



HAL
open science

Deriving semantic objects from the structured web

Marilena Oita

► **To cite this version:**

Marilena Oita. Deriving semantic objects from the structured web. Other [cs.OH]. Télécom ParisTech, 2012. English. NNT : 2012ENST0060 . tel-01124278

HAL Id: tel-01124278

<https://pastel.hal.science/tel-01124278>

Submitted on 6 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité “Informatique et Réseaux”

présentée et soutenue publiquement par

Marilena OITA

le 29 October 2012

Inférer des objets sémantiques du Web structuré

Directeur de thèse: **Pierre SENELLART**

Jury

M. Stéphane GANÇARSKI, Maître de conf. Université Paris 6
M. Julien MASANÈS, PDG Internet Memory
M. Paolo MERIALDO, Prof. Université Roma Tre
M. Elie NAJM, Prof. Télécom ParisTech
M. Pierre SENELLART, Maître de conf. Télécom ParisTech
M. Marc TOMMASI, Prof. Université Lille 3

Rapporteur

Rapporteur

Directeur de thèse

TELECOM ParisTech

école de l'Institut Télécom - membre de ParisTech

Inférer des Objets Sémantiques du Web Structuré

Marilena Oita

Résumé

Cette thèse se concentre sur l'extraction et l'analyse des objets du Web, selon différents points de vue: temporel, structurel, sémantique.

Nous commençons par une étude qui porte sur la compréhension des différentes stratégies et meilleures pratiques pour inférer les aspects temporels des pages Web. Pour cette finalité, on présente plus en détail une approche qui utilise des statistiques sur les flux du Web.

Nous continuons par la présentation de deux techniques basées sur des mots-clés pour l'extraction d'objets, dans le cadre des pages Web générées dynamiquement par des systèmes de gestion du contenu. Les objets que nous étudions dans ce contexte correspondent à des articles du Web. Les mots-clés, acquis automatiquement par une analyse *Tf-Idf* des pages Web ou extraits en utilisant les flux Web associés à ces pages, guident le processus d'identification d'objets, soit au niveau d'une seule page Web (SIGFEED), soit sur plusieurs pages différentes qui partagent le même modèle (FOREST).

Nous décrivons également un cadre général qui vise à découvrir le modèle sémantique des objets du Web caché. Dans ce contexte, l'objets sont représentés par des enregistrements de données. Ce cadre utilise FOREST pour l'identification des enregistrements dans la page et se base sur l'alignement des instances extraites et des objets mêmes, par rapport à des similitudes de type représentées avec *rdf:type* dans un graphe étiqueté. Ce graphe est ensuite aligné avec une ontologie générique comme YAGO, pour la découverte des types et leur relations par rapport à l'entité de base qui est résumé par le formulaire Web.

Mots Clefs: extraction de l'information, recherche d'information, détection de changement, Web profond, ontologies

Termes: algorithmes, expérimentation

Cette thèse est rédigée en anglais, mais nous fournissons un résumé en français dans la Partie 4 de ce manuscrit.

Deriving Semantic Objects from the Structured Web

Marilena Oita

ABSTRACT

This thesis focuses on the extraction and analysis of Web data objects, investigated from different points of view: temporal, structural, semantic.

We first survey different strategies and best practices for deriving *temporal* aspects of Web pages, together with a more in-depth study on Web feeds for this particular purpose, and other statistics.

Next, in the context of dynamically-generated Web pages by content management systems, we present two keyword-based techniques that perform article extraction from such pages. Keywords, either automatically acquired through a *Tf-Idf* analysis, or extracted from Web feeds, guide the process of object identification, either at the level of a single Web page (SIGFEED), or across different pages sharing the same template (FOREST).

We finally present, in the context of the deep Web, a generic framework that aims at discovering the semantic model of a Web object (here, data record) by, first, using FOREST for the extraction of objects, and second, representing the implicit *rdf:type* similarities between the object attributes and the entity of the form as relationships that, together with the instances extracted from the objects, form a labeled graph. This graph is further aligned to an ontology like YAGO for the discovery of the unknown types and relations.

Keywords: Information Extraction, Information Retrieval, Change Detection, Deep Web, Ontologies

Terms: Algorithms, Experimentation

This thesis is written in English, excepting for Part 4, where we provide a French extended abstract.

Contents

| | |
|---|-------------|
| List of Tables | xi |
| List of Figures | xiii |
| List of Algorithms | xv |
| | |
| I. Introduction | 3 |
| § 1. Research Context | 5 |
| § 2. Outlining Contributions | 6 |
| § 3. Global Vision | 9 |
| | |
| II. Contributions | 13 |
| | |
| 1. Deriving Temporal Aspects from Web Pages | 15 |
| § 1. Web Dynamics | 15 |
| § 2. Survey | 17 |
| § 2.1. Static: Timestamping | 17 |
| § 2.2. Comparative Methods | 20 |
| § 2.2.1. Document Models | 20 |
| § 2.2.2. Similarity Metrics | 23 |
| § 2.2.3. Types of Changes | 26 |
| § 2.2.4. Change Representation | 27 |
| § 2.3. Statistical: Change Estimation | 28 |
| § 2.4. Towards an Object-Centric Model for Change Detection | 30 |
| § 3. Web Feeds | 31 |
| § 3.1. Ephemeral Content | 31 |
| § 3.2. Feed Files Structure | 31 |
| § 3.3. Characteristics | 32 |
| § 3.4. Web Page Change Detection | 33 |
| § 4. Perspectives | 37 |

| | |
|---|-----------|
| 2. Web Objects Extraction | 39 |
| § 1. Context | 39 |
| § 2. Related Work | 40 |
| § 3. Signifier-based Methods | 45 |
| § 3.1. Signifiers | 45 |
| § 3.2. Acquisition | 45 |
| § 4. Preliminary Notions | 47 |
| § 5. SIGFEED | 48 |
| § 5.1. Intuitions and Implementation | 48 |
| § 5.2. Experiments | 53 |
| § 6. FOREST | 56 |
| § 6.1. Introduction | 56 |
| § 6.2. Methodology | 56 |
| § 6.2.1. Structural Patterns | 57 |
| § 6.2.2. Informativeness Measure | 59 |
| § 6.2.3. Combining Structure and Relevance | 62 |
| § 6.3. Experiments | 64 |
| 3. Discovering the Semantics of Objects | 73 |
| § 1. The Deep Web | 73 |
| § 2. Related Work | 75 |
| § 3. Envisioned Approach | 78 |
| § 3.1. Form Analysis and Probing | 78 |
| § 3.2. Record Identification | 79 |
| § 3.3. Output Schema Construction | 79 |
| § 3.4. Input and Output Schema Mapping | 81 |
| § 3.5. Labeled Graph Generation | 81 |
| § 3.6. Ontology Alignment using PARIS | 82 |
| § 3.7. Form Understanding and Ontology Enrichment | 83 |
| § 4. Preliminary Experiments | 84 |
| III. Discussion | 87 |
| § 1. Conclusions | 89 |
| § 2. Further Research | 90 |
| IV. Résumé en français | 95 |
| § 1. Contexte de recherche | 97 |
| § 2. Description des contributions | 99 |

