}, /, *, /, *, \$ \rangle$	Paper
$\langle \wedge, \text{http}://, \langle \text{MSAS} \rangle, /, \text{Author}, /, *, /, *, \$ \rangle$	Author
$\langle \wedge, \text{http}://, \langle \text{MSAS} \rangle, /, \text{Journal}, /, *, /, *, \$ \rangle$	Journal
$\langle \wedge, \text{http}://, \langle \text{MSAS} \rangle, /, \text{Detail}, ?, \text{entityType}, =, 1, \&, \text{searchType}, =, 5, \&, \text{id}, =, *, \$ \rangle$	Citation

**Table 7.1:** Patterns built for the running example.

*estimator that are not significantly higher than the  $p$ -estimators of their siblings and whose distance to 1.00 is higher than 0.05 is an appropriate criterion to build patterns, i.e., patterns built using this criterion allow classifying URLs achieving high precision and recall.*

The siblings that fulfill the wildcarding criterion are added to the set of frequent siblings  $S'$  for further processing at line 23. These siblings and their descendants, i.e., the prefixes that have an element of  $S'$  as a prefix, are added to the set of prefixes to be wildcarded  $W$  at line 27. Afterwards, these prefixes are wildcarded at line 28 by replacing the element at position  $\#p$  with a wildcard. Finally, the prefix set  $P$  is updated by withdrawing the prefixes that have already been processed in this iteration and adding those of their descendants that have been wildcarded at line 30.

In our running example, algorithm `buildPatterns` outputs four patterns from our running example, cf. Table 7.1, where  $\langle \text{MSAS} \rangle$  denotes domain name `academic.research.microsoft.com`:

In the following subsections, we describe the ancillary functions on which the algorithm relies.

### 7.3.1 Initialising the prefix set

Function `initialisePrefixSet` takes a hubset as input and returns a sequence with the prefixes of the patterns in flat  $hs$ , ordered in increasing order of size. We formally define this function as follows:

$$\begin{array}{|l} \text{initialisePrefixSet} : \text{Hubset} \rightarrow \text{seq Prefix} \\ \hline \forall hs : \text{Hubset} \bullet \\ \text{initialisePrefixSet}(hs) == \\ \text{sortSet}\{p : \text{Prefix} \mid \#p \geq 2 \wedge (\exists q : \text{Pattern} \mid q \in \text{flat } hs \bullet p \text{ prefix } q)\} \end{array}$$

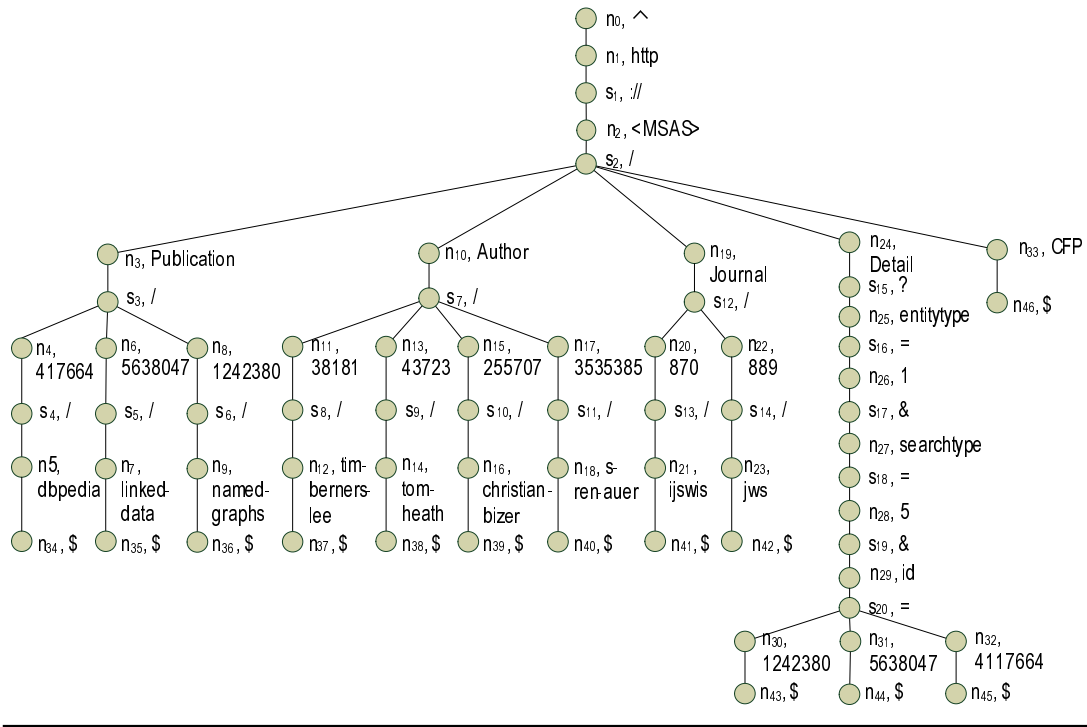


Figure 7.4: Partial view of the initial prefix set in the running example.

As an example, Figure 7.4 represents a subset of prefixes in our running example, using our tree notation. It is not possible to show the complete set since it has thousands of prefixes. Every node in the tree has a label, which we denote as  $n_i$  or  $s_i$ . Nodes labelled with  $n_i$  correspond to tokens, whereas nodes labelled with  $s_i$  correspond to separators. Each node, actually, represents a prefix, which is the sequence of tokens and separators in the path from the tree root  $n_0$  to the node itself. Note that each path from the tree root to a leaf represents a pattern. In the following examples, we refer to prefixes in the tree by means of its corresponding node.

### 7.3.2 Computing siblings

Function `computeSiblings` takes a hubset  $hs$  and a prefix  $p$  as input and returns the set of prefixes in  $flat\ hs$  that share a common prefix with  $p$  up to its penultimate element, including  $p$  itself. We formally define this function as follows:

$$\begin{array}{|l} \text{computeSiblings} : \text{Hubset} \times \text{Prefix} \rightarrow \text{seq Prefix} \\ \hline \forall \text{hs} : \text{Hubset}, \text{p} : \text{Prefix} \mid \text{hs} \neq \emptyset \wedge \#\text{p} \geq 2 \bullet \\ \text{computeSiblings}(\text{hs}, \text{p}) = \{q : \text{Prefix}; r : \text{Pattern} \mid \\ r \in \text{flat hs} \wedge q \text{ prefix } r \wedge \text{last } q \neq * \wedge (\text{front } \text{p}) \text{ prefix } q \bullet \\ \text{subseq}(q, 1, \#\text{p})\} \end{array}$$

In our running example, prefix  $n_4 = \langle \wedge, \text{http}, \text{//}, \langle \text{MSAS} \rangle, /, \text{Publication}, /, 4117664 \rangle$  has siblings  $n_6 = \langle \wedge, \text{http}, \text{//}, \langle \text{MSAS} \rangle, /, \text{Publication}, /, 5638047 \rangle$  and  $n_8 = \langle \wedge, \text{http}, \text{//}, \langle \text{MSAS} \rangle, /, \text{Publication}, /, 1242380 \rangle$ . In Figure 7.4, it is easy to find the siblings of a prefix represented by a node  $n_i$  by looking for nodes that share an ancestor with  $n_i$ .

### 7.3.3 Computing p-estimators

Function p-estimator takes a hubset  $hs$  and a prefix  $p$  as input and returns an estimator of the probability of finding at least one pattern prefixed by  $p$  in a hubset. Since the underlying distribution of prefixes is unknown and heavily dependent on the web site being analysed, we can compute the p-estimator as the relative frequency of every prefix in  $hs$  [56]. We formally define this function as follows:

$$\begin{array}{|l} \text{p-estimator} : \text{Hubset} \times \text{Prefix} \rightarrow \mathbb{R} \\ \hline \forall \text{hs} : \text{Hubset}, \text{p} : \text{Prefix} \mid \text{hs} \neq \emptyset \bullet \\ \text{let } H == \{h : \text{Hub} \mid h \in \text{hs} \wedge (\exists q : \text{Pattern} \bullet q \in h \wedge p \text{ prefix } q)\} \bullet \\ \text{p-estimator}(\text{hs}, \text{p}) = \#H/\#\text{hs} \end{array}$$

P-estimators range from 0.00 to 1.00. The more frequent a prefix, the higher its corresponding p-estimator. We state the following conjecture regarding p-estimators:

**Conjecture 7.2 (Distribution of p-estimators)** *P-estimators that are not near 1.00 are most probably near  $1/\#\text{hs}$ , whose limit is 0.00 as the number of hubs increases. In other words, the distribution of p-estimators has two peaks at 0.00 and 1.00.*

In our running example, the p-estimators computed for the prefixes in Figure 7.4 are shown in Table 7.2. As an example, prefix  $n_3 = \langle \wedge, \text{http}, \text{//}, \langle \text{MSAS} \rangle, /, \text{Publication} \rangle$  appears in every hub in  $hs$  since this prefix refers to publications and every hub in our running example includes links to publications. Therefore, its p-estimator is 1.00. Every hub

Node	Last element	P-estimator	Node	Last token	P-estimator
n <sub>1</sub>	http	1.00	s <sub>1</sub>	://	1.00
n <sub>2</sub>	<MSAS>	1.00	s <sub>2</sub>	/	1.00
n <sub>3</sub>	Publication	0.99	s <sub>3</sub>	/	1.00
n <sub>4</sub>	4117664	0.01	s <sub>4</sub>	/	1.00
n <sub>5</sub>	dbpedia	0.01	n <sub>6</sub>	5638047	0.01
s <sub>5</sub>	/	1.00	n <sub>7</sub>	linked-data	0.01
n <sub>8</sub>	1242380	0.01	s <sub>6</sub>	/	1.00
n <sub>9</sub>	named-graphs	0.01	n <sub>10</sub>	Author	0.99
s <sub>7</sub>	/	1.00	n <sub>11</sub>	38181	0.01
s <sub>8</sub>	/	1.00	n <sub>12</sub>	tim-berners-lee	0.01
n <sub>13</sub>	43723	0.02	s <sub>9</sub>	/	1.00
n <sub>14</sub>	tom-heath	0.02	n <sub>15</sub>	255707	0.01
s <sub>10</sub>	/	1.00	n <sub>16</sub>	christian-bizer	0.01
n <sub>17</sub>	3535385	0.01	s <sub>11</sub>	/	1.00
n <sub>18</sub>	s-ren-auer	0.01	n <sub>19</sub>	Journal	0.89
s <sub>12</sub>	/	1.00	n <sub>20</sub>	870	0.01
s <sub>13</sub>	/	1.00	n <sub>21</sub>	ijswis	0.01
n <sub>22</sub>	889	0.01	s <sub>14</sub>	/	1.00
n <sub>23</sub>	jws	0.01	n <sub>24</sub>	Detail	1.00
s <sub>15</sub>	?	1.00	n <sub>25</sub>	entityType	1.00
s <sub>16</sub>	=	1.00	n <sub>26</sub>	1	1.00
s <sub>17</sub>	&	1.00	n <sub>27</sub>	searchType	1.00
s <sub>18</sub>	=	1.00	n <sub>28</sub>	5	1.00
s <sub>19</sub>	&	1.00	n <sub>29</sub>	id	1.00
s <sub>20</sub>	=	1.00	n <sub>30</sub>	1242380	0.01
n <sub>31</sub>	5638047	0.01	n <sub>32</sub>	4117664	0.01
n <sub>33</sub>	CFP	0.17			

Table 7.2: *P*-estimators in our running example.

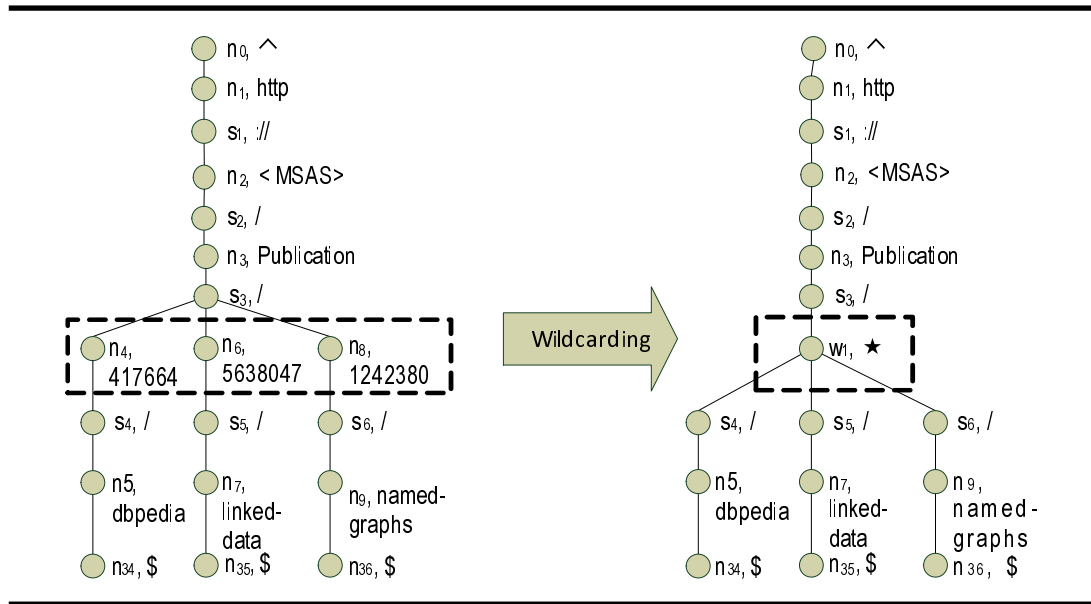


Figure 7.5: Wildcarding example.

includes a list of publications, and each publication has at least one author, which means that we can find at least one pattern prefixed by  $n_3$  in every hub. Contrarily, some publications are published in journals and some others are published in conferences, which means that only some of the hubs contain patterns prefixed by  $n_{19}$ . Therefore, the p-estimator of prefix  $n_{19} = \langle \wedge, \text{http}, //, \langle \text{MSAS} \rangle, /, \text{Journal} \rangle$  is not 1.00, but 0.89 in this case; this estimator is significantly high, but not as high as the p-estimator of prefix  $n_3$ .

### 7.3.4 Wildcarding prefixes

Function `wildcardPrefixes` takes a set of prefixes and a natural number  $i$  as input, and it returns a set in which the input prefixes have been wildcarded at the  $i$ -th position. We formally define this function as follows:

$$\begin{array}{|l} \text{wildcardPrefixes} : \text{seq Prefix} \times \mathbb{N} \rightarrow \text{seq Prefix} \\ \hline \forall C : \text{seq Prefix}; i : \mathbb{N} \mid i \geq 1 \bullet \\ \text{wildcardPrefixes}(C, i) = \{p : \text{Prefix} \mid p \in C \wedge \#p \geq 2 \wedge i \leq \#p \bullet \\ \text{subseq}(p, 1, i-1) \frown \langle \star \rangle \frown \text{subseq}(p, i+1, \#p)\} \end{array}$$

In our running example, the p-estimator of prefix  $n_4 = \langle \wedge, \text{http}, //, \langle \text{MSAS} \rangle, /, \text{Publication}, /, 4117664 \rangle$  and its siblings  $n_6 = \langle \wedge, \text{http}, //, \langle \text{MSAS} \rangle, /, \text{Publication}, /, 5638047 \rangle$  and  $n_8 = \langle \wedge, \text{http}, //, \langle \text{MSAS} \rangle, /, \text{Publication}, /, 1242380 \rangle$  is not 1.00, but 0.89 in this case; this estimator is significantly high, but not as high as the p-estimator of prefix  $n_3$ .