Contents

Contents

| | |
|--|------------|
| § 3. Extraction du contenu pertinent | 102 |
| § 4. FOREST | 105 |
| § 5. Autres directions de recherche étudiées | 114 |
| Bibliography | 115 |

List of Tables

- 1.1. Dataset feed types 33
- 1.2. Feed statistics per domain 37

- 2.1. SIGFEED experimental results 55
- 2.2. FOREST: mean precision, recall, and their corresponding F_1 -measure . . 68

List of Figures

| | |
|--|----|
| 0.1. Global vision | 11 |
| 1.1. Summary of <i>comparative</i> change detection approaches | 29 |
| 1.2. Number of items per Web feed | 36 |
| 1.3. Quartile value of interval between feed updates (logarithmic scale) . . . | 36 |
| 2.1. Two sibling news pages (BBC <i>travel</i> blog) | 41 |
| 2.2. Match of keywords in a Web article | 46 |
| 2.3. A typical Web article and its corresponding data item from the feed . . . | 51 |
| 2.4. The incidence of signifiers coming from the article title | 52 |
| 2.5. Simplified example of two LCBAAs | 54 |
| 2.6. The <i>pos vectors</i> across three sample pages | 60 |
| 2.7. Example of DOM element types occurring in terminal paths | 60 |
| 2.8. Article extraction: effectiveness results for all the considered techniques | 67 |
| 2.9. Evolution of the F_1 measure in function of the (maximum) number of pages considered, for each Web source, to share a similar layout | 67 |
| 2.10. Evolution of the F_1 measure in function of the (maximum) number of signifiers used in the process of DOM path selection | 70 |
| 2.11. Influence of J and U on mean precision, recall, and the F_1 -measure . . . | 71 |
| 3.1. Deep Web form understanding and ontology enrichment: overview of the envisioned approach | 80 |
| 3.2. Triples generation using deep Web IE results | 85 |

List of Algorithms

- 1. Feed-based object extraction 49
- 2. FOREST: *Tf-Idf* keyword acquisition 57
- 3. FOREST: structural clustering of significant DOM elements 57
- 4. FOREST: ranking of structural patterns 63

Acknowledgments

I'd like to say thanks to the many people who made this thesis either possible or enjoyable and an enriching experience.

I am extremely grateful to my PhD supervisor, Pierre Senellart, for offering me his continuous advice and academic experience. I cannot thank him enough for the opportunities he has given me to learn and for the freedom I had in my exploration. The passion that he invests in his work is a real inspiration to me.

This is an occasion to thank two other persons who had a big influence in the way I was perceiving the research in the first place and who gave me the idea and possibility to start a thesis: Serge Abiteboul and Alban Galland.

During the last part of my research, I had the occasion to collaborate with Antoine Amarilli, and I want to thank him for his implication and constructive arguments.

I am also grateful to Julien Masanès, Fabian Suchanek, Jean Louis Dessalles for interesting discussions about my research.

I would like to thank my English teachers, in particular James Benenson for his modernity in thinking and for being close to his students.

I am indebted to my friends from Télécom ParisTech for the stimulating environment that they created. Thank you Asma for the morning pancakes and coffee. Thanks Nora and Imen for the kind words. Special thanks to Sylvain, Jo, Simon, Mike and Hayette for making Télécom a fun place. Thank you Hoa and Antoine S. for the interesting discussions during the 4pm breaks. Thank you all for putting up with me!

I am particularly thankful to Fabian Suchanek who showed me an invaluable support. I totally appreciated his enthusiasm, curiosity and great company.

I am also grateful to many persons from the Internet Memory Foundation: Julien Masanès, Leïla, Nathalie, France, Chloé and Radu. Beside their kindness and hospitality, they also provided me with a computer to work on during the PhD and experimental datasets.

I wish to thank the members of the jury for their interest and time, and in particular to Paolo Merialdo and Stéphane Gançarski for proof-reading the preliminary version of this thesis. Their feedback was precious in making this manuscript a better work.

Big thanks to my family for being there for me. Having my sister Nico near for 10 months was a source of motivation and happiness.

Most importantly, I want to thank my beloved Damien for his support and for accom-

panying me on this journey. He helped me see things from a different perspective and filled these years with wonderful moments, I therefore dedicate this thesis to him.

Lastly, thank you all for sharing with me cups of tea along the way.

Part I.
Introduction

§ 1. Research Context

Ephemeral Web content The need for fully automatic techniques to index, query, and process Web pages for the extraction of significant content is growing with the size of the Web. Ways to control the information explosion and to make sense out of it is an important axis of research, whose relevance increases with everyone's possibility of producing information.

In the "communication revolution", distributing content so that it can have the greatest impact is one of the main concerns of content producers. On the other hand, finding the best information for one's needs by monitoring¹ and filtering publishing sources based on interests² is essential for a Web content consumer.

In this context, which underlies a multi-optimization problem, Web feeds have a great potential.

More and more present on the Web, Web feeds (in either RSS or Atom formats), are XML-based documents that characterize the dynamic part ("hub") of a Web site. By revealing a short summary of each new article which is published on a Web site, Web feeds help subscribers to keep up with what is the most recent and interesting Web content for them.

We will show in this thesis their usefulness not only in the identification of relevant sources of information and monitoring, but also for Web content extraction.

The Structured Web The Web's continuous growth is caused by the multiplication of sources, but more importantly to the automation of content generation means. Web content of a particular source (e.g., Web site) is indeed getting more dynamic and difficult to "catch", but, at the same time, exhibits more structural patterns, a consequence of the usage of common templates.

However, considering in the analysis various sample Web pages that share the same structural patterns increases in practice not only the possibility of relevant content precise identification, but also its semantic annotation. Having the fully automatic means to exploit the patterns of the Structured Web for the enrichment of the Semantic Web, and conversely, discover the semantics behind structural pattern similarities, would have a big impact in the way information is processed on the Web.