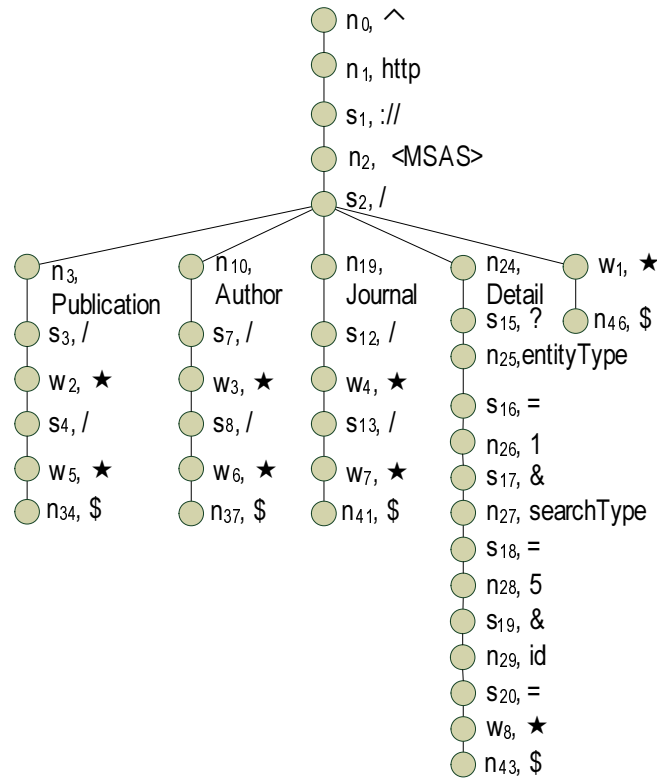


Figure 7.6: Prefixes in the running example after building patterns.

$/, 5638047$ ) and  $n_8 = \langle ^, \text{http}, ://, \text{<MSAS>, /, Publication, /, 1242380} \rangle$  is 0.01 according to Table 7.2. Since they all have the same value, their mean value is 0.01, and their standard deviation is 0.00. Therefore, the upper threshold is 0.01 in this case. Since none of the siblings has a p-estimator higher than the threshold, they are added to the set of prefixes to be wildcarded, including the prefixes that have them as a prefix, namely  $s_4, s_5, s_6, n_5, n_7, n_9, n_{34}, n_{35}$ , and  $n_{36}$ . Figure 7.5 shows the siblings of  $n_4$  and their descendants after wildcarding them. Note that we change the name of node  $n_4$  to  $w_1$  to emphasise the fact that the last tokens in prefixes  $n_4, n_6$ , and  $n_8$  have been replaced with a wildcard, i.e., they three have become a single new prefix that is represented by node  $w_1$ . The descendants of former prefixes  $n_4, n_6$ , and  $n_8$  now share the same prefix  $w_1$ .

Figure 7.6 presents the tree with the resulting prefix set after executing algorithm `buildPatterns` on the running example. Some the prefixes in the tree are the result of wildcarding one or more of the original prefixes using func-

tion `wildcardPrefixes`. Note that every path from the tree root to a leaf represents a different pattern.

### 7.3.5 Updating the prefix set

Function `updatePrefixSet` takes four set of prefixes as input:  $P$ ,  $S$ ,  $W$ , and  $W'$ . In each iteration of algorithm `buildPatterns` these sets represent, respectively, the ordered prefix set in algorithm `buildPatterns`, the set of siblings that have been processed, the set of prefixes to be wildcarded, and another set in which the prefixes have already been wildcarded. The function returns a prefix set that is the result of updating  $P$  by replacing the prefixes to be wildcarded with their wildcarded version, and subtracting the sibling prefixes that have been processed in that iteration. We formally define this function as follows:

$$\begin{array}{|l} \text{updatePrefixSet} : \text{seq Prefix} \times \text{seq Prefix} \times \text{seq Prefix} \rightarrow \text{seq Prefix} \\ \hline \forall P, S, W, W' : \text{seq Prefix} \bullet \\ \text{updatePrefixSet}(P, S, W, W') = \\ \text{sortSet}((P \setminus W \setminus S) \cup \{q : \text{Prefix} \mid q \in W' \setminus S\}) \end{array}$$

## 7.4 Analysis

In the following subsections, we analyse the worst-case complexity of the previous ancillary functions and algorithm `buildPatterns`. In our proofs, we make the same assumptions regarding arithmetical and set operations as in the previous chapter, i.e., that they can be implemented in  $O(1)$  time with respect to the other operations. Likewise, we assume that  $O(k \log k)$  is an upper bound to the worst case complexity of the algorithm used to sort a set of size  $k$ , e.g., using Merge Sort.

### 7.4.1 Ancillary functions

**Proposition 7.1** (`initialisePrefixSet`) *Let  $hs$  be a hubset. The size of the set of every possible prefix in the patterns of `flat hs` is  $m n$  in the worst case, where  $m$  and  $n$  denote the number of patterns and the maximum size of a pattern in `flat hs`, respectively.*

**Proof** Function `initialisePrefixSet(hs)` computes a new prefix for each subsequence of tokens in each pattern in `flat hs`, starting from the beginning of

each pattern. Therefore, for each pattern the function computes as many prefixes as the size of the pattern. In the worst case, all subsequences are different (i.e., there does not exist two patterns with a common subsequence), so the total number of prefixes is the sum of all of the patterns sizes. The worst case happens when every pattern has the maximum length, which we denote as  $m$ . As a conclusion, the number of prefixes is  $m \cdot n$  in the worst case. Finally, the function has to order this set, which can be implemented in  $O(k \log k)$  time, where  $k$  is the size of the set. As a conclusion, `initialisePrefixSet` terminates in  $O(m \cdot n + m \cdot n \log(m \cdot n))$  time in the worst case.  $\square$

**Proposition 7.2 (computeSiblings)** *Let  $hs$  be a hubset and  $p$  be a prefix of a pattern in `flat hs`. `computeSiblings(hs, p)` terminates in  $O(k)$  time in the worst case, where  $k$  denotes the number of patterns in `flat hs`.*

**Proof** Function `computeSiblings(hs, p)` has to iterate through the set of patterns of `flat hs` and check whether they have a common prefix with  $p$  up to its penultimate token. Therefore, it has to process each of the  $k$  patterns of `flat hs`. As a conclusion, function `computeSiblings(hs, p)` terminates in  $O(k)$  time in the worst case.  $\square$

**Proposition 7.3 (Number of siblings)** *Let  $hs$  be a hubset and  $p$  be a prefix of a pattern in `flat hs`. `computeSiblings(hs, p)` has  $k$  prefixes in the worst case, where  $k$  denotes the number of patterns in `flat hs`.*

**Proof** Function `computeSiblings(hs, p)` has to iterate through the set of patterns of `flat hs` and check whether they have a common prefix with  $p$  up to its penultimate token. In the worst case, every pattern in `flat hs` shares that common prefix. As a conclusion, function `computeSiblings(hs, p)` returns  $k$  prefixes in the worst case.  $\square$

**Proposition 7.4 (upperThreshold)** *Let  $R$  be a set of real numbers and let  $k = \#R$ . Function `upperThreshold(R)` terminates in  $O(k)$  time in the worst case.*

**Proof** The function has to compute the mean and standard deviation of  $R$ , which we can safely assume terminate in  $O(k)$  time in both cases. Then, it has to perform several arithmetical operations, which terminate in  $O(1)$  time. As a conclusion, function `upperThreshold` terminates in  $O(k)$  time, in the worst case.  $\square$

**Proposition 7.5 (p-estimator)** *Let  $hs$  be a hubset, and  $p$  be a prefix of a pattern in `flat hs`. Function `p-estimator(hs, p)` terminates in  $O(k)$  time in the worst case, where  $k$  denotes the number of patterns in `flat hs`.*

**Proof** Function `p-estimator(hs, p)` has to calculate the ratio of hubs in flat `hs` that have at least one pattern whose prefix is `p`. It has to iterate through every hub in `hs`. In each hub, it must find at least one pattern of which `p` is a prefix. In the worst case, the function needs to iterate through the complete set of patterns of flat `hs`. As a conclusion, this function terminates in  $O(k)$  time in the worst case.  $\square$

**Proposition 7.6** (`wildcardPrefixes`) *Let  $S'$  be a set of prefixes, and  $i \in \mathbb{N}$  be a number. Function `wildcardPrefixes( $S', i$ )` terminates in  $O(k)$  time in the worst case, where  $k$  denotes the size of  $S'$ .*

**Proof** Function `wildcardPrefixes` must iterate through the  $k$  elements in  $S'$ . In each iteration, the function concatenates three subsequences, which can be implemented in  $O(1)$  time. As a conclusion, `wildcardPrefixes` terminates in  $O(k)$  time, in the worst case.  $\square$

**Proposition 7.7** (`updatePrefixSet`) *Let  $P, S, W$  and  $W'$  be sets of prefixes, and  $k$  be the maximum size of set  $P$ . Function `updatePrefixSet( $P, S, W, W'$ )` terminates in  $O(k \log k)$  time in the worst case.*

**Proof** Function `updatePrefixSet` must update the prefix set  $P$  by subtracting a number of elements in  $S$  and  $W$ , and adding the elements in  $W'$ . According to Proposition 7.8, after this update,  $P$  has decreased its size in at least one element, i.e., it has size  $k - 1$  in the worst case. The former set operations can be implemented in  $O(1)$  time. Finally, the function sorts  $P$ , which can be implemented in  $O((k - 1) \log(k - 1)) \subseteq O(k \log k)$  time. As a conclusion,  $O(k \log k)$  is an upper bound to the worst-case time complexity of `updatePrefixSet( $P, S, W, W'$ )`.  $\square$

## 7.4.2 Algorithm

**Proposition 7.8** (**Iterations in algorithm `buildPatterns`**) *Let  $hs$  be a hubset,  $n = \#\text{flat } hs$ , and  $m$  the size of the longest pattern in flat  $hs$ . Algorithm `buildPatterns(hs)` terminates in  $m n$  iterations in the worst case.*

**Proof** Algorithm `buildPatterns` has to iterate until the whole prefix set  $P$  is empty. Initially,  $P$  has  $n m$  elements in the worst case, according to Proposition 7.1; in each iteration, a prefix `p` is removed from  $P$  and processed. On the one hand, if the last element of `p` is `$` (lines 12 and 13), then one element is withdrawn from  $P$  and no elements are added, so  $P$  decreases its size in one element. On the other hand, if the last element of `p` is not `$`,  $P$  it is modified by function `updatePrefixSet`, at line 30. We can distinguish between two cases:

**Case 1:** If the set of prefixes to be wildcarded  $W$  is empty, no elements are added to  $P$  at line 30.  $W$  can be empty when none of the siblings of  $p$  that are being analysed have a significantly high  $p$ -estimator. The worst case is that  $p$  has no siblings, so the set of siblings  $S$  contains only  $p$  (recall that every prefix is a sibling of itself). Therefore, in this case only one element is withdrawn from  $P$  at line 30.  $W'$  has the same size as  $W$ , since the wildcarding function at line 28 creates a new wildcarded prefix for each prefix in the original set. Therefore,  $W'$  is also empty, and no elements are added to  $P$ .

**Case 2:** If  $W$  is not empty, let  $k = \#W$ . The worst case is that the set of siblings  $S$  has size 1 (as we proved before), so a total of  $k + 1$  elements are withdrawn from  $P$  at line 28. The size of  $W'$  is  $k$ , as seen before. The prefixes in  $W'$  that have not been processed in this iteration ( $W' \setminus S'$ ) are added to  $P$  at line 28. An upper bound to the number of elements in  $W' \setminus S'$  is the size of  $W$ , in which case we add  $k$  elements to  $P$ . Therefore, at the end of the iteration,  $P$  has decreased its size in  $(k + 1) - k = 1$  element.

As a conclusion, in each iteration the size of  $P$  decreases in at least one prefix. Since  $P$  is a set whose initial size is  $n m$ , we can conclude that, in the worst case, the main loop of algorithm `buildPatterns` (lines 9–32) iterates a maximum of  $n m$  times.  $\square$

**Theorem 7.1** (`buildPatterns`) *Let  $hs$  be a hubset.  $O((m + 2 m^2) n^3 + (3 m + m^2 \log(m n)) n^2 + (2 m + m \log n) n)$  is an upper bound to the worst-case time complexity of Algorithm `buildPatterns(hs)`, where  $n$  denotes the number of patterns in flat  $hs$ , and  $m$  denotes the size of the longest pattern.*

**Proof** Algorithm `buildPatterns` starts by initialising the set of prefixes in flat  $hs$ , which terminates in  $O(m n + m n \log(m n))$  time in the worst case according to Proposition 7.1. Then, it has to iterate until the prefix set  $P$  is empty (lines 9–32). In the worst case, that means  $m n$  iterations, according to Proposition 7.8. In each iteration, the worst case is that the last token in the prefix is not  $\$$ , so that it has to perform the following steps:

- Select the shortest prefix in the prefix set  $P$ , which terminates in  $O(1)$  time.
- Compute the siblings of a prefix at line 16, which terminates in  $O(n)$  time in the worst case according to Proposition 7.2.

- Calculate the threshold for p-estimators that are upper outliers at line 18. The algorithm calls function `upperThreshold` on the set of p-estimators of the prefixes in the set of siblings  $S$ . First, the algorithm has to call function `p-estimator` to compute the p-estimator of each prefix in  $S$ , which has  $n$  prefixes in the worst case. Function `p-estimator` terminates in  $O(n)$  time in the worst case according to Proposition 7.5. Therefore, the computation of p-estimators terminates in  $O(n^2)$  time in the worst case. Then, function `upperThreshold` terminates in  $O(n)$  time according to Proposition 7.4, since  $n$  is the size of the set of siblings  $S$  in the worst case according to proposition 7.3. Therefore, the computation of the threshold terminates in  $O(n^2 + n)$  time in the worst case.
- For each prefix in the set of siblings, the algorithm has to check if the p-estimator of the prefix is lower than the threshold and, in that case, add the prefix to the set of non-frequent siblings  $S'$  at line 23. Since the p-estimators of each prefix were already computed in the previous step, it is not necessary to compute them again, so we assume that the operations in the loop at lines 20–25 terminate in  $O(1)$  time, and the loop itself terminates then in  $O(n)$  time in the worst case.
- Then, the algorithm computes the set of prefixes that must be wildcarded ( $W$ ) at line 27. The algorithm compares each element in  $S'$  to each element in  $P$  to check if the former is a prefix of the latter. Since  $S'$  is a subset of  $S$  and  $S$  has a maximum size of  $n$ ,  $S'$  has a maximum size of  $n$ , in the worst case. In addition,  $P$  has a maximum size of  $m n$  according to Proposition 7.1, so the total number of elements in  $W$  is the cartesian product of the two sets, i.e.,  $m n^2$ .
- Then, the elements in  $W$  are wildcarded at line 28, which terminates in  $O(k)$  time according to Proposition 7.6, where  $k$  is the size of the input set. The size of  $W$  is always less or equal than  $m n^2$  as we proved in the previous step, so the execution of `wildcardPrefixes(W, i)` terminates in  $O(m n^2)$  time in the worst case.
- Finally, the set of prefixes  $P$  is updated by function `updatePrefixSet` at line 30, which terminates in  $O(k \log k)$  time in the worst case according to Proposition 7.7, where  $k$  is the maximum size of  $P$ . Therefore, `updatePrefixSet` terminates in  $O((m n) \log(m n))$  time in the worst case.

Therefore,  $O(m n + m n \log(m n) + m n(1 + n + n^2 + n + n + m n^2 + m n^2 + (m n) \log(m n))) = O((m + 2 m^2) n^3 + (3 m + m^2 \log(m n)) n^2 + (2 m +$

$m \log n) n)$  is an upper bound to the worst-case time complexity of Algorithm `buildPatterns`.  $\square$

**Corollary 7.1** *Let  $hs$  be hubset.  $O(n^3)$  is an upper bound to the worst-case time complexity of Algorithm `buildPatterns`( $hs$ ), where  $n$  denotes the number of patterns in  $hs$ .*

**Proof** Since the size of an URL is always significantly smaller than the number of URLs in a hub page, i.e.,  $m \ll n$ , we can conclude that the time complexity is  $O((m + 2m^2)n^3 + (3m + m^2 \log(mn))n^2 + (2m + m \log n)n) \subseteq O(3n^3 + n^2 \log n + n \log n) \subseteq O(n^3)$  is an upper bound to the worst-case time complexity of the algorithm.  $\square$

## 7.5 Summary

In this chapter, we have described our pattern builder, which is responsible for building a set of patterns that represent the URLs of a web site. We use a statistical approach to discern which parts of a URL are significant and should be a part of the pattern, and which parts are not significant and can be abstracted. We have proved that our proposal is computationally tractable.





---

# Chapter 8

## Evaluation

---

*Cleverly designed experiments are the key.  
Wonder and Skepticism, 1995.*

*Carl Edward Sagan, astrophysicist (1934–1996)*

**I**n this chapter, we present the evaluation of our proposal and corroborate the conjectures on which it is based. It is organised as follows: in Section 8.1, we introduce it; Section 8.2 describes the experimental design, the results of the experiments, and draws some intuitive conclusions from them; Section 8.3 presents some statistical tests that corroborate our conclusions; Section 8.4 corroborates the conjectures on which our proposal is based; finally, we summarise the chapter in Section 8.5.

## 8.1 Introduction

The goals of our evaluation are threefold: first, we aim to prove that our technique is able to classify web pages with good precision and recall, with regard to other baseline classification techniques; second, we aim to prove that our technique is very efficient with regard to other techniques; finally, we aim to corroborate the conjectures on which our proposal relies.

## 8.2 Experimental evaluation

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 first describe our experimentation environment, then we present the experimental results, and finally we corroborate this results with a number of statistical tests.

### 8.2.1 Experimentation environment

The experiments were run on a cloud computer that was equipped with a four-threaded 64-bit 2.93 GHz Intel i7 processor, 16 GiB of RAM, Oracle Java Development Kit 1.6.0\_25, and Windows 7 Pro 64-bit.

We have carried out our experimentation with a collection of datasets that were gathered from the top 41 web sites according to Alexa on February 14, 2011, excluding search engines since their hubs provide links to external sites; the dataset included four additional academical sites, namely: TDG Scholar, Google Scholar, Microsoft Academic Search, and Arxiv.org. A detailed description of our datasets can be found in Appendix C.1. A demo of our technique and the datasets used are available at the author's web site<sup>†1</sup>.

We also compared our technique to two state-of-the-art classification techniques. We did not find any available implementation of the URL-based techniques described in section 5.5. We had to resort to the following baselines:

**Template Page Model Classification Scheme, or TPM for short [21]:** It is a supervised structural web page classifier, that detects common templates in a small set of training pages. We implemented this proposal in the lab using Java. We used four randomly-selected pages in each dataset to train a classifier, 50 randomly-selected pages to evaluate it, and set the similarity threshold to 0.75; these figures were recommended by the authors in their article.

---

<sup>†1</sup><http://tdg-seville.info/inmahernandez>

**Support Vector Clustering, or SVC for short [15]:** It is an unsupervised clustering technique that is based on Support Vector Machines. We chose this clustering technique because it does not require the user to set a number of clusters beforehand. Note that SVC is particularly suitable for clustering web pages since it can deal with large sets of classification features efficiently; in this context, each feature corresponds to a token in a pattern and counts its frequency in the datasets. We used the implementation provided by RapidMiner [99] using the suggested parameters values: radial kernel with  $\gamma = 1.00$ , a kernel cache of size 200, a minimum of 2 pages per cluster, a convergence of 0.001, and a maximum of 100 000 iterations.

Our goal was to prove that our technique is able to perform as well as a supervised technique without needing a training set of annotated pages, and that it performs better than other unsupervised classification techniques. Furthermore, we aimed to prove that our technique is as effective as other techniques that use internal features, which need to download the pages before classifying them.

### 8.2.2 Experimental results

For each dataset, we used our pattern builder and evaluated its effectiveness. We used 50% of the hubs in each dataset to infer a set of patterns (training set) and the remaining 50% to validate our proposal (test set). We set both the confidence level for calculating outliers and the similarity threshold to 0.05. Regarding the confidence level, recall that it defines the fraction of data in a distribution that are considered outliers; we selected the standard value in the literature. Regarding the similarity threshold, recall that it is a small value that allows to consider two close numbers equal; there is not a standard in the literature, but we found out through repeated experimentation that setting it to other values had a slight negative impact on the effectiveness of our proposal.

Regarding effectiveness, we computed precision (P), recall (R), and the F1 measure (F1); regarding efficiency, we computed the CPU learning times in seconds (T). Precision, recall and F1 were measured for each class. However, the learning time could not be measured for each class in the unsupervised techniques, since the number of classes is not known beforehand; therefore, the time is measured for each site.

In the case of TPM, it was easy to compute the precision and recall since it is a supervised technique, i.e., it requires the user to provide annotated web pages for each class. Contrarily, CALA and SVC are unsupervised,

		CALA				TPM				SVC			
Summary		P	R	F1	T	P	R	F1	T	P	R	F1	T
Mean		0.98	0.91	0.92	9.24	1.00	0.65	0.70	4.17	0.66	0.93	0.72	10 613.52
Std. Deviation		0.09	0.19	0.18	10.80	0.00	0.39	0.38	5.81	0.31	0.20	0.26	32 019.15
Site	Class	P	R	F1	T	P	R	F1	T	P	R	F1	T
Amazon	Products	1.00	0.94	0.97		1.00	0.46	0.63		-	-	-	
	Reviews	1.00	0.99	1.00	34.06	1.00	0.22	0.36	16.78	-	-	-	-
	Authors	1.00	1.00	1.00		1.00	1.00	1.00		-	-	-	
Daily Motion	Videos	1.00	1.00	1.00	2.78	1.00	0.50	0.67	1.53	0.73	1.00	0.85	587.00
	User Profiles	1.00	1.00	1.00		1.00	0.91	0.95		0.24	1.00	0.38	
Ehow	Articles	1.00	0.99	0.99	2.06	1.00	0.00	0.00	0.86	1.00	0.78	0.88	1 113.00
Answers	Topics	0.92	0.99	0.95	5.06	1.00	0.50	0.67	6.03	0.22	1.00	0.36	33 553.00
	Questions	1.00	1.00	1.00		1.00	0.98	0.99		0.76	1.00	0.86	
Digg	Authors	0.99	1.00	1.00		1.00	0.49	0.66		0.26	1.00	0.42	
	Articles	1.00	1.00	1.00	3.80	1.00	0.00	0.00	6.59	0.26	1.00	0.42	453.00
	Comments	1.00	1.00	1.00		1.00	0.33	0.49		0.33	1.00	0.49	
India Times	Articles	1.00	0.45	0.62	5.42	1.00	0.00	0.00	0.53	0.99	0.70	0.82	1 240.00
Daily Mail	Authors	1.00	1.00	1.00	9.48	1.00	0.50	0.67	7.19	0.10	0.99	0.19	203 114.00
	Articles	1.00	0.45	0.62		1.00	1.00	1.00		0.45	0.99	0.62	
Deviantart	Photos	1.00	1.00	1.00	8.86	1.00	1.00	1.00	4.44	0.86	1.00	0.92	11 059.00
	Tags	0.94	0.31	0.47		1.00	0.50	0.67		0.12	0.98	0.21	
Filestube	Files	1.00	0.94	0.97	7.97	1.00	1.00	1.00	0.34	0.91	1.00	0.95	175.00
The Huffington Post	Articles	1.00	0.18	0.31	3.73	1.00	0.83	0.90	5.30	0.99	0.99	0.99	858.00
Sourceforge	Projects	1.00	1.00	1.00	8.02	1.00	0.96	0.98	0.84	0.61	1.00	0.76	420.00
	Reviews	1.00	1.00	1.00		1.00	0.49	0.66		0.36	1.00	0.53	
Squidoo	Articles	1.00	0.99	1.00	2.08	1.00	0.00	0.00	2.27	0.58	1.00	0.73	137.00
	User Profiles	1.00	1.00	1.00		1.00	0.49	0.66		0.36	1.00	0.53	
Torrentz	Files	1.00	1.00	1.00	3.33	1.00	1.00	1.00	0.17	0.98	1.00	0.99	914.00
The Guardian	Authors	0.65	1.00	0.99	20.94	1.00	1.00	1.00	7.55	0.35	0.97	0.51	6 028.00
	Articles	0.35	1.00	0.02		1.00	0.87	0.93		0.56	0.72	0.63	
Archive	Articles	1.00	0.98	0.99	13.28	1.00	0.98	0.99	0.23	0.98	1.00	0.99	263.00
Isohunt	Files	1.00	0.97	0.98	4.94	1.00	0.98	0.99	2.69	0.49	1.00	0.66	524.00
	Comments	1.00	1.00	1.00		1.00	1.00	1.00		0.49	1.00	0.66	
Yelp	Businesses	1.00	0.79	0.88	7.59	1.00	1.00	1.00	24.61	1.00	1.00	1.00	899.00
Metacafe	Videos	1.00	0.76	0.86		1.00	0.25	0.40		0.50	1.00	0.67	
	Topics	1.00	0.97	0.99	4.20	1.00	1.00	1.00	4.41	0.28	1.00	0.44	2 647.00
	User Profiles	1.00	0.91	0.95		1.00	0.48	0.65		0.19	1.00	0.32	
Etsy	Products	1.00	0.95	0.97	21.55	1.00	0.52	0.69	2.78	-	-	-	-
	Stores	1.00	1.00	1.00		1.00	0.42	0.60		-	-	-	
BBC	News	1.00	0.57	0.73	3.66	1.00	0.00	0.00	2.31	0.39	0.97	0.56	37.00
	Videos	1.00	1.00	1.00		1.00	0.00	0.00		0.43	1.00	0.60	
Alibaba	Products	1.00	0.89	0.94	22.23	1.00	1.00	1.00	4.36	0.97	0.25	0.39	3 746.00

P = Precision; R = Recall; F1 = F1-measure; T = CPU learning time;

**Table 8.1:** Results of the evaluation.

Site	Class	CALA				TPM				SVC			
		P	R	F1	T	P	R	F1	T	P	R	F1	T
Target	Products	1.00	1.00	1.00	47.50	1.00	1.00	1.00	12.77	0.98	1.00	0.99	31917.00
TDG Scholar	Authors	1.00	1.00	1.00		1.00	1.00	1.00		0.60	1.00	0.75	
	Hosts	1.00	1.00	1.00	6.03	1.00	1.00	1.00	1.38	0.77	0.92	0.83	38081.00
	Papers	1.00	0.25	0.40		1.00	0.00	0.00		1.00	0.88	0.93	
MS Academic	Authors	1.00	0.85	0.92	7.89	1.00	0.96	0.98	2.83	0.73	1.00	0.85	354.00
	Papers	1.00	0.98	0.99		1.00	0.50	0.67		0.21	1.00	0.35	
Google Scholar	Citations	1.00	1.00	1.00	5.28	1.00	1.00	1.00	1.23	1.00	1.00	1.00	66.00
Arxiv	Authors	1.00	1.00	1.00		1.00	0.25	0.40		0.73	0.18	0.29	
	Papers	1.00	1.00	1.00	20.81	1.00	0.00	0.00	9.53	0.17	0.18	0.17	3124.00
	Abstracts	1.00	0.88	0.94		1.00	0.92	0.96		0.10	0.19	0.13	
Livejournal	News	1.00	0.64	0.78	3.30	1.00	0.00	0.00	1.22	1.00	1.00	1.00	6372.00
Xing	User Profiles	1.00	1.00	1.00	2.08	1.00	1.00	1.00	0.77	1.00	1.00	1.00	1300.00
Odesk	User Profiles	1.00	1.00	1.00	4.70	1.00	0.49	0.66	2.61	0.55	1.00	0.71	22414.00
	Skills	1.00	1.00	1.00		1.00	1.00	1.00		0.41	1.00	0.58	
Articles Base	Authors	1.00	1.00	1.00	3.41	1.00	1.00	1.00	2.00	0.91	1.00	0.95	133.00
Freelancer	Projects	1.00	1.00	1.00	4.47	1.00	1.00	1.00	26.33	1.00	0.92	0.96	26560.00
Plenty Of Fish	User Profiles	1.00	1.00	1.00	6.16	1.00	0.85	0.92	0.25	1.00	1.00	1.00	3766.00
Slideshare	Files	0.92	1.00	0.96	3.00	1.00	1.00	1.00	1.30	0.92	1.00	0.96	171.00
Netlog	User Profiles	1.00	1.00	1.00	3.72	1.00	0.85	0.92	0.59	1.00	1.00	1.00	49.00
Drupal	Projects	1.00	0.68	0.81	3.14	1.00	0.00	0.00	1.14	0.53	1.00	0.70	854.00
	Authors	1.00	1.00	1.00		1.00	1.00	1.00		0.34	1.00	0.50	
Newegg	Products	1.00	1.00	1.00	47.38	1.00	0.91	0.95	3.02	1.00	1.00	1.00	3068.00
Overblog	Articles	1.00	0.78	0.88	0.66	1.00	1.00	1.00	0.53	0.91	1.00	0.96	80.00
Chip	Articles	1.00	1.00	1.00	6.75	1.00	0.98	0.99	2.48	1.00	1.00	1.00	19.00
Battle.net	Forum Posts	1.00	1.00	1.00	1.95	1.00	0.74	0.85	3.36	1.00	1.00	1.00	179.00
Fiverr	Ads	0.96	1.00	0.98	4.16	1.00	0.91	0.95	3.36	0.96	1.00	0.98	1798.00
Fotolia	Photos	1.00	1.00	1.00	12.00	1.00	1.00	1.00	0.84	1.00	1.00	1.00	14324.00
People	Styles	1.00	0.70	0.74		1.00	0.00	0.00		0.52	0.92	0.66	
	Babies	1.00	0.69	0.73	4.25	1.00	0.00	0.00	3.77	0.48	0.96	0.64	8439.00
	Covers	1.00	1.00	1.00		1.00	0.33	0.50		0.61	0.87	0.71	
	Articles	1.00	0.84	0.84		1.00	1.00	1.00		0.91	0.94	0.93	
Indeed	Jobs	1.00	1.00	1.00	4.27	1.00	0.00	0.00	2.45	1.00	1.00	1.00	2271.00
Game FAQs	Boards	1.00	1.00	1.00	8.61	1.00	0.98	0.99	0.48	1.00	1.00	1.00	12662.00

Table 8.1: Results of the evaluation. (Cont'd)

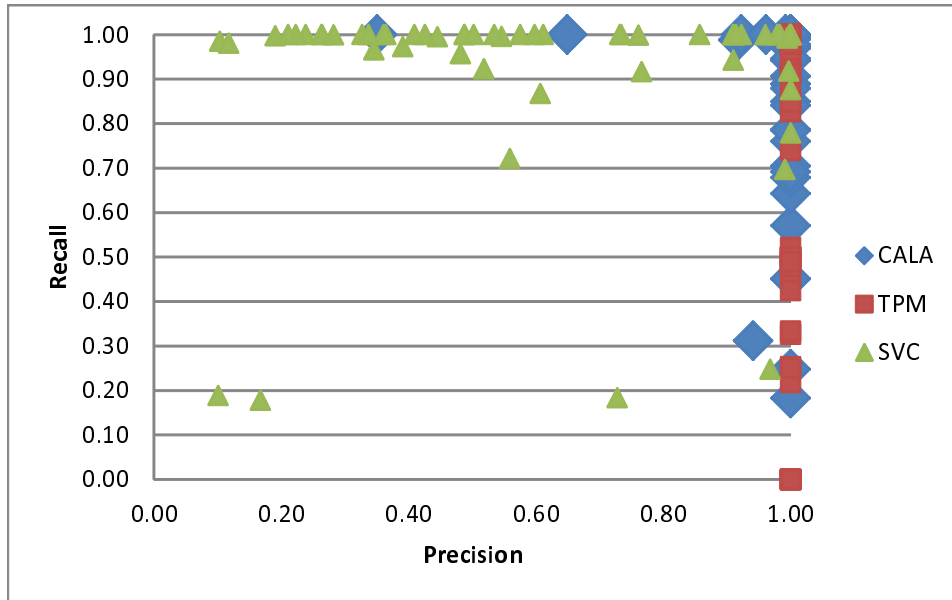
i.e., they cluster web pages, give each cluster a computer-generated class, and it is the responsibility of the user to assign a meaning to these classes. To validate CALA and TPM, we handcrafted annotations for every web page in our test sets, so we could find which cluster was the closest to each annotation. To do so, we compared each web page in each cluster to every annotation, and computed the number of true positives (tp), false negatives (fn), and false positives (fp), since this allowed us to compute precision as  $P = \frac{tp}{tp+fp}$ , recall as  $R = \frac{tp}{tp+fn}$ , and the F1 measure as  $F1 = 2 \frac{PR}{P+R}$ . Given an annotation, we can consider that the precision and recall to classify it corresponds to the cluster with the highest F1 measure.