The Web's dynamic nature is visible through its content, which is continuously added, deleted, or changed. Since not all changes are significant ones, a special attention must be paid to the *type of content* that is affected by the change in order to avoid wasting precious time and space processing.

From the semantic point of view, a Web page can include various topically-coherent parts or, on the contrary, a single subject can spread over various pages of the same Web

¹http://www.timeatlas.com/web_sites/general/easily_monitor_web_page_changes

²http://www.timeatlas.com/web_sites/general/how_to_use_rss_in_your_job_search

site. Despite of the obvious Web page content heterogeneity, current information retrieval systems still indivisibly operate at the level of a Web page, this latter being considered the basic information unit. We aim at presenting throughout this thesis the importance of operating at Web data object level, at least in the analysis stage.

A Web data object represents for us a logical fragment in a single Web page or which spreads across various pages of the same Web site, that has a well-defined structure and a semantically-coherent topic. We study in this thesis ways to automatically identify data objects in Web pages and to analyse them depending on their various dimensions, e.g., temporal, semantic.

The Semantic Web From the point of view of humans, the Web represents the largest knowledge base ever built. However, for automated agents such as Web crawlers, the semantics that humans inherently recognize is not explicit. Semantics has to be then provided, or directly inferred during the processing. We will develop in this thesis the vision of a hybrid solution, that has, in our opinion, most chances to adapt to the complexity of Web topics and their evolution.

The most popular choice in assisting programs for semantics discovery is to provide them, a priori of any actual analysis on the data, a specific domain knowledge, that is adapted to the data source. Assuming a fixed domain schema, the use of specific ontologies, either manually or semi-automatically built, does not scale in practice to the size or to the heterogeneity of the Web.

In this thesis, we advocate for the exploration of the rich context created by Web objects generated by the same source *in collaboration* with a generic ontology. Towards this achievement, we propose a generic framework in which the data semantics would not be given from the beginning, but inferred. Due to the shared properties of objects (i.e., subject, schema), it could be possible to first, discover the type and other attributes of Web objects that are already canonically described in an existent generic ontology, and second, infer new semantic infrastructures to be integrated to the ontology.

§ 2. Outlining Contributions

Chapter 1 reviews major approaches for assessing dynamics of Web pages through extraction or estimation of their temporal properties (e.g., timestamps, frequency of change). We focus our attention on techniques and systems that have been proposed in the last ten years and we analyze them to get some insight into the practical solutions and best available practices. We aim at providing an analytical view of the range of methods for assessing dynamics of Web pages, distinguishing them into several types, based on their static or dynamic nature. For dynamic methods relying on comparison of successive versions of a page, we detail the Web page modeling and the similarity metrics used in the process.

We continue with a more in-depth study on Web feeds, considered as instruments for Web detection. A Web feed usually includes timestamp metadata for the Web pages to which it is associated. By monitoring a corpus of Web feeds and continuously crawling their new items during a specific time period, we present some statistics and the resulting observations on the refreshing strategies of the Web channels that we monitored.

As not all changes that occur in a Web page are relevant, main content identification happens to be a key point for efficient change detection. The intuitions that we get from the study on Web feeds is that, using the feed's structured metadata, we can infer some valuable keywords that can be further used for the extraction of interesting content from associated Web pages.

Chapter 2 presents the main contribution of this thesis related to the identification and extraction of Web data objects from Web pages. The ideas and practices that we present in this chapter have been fully implemented and extensively experimented, using state-of-the-art methods and various baselines.

Web objects issue from data-intensive Web sites. Content management systems usually generate these data-intensive Web sites, e.g., of blog posts, news items, forum or dictionary entries, tweets, etc. The simplest vision of a Web object is a fixed-template structure filled with variable content. A data object also has the property of being structurally minimal, while expressing the maximum of informative content. A vast source of objects is also the deep Web. Here, "hidden" behind Web forms, Web pages are dynamically-generated during form submission. These pages contain a rather structured form, presenting data records in response to the user query. The Web pages that populate the *deep Web* have mostly been left unindexed by search engine crawlers due to the missing *a priori* knowledge about the type of objects that the Web interface models.

Unsupervised techniques for the extraction of Web objects generally leverage the structural similarities of Web pages in which objects occur in order to exclude the template that is common to all (i.e., static layout). This is the case for wrapper induction, a popular technique in information extraction for record data extraction. However, due to the sophistication of templates and the rather common practice of including dynamic components such as scripts and applications, a standard approach in wrapper induction is not sufficient on its own to identify the content of interest. Also, we are extending the type of a data object from standard response records (e.g., products) to Web articles, where the information is less obviously structured, but follows the same generation principles.

The techniques that we develop for object identification are keyword-based. Chapter 2 describes first SIGFEED, a bottom-up, DOM-based strategy that uses feed *meta* clues to identify, at HTML DOM level, the element that contains the article of interest. The data sources that we consider for SIGFEED are Web sites that summarize their new content through feed items. SIGFEED extracts the full content of interest from the Web pages

referenced by the items of a feed.

Further in this chapter, we use the intuitions obtained by studying the problem of object extraction using Web feeds in a more general context. Independently of the existence of Web feeds, we present FOREST algorithm that receives as input various Web pages that share a similar template and outputs a generic XPath expression that targets the Web article of each. The hypothesis of having various sample pages available is current in the literature; this condition is indeed easily achieved through a prior structural clustering of pages in a Web site.

FOREST, for “*Focused Object Retrieval by Exploiting Significant Tag paths*”, is a keyword-based method that performs Web page main content extraction. This algorithm identifies, for each of the input Web pages, DOM nodes that are relevant to a bag of keywords (automatically acquired through a *Tf-Idf* analysis). Due to the structural similarity of the input pages, location patterns of these nodes overlap. We acquire then frequently occurring *types* of DOM elements that contain, partially or redundantly, the information of interesting. As these elements are hierarchically distributed, the challenge becomes to identify the boundaries of the Web article, knowing that keywords can pinpoint different regions of a Web page (e.g., comments, related articles, etc.). We introduce a measure of relevance to rank these element types. The measure of informativeness takes into account the keyword density of a given DOM element across different Web pages, but also its frequency of occurrence and its level in the DOM. We improve the results in practice by adding the notion of content “unexpectedness”.

We believe that the ideas and measures presented in FOREST for object extraction are also applicable to the deep Web context, although more extensive experiments should be done in order to thoroughly verify this hypothesis.

Chapter 3 considers the extraction of data objects from the deep Web and their semantic annotation, while aiming at presenting a novel perspective on deep Web comprehension.