Table 8.1 reports on the results of our experiments. The columns report on the number of URLs of each class in each dataset (U), average page size in KiB (S), precision (P), recall (R), the F1 measure (F1), and learning time in CPU seconds (T). 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 one week.

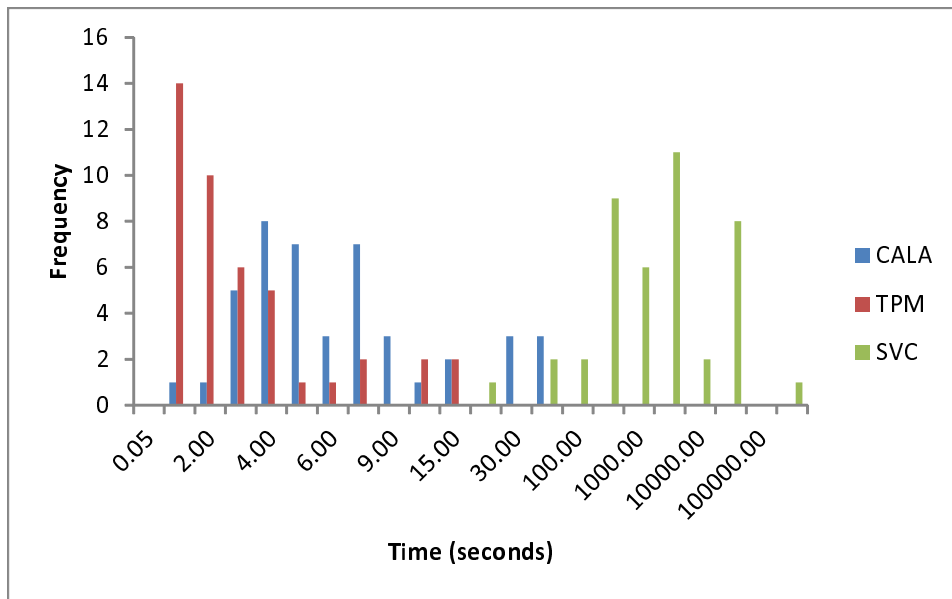
Figure 8.1(a) illustrates the effectiveness measures in a scatter plot; intuitively, the closer the points to (1.00,1.00) the higher the F1 measure; similarly, the closer to (0.00,0.00) the lower the F1 measure. Figures 8.1(b) and 8.2 illustrate the efficiency measures in a histogram and a box plot, respectively; intuitively, this helps compare the learning times taking into account the whole range of values. Note that the X axis in the former histogram is expressed in logarithmic scale because of its large extension.

According to Table 8.1, CALA outperforms the other techniques regarding F1; Figure 8.1(a) illustrates this conclusion since the majority of points that correspond to CALA are very close the upper right corner, whereas the points that correspond to the other techniques are more scattered. Note too that the results in Table 8.1 support the idea that CALA is more effective regarding learning time than the other unsupervised technique (SVC), and that it is comparable to TPM which is supervised and requires to download a page before classifying it. Figures 8.1(b) and 8.2 illustrate this idea. Note that for CALA, times range from 0.53 to 49.00 seconds and for TPM, they range from 0.17 to 26.33 seconds. Therefore, in both cases the learning times are below 60 seconds. On the contrary, for SVC learning times range from 19.00 to 203,114 seconds (more than 2 days). Therefore, CALA and TPM behave more homogeneously regarding learning times, and their learning times are significantly lower than those of SVC.

The drawbacks and the weak points of the other techniques were reflected in their effectiveness. TPM classifies pages according to the sequences

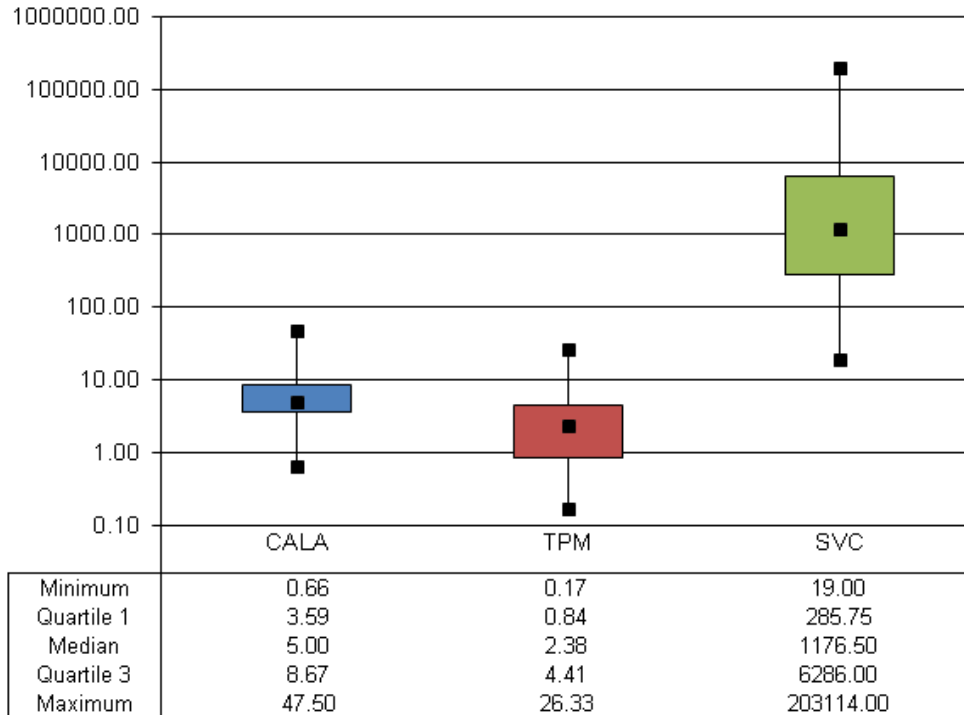


(a) Trade-off precision-recall of the three techniques.



(b) Histogram with the execution times.

**Figure 8.1:** Performance of CALA with regard to the other techniques.



**Figure 8.2:** Boxplot of the learning times of CALA, TPM, and SVC.

of tags in their DOM trees, therefore, it is highly influenced by small changes in the DOM trees (e.g., an advertisement that is present in one page but not in another one); this is the reason why its recall is relatively low and the standard deviation is relatively high. SVC learns a degenerated clustering in many cases, i.e., a clustering that consists in a single cluster that includes every page, cf., Table 8.2; therefore, its precision is relatively low, its recall deceptively high, and the standard deviation relatively high in both cases.

Their drawbacks were also reflected in the efficiency results achieved. TPM is able to learn a classifier in less time than CALA, but it needs to download a page before classifying it, which increases the classification time. SVC relies on a Support Vector Machine that needs too much time to create a clustering in some cases, depending on the size of the training set. That is the reason why SVC failed to create a clustering for Amazon and Etsy after one week.



---

Site	CALA	TPM	SVC
Amazon	5	3	-
Daily Motion	2	2	1
eHow	2	1	3
Answers	4	2	3
Digg	3	3	1
India Times	5	1	5
Daily Mail	10	2	2
Deviantart	2	2	2
Filestube	2	1	2
The Huffington Post	4	1	3
Sourceforge	2	2	1
Squidoo	3	2	1
Torrentz	1	1	1
The Guardian	2	2	5
Archive	3	1	1
Isohunt	2	2	1
Yelp	4	1	2
Metacafe	7	3	2
Etsy	3	2	-
BBC	5	2	3
Alibaba	3	1	2

---

**Table 8.2:** Number of classes/clusters created by each technique.

### 8.3 Statistical analysis

To confirm that the conclusions we have drawn from our empirical evaluation are valid, we need to perform a statistical ranking [51, 119].

The first step is to determine if the evaluation results are normally distributed and have equal variances; in such a case we must perform a parametrical analysis and in other case a non-parametrical analysis. 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 CALA's precision is  $W = 0.22$ , whose p-value is 0.00; this is a strong indication that the

---

Site	CALA	TPM	SVC
Target	2	1	2
TDG Scholar	3	3	4
MS Academic	6	2	1
Google Scholar	1	1	2
Arxiv	4	3	2
Livejournal	3	1	2
Xing	1	1	2
Odesk	4	2	2
Articles Base	1	1	1
Freelancer	1	1	2
Plenty Of Fish	1	1	2
Slideshare	1	1	1
Netlog	2	1	1
Drupal	5	2	2
Newegg	2	1	3
Overblog	1	1	1
Chip	1	1	3
Battle.net	1	1	2
Fiverr	2	1	2
Fotolia	1	1	2
People	8	4	4
Indeed	2	1	2
Game FAQs	1	1	2

---

**Table 8.2:** Number of classes/clusters created by each technique. (Cont'd)

data is not distributed normally. This is not surprising at all; a quick look at the scatter plot in Figure 8.1(a) makes it clear that these cloud of points are far from a Gaussian circle.

As a conclusion, we have performed a non-parametric analysis, which consists of the following steps: i) compute the rank of each technique from the evaluation data; ii) determine if the differences in ranks are significant or not using Iman-Davenport's test; iii) if the differences are significant, then compute the statistical ranking using Bergmann-Hommel's test on every

	Sample Ranking		Iman-Davenport	Bergmann-Hommel's				Statistical Ranking	
	Tech	Rank	P-value	Tech	CALA	TPM	SVC	Tech	Rank
P	TPM	1.56	8.71E-18	CALA	-	4.47E-01	2.32E-10	TPM	1
	CALA	1.68		TPM	-	-	3.76E-12	CALA	1
	SVC	2.76		SVC	-	-	-	SVC	2
R	SVC	1.71	7.95E-06	CALA	-	3.28E-04	4.22E-01	SVC	1
	CALA	1.84		TPM	-	-	3.33E-05	CALA	1
	TPM	2.45		SVC	-	-	-	TPM	2
F1	CALA	1.53	2.41E-06	CALA	-	3.86E-04	8.17E-06	CALA	1
	TPM	2.13		TPM	-	-	2.54E-01	TPM	2
	SVC	2.33		SVC	-	-	-	SVC	2
T	TPM	1.18	2.61E-36	CALA	-	0.28E-02	2.97E-08	TPM	1
	CALA	1.81		TPM	-	-	2.97E-08	CALA	2
	SVC	3.00		SVC	-	-	-	SVC	3

Tech = Technique;

**Table 8.3:** Results of our statistical ranking.

pair of techniques.

Table 8.3 presents the results of the analysis. 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 F1, which means that it achieves a better trade-off between precision and recall than the other techniques. Regarding P and R the differences with TPM and SVC (which are the proposals that ranks the first in each variable, respectively) do not seem to be statistically significant. Regarding efficiency, our proposal is faster than SVC a thousand orders of magnitude. Although it is slower than TPM, note that since TPM uses structure-based features, it has to download a page before classifying it. Therefore, although the learning time is faster, the classification time is slower. As a conclusion, our experiments prove that there is enough statistical evidence to conclude that our proposal outperforms the others.

## 8.4 Corroboration of conjectures

In this section, we corroborate the conjectures on which our proposal relies.

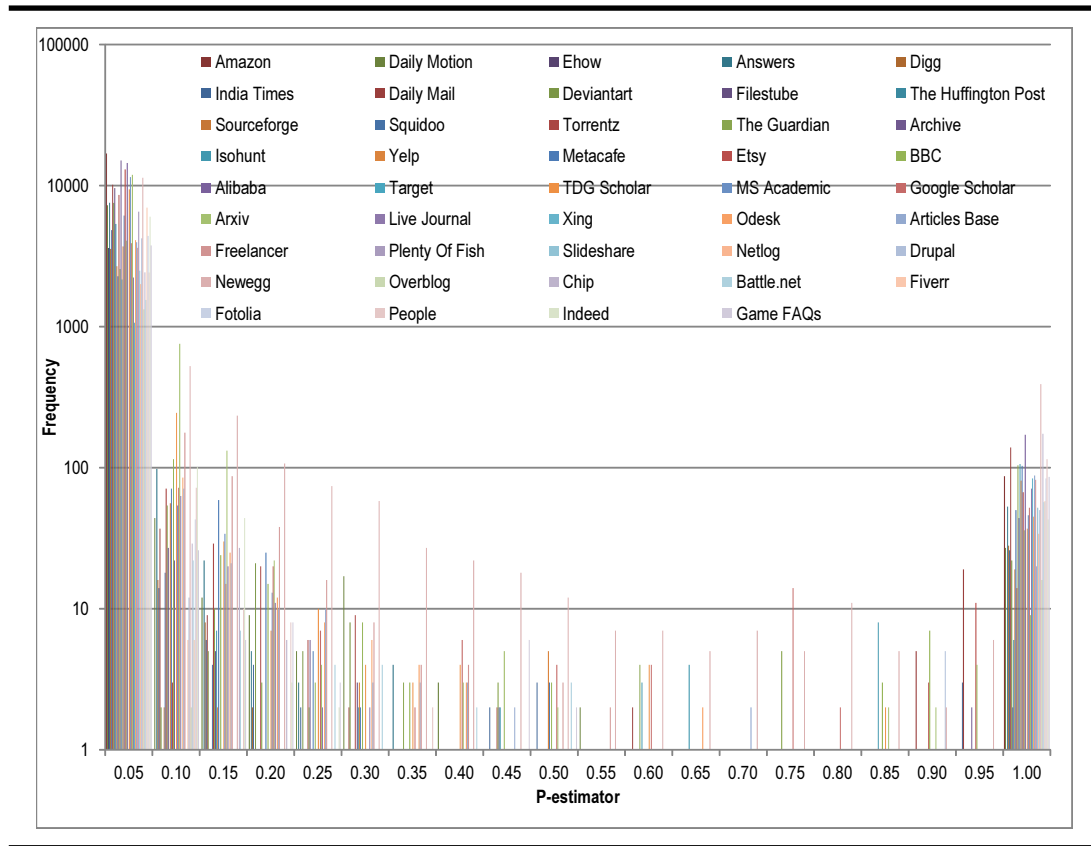


Figure 8.3: Distribution of  $p$ -estimators in our experiments.

**Conjecture 6.1 (Value of  $M$ ):** We conjectured that a relatively small number of hubs suffices to achieve a good effectiveness. Our experiments corroborate this hypothesis since examining more than 100 hubs did not result in better efficiency.

**Conjecture 6.2 (Value of  $T$ ):** We set a maximum number of attempts in Algorithm gatherHubset and conjectured that a relatively small number of attempts suffices to gather enough hubs. In our experiments, we managed to gather 100 hubs per site in a maximum of five attempts, which corroborates our conjecture.

**Conjecture 6.3 (Value of  $L$ ):** We conjectured that discarding words with a size lower than  $L$  suffices to gather enough words (excluding stop words) to be used as keywords. Our experiments corroborate this hypothesis since we got enough keywords to gather 100 hubs per site in every case.

---

Site	h	e	p
Amazon	100	451 000 000	~ 0.00
Daily Motion	100	67 600 000	~ 0.00
eHow	100	3 700 000	~ 0.00
Answers	100	106 000 000	~ 0.00
Digg	100	13 000 000	~ 0.00
India Times	100	20 600 000	~ 0.00
Daily Mail	100	237 000	~ 0.04
Deviantart	100	237 000 000	~ 0.00
Filestube	100	184 000 000	~ 0.00
The Huffington Post	100	9 320 000	~ 0.00
Sourceforge	100	17 200 000	~ 0.00
Squidoo	100	552 000	~ 0.02
Torrentz	100	86 100 000	~ 0.00
The Guardian	100	90 800 000	~ 0.00
Archive	100	18 600 000	~ 0.00
Isohunt	100	25 800 000	~ 0.00
Yelp	100	73 000 000	~ 0.00
Metacafe	100	6 750 000	~ 0.00
Etsy	100	155 000 000	~ 0.00
BBC	100	24 100 000	~ 0.00
Alibaba	100	630 000 000	~ 0.00

h = Number of hubs gathered; e = Estimated number of pages; p = Percentage of pages gathered.

---

**Table 8.4:** *Number of pages from each web site in the experiment.*

**Conjecture 6.4 (Value of N):** We conjectured that a relatively small number of keywords from the initial search page and subsequent hubs suffices to gather enough hubs. Our experiments corroborate this hypothesis since we managed to gather 100 hubs per site using 10 keywords in every case.

**Conjecture 7.1 (Wildcarding criterion):** We conjectured that wildcarding prefixes with a p-estimator that deviates from the p-estimators of their siblings and whose distance to 1.00 is higher than 0.05 suffices to build

---

Site	h	e	p
Target	100	6 090 000	~ 0.00
TDG Scholar	100	452 000	~ 0.02
MS Academic	100	83 800	~ 0.12
Google Scholar	100	7 520 000	~ 0.00
Arxiv	100	2 110 000	~ 0.00
Live Journal	100	189 000 000	~ 0.00
Xing	100	8 930 000	~ 0.00
Odesk	100	11 200 000	~ 0.00
Articles Base	100	3 420 000	~ 0.00
Freelancer	100	4 720 000	~ 0.00
Plenty Of Fish	100	15 100 000	~ 0.00
Slideshare	100	47 700 000	~ 0.00
Netlog	100	434 000 000	~ 0.00
Drupal	100	2 910 000	~ 0.00
Newegg	100	14 000 000	~ 0.00
Overblog	100	31 200 000	~ 0.00
Chip	100	3 320 000	~ 0.00
Battle.net	100	20 400 000	~ 0.00
Fiverr	100	1 040 000	~ 0.01
Fotolia	100	110 000 000	~ 0.00
People	100	345 000	~ 0.03
Indeed	100	161 000 000	~ 0.00
Game FAQs	100	27 400 000	~ 0.00

h = Number of hubs gathered; e = Estimated number of pages; p = Percentage of pages gathered.

---

**Table 8.4:** *Number of pages from each web site in the experiment. (Cont'd)*

patterns that allow classifying URLs achieving high precision and recall. Our experiments corroborate this hypothesis, cf. Table 8.1.

**Conjecture 7.2 (Distribution of p-estimators):** We conjectured that the distribution of p-estimators has two peaks at 0.00 and 1.00, and that it is more dense at these peaks than in between them. Figure 8.3 depicts the distribution of p-estimators in our datasets; roughly 94.62% of the val-

ues range from 0.00 to 0.05, 1.63% range from 0.05 to 0.10 and 2.24% range from 0.95 to 1.00.

## **8.5 Summary**

In this chapter, we have presented the evaluation of our proposal. First, have we described the experimental design and presented the results of the experiments. From the previous results, we could draw some preliminary conclusions regarding the effectiveness and efficiency of our proposal. Then, we have described the statistical tests we performed to test the former conclusions. Regarding effectiveness, the tests confirm that our proposal achieves a better trade-off between precision and recall than the other techniques. Regarding efficiency, the tests confirm that its learning time is slightly slower than that of the supervised technique, and significantly faster than that of the unsupervised technique. However, its execution time is faster than that of the supervised technique, which requires downloading each page before classifying it.





---

*Part IV*

*Final Remarks*

---



---

# Chapter 9

## Conclusions

---

*Everything has to come to an end, sometime*  
*The Marvelous Land of Oz, 1900.*

*Lyman Frank Baum, writer (1856-1919)*

In this dissertation, we present CALA, a proposal to automatically generate URL-based web page classifiers that can be used to implement navigators for enterprise web information integration. Our proposal takes the URL of a web page with a search form as input, and it outputs a set of patterns that represent the URLs of pages that belong to each semantic class. It is based on a statistical outlier detection technique to decide which parts of an URL are significantly representative from each URL pattern and which parts can be abstracted.

We have validated our proposal using datasets gathered from 44 real-world web sites that we have made publicly available. We have built a set of patterns for each web site, and we have used them to classify further pages; we achieved an average precision of 98% and average recall of 91%. The times required to build those patterns were reasonable, similar to those of a supervised technique with which we compare ours. These results suggest that our proposal seems promising enough for real-world web page classification, that it is efficient in practice, that the patterns it builds are able to classify web pages accurately, and that it minimises the number of irrelevant pages that are downloaded, which makes it suitable for enterprise web information integration contexts.

In addition to enterprise web information integration, the patterns built by our proposal have more potential uses in other fields, and it is our plan to

research on them as future work: in the context of web site model discovery, the patterns built from a web site can be used as a first approach for the entities in a web site model. Then, other algorithms are needed to detect the relationships between those entities, to create the complete model. More details can be found in Appendix D. We have presented preliminary results elsewhere [66].

As a future work, we think that the following problems deserve further research: joining two different patterns that actually represent the same class, splitting a pattern that represents two different classes into two more cohesive patterns, and discarding patterns that do not improve classification (for example, patterns that match only a few URLs). Other research directions include considering extended features for classification. Since we get the URLs for learning the patterns from hubs, some other features might be computed from them, like the anchor text of the link, or the text surrounding the link.

---

*Part V*  
*Appendices*

---



---

# Appendix A

## Notation

---

Throughout this dissertation, we use a simple notation to present formal definitions. It builds on a subset of the  $\mathcal{Z}$  mathematical language standard (ISO/IEC 13568:2002) [73] that is presented in Table A.1. We also rely on a number of data types that we describe below.

Our proposal relies on the analysis of the URLs of the web pages provided by a number of hubs. Generally speaking, a URL describes the access protocol and the location of a resource. The URL syntax was defined by the IETF in RFC 3986 [130]. According to this recommendation, a URL like `http://academic.research.microsoft.com/Detail?entitytype=1&searchtype=5&id=48814179#stats` is composed of the following segments: first, a protocol (`http`); then, an authority or domain name (`academic.research.microsoft.com`); afterwards a sequence of path segments separated by slash characters (`Detail`); and two optional sections: a question mark symbol followed by a query string, and/or a sharp symbol followed by a fragment. A query string provides information about the names and the values of a number of parameters sent to the web server (`entitytype=1&searchtype=5&id=48814179` includes parameters `entityType`, `searchType` and `id` with values 1, 5 and 48814179, respectively). Finally, the fragment is a sequence of characters that indicates a specific section inside a page (`stats`).

We define a token as a subsequence of characters in a URL that is delimited by separators `'://', '/', '?', '&', '=',` and `'#'`. In other words, a token is a string of characters that denotes a protocol, a path segment, a parameter, a value of a parameter, or a fragment. We introduce URLs and tokens as three given sets in our framework since we need not delve into their structure:

Separator == `{://, /, ?, &, =, #}`  
[URL, Token]

Notation	Description
$\mathbb{N}$	Set of natural numbers, including zero.
$\mathbb{R}$	Set of real numbers.
$[A]$	Introduces a given set $A$ .
$\text{seq } A$	Finite set of elements of type $A$ .
$\text{bag } A$	Finite bag of elements of type $A$ .
$\text{seq } A$	Sequence of elements of type $A$ .
$\langle a \rangle$	Sequence consisting of element $a$ .
$A \setminus B$	Creates a new bag with the elements of bag $A$ (their frequency is not altered) that are not in set $B$ .
$\#A$	Size of set or sequence $A$ .
$\text{flat } A$	Flattening of the set of sets $A$ .
$\text{last } S$	Last element in sequence $S$ .
$\text{front } S$	Subsequence of $S$ without its last element.
$P \text{ prefix } Q$	Sequence $P$ is a prefix for sequence $Q$ .
$\text{subseq}(S, i, j)$	Subsequence of sequence $S$ between positions $i$ and $j$ .
$\text{sortSet } P$	Sorting of $P$ , a set of sequences, by size in ascending order.
$\text{sortBag } Q$	Sorting of $Q$ , a bag of elements, by frequency in ascending order.
$S \frown T$	Concatenation of sequences $S$ and $T$ .
$\text{mean } A$	Mean of the non-empty set of reals $A$ .
$\text{stdev } A$	Standard deviation of the non-empty set of reals $A$ .
$a \approx b$	Numerical values $a$ and $b$ are approximately equal, i.e. $a = b \pm \beta$ , where $\beta$ is close to 0.0.
$f : A \rightarrow B$	Function $f$ is a partial mapping defined from domain $A$ to range $B$ .
$f : A \rightarrow B$	Function $f$ is a total mapping defined from domain $A$ to range $B$ .
$\{a : A \mid p(a)\}$	Subset of elements of $A$ that satisfy predicate $p$ . If omitted, $p(a)$ is assumed to be true.
$\{a : A \mid p(a) \bullet f(a)\}$	Subset of elements of $A$ that satisfy predicate $p$ transformed by function $f$ .

**Table A.1:** Some mathematical notation used throughout this dissertation.



We define a pattern as a sequence of tokens, separators, and wildcards that ends with a \$ symbol. A wildcard, which we denote as  $\ast$ , is a placeholder that accounts for any token. Note that given a URL, it is straightforward to transform it into a pattern; thus, we do not provide any additional details on this procedure.

A prefix refers to a subsequence of a pattern that starts at the first token and extends up to any token in the pattern, but the last one. Note that a pattern is similar to a prefix, since both of them are sequences of tokens, separators, and wildcards; the only difference between them is that a pattern ends with a \$. We formally define the previous concepts as follows:

$$\begin{aligned} \text{Marker} &== \{\ast, \wedge, \$\} \\ \text{Prefix} &== \{p : \text{seq}(\text{Token} \cup \text{Separator} \cup \text{Marker}) \mid \\ &\quad \#p \geq 1 \wedge p(1) = \wedge \wedge (\forall i : \mathbb{N} \mid i > 1 \wedge i \leq \#p - 1 \bullet p(i) \notin \{\wedge, \$\})\} \\ \text{Pattern} &== \{p : \text{Prefix} \mid \text{last } p = \$\} \end{aligned}$$

A hub page is a web page that results from submitting a keyword-based search form using some keywords as query, and provides summaries and links to other web pages [80]. Note that hub pages usually contain a larger number of URLs than other pages since their goal is to offer the users as many results related to their queries as possible. Regarding our proposal, a hub can be abstracted as a set of patterns that result from the URLs in the links provided by that hub page. A hubset is a collection of hubs that result from submitting a search form using different words. We formally define the previous concepts as follows:

$$\begin{aligned} &[\text{Word}, \text{WebPage}] \\ \text{Hub} &== \text{seq Pattern} \\ \text{Hubset} &== \text{seq Hub} \end{aligned}$$



---

## Appendix B

### Detecting outliers

---

An outlier is a value in a set that deviates markedly from other members of the sample in which it occurs [28]. The simplest technique to identify them in unidimensional sets builds on the Cantelli inequality, according to which for any random variable  $X$  with finite mean  $\mu$  and finite non-zero standard deviation  $\sigma$  it holds that:

$$P(X - \mu \geq k \sigma) \leq \frac{1}{1 + k^2}$$

or, otherwise,

$$P(X \geq \mu + k \sigma) \leq \frac{1}{1 + k^2}$$

$\mu + k \sigma$  is an upper threshold that defines the frontier from the values of  $X$  that can be considered “normal” to the values that can be considered upper outliers. We generally wish this frontier to help identify values whose probability is very low, typically less than or equal to  $\alpha = 0.05$ . If we set

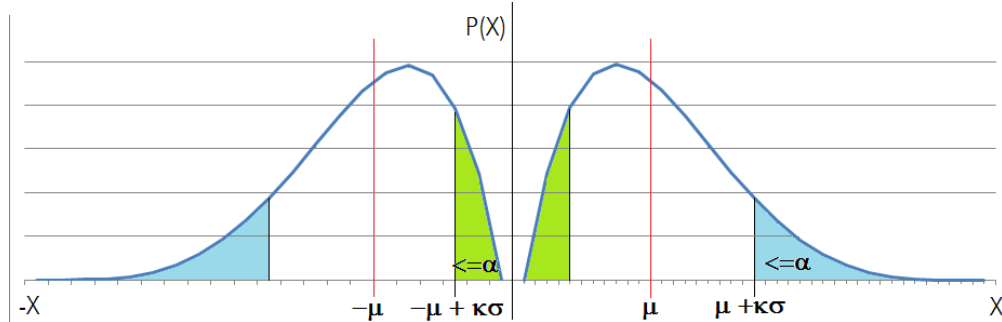
$$\alpha = \frac{1}{1 + k^2},$$

we then can conclude that

$$k = \sqrt{\frac{1 - \alpha}{\alpha}}.$$

Therefore, every value that is greater or equal than

$$\mu + \sqrt{\frac{1 - \alpha}{\alpha}} \sigma$$



**Figure B.1:** Sample distribution and its symmetric.

can be considered an upper outlier.

We also need to identify lower outliers. Unfortunately, the Cantelli inequality does not provide a lower threshold, but it is not difficult to compute building on the following idea: instead of studying variable  $X$ , we need to focus on variable  $-X$  whose mean value is  $-\mu$  and standard deviation is  $\sigma$ , cf. Figure B.1. If we apply the Cantelli inequality to variable  $-X$ , we conclude that

$$\begin{aligned} P(-X - (-\mu) \geq k\sigma) &\leq \frac{1}{1+k^2} \\ P(\mu - X \geq k\sigma) &\leq \frac{1}{1+k^2} \\ P(X \leq \mu - k\sigma) &\leq \frac{1}{1+k^2} \end{aligned}$$

In other words,  $\mu - k\sigma$  is the lower extreme value, where  $k$  is computed as previously.

Building on the previous argumentation, we formally define the following functions that help us identify both lower and upper outliers:

lowerThreshold : seq  $\mathbb{R} \rightarrow \mathbb{R}$   
 upperThreshold : seq  $\mathbb{R} \rightarrow \mathbb{R}$

$\forall R : \text{seq } \mathbb{R} \mid R \neq \emptyset \bullet$

let  $\mu == \text{mean } R$ ;  $\sigma == \text{stdev } R$ ;  $k == \sqrt{(1 - \alpha)/\alpha} \bullet$

lowerThreshold( $R$ ) =  $\mu - k\sigma$

upperThreshold( $R$ ) =  $\mu + k\sigma$

---

# *Appendix C*

## *Datasets*

---

In our experimentation we created 44 datasets, each of them composed of 100 hubs from real-world web sites. The sites were chosen from the top 500 most visited sites according to Alexa directory. We give a detailed description of each dataset in Table C.1. The datasets, together with the keywords that were used to fill in the forms in each case, are available at the author's web site<sup>†1</sup>.