Traditionally, a specific domain knowledge has been assumed in order for automatic agents to understand the content that they have to crawl and submit the form for valid response pages. However, that approach does not scale: it is difficult to imagine how we could, in a non-preferential way, achieve the extraction of new and meaningful data without constantly adapting the virtual schema to changes in and across Web sources. Chapter 3 presents the vision and preliminary results of a framework that achieves the fully automatic and *generic* annotation of data objects.

We have prototyped an application that performs the semantic description of Web objects obtained through the *Amazon* advanced search form. Similar to pages summarized by feed items, pages obtained through from submission present a clear structural similarity, but in contrast, tend to exhibit the various objects (i.e., response records) in a list form, possibly on the same response page. We apply FOREST for record identification and data extraction in this prototype. Keywords that we had previously acquired through

a classical *Tf-Idf* analysis come here directly from the user *query terms*. An important observation is that query terms are reflected in the response records if the form submission was valid, which creates the perfect conditions for FOREST. This is however only a small step in the workflow that we present: next, two other typical steps in information extraction are accomplished: *attribute alignment* for output schema construction, and *attribute labeling* for the semantic annotation of objects.

An interesting feature of a Web form is that of validating the input terms. Using the form as an instrument of validating form input-output schema mappings, we distinguish ourselves from other techniques by the result of this mapping, which mingles data with schema in a labeled graph. Here, we represent the data values extracted from objects as literals in a labeled graph. This graph also includes important implicit *rdf:type* similarities not only between aligned data values, but also between the the response objects themselves. Unknown types and relationships are then discovered using an ontology alignment algorithm on this graph

Outlining the main contributions of this thesis

1. a survey on strategies and best practices for deriving *temporal aspects* of Web pages, together with a more in depth study on Web feeds for this particular purpose (Chapter 1);
2. two keyword-based techniques for *article extraction* from dynamically-generated Web pages; keywords, either automatically acquired or extracted from Web feeds, guide the process of relevant content extraction (Chapter 2);
3. a generic framework in which deep Web form understanding, data object semantics discovery and ontology enrichment cross-fertilize (Chapter 3).

§ 3. Global Vision

Generally, the aim of our thesis was to add exploitable meaning at finer-grained levels to collections of Web content, in order for them to become more expressive and adapted to user needs. In Figure 0.1, various modules that are discussed throughout this thesis are put together: the acquisition of Web pages in time using feed streams and identification of different temporal features using methods defined in Chapter 1, main content extraction using keyword-based techniques presented in Chapter 2, and objects' semantics discovery following the vision plan outlined in Chapter 3.

The content of this thesis is based on works already published or under review, and more specific details will be given at the beginning of each chapter. We do not provide a separate chapter for the related work, but prefer to refer to it as research context at the beginning of each treated subject.

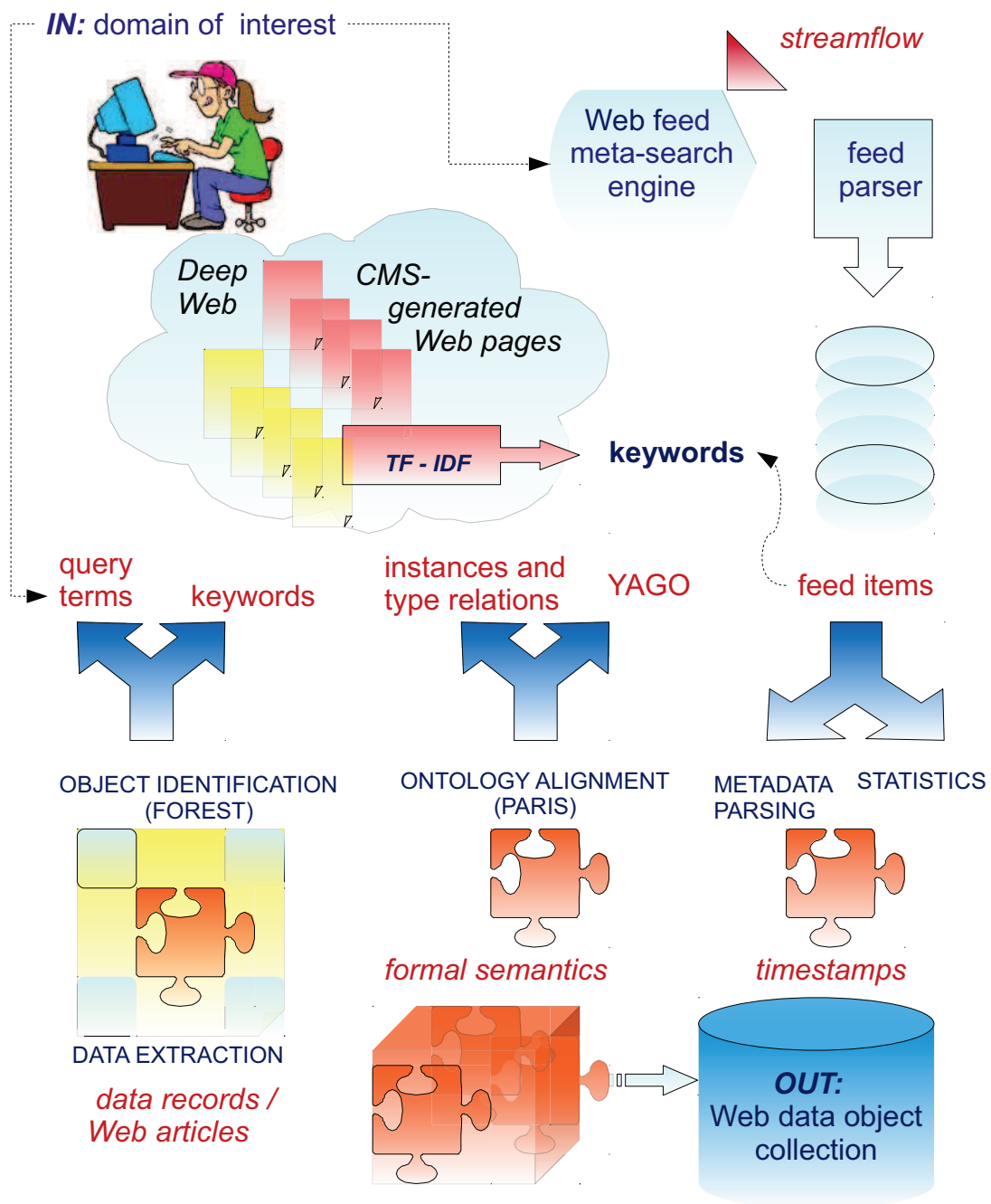


Figure 0.1.: Global vision

Part II.
Contributions

1. Deriving Temporal Aspects from Web Pages

We review in this chapter major approaches to change detection in Web pages and extraction of temporal properties (especially, timestamps) of Web pages. We focus our attention on techniques and systems that have been proposed in the last ten years and we analyze them to get some insight into the practical solutions and best practices available. We aim at providing an analytical view of the range of methods that can be used, distinguishing them on several dimensions, especially, their static or dynamic nature, the modeling of Web pages, or, for more dynamic methods relying on comparison of successive versions of a page, the similarity metrics used. We study more in detail the method using Web feeds for change detection, and finally highlight the need for detecting change at Web object level.