---

<sup>†1</sup><http://tdg-seville.info/inmahernandez>

Class	XPath	URLs	Tokens/URL	Page size
<b>amazon.com</b>				
Products	//a[@class="title"]//div[@class="image"]//a	6273	28.08 ± 0.02	242.69
Reviews	//div[@class="starsAndPrime"]//a	7846	24.34 ± 0.05	149.90
Authors	//span[@class="ptBrand"]//a	1558	24.02 ± 0.01	140.64
<b>dailymotion.com</b>				
Videos	//div[@id="dual_list"]/div/h3//a   //div[@class="dmpi_video_preview image_border "]/a	2632	10.00 ± 0.00	63.91
User Profiles	//a[@class="login name"]	855	8.00 ± 0.00	68.56
<b>eHow.com</b>				
Articles	//li[@class="item"]//a//a[@id="search"]	869	10.00 ± 0.00	47.45
<b>wiki.answers.com</b>				
Topics	//div[@id="search-results"]/div/p[3]/a	1510	12.00 ± 0.00	136.63
Questions	//div[@id="search- results"]//a[@class="internal"]	4898	10.15 ± 0.02	55.43
<b>digg.com</b>				
Articles	//h3[@class="story-item-title"]//a   //a[@class="story-item-thumb"]	1366	10.00 ± 0.00	70.34
Authors	//li[@class="story-item-submitter"]//a	1011	6.00 ± 0.00	64.13
Comments	//li[@class="story-item-comments"]//a	1366	10.00 ± 0.00	14.45
<b>search.indiatimes.com</b>				
Articles	id('netspidersosh')/tbody/tr[2]/td/table /tbody/tr[4]/td/table/tbody/tr/td/table[2]/tbody /tr/td/table[3]/tbody/tr/td/table/tbody/tr/td/a[1]	631	15.18 ± 0.26	70.63
<b>dailymail.co.uk</b>				
Articles	//h3[@class="sch-res-title"]//a   //div[@class="sch-res-content"]/h2/a   //a[@class="boxed-image"]	1976	16.37 ± 0.03	124.95
Authors	//h4[@class="sch-res-info"]//a	468	18.43 ± 0.03	104.44
<b>deviantart.com</b>				
Photos	//a[@class="thumb"]   //a[@class="t"]	3677	10.00 ± 0.00	115.08
Tags	//span[@class="tt-w"]/small//a	524	13.61 ± 0.13	86.22
<b>filestube.com</b>				
Files	//a[@class="resultsLink"]   //div[@id="newresult"]/div/a[2]	1033	12.19 ± 0.06	26.16

Table C.1: Datasets description

Class	XPath	URLs	Tokens/URL	Page size
<b>huffingtonpost.com</b>				
Articles	//a[@property="f:title"]	754	14.55 ± 0.17	204.09
<b>sourceforge.net</b>				
Projects	//td[@class="description"]//a	2195	10.00 ± 0.00	32.11
Reviews	//div[@class="recommended"]//a	1294	8.00 ± 0.01	27.61
<b>squidoo.com</b>				
Articles	//dl[@class="searchresults search_list_dl"]/dt/a	969	8.00 ± 0.01	137.56
User profiles	//dd[@class="search_results_url"]/a[2]	613	10.00 ± 0.01	58.67
<b>torrentz.eu</b>				
Files	//div[@class="results"]/dl//a	4739	6.00 ± 0.00	16.60
<b>guardian.co.uk</b>				
Authors	//li[@class="l1"]/p/a	315	16.00 ± 0.00	82.46
Articles		682	22.92 ± 0.10	143.03
<b>archive.org</b>				
Articles	//h3[@class="t2"]//a[@class="link-text"]	4981	10.08 ± 0.02	19.09
<b>isohunt.com</b>				
Files	//table[@id="serps"]//td/a[2]	1996	14.00 ± 0.00	57.33
Comments	//table[@id="serps"]/tbody/tr/td[3]/a[1]	1996	14.00 ± 0.00	59.59
<b>yelp.com</b>				
Businesses	//div[@class="businessresult clearfix"]/div/h4/a	995	10.97 ± 0.06	159.49
<b>metacafe.com</b>				
Videos	//h3[@class="itemtitle"]//a   //p[@class="itemtitle"]//a	1971	13.17 ± 0.04	102.19
Topics	//h3[@class="itemtitle"]//a   //p[@class="itemtitle"]//a	1109	11.45 ± 0.07	68.75
User Profiles	//h3[@class="itemtitle"]//a   //p[@class="itemtitle"]//a	750	11.31 ± 0.10	55.67
<b>etsy.com</b>				
Products	//p[@class="listing-title"]//a   //a[@class="listing-thumb"]	4170	38.97 ± 0.15	86.93
Stores	//p[@class="listing-maker"]//a	4120	33.01 ± 0.15	110.65

**Table C.1: Datasets description (Cont'd)**

Class	XPath	URLs	Tokens/URL	Page size
<b>bbc.co.uk</b>				
News	//div[@id="news-content"]//a	220	12.21 ± 0.17	69.85
Videos		233	17.03 ± 0.03	49.89
<b>alibaba.com</b>				
Products	//a[@class="qrptitle"]	3912	14.30 ± 0.03	93.33
<b>target.com</b>				
Products	//span[@class="producttitle"]//a	3713	14.01 ± 0.00	245.09
<b>scholar.tdg-seville.info</b>				
Authors	//span[@class="person"]//a	2061	15.16 ± 0.04	29.36
Papers	//span[@class="title"]//a	759	14.98 ± 0.26	65.32
Hosts	//span[@class="host"]//a	336	15.09 ± 0.07	38.96
<b>academic.research.microsoft.com</b>				
Authors	//a[@class="author-name-tooltip"]	3767	14.00 ± 0.00	98.81
Papers	//div[@class="title-download"]//h3/a	1160	14.01 ± 0.01	78.90
<b>scholar.google.com</b>				
Citations	//span[@class="gs_fl"]//a[1]	1014	25.99 ± 0.03	102.87
<b>arxiv.org</b>				
Authors	//div[@class="list-authors"]//a	15529	27.00 ± 0.00	41.42
Papers	//a[@title="download pdf"]   //a[@title="download postscript"]	3681	10.00 ± 0.00	826.72
Abstracts	//a[@title="Abstract"]	2094	10.00 ± 0.00	14.65
<b>livejournal.com</b>				
News	//ol[@class="tagged-list"]/li/dl/dt/a	1333	11.33 ± 0.10	55.58
<b>xing.com</b>				
User profiles	//table[@class="data-table"]/tbody/tr[string-length(@style)=0]/td/a	993	10.00 ± 0.00	33.31
<b>odesk.com</b>				
User profiles	//div[@class="searchresult"]/div/div/h3/a	902	22.00 ± 0.00	31.44
Skills	//a[@class="skill"]	761	13.73 ± 0.14	44.32
<b>articlesbase.com</b>				
Authors	//span[@class="name"]//a	1008	12.00 ± 0.00	66.19

Table C.1: Datasets description (Cont'd)



Class	XPath	URLs	Tokens/URL	Page size
<b>freelancer.com</b>				
Projects	//div[@class="title"]/a	2207	14.06 ± 0.05	174.16
<b>pof.com</b>				
User profiles	//a[@class="mic"]	6542	14.00 ± 0.00	19.59
<b>slideshare.net</b>				
Files	//ol[@class="searchresults"]/li/a	1190	10.04 ± 0.02	82.55
<b>en.netlog.com</b>				
User profiles	//table[@class="searchresults"]/h4/a1	968	8.00 ± 0.00	36.28
<b>drupal.org</b>				
Projects	//dt[@class="title"]//a	2464	8.37 ± 0.03	24.89
Authors	//p[@class="submitted"]//a	1514	8.00 ± 0.00	14.59
<b>newegg.com</b>				
Products	//div[@class="itemtext"]//a	1832	16.00 ± 0.00	153.62
<b>over-blog.com</b>				
Articles	//h2[@class="title"]//a	1063	10.13 ± 0.04	53.02
<b>chip.de</b>				
Articles	//span[text()='Artikel']/../span//a	161	12.22 ± 0.11	112.93
<b>battle.net</b>				
Forum Posts	//div[@class="results post-results"]/h4/a	476	16.11 ± 0.06	87.34
<b>fiverr.com</b>				
Ads	//div[@class="gig-title"]//a	2428	8.01 ± 0.01	42.47
<b>fotolia.com</b>				
Photos	//div[@class="list"]/div/a	4161	10.00 ± 0.00	39.92
<b>people.com</b>				
Articles	//div[@id="resultsNews"] //a[@class="entry-permalink"]	311	19.53 ± 0.14	39.12
Babies	//div[@id="resultsBabies"] //a[@class="entry-permalink"]	301	15.00 ± 0.00	76.97
Covers	//div[@id="resultsCovers"] //a[@class="entry-permalink"]	219	21.00 ± 0.00	35.14
Styles	//div[@id="resultsStyleWatch"] //a[@class="entry-permalink"]	336	15.09 ± 0.13	54.55
<b>indeed.com</b>				
Jobs	//h2[@class="jobtitle"]//a	993	14.00 ± 0.00	21.05
<b>gamefaqs.com</b>				
Boards	//tr/td[11]/a	3371	10.00 ± 0.00	20.77

Table C.1: Datasets description (Cont'd)



---

## *Appendix D*

# *Web site model discovery using CALA*

---

The Web comprises a number of web sites that expose data stored in a back-end database, publishing them in a friendly format [30]. Entry points to these web sites are submittable query forms, which return as a response a number of web pages that are generated by filling a template with data [19, 79]. The data that fill each template is the result of executing a view over the back-end database [6].

Since query forms are the unique entry points to most sites in the Web, the different views that provide the data to fill each template are not accessible. Therefore, the conceptual model of the database, which comprises a number of entities and a number of relationships amongst these entities, remains hidden.

Having access to the conceptual model of a web site is mandatory to perform several tasks, such as integrating different (semantic or non-semantic) web sites [6, 110, 114], extracting information from the web without supervision [5, 29, 79], or creating ontologies by means of query forms [123].

As a consequence, there are many proposals in the literature that deal with discovering conceptual models behind web sites [5, 6, 19, 21, 22, 35, 65, 79, 98, 104, 123]. Some of these proposals deal with models composed solely of entities, without taking the relationships between them into account [19, 21, 22, 65, 98, 104]. Other proposals discover models with entities and relationships [6, 123], but they are supervised and require the intervention of the user, providing expert knowledge about each web site. Finally, the rest of the proposals focus on a single template, discovering only one view of the model [5, 35, 79].

As an application of our proposal, we propose a technique to discover the conceptual model behind a web site. The model our technique is able to discover from each web site does not represent the complete, hidden conceptual model of the back-end database, but the union of the views over that conceptual model, composed of those entities and relationships that are exposed in the web site.

Our technique takes a set of URL patterns as input, each of which represents an entity in a particular web site. It follows a statistical approach to detect relationships between those entities. Our hypothesis is that each relationship is materialised in HTML links that go from pages of one class to pages of another class, so an XPath pattern targeting those links is created to represent each relationship. The URL patterns that support our technique can be either handcrafted by the user, or automatically built by any of the former proposals [7, 22, 65, 104].

Our proposal presents some advantages: it creates a conceptual model consisting not only of entities, but also of relationships between those entities; it is not supervised, which saves the user a significant amount of time in annotating training sets, and does not require the user to have expert knowledge; and it integrates different views from the different templates in the site. Furthermore, our proposal discovers all the possible anonymous relationships in the model, and we leave the user the task of annotating those relationships with an appropriate name and selecting those relationships that are useful for his or her model. Therefore, the set of relationships we automatically discover can be used as a first approach to the model, which can be refined by an expert data modeller, with a significant reduction in time investment [125]. Our technique takes a set of URL patterns that describe all classes of information offered in a web site as input, and discovers relationships between the classes.

We base the discovery of relationships between two classes on the detection of HTML links in pages of one class whose target is a page of another class. We extract the XPath locators of those links, and we apply a statistical-based technique to estimate the variability of each token in each locator. Then, we abstract the tokens with a high variability (again, using a statistical criterion), learning XPath patterns. Finally, each XPath pattern represents a particular relationship between the former classes.

There is an abstract relationship between two classes  $a$  and  $b$ , if there are links to pages of class  $b$  in most pages of class  $a$ . However, more than one type of relationship may exist between any given pair of classes  $a$  and  $b$ . For

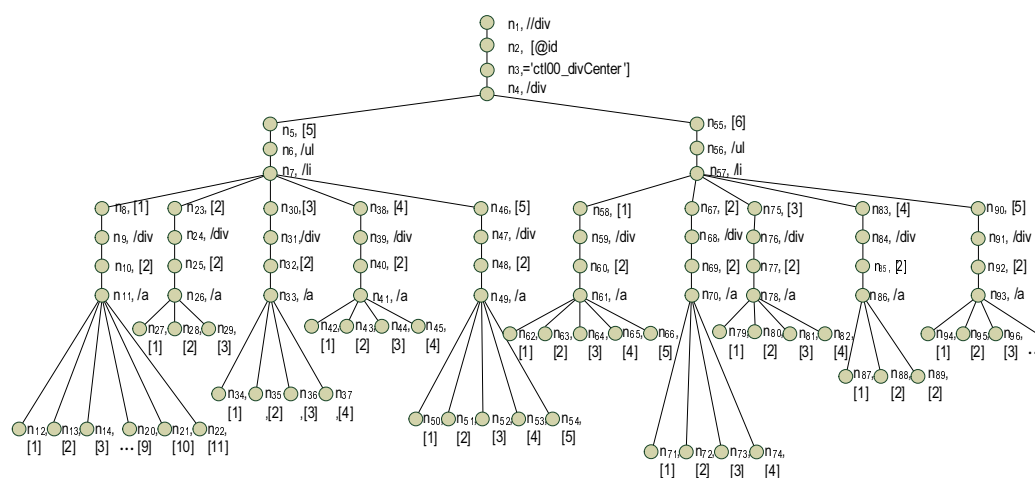


Figure D.1: XPathTree.

Node	token	$\Omega$	$V(n_i)$
n <sub>11</sub>	/a	11, 3, 3	4.62
n <sub>26</sub>	/a	3, 4, 3	0.58
n <sub>33</sub>	/a	4, 3, 3	0.58
n <sub>41</sub>	/a	2, 4, 1	1.53
n <sub>49</sub>	/a	1, 5, 2	2.08
n <sub>61</sub>	/a	4, 5, 5	0.58
n <sub>70</sub>	/a	4, 3, 4	0.58
n <sub>78</sub>	/a	3, 4, 3	0.58
n <sub>86</sub>	/a	3, 3, 2	0.58
n <sub>93</sub>	/a	4, 5, 8	2.08

Table D.1: Variability estimator values.

example, pages of class Author in MsAcademic contain both a list of publications, which include coauthors of the publication, and a list of citations, which includes authors that cited this author, as shown in Figure D.3. Therefore, there are two different types of relationships in this model between class Author and itself: 1) isCoauthorOf and 2) cites.

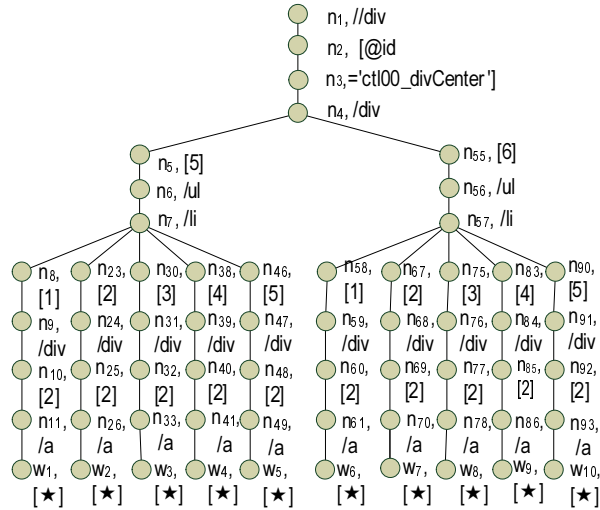


Figure D.2: XPathTree, after compression.

**Publications**

[On benchmarking data translation systems for semantic-web ontologies](#)  
 Carlos R. Rivero, **Inma Hernandez**, David Ruiz, Rafael Corchuelo  
 Conference: International Conference on Information and Knowledge Management - CIKM, Published in 2011.