The content of this chapter is the aggregation of a survey paper [Oita and Senellart, 2011] on the deriving of temporal properties from Web pages and another publication [Oita and Senellart, 2010b], in which we study the value of Web feeds in the change detection process, and make some other statistics on feeds.

§ 1. Web Dynamics

The World Wide Web challenges our capacity to develop tools that can keep track of the huge amount of information that is getting modified at speed rate. The ability to capture the temporal dimension of textual information on the Web is essential to many natural language processing (NLP) applications, such as Question Answering, Automatic Summarization, and Information Retrieval (IR).

The subject of inferring temporal aspects is of interest in various applications and domains, such as: large-scale information monitoring and delivery systems [Douglis et al., 1998; Flesca and Masciari, 2007; Jacob et al., 2004; Lim and Ng, 2001; Liu

et al., 2000] or services¹, Web cache improvement [Cho and Garcia-Molina, 2000], version configuration and management of Web archives [Pehlivan et al., 2009], active databases [Jacob et al., 2004], servicing of continuous queries [Abiteboul, 2002], etc.

These applications use change detection techniques with the same aim as temporal processing at textual level, but at semi-structured data level instead. The temporal dimension of Web content can be computed in a *static* manner, by deriving it using means inherent to the Web page itself (see § 2.1, or in a more *dynamic* manner which implies comparisons of versions of the same Web page or data object (see § 2.2) or through estimations (see § 2.3).

The bridge between these methods is more visible in Web crawling applications. In this field, which is closely related to Web archiving, substantial efforts are made to maximize the freshness of a Web collection or index.

Performing a coherent crawl of Web sites that are updating their supply of information very rapidly is not an easy task. The common choice of crawling in *snapshots*, i.e., crawling the entire collection at distant (but frequent enough) intervals of time, can be quite expensive in terms of network bandwidth, and in the end, both redundant for some zones of the site, and incomplete for other zones that are more dynamic. Additionally, as running an integral crawl takes time and in the meanwhile the resource might change, it raises issues of temporal incoherence of a given snapshot [Spaniol et al., 2008].

On the other hand, an incremental crawler crawls once a full version of the site and, from that point, future crawls will capture only the content that has been modified meanwhile (supposing the existence of a trigger that informs the crawler that content has been added or updated). An important issue is to clarify what “frequent enough” means, that is, how often new Web pages are added or current ones are modified, and to make this change detection effective, knowing that there are a lot of factors that influence the detection process.

There are multiple studies [Adar et al., 2009; Baeza-Yates et al., 2004; Douglis et al., 1997; Fetterly et al., 2003; Ntoulas et al., 2004] in the literature that try to understand *the dynamics of Web pages*, i.e., how fast the Web content changes, the nature of the changes, its implications on the structure of the Web, and the possible correlations that exists, for instance with the topic of the Web pages. These studies are delivering results concerning the analysis of large Web crawls during an extended period of time. They have already chosen a method of assessing change at Web page level and respond directly to these enumerated questions.

Our study in this chapter is, in its first phase, a survey: we outline the methods available, discuss their nature and use cases. We stress the importance of relevant change detection and make a more in depth analysis on Web feeds from this specific point of view.

¹Examples of these include <http://timelyweb.en.softonic.com/>, <http://www.urlywarning.net/>, <http://www.changealarm.com/>, <http://www.changedetect.com/>.

§ 2. Survey

We perform in this section an analysis of different state-of-the-art strategies for change detection.

We mention that there is a large body of work on the related problem of change detection in XML documents [Cobéna et al., 2002; Wang et al., 2003], particularly for purposes of data integration and update management in XML-centric databases. However, the solutions developed for XML documents cannot be applied without revisions for HTML pages. While an XML document defines, by its tags, the semantics of the content within, HTML deals with the presentation of content. In addition to the challenges that exist in XML documents, Web pages add some more by their lack of formal semantics, a fuzziness regarding the formatting or by the embedding of multimedia and scripts. We refer the reader to [Cobéna and Abdesslem, 2009] for a survey of XML change detection algorithms. There are also works which study the problem of change detection on \LaTeX documents [Chawathe et al., 1996], but the focus on our survey is on HTML documents.

We begin by illustrating the main approaches for change detection:

1. *static*: infer the date of last modification of content from the Web page itself or from its neighborhood;
2. *comparative*: compare successive versions of a Web page in order to detect if a change has occurred;
3. *estimative*: approximate the change frequency by using statistics or learning.

§ 2.1. Static: Timestamping

Quoting [Maršić, 2011], *temporal processing* is “a field of Computational Linguistics which aims to access the temporal dimension and derive a precise temporal representation of a *natural language text* by extracting time expressions, events and temporal relations, and then representing them according to a chosen knowledge framework”.

We overview in this part of the thesis methods for inferring temporal properties of Web pages by directly applying temporal processing methods on the page itself, or through any cross-analysis of it (neighborhood, common descriptive documents attached, etc.). We do not cover, however, the extraction of temporal relations, not their representation in a specific framework, although we believe this is a very interesting line of research.

We next refer to methods that operate at the level of a Web page (or its extensions, e.g., metadata) as *static*, as opposed to the more *dynamic*, commonly-used difference computation between successive versions of a Web page. The ultimate goal of this type of analysis is to infer the creation or the last modification date of a Web page (or, possibly, some parts of it).

HTTP metadata HTTP/1.1, the main protocol used by Web clients and servers to exchange information, offers several features of interest for timestamping, the foremost of which are the ETag and Last-Modified HTTP response headers. The canonical way for timestamping a Web page is to use the Last-Modified HTTP header. Unfortunately, studies have shown this approach is not reliable in general [Clausen, 2004].

Entity tags (or ETags) are unique identifiers for a given version of a particular document. They are supposed to change if and only if the document itself changes. Servers can return this with any response, and clients can use the If-Match and If-None-Match HTTP requests headers to condition the retrieval of the document to a change in the ETag value, avoiding then to retrieve already known contents. If-Modified-Since and If-Unmodified-Since provide conditional downloading features, in a similar way as for ETags. Even when conditional downloading is not possible, Etags and HTTP timestamps can be retrieved by a Web client without downloading a whole Web page by making use of the head HTTP method. The problem is that, while this information is generally provided and is very reliable for static content (e.g., static HTML pages or PDF), it is most of the time missing or changed at every request (the timestamp given is that of the request, not of the content change) when the content is dynamic (generated by content management systems, etc.). Some CMSs do return correct HTTP timestamps, such as MediaWiki², but they seem to be a minority.

[Clausen, 2004] presents an experimental study of the reliability of Etags and HTTP timestamps on a collection of a few million Danish Web pages. He finds out that the best strategy for avoiding useless downloads of versions of Web pages already available in a Web archive is to always download when the ETag server is missing, and otherwise download only if the Last-Modified header indicates change. This rather counterintuitive result yielded in this experimental study an almost perfect prediction of change, and a 63% accuracy in predicting non-change. Given that the majority of Web servers run some version of the open-source Apache³ HTTP server [Netcraft, 2011], it would be interesting to see whether this strategy is correlated with some inherent behavior of this software. Furthermore, repeating this experiment on a larger scale and with a more recent set of Web pages would be of interest.

HTTP also provides the Cache-Control and Expires response headers. This information is often given, but with a zero or very low expiration delay, which means that nothing interesting can be derived from it. In some specific and controlled environments (e.g., Intranets), it might still be useful to look at these two pieces of information to estimate the refresh rate of a Web page.

Timestamps in Web content CMSs, as well as Web authors, often provide in the content of a Web page some human-readable information about its last date of modification.

²<http://www.mediawiki.org/>

³<http://www.apache.org/>

This can be a global timestamp (for instance, preceded by a “Last modified” string, in the footer of a Web page) or a set of timestamps for individual items in the page (such as news stories, blog posts, comments, etc.). In the latter case, the global timestamp might be computed as the maximum of the set of individual timestamps.

It is actually quite easy to recognize and extract such information, either with date entity recognizers based on regular expressions, or built-in temporal taggers⁴. However, this timestamp is often quite informal and partial: there is sometimes no time indication, and most of the time no timezone.

To the best of our knowledge, no formal study of the precision reached by extracting timestamps from Web content has been carried out.

Semantic temporal associations In addition to these timestamps provided to humans, documents on the Web may include additional semantic timestamps meant for machines. No mechanism for this exists in HTML *per se* (excluding recent additions to the HTML 5 standard, with the <time> tag and its `pubdate` attribute, the specification of which is not yet entirely finalized), but the HTML specification [W3C, 1999] allows for arbitrary metadata in the form of <meta> tags, one particular profile of such metadata being Dublin Core⁵ whose `modified` term indicates the date of last modification of a Web page. Both content management systems and Web authors occasionally use this possibility.

When attached to Web pages, descriptive documents can be used for dating an HTML Web page. Web *feeds* (in RSS or Atom formats) also have semantic timestamps, which are quite reliable, since they are essential to the working of applications that exploit them, such as feed readers. In this case, a Web feed containing descriptions (i.e., items) about the Web pages of a particular site can be used in this direction. More details on the temporal aspects of feeds can be found in [Oita and Senellart, 2010b].

Another approach is based on *sitemaps* [sitemaps.org, 2008]. Sitemaps are files that can be provided by the owner of a Web site to describe its organization, so as to improve its indexing by Web search engines. Sitemaps allow for both timestamps and change rate indications (*hourly*, *monthly*, etc.), but these features are not often used. Very few content management systems produce all of this, although it would have been the ideal case: the download of a single file would suffice to get all timestamping information about the whole Web site.

Using the neighborhood When no source of reliable timestamps is found for a given page using one of the technique described above, its timestamp is set to some form of average of the timestamps of pages pointed to and by this page. Using the graph structure of the Web, timestamping a Web page can be estimated according to the timestamps of other pages with whom this particular page is in relation. The hypothesis of such

⁴<http://dbs.ifi.uni-heidelberg.de/index.php?id=129>

⁵<http://dublincore.org/documents/dcmi-terms/>

an approach is that pages linked together tend to have a similar update patterns. The precision is not very high [Nunes et al., 2007], but better than nothing when no other information is available.

§ 2.2. Comparative Methods

We do not have the means to infer in advance the perfect frequency of crawl for a given Web page or site. When change concerns the same Web page, and it is not related to new content that appear on a specific Web site, then a method to derive the dynamics of it is to perform a *frequent-enough* crawl. An a posteriori analysis of this crawl compares the versions to detect the frequency of the change and its nature. The majority of works in the literature that study change detection consider that they have an access to an archive of Web pages, and focus on techniques that can efficiently compare successive versions of pages.

Static techniques of timestamping that we have previously described can be incomplete (e.g., timestamps appear in a non-consistent manner), or not applicable (e.g., timestamps are missing or give incoherent results). In this case, timestamps can be roughly approximated to *Last-Modified*, by deriving them from the crawl interval as the date where the change was identified. Of course, the smaller the crawl interval, the better approximation of a timestamp we get.

The frequency of change is difficult to estimate due to the fact that Web pages have different patterns of change. Many factors determine variations in the frequency of change for a given Web page: the CMS, the subject of writing, the time of the year (even the time of the day), etc. Estimating the frequency of change based on a previous crawl of Web pages during a more or less extended period, is the subject of many studies [Adar et al., 2009; Cho and Garcia-Molina, 2000; Fetterly et al., 2003; Jatowt et al., 2007; Ntoulas et al., 2004]. The reported results are however quite variable, as multiple methods exist, and the technique of change detection utilized has important consequences on the:

1. *dimension of change*: content, structural, presentational, behavioral, etc.;
2. *description of change*: qualitative (the types of changes that have occurred) or quantitative (a number depicting the magnitude of change);
3. *complexity* of the method.

§ 2.2.1. Document Models

We overview here the typical Web page models that are frequently considered in change detection algorithms that perform comparison of successive versions of Web pages. We first discuss the simplest model of a Web page, that is, flat files (strings); then we describe tree models, a popular choice in the literature. Finally, we present approaches based on a

Page Digest manner of encoding Web pages which clearly separates the structure of Web documents from their content, while remaining highly compact and reversible.

Flat-files Some early change detection systems model Web pages as flat files [Douglass et al., 1998]. As these models do not take into account the hierarchical structure of HTML documents and neither the characteristics of the layout, they can detect only *content changes* – and this without making any semantic difference in the content.