[A Conceptual Framework for Efficient Web Crawling in Virtual Integration](#)  
**Inma Hernandez**, Hassan A. Sleiman, David Ruiz, Rafael Corchuelo  
 Conference: Web Information Systems and Mining - WISM, pp. 282-291, 2011

[A Reference Architecture for Building Semantic-Web Mediators](#)  
 Carlos R. Rivero, **Inma Hernandez**, David Ruiz, Rafael Corchuelo  
 Published in 2011.

**Co-authors**

[Integrating Deep-Web Information Sources](#) (Citations: 1)  
 Inaki Fernández de Viana, **Inma Hernandez**, Patricia Jiménez, Carlos R. Rivero, Hassan A. Sleiman  
 Published in 2010.

**Citations**

[Toward One Class Classifier techniques applied to verifier](#)  
 Inaki Fernandez de Viana, Pedro J. Abad, Jose L. Alvarez, Jose L. Arjona  
 Published in 2011.

[A Roadmap on Integrating Applications and Data on the Web](#)  
 Rafael Corchuelo, Jose Luis Arjona, David Ruiz, Jose L. Alvarez, Rafael Z. Frantz, Carlos Molina-jimenez, Inaki Fernandez De Viana, Patricia Carlos R. Osuna, Antonia M. Reina, Hassan Sleiman,  
 Conference: Jornadas de Ingeniería del Software y Bases de Datos - JISBD, 2010

**Citing authors**

Figure D.3: Detail page of class Author.

Using only URL patterns, we are not able to discern between these different types of relationships, since all URLs match the same pattern, regardless of the type of relationship they represent. Therefore, other features must be extracted from the URLs to classify them according to their role.

We assume that links whose URL matches the same URL patterns may appear in different locations in the page, but all links representing the same relationship appear in similar locations. Therefore, we use the XPath of the different links, which denotes their location in the page. We apply a technique to build patterns for those XPath, which starts by tokenising all XPath locators and inserting all their tokens in order in an XPathTree. Then, we use some criterion to discern tokens that must be abstracted (replaced by a wildcard), based on the concept of variability of a token.

The variability of a token refers to how spread the numbers of tokens that follow that token in different XPath locators in different pages of the same class (i.e., the different numbers of children of the node representing that token in each page) are. Since we do not analyse all pages of a site, but only a representative sample, we estimate the variability by means of the following definition.

**Definition D.1 (Variability estimator:)** *Let  $D^c$  be a set of detail pages of class  $c$ ,  $x$  an XPath expression and  $n_i$  be a tree node referring a token  $t$ , we define the variability estimator of node  $n_i$ , and we denote it as  $V(n)$  as the standard deviation of the numbers of children of node  $n_i$  in the different pages of  $D^c$ .*

Based on these variability estimators, we define a process to generate XPath patterns. For each node  $n_i$  in the XPathTree, we check if its variability estimator is significantly high, and in that case, all of its children nodes have their token replaced with a wildcard, and the subtrees rooted at them are merged. Contrarily, children of nodes with a low variability are probably part of a pattern, so they are not abstracted, but kept as literals.

Our technique to mine relationships between classes  $a$  and  $b$  consists of two steps: XPathTree building and XPathTree compressing.

In the first step, we extract all URLs matching pattern  $\Phi(b)$  in pages from  $D^a$ , and we calculate an XPath locator for each of them. XPath locators are tokenised, and each token is inserted in an XPathTree as a node with a variability estimator. An example of an XPathTree built using this technique is presented in Figure D.1. It contains XPath expressions of URLs matching  $\Phi(\text{Author})$  in the running example, extracted from detail pages of class Author.

In the second step, we apply a compressing algorithm that performs a depth-first traversal on the XPathTree, and for each visited node, uses its variability estimator to discern nodes with a variability higher than a given parameter  $\theta > 0$ . Those nodes have their token abstracted into a wildcard (\*). As an example, nodes with variability higher than 0.5 are presented in Table D.1.

After the whole tree has been traversed and processed, each of the resulting tree branches represents a different pattern. Furthermore, each pattern refers to a different type of relationship between classes a and b. As an example, in Figure D.2 we show the example tree containing XPath expressions of links between class Author and itself, after processing all of its nodes. The tree contains ten branches, which correspond to ten XPath patterns.

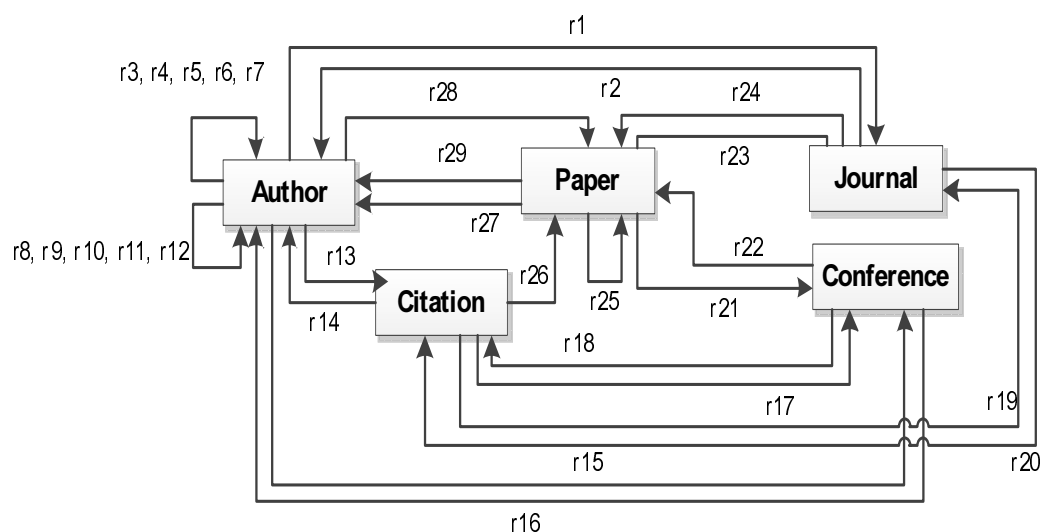
At the end of this process, for each pair of classes a and b, we have learnt a set of XPath patterns, that represent the different relationships between them. These relationships are anonymous, and it is left to the user the task of assigning them to an appropriate class. Furthermore, we have identified all the possible relationships, but some of them might be duplicated (i.e., we discover a relationship between a and b, which is the same as another relationship between b and a). Therefore, the user has the opportunity to select the relationships that are most suitable for his or her model, discarding the rest. Therefore, although we are indeed automatically discovering the relationships between entities, the user still has the complete control over the final model.

As an example, consider the former patterns discovered in Figure D.2. The first five patterns correspond to links to authors that co-author, respectively, the five most recent papers of an author. Meanwhile, patterns sixth to tenth correspond to links to authors of, respectively, the five most recent papers that cite the author. Therefore, the five first patterns correspond to a particular relationship between class Author and itself(isCoauthorOf), while the five last patterns correspond to a different relationship (cites).

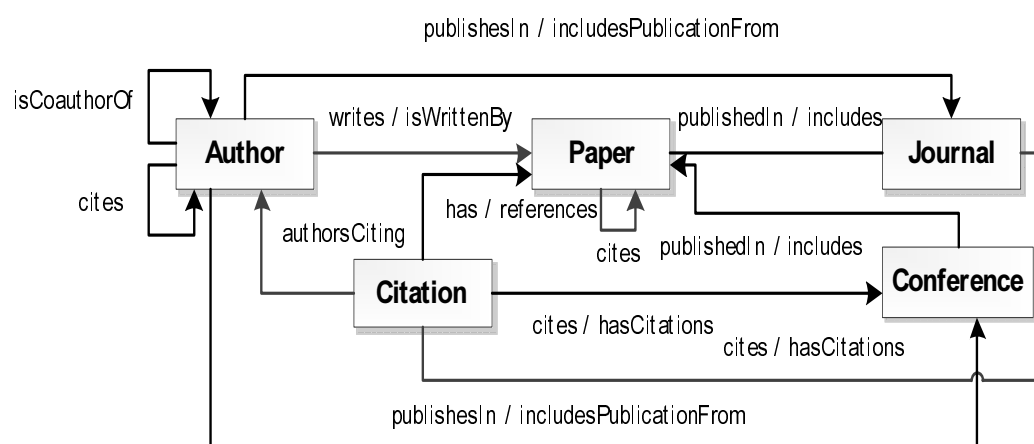
We performed an experiment to validate our technique. Microsoft Academic Search was analysed to discover the conceptual model behind it, by means of two steps: in the first step, we discovered the entities in the model, using the URL patterns obtained by CALA; in the second step we discovered the relationships between these entities, with the former URL patterns as input, and using the technique described in this paper.

We show the relationships discovered for this site in Figure D.4, using a UML class diagram. After the intervention from the user, a possible model





**Figure D.4:** Relationships for MsAcademic.



**Figure D.5:** Model for MsAcademic.

obtained from the former relationships is presented in Figure D.5. For example, relationships r3, r4, r5, r6 and r7, represent respectively the co-authors of the most recent paper of an author, the co-authors of the second most recent paper, and so on. The user analyses these relationships and decides that all these relationships are actually the same, and labels it *isCoauthorOf*.

Using our technique, it is also possible to infer hierarchical relationships

between classes, by identifying classes that share a common group of relationship with other classes. For example, in the former example model for MsAcademic, classes Journal and Conference both share exactly the same types of relationships (Journal is related to Paper by means of r23 and r24, to Author by means of r1 and r2 and to Citations by means of r19 and r20. Similarly, Conference is related to the same set of classes, with two relationships with each class). Therefore, our technique proposes the user the generalisation of Journal and Conference into another class, and lets the user name it (e.g., Host).

---

## Bibliography

---

- [1] *Webdriver*. [http://seleniumhq.org/docs/03\\_webdriver.html](http://seleniumhq.org/docs/03_webdriver.html), 2012.
- [2] S. Abiteboul and O. M. Duschka. *Complexity of answering queries using materialized views*. In *PODS*, pages 254–263, 1998.
- [3] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. *On the design of a learning crawler for topical resource discovery*. *ACM Trans. Inf. Syst.*, 19(3): 286–309, 2001.
- [4] V. Anupam, J. Freire, B. Kumar, and D. F. Lieuwen. *Automating web navigation with the WebVCR*. *Computer Networks*, 33(1-6):503–517, 2000.
- [5] A. Arasu and H. Garcia-Molina. *Extracting structured data from web pages*. In *SIGMOD Conference*, pages 337–348, 2003.
- [6] P. Atzeni, G. Mecca, and P. Merialdo. *Managing web-based data: Database models and transformations*. *IEEE Internet Computing*, 6(4): 33–37, 2002.
- [7] Z. Bar-Yossef, I. Keidar, and U. Schonfeld. *Do not crawl in the DUST: Different URLs with similar text*. *TWEB*, 3(1):3, 2009.
- [8] Z. Bar-Yossef and S. Rajagopalan. *Template detection via data mining and its applications*. In *WWW*, pages 580–591, 2002.
- [9] R. Baumgartner, M. Ceresna, and G. Ledermuller. *Deep Web navigation in web data extraction*. In *CIMCA/IAWTIC*, pages 698–703, 2005.
- [10] R. Baumgartner, S. Flesca, and G. Gottlob. *Visual web information extraction with Lixto*. In *VLDB*, pages 119–128, 2001.
- [11] E. Baykan, M. Henzinger, L. Marian, and I. Weber. *A comprehensive study of features and algorithms for url-based topic classification*. *TWEB*, 5(3):15, 2011.

- [12] E. Baykan, M. R. Henzinger, L. Marian, and I. Weber. *Purely URL-based topic classification*. In *WWW*, pages 1109–1110, 2009.
- [13] E. Baykan, M. R. Henzinger, and I. Weber. *Web page language identification based on URLs*. *PVLDB*, 1(1):176–187, 2008.
- [14] F. Beil, M. Ester, and X. Xu. *Frequent term-based text clustering*. In *KDD*, pages 436–442, 2002.
- [15] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. *Support vector clustering*. *Journal of Machine Learning Research*, 2:125–137, 2001.
- [16] A. L. Berger, S. D. Pietra, and V. J. D. Pietra. *A maximum entropy approach to natural language processing*. *Computational Linguistics*, 22(1):39–71, 1996.
- [17] C. Bertoli, V. Crescenzi, and P. Merialdo. *Crawling programs for wrapper-based applications*. In *IRI*, pages 160–165, 2008.
- [18] S. Bhagat, G. Cormode, and I. Rozenbaum. *Applying link-based classification to label blogs*. In *WebKDD/SNA-KDD*, pages 97–117, 2007.
- [19] L. Blanco, M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti. *Automatically building probabilistic databases from the Web*. In *WWW*, pages 185–188, 2011.
- [20] L. Blanco, V. Crescenzi, and P. Merialdo. *Efficiently locating collections of web pages to wrap*. In *WEBIST*, pages 247–254, 2005.
- [21] L. Blanco, V. Crescenzi, and P. Merialdo. *Structure and semantics of data-intensive web pages: An experimental study on their relationships*. *J. UCS*, 14(11):1877–1892, 2008.
- [22] L. Blanco, N. Dalvi, and A. Machanavajjhala. *Highly efficient algorithms for structural clustering of large websites*. In *WWW*, pages 437–446. ACM, 2011.
- [23] J. Blythe, D. Kapoor, C. A. Knoblock, K. Lerman, and S. Minton. *Information integration for the masses*. *J. UCS*, 14(11):1811–1837, 2008.
- [24] N. Bruno and S. Chaudhuri. *Exploiting statistics on query expressions for optimization*. In *SIGMOD Conference*, pages 263–274, 2002.

- [25] S. Chakrabarti. *Focused web crawling*. In *Encyclopedia of Database Systems*, pages 1147–1155. 2009.
- [26] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. M. Kleinberg. *Automatic resource compilation by analyzing hyper-link structure and associated text*. *Computer Networks*, 30(1-7):65–74, 1998.
- [27] S. Chakrabarti, M. van den Berg, and B. Dom. *Focused crawling: A new approach to topic-specific web resource discovery*. *Computer Networks*, 31(11-16):1623–1640, 1999.
- [28] V. Chandola, A. Banerjee, and V. Kumar. *Anomaly detection: A survey*. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.
- [29] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. *A survey of web information extraction systems*. *IEEE Trans. Knowl. Data Eng.*, 18(10):1411–1428, 2006.
- [30] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. *Structured Databases on the Web: Observations and Implications*. *SIGMOD Record*, 33(3):61–70, 2004.
- [31] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papanikolaou, J. D. Ullman, and J. Widom. *The TSIMMIS project: Integration of heterogeneous information sources*. In *IPSJ*, pages 7–18, 1994.
- [32] C.-M. Chen and N. Roussopoulos. *Adaptive selectivity estimation using query feedback*. In *SIGMOD Conference*, pages 161–172, 1994.
- [33] W. W. Cohen. *Improving a page classifier with anchor extraction and link analysis*. In *NIPS*, pages 1481–1488, 2002.
- [34] J. Conallen. *Modeling web application architectures with UML*. *Commun. ACM*, 42(10):63–70, 1999.
- [35] V. Crescenzi and G. Mecca. *Automatic information extraction from large websites*. *J. ACM*, 51(5):731–779, 2004.
- [36] V. Crescenzi, G. Mecca, and P. Merialdo. *RoadRunner: Towards automatic data extraction from large web sites*. In *VLDB*, pages 109–118, 2001.