Some works [Sigurðsson, 2005] try to filter first irrelevant content by using heuristics on the type of content and regular expressions. After this basic filtering, the Web page content is directly hashed and compared between versions. Unfortunately, we can never filter all kind of inconvenient Web content, especially since its type and manner of HTML encoding evolve in time.

Trees An intuitive approach to represent Web pages is to refer to their DOM (Document Object Model) and process them as trees. By taking into account the tree hierarchy and the attributes of the nodes, more types of changes can be detected [Buttler, 2004]. The differences that appear in methods that use the tree model of a Web page to detect change appear in the:

1. *ordered* characteristic of the tree;
2. *granularity* i.e., the level at which change is detected (node, branch, subtree or “object”).

Pre-processing First, the modeling of a Web page into a tree requires a pre-processing step of cleaning. This corrects missing or mismatching, out-of-order end tags, as well as all other syntactic ill-formedness of the HTML page and typically transform it into a XML-compliant document. Sometimes also a pruning of elements is done; for instance, [Khoury et al., 2007] filters out scripts, applets, embedded objects and comments. The end result is manipulated using implementations of the DOM standard (that usually employ XSLT and XPath).

Many works do not specify how they realize the cleaning, so either they assume it to be done in advance, or they solve this issue by using an HTML cleaning tool [Artail and Fawaz, 2008] like HTML Tidy⁶. However, there are also techniques that operate on trees that do not need the structure to be enforced [Lim and Ng, 2001] because they operate rather at syntactic level.

⁶<http://tidy.sourceforge.net/>

Ordered trees The *ordered* characteristic of trees implies that the position of nodes or their order of enumeration is important in the model. This is taken into account in various algorithms for model entity identification [Augsten et al., 2005; Khandagale and Halkarnikar, 2010], level order tree transversal and parsing [Yadav et al., 2007] or detecting changes in relevant hierarchy structures [Lim and Ng, 2001].

In the SCD [Lim and Ng, 2001] algorithm the model entity is a branch (or path) that gives the location of a node in the tree, but also its meaning. A data hierarchy is obtained by formalizing the branch as an ordered multiset, in which the nodes appearing in the branch are designated by either their *tag name* if the node is a non-leaf, or by its *text content* otherwise. Take for instance the data hierarchy `book.author.name.Dante`; this is more meaningful than a *markup* hierarchy like `div.p.b.#PCDATA`. At the same time, if we change the order of nodes, the semantic interpretation of the data changes also, producing possible incoherencies.

Unordered trees The *unordered* labeled tree model does not consider the order of appearance of elements in the tree as relevant; instead, only hierarchy-related (i.e., parent-child) relationships are captured. Typically, methods like CMW [Flesca and Masciari, 2007] or MH-Diff [Chawathe and Garcia-Molina, 1997] use an unordered tree model of the Web page to perform adaptations of the Hungarian algorithm. This algorithm is typically used in linear programming to find the optimal solution to an assignment problem. Here, the change detection problem by comparing two trees is transformed into that of finding a minimum cost edge cover on a bipartite graph. The bipartite graph is constructed by considering the trees as two independent sets of entities, each corresponding to a document version. Model entities that populate the two sets represent either nodes or subtrees, possibly acquired after pruning the identical parts of initial trees. Subtrees are chosen over nodes in [Flesca and Masciari, 2007; Khoury et al., 2007]: the argument is that we might be interested more in which subtree the change occurs, rather than in which specific node.

The elements in the bipartite graph are connected through cost edges. The cost of an edge is that of the *edit scripting* needed to make a model entity of the first set *isomorphic* with one of the second set. The similarities between all entities in the first tree and all those of the second tree are computed and placed in a cost matrix.

Having this matrix, the Hungarian [Bertsekas and Castañon, 1993] algorithm is used to find in polynomial time a minimum-cost bijection between the two partitions.

Optimizations are brought out in [Khoury et al., 2007] in comparison with the work in [Bertsekas and Castañon, 1993], although the general aim and similarity metrics remain the same as in [Artail and Fawaz, 2008]. This latter approach works also on unordered trees, but performs a special generation of subtrees based on the level of a node. More specifically, [Artail and Fawaz, 2008] apply a tree segmentation that starts from the root until the third level; then, the node at $(\text{level}\%3 + 1)$ becomes the *local* root

of the following subtree, and so on, iteratively until the end of the hierarchy. Each subtree is marked with the tag name of its local root and indexed based on its start and end node identifiers. A hashtable will index metadata about nodes (like tag name, content, path to the root, attributes pair values, etc.) in order for these features to be efficiently searched when performing a comparison between versions.

The change detection problem for unordered trees is considered to be harder than for ordered ones [Chawathe and Garcia-Molina, 1997].

Page Digest The *Page Digest* model⁷ represents a more compact encoding of data than HTML and XML formats, while preserving all their advantages. To construct the *Page Digest* encoding of a Web page, various general steps need to be performed: node counting, children enumeration in a depth-first manner (tmeant to capture the structure of the document), and content mapping for each node. It is also possible to include other features as tag type and attribute information in the model [Rocco et al., 2003]. Change is then detected by performing comparisons on these features.

The advantages of Page Digest over the DOM model of a Web page are minimality and execution performance through the reduction of tag redundancy, resulting in a more compact model (therefore a faster document traversal). Algorithms using this model [Rocco et al., 2003; Yadav et al., 2007] run in $O(n)$ and make, comparatively with other techniques less heuristics or restrictions in the types of changes detected.

§ 2.2.2. Similarity Metrics

In methods which compare two Web pages to assess their dynamics, the change is detected by identifying *dissimilarity* between versions. This gives rise to the necessity of adequate similarity metrics. We present next some popular choices in the literature. Note that we do not exhaustively cover here all existing techniques for approximate string or tree matching: we only classify from this point of view the main approaches that we study in this chapter.

The basic instance of comparison is, for techniques using a flat-file model of a Web page, a string. For tree models, structural features are also checked for change. Very often, model elements that are the identical between page versions are pruned. Indeed, total dissimilar model elements do not represent instances of the same entity that has evolved. This determines the focus on model elements which are not different for all dimensions, in the same time.

Jaccard One simple technique to verify if two string sequences are similar is to count the number of words that are common (regardless of their position) and to divide the

⁷<http://www.cc.gatech.edu/projects/dis1/PageDigest/>

result by the number of distinct words. Another possibility is to normalize by the length of the first string [Pehlivan et al., 2009].

Hash-based methods The basic idea of hashing is that content that remains identical between versions continue to have the same hash value. By comparing the hashed value of the strings, dissimilarity is detected; the downside of hashing is that if the string slightly changes (say, only a single letter is different), the hash will be totally different [Artail and Fawaz, 2008].