- [37] D. R. Cutting, J. O. Pedersen, D. R. Karger, and J. W. Tukey. *Scatter/gather: A cluster-based approach to browsing large document collections*. In *Research and Development in Information Retrieval*, pages 318–329, 1992.
- [38] H. Davulcu, J. Freire, M. Kifer, and I. V. Ramakrishnan. *A layered architecture for querying dynamic web content*. In *SIGMOD Conference*, pages 491–502, 1999.
- [39] G. T. de Assis, A. H. F. Laender, M. A. Gonçalves, and A. S. da Silva. *Exploiting genre in focused crawling*. In *SPIRE*, pages 62–73, 2007.
- [40] L. M. de Campos, J. M. Fernández-Luna, J. F. Huete, and A. E. Romero. *Probabilistic methods for link-based classification at INEX 2008*. In *INEX*, pages 453–459, 2008.
- [41] D. de Castro Reis, P. B. Golgher, A. S. da Silva, and A. H. F. Laender. *Automatic web news extraction using tree edit distance*. In *WWW*, pages 502–511, 2004.
- [42] I. F. de Viana, I. Hernández, P. Jiménez, C. R. Rivero, and H. A. Sleiman. *Integrating deep-web information sources*. In *PAAMS*, pages 311–320, 2010.
- [43] M. M. Deza and E. Deza. *Encyclopedia of distances*. Springer, edition 3, 2012.
- [44] A. Doan and A. Y. Halevy. *Semantic integration research in the database community: A brief survey*. *AI Magazine*, 26(1):83–94, 2005.
- [45] S. T. Dumais and H. Chen. *Hierarchical classification of web content*. In *SIGIR*, pages 256–263, 2000.
- [46] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. *Data exchange: semantics and query answering*. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
- [47] E. Fersini, E. Messina, and F. Archetti. *Enhancing web page classification through image-block importance analysis*. *Inf. Process. Manage.*, 44(4):1431–1447, 2008.
- [48] R. Z. Frantz, A. M. R. Quintero, and R. Corchuelo. *A domain-specific language to design enterprise application integration solutions*. *Int. J. Cooperative Inf. Syst.*, 20(2):143–176, 2011.

- [49] T. Friedman, D. Newman, D. Feinberg, and W. Andrews. *Cool vendors in data management and integration*. Gartner Research, 2007.
- [50] J. Fürnkranz. *Hyperlink ensembles: a case study in hypertext classification*. *Information Fusion*, 3(4):299–312, 2002.
- [51] 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*. *Inf. Sci.*, 180(10): 2044–2064, 2010.
- [52] L. Getoor, E. Segal, B. Taskar, and D. Koller. *Probabilistic Models of Text and Link Structure for Hypertext Classification*. In *IJCAI Workshop on Text Learning: Beyond Supervision*, 2001.
- [53] T. Gottron. *Clustering template based web documents*. In *European Colloquium on IR Research*, pages 40–51, 2008.
- [54] A. Gounaris, N. W. Paton, A. A. A. Fernandes, and R. Sakellariou. *Adaptive query processing: A survey*. In *BNCOD*, pages 11–25, 2002.
- [55] P. Grünwald. *Advances in minimum description length: Theory and applications*. MIT Press, 2005.
- [56] I. Hacking. *An Introduction to Probability and Inductive Logic*. Cambridge University Press, 2001.
- [57] A. Y. Halevy. *Answering queries using views: A survey*. *VLDB J.*, 10(4): 270–294, 2001.
- [58] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. *Data integration: The teenage years*. In *VLDB*, pages 9–16, 2006.
- [59] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. *Accessing the deep web*. *Commun. ACM*, 50(5):94–101, 2007.
- [60] H. He, W. Meng, Y. Lu, C. T. Yu, and Z. Wu. *Towards deeper understanding of the search interfaces of the deep web*. In *WWW*, pages 133–155, 2007.
- [61] R. Hecht-Nielsen. *Neural networks for perception (vol. 2)*. chapter Theory of the backpropagation neural network, pages 65–93. Harcourt Brace & Co., 1992.

- [62] I. Hernández, C. R. Rivero, D. Ruiz, and J. L. Arjona. *An experiment to test url features for web page classification*. In *PAAMS*, pages 109–116, 2012.
- [63] I. Hernández, C. R. Rivero, D. Ruiz, and R. Corchuelo. *A tool for link-based web page classification*. In *CAEPIA*, pages 443–452, 2011.
- [64] I. Hernández, C. R. Rivero, D. Ruiz, and R. Corchuelo. *An architecture for efficient web crawling*. In *CAiSE Workshops*, pages 228–234, 2012.
- [65] I. Hernández, C. R. Rivero, D. Ruiz, and R. Corchuelo. *A statistical approach to url-based web page clustering*. *WWW Companion*, pages 525–526. ACM, 2012.
- [66] I. Hernández, C. R. Rivero, D. Ruiz, and R. Corchuelo. *Towards discovering conceptual models behind web sites*. In *ER*, 2012. TBP.
- [67] I. Hernández, H. A. Sleiman, D. Ruiz, and R. Corchuelo. *A conceptual framework for efficient web crawling in virtual integration contexts*. In *WISM*, pages 282–291, 2011.
- [68] I. Hernández, H. A. Sleiman, D. Ruiz, and R. Corchuelo. *A tool for web links prototyping*. In *ICAI*, pages 951–957, 2011.
- [69] L. Hirsch. *How big is e-commerce?* 2002.
- [70] A. Holmes and M. Kellogg. *Automating functional tests using selenium*. In *AGILE*, pages 270–275, 2006.
- [71] A. Hotho, A. Maedche, and S. Staab. *Ontology-based text document clustering*. *KI*, 16(4):48–54, 2002.
- [72] A. Hotho, S. Staab, and G. Stumme. *WordNet improves text document clustering*. In *SIGIR Semantic Web Workshop*, pages 541–544, 2003.
- [73] *Z formal specification notation: syntax, type system and semantics*, 2002.
- [74] Z. G. Ives, A. Y. Halevy, and D. S. Weld. *Adapting to source properties in processing data integration queries*. In *SIGMOD Conference*, pages 395–406, 2004.
- [75] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, 1988.



- [76] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.
- [77] M.-Y. Kan and H. O. N. Thi. *Fast webpage classification using URL features*. In *CIKM*, pages 325–326, 2005.
- [78] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. John Wiley and Sons, 1990.
- [79] 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.
- [80] J. M. Kleinberg. *Authoritative sources in a hyperlinked environment*. *J. ACM*, 46(5):604–632, 1999.
- [81] C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. *Accurately and reliably extracting data from the web: A machine learning approach*. *IEEE Data Eng. Bull.*, 23(4):33–41, 2000.
- [82] W. Koehler. *An analysis of web page and web site constancy and permanence*. *JASIS*, 50(2):162–180, 1999.
- [83] H. S. Koppula, K. P. Leela, A. Agarwal, K. P. Chitrapura, S. Garg, and A. Sasturkar. *Learning URL patterns for webpage de-duplication*. In *WSDM*, pages 381–390. ACM, 2010.
- [84] G. Korfiatis and G. Paliouras. *Modeling web navigation using grammatical inference*. *Applied Artificial Intelligence*, 22(1&2):116–138, 2008.
- [85] M. Kovacevic, M. Diligenti, M. Gori, and V. M. Milutinovic. *Recognition of common areas in a web page using visual information: a possible application in a page classification*. In *IEEE International Conference on Data Mining*, pages 250–257, 2002.
- [86] N. Kushmerick. *Wrapper verification*. *World Wide Web*, 3(2):79–94, 2000.
- [87] O.-W. Kwon and J.-H. Lee. *Text categorization based on k-nearest neighbor approach for web site classification*. *Inf. Process. Manage.*, 39(1):25–44, 2003.
- [88] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. *A brief survey of web data extraction tools*. *SIGMOD Record*, 31(2):84–93, 2002.

- [89] J. P. Lage, A. S. da Silva, P. B. Golgher, and A. H. F. Laender. *Automatic generation of agents for collecting hidden web pages for data extraction*. *Data Knowl. Eng.*, 49(2):177–196, 2004.
- [90] A. Y. Levy, A. Rajaraman, and J. J. Ordille. *Querying heterogeneous information sources using source descriptions*. In *VLDB*, pages 251–262, 1996.
- [91] J. Li, K. Furuse, and K. Yamaguchi. *Focused crawling by exploiting anchor text using decision tree*. In *WWW*, pages 1190–1191, 2005.
- [92] S. W. Liddle, D. W. Embley, D. T. Scott, and S. H. Yau. *Extracting data behind web forms*. In *ER (Workshops)*, pages 402–413, 2002.
- [93] J. B. MacQueen. *Some methods for classification and analysis of multivariate observations*. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.
- [94] J. Madhavan, L. Afanasiev, L. Antova, and A. Y. Halevy. *Harnessing the deep web: Present and future*. In *CIDR*, 2009.
- [95] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Y. Halevy. *Google’s deep web crawl*. *PVLDB*, 1(2):1241–1252, 2008.
- [96] L. M. Manevitz and M. Yousef. *One-class SVMs for document classification*. *Journal of Machine Learning Research*, 2:139–154, 2001.
- [97] R. McCann, B. K. AlShebli, Q. Le, H. Nguyen, L. Vu, and A. Doan. *Mapping maintenance for data integration systems*. In *VLDB*, pages 1018–1030, 2005.
- [98] G. Mecca, S. Raunich, and A. Pappalardo. *A new algorithm for clustering search results*. *Data Knowl. Eng.*, 62(3):504–522, 2007.
- [99] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. *YALE: rapid prototyping for complex data mining tasks*. In *KDD*, pages 935–940, 2006.
- [100] P. Montoto, A. Pan, J. Raposo, F. Bellas, and J. Lopez. *Web navigation sequences automation in modern websites*. In *DEXA*, pages 302–316, 2009.
- [101] S. Mukherjea. *Discovering and analyzing world wide web collections*. *Knowl. Inf. Syst.*, 6(2):230–241, 2004.

- [102] J. Neville, D. Jensen, and B. Gallagher. *Simple estimators for relational bayesian classifiers*. In *ICDM*, pages 609–612, 2003.
- [103] K. Nigam, A. K. Mccallum, S. Thrun, and T. Mitchell. *Text classification from labeled and unlabeled documents using em*. In *Machine Learning*, pages 103–134, 1999.
- [104] D. P and D. Khemani. *Unsupervised learning from URL corpora*. In *COMAD*, pages 128–139, 2006.
- [105] A. Pan, J. Raposo, M. Álvarez, J. Hidalgo, and Á. Viña. *Semi-automatic wrapper generation for commercial web sources*. In *Engineering Information Systems in the Internet Context*, pages 265–283, 2002.
- [106] G. Pant and P. Srinivasan. *Learning to crawl: Comparing classification schemes*. *ACM Trans. Inf. Syst.*, 23(4):430–462, 2005.
- [107] G. Pant and P. Srinivasan. *Link contexts in classifier-guided topical crawlers*. *IEEE Trans. Knowl. Data Eng.*, 18(1):107–122, 2006.
- [108] H.-S. Park and C.-H. Jun. *A simple and fast algorithm for k-medoids clustering*. *Expert Syst. Appl.*, 36(2):3336–3341, 2009.
- [109] I. Partalas, G. Paliouras, and I. P. Vlahavas. *Reinforcement learning with classifier selection for focused crawling*. In *ECAI*, pages 759–760, 2008.
- [110] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. *Translating web data*. In *VLDB*, pages 598–609, 2002.
- [111] X. Qi and B. D. Davison. *Web page classification: Features and algorithms*. *ACM Comput. Surv.*, 41(2), 2009.
- [112] S. Raghavan and H. Garcia-Molina. *Crawling the Hidden Web*. In *WWW*, 2001.
- [113] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. *Benchmarking data exchange amongst semantic-web ontologies*. In *TKDE*, pages 1613–1618, 2011.
- [114] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. *Generating SPARQL executable mappings to integrate ontologies*. In *ER*, pages 118–131, 2011.

- [115] H. Schütze and C. Silverstein. *Projections for efficient document clustering*. In *SIGIR*, pages 74–81, 1997.
- [116] A. Selamat and S. Omatu. *Web page feature selection and classification using neural networks*. *Inf. Sci.*, 158:69–88, 2004.
- [117] P. Senkul and S. Salin. *Improving pattern quality in web usage mining by using semantic information*. *Knowl. Inf. Syst.*, 30(3):527–541, 2012.
- [118] D. Shen, Z. Chen, Q. Yang, H.-J. Zeng, B. Zhang, W.-Y. Ma, and Y. Lu. *Web-page classification through summarization*. In *SIGIR*, pages 242–249, 2004.
- [119] D. J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman and Hall/CRC, edition 5, 2011.
- [120] L. K. Shih and D. R. Karger. *Using urls and table layout for web classification tasks*. In *WWW*, pages 193–202, 2004.
- [121] H. A. Sleiman and R. Corchuelo. *A survey on region extractors from web documents*. *IEEE Trans. Knowl. Data Eng.*, 99, 2012. TBP.
- [122] P. Sneath and R. Sokal. *Numerical taxonomy. the principles and practice of numerical classification*. Freeman, 1973.
- [123] C. Tao, D. W. Embley, and S. W. Liddle. *Focih: Form-based ontology creation and information harvesting*. In *ER*, pages 346–359, 2009.
- [124] S. Thakkar, J. L. Ambite, and C. A. Knoblock. *Composing, optimizing, and executing plans for bioinformatics web services*. *VLDB J.*, 14(3):330–353, 2005.
- [125] O. Thonggoom, I.-Y. Song, and Y. An. *Semi-automatic conceptual data modeling using entity and relationship instance repositories*. In *ER*, pages 219–232, 2011.
- [126] R. Trillo, L. Po, S. Ilarri, S. Bergamaschi, and E. Mena. *Using semantic techniques to access web data*. *Inf. Syst.*, 36(2):117–133, 2011.
- [127] J. Turmo, A. Ageno, and N. Català. *Adaptive information extraction*. *ACM Comput. Surv.*, 38(2), 2006.
- [128] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and J. M. B. Cavalcanti. *Structure-based crawling in the Hidden Web*. *J. UCS*, 14(11):1857–1876, 2008.

- [129] K. Vieira, A. S. da Silva, N. Pinto, E. S. de Moura, J. M. B. Cavalcanti, and J. Freire. *A fast and robust method for web page template detection and removal*. In *CIKM*, pages 258–267, 2006.
- [130] W3C. *Uniform resource identifier URI: Generic syntax*. <http://www.ietf.org/rfc/rfc3986.txt>, 2005.
- [131] J. Weiss. *Aligning relationships: Optimizing the value of strategic outsourcing*. Technical report, IBM, 2005.
- [132] W. Xie, M. A. Mammadov, and J. Yearwood. *Using links to aid web classification*. In *ACIS-ICIS*, pages 981–986, 2007.
- [133] L. Xu and D. W. Embley. *Combining the best of global-as-view and local-as-view for data integration*. In *ISTA*, pages 123–136, 2004.
- [134] R. Xu and D. C. W. II. *Survey of clustering algorithms*. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.
- [135] Y. Yang and X. Liu. *A re-examination of text categorization methods*. In *SIGIR*, pages 42–49, 1999.
- [136] Y. Y. Yao, H. J. Hamilton, and X. Wang. *Pageprompter: An intelligent web agent created using data mining techniques*. In *Rough Sets and Current Trends in Computing*, pages 506–513, 2002.
- [137] Z. Zhang, B. He, and K. C.-C. Chang. *Understanding web query interfaces: Best-effort parsing with hidden syntax*. In *SIGMOD Conference*, pages 107–118, 2004.
- [138] M. Zhu, W. Hu, O. Wu, X. Li, and X. Zhang. *User oriented link function classification*. In *WWW*, pages 1191–1192, 2008.
- [139] S. Zhu, K. Yu, Y. Chi, and Y. Gong. *Combining content and link for classification using matrix factorization*. In *Research and Development in Information Retrieval*, pages 487–494, 2007.



This document was typeset on October 1, 2012 at 09:59 using class **RC-BOOK**  $\alpha$ 2.14 for **L<sup>A</sup>T<sub>E</sub>X**<sub>2 $\epsilon$</sub> . 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](mailto:contact@tdg-seville.info)