Signatures The signature of an entity is a function of its hashed value. When the model is hierarchical, this function is related to the position of a model entity (e.g., node, branch) in the tree. In [Khandagale and Halkarnikar, 2010], the signature of a non-leaf node represents the sum of the signatures of its children, in a recursive manner, while the signature of a leaf node is directly the hash of its content. Only nodes that have different signatures from those in the reference version of the Web page are compared. Change detection algorithms that employ signatures have the disadvantage that false negatives are possible: change exists, but a different signature for it does not. Therefore they should be rather careful concerning the hashing space, and possibly the application: if it can tolerate false results or not.

Shingling Another method based on hashing, but more flexible is shingling [Broder et al., 1997]. A *shingle* is a contiguous subsequence (w -gram) of the reference string, where w denotes the number of tokens in each shingle in the set. For the two strings to be compared, their shingles are *hashed*. In practice, [Broder, 1997] reduces the resemblance and containment to/of a document in another to a set intersection problem.

Longest common subsequence *Diff* tools are based on computing the longest common subsequence (LCS) between two strings [Sahinalp and Utis, 2004], an NP-hard problem in the general case. A *diff* tool is in general a file comparison program that outputs the differences between two files. For instance, *HTMLDiff*⁸ uses GNU *diff* utility adapted for HTML page change detection. This program models Web pages as strings and, after processing, highlights the changes by merging the two versions in a single document. As mentioned in [Douglis et al., 1998], *HTMLDiff* can consume significant memory and computation resources, and this might have an influence on the scalability of the tool. Another example of system that uses GNU *diff* tool is WebVigiL [Jacob et al., 2004, 2005].

Diff techniques may be also applied on tree models of a Web page, with a preference for subtree model elements, which tend to make the computation less complex. Another option evidenced by WebCQ [Liu et al., 2000] is to use *diff* at *object* level, where the object is user-defined: a fragment of a Web page, specified either by means of regular

⁸<http://htmldiff.codeplex.com/>

expressions or by directly marking for monitoring elements of the DOM tree (e.g., a table, list, links).

Edit scripting The edit distance between two strings of characters represents the number of operations required to transform one string into another. There are different ways of defining an edit distance, depending on which the operations taken into account: delete, insert, etc. In string edit scripting, the atomic element is a single character and the cost is usually unitary, for every edit operation defined. More on the large topic of approximate string matching can be found in [Navarro, 2001].

Edit scripting on trees has a similar formal definition as in the case of strings. However, the edit operations can be node insertion, node deletion or label change. Cost models of edit operations become more complex: the cost can be proportional to the complexity of the operation, or moderated by some constants based on heuristics. The execution of an edit script as a sequence of operations returns a cumulated cost, and the similarity measure is then computed as the inverse of this total cost, with the following interpretation: Less *unimportant* modifications we perform on a data structure in order to make it isomorphic with its previous version, the more similar the two structures are.

The problem of determining the distance between two trees is referred to as the *tree-to-tree correction problem*, more in depth covered in [Barnard et al., 1995].

Regarding the complexity, we note that the edit scripting with moving operations is NP-hard [Chawathe and Garcia-Molina, 1997] in its general case, and therefore seldom considered in the change detection literature. The edit scripting on trees in general has a $O(n^2)$ runtime complexity, where n is the number of nodes. Usually, better complexities can be achieved by suboptimal solutions by introducing model restrictions or computational thresholds. As an example, [Lim and Ng, 2001] computes an edit scripting on branches rather than on trees.

A bottom-up traversal of the tree for distance computation can decrease the complexity to $O(n^2)$, but makes the model very sensitive to differences in the leaf nodes contents. Variants of the tree edit distance are considered in [Lee et al., 2004], where $O(ne)$ is obtained (e being the edit distance between the trees). Approaches exist also to approximate the tree edit distance with a pq -gram distance [Augsten et al., 2005]. The pq -grams of a tree represent all its subtrees of a particular shape, where p is related to the number of ancestors and q to the number of children of the root of a subtree. Here, the similarity between trees is defined in relation to the number of pq -grams that the trees have in common.

Quantitative measures of change give a numerical approximation of the change, rather than a description of it. Various works [Artail and Fawaz, 2008; Flesca and Masciari, 2007; Khoury et al., 2007] use a composed measure of similarity that tries to better adapt to the specific of the types of changes considered.

The change is measured by a formula that incorporates three sub-measures of specific similarity: on the content (*intersect*: the percentage of words that appear in both textual content of subtrees), attributes (*attdist*: the relative weight of the attributes that have the same value in the model elements) and on the types of elements considered in the path (*typedist* emphasizes differences on the tag names when going up in the hierarchy). The final measure incorporates all above-defined types of similarity, together with some parameters that are meant to influence the importance of certain types of changes over others. The advantage of this measure is that it captures the symbiosis of different types of changes that occur in a certain way independently: content changes in leaf nodes, attribute changes in internal nodes; the third submeasure is more focused on the position of nodes in the structure.

Another quantitative measure of change is proposed in [Lim and Ng, 2001]. Here, a weighted measure that determines the magnitude of the difference between two branches represented as ordered multisets is employed. In an ordered multiset, the weight of the i th node is defined as $(2^i)^{-1}$ (where i represents the depth of an element of the branch considered). Finally, the quantity of change is measured by computing the sum of the weights of the nodes that appear in the symmetric difference set.

An original way of compute similarity between two strings is to use RMS (Root Mean Square), i.e., the quadratic mean of the ASCII values of the text characters composing that string. RMS represents a statistical measure for the *magnitude* of a varying quantity, often used in engineering to do an estimation of the similarity between a canonical model and an empirical one (in order to see the precision of the experiment). RMS permits a “tolerant” change quantification in the HTML context as it can ignore the effect of certain special characters or spaces. However, changes concerning the position of a character in a string will not be detected. Variants of this measure are used in [Yadav et al., 2007, 2008].

§ 2.2.3. Types of Changes

We summarize here the types of changes detected by the different comparative approaches that we study. First, the most popular are *content* changes. This type of change is generally considered by all methods, and in particular is the only type of change that is detected by models in which the Web page is seen as simply a string.

Works that model a HTML document as a tree or using page digest encoding are aware of more dimensions (or features) of the Web page; as a result, they tend to detect also other type of changes, typically structural. Unlike flat-file models of Web pages, the output of content change detection in hierarchical models is more meaningful because we can identify the location and other particularities of the node in which the content change occurs.

Structural changes occur when the position of elements in the page is modified. Typically, techniques detect the operations of *insert*, *delete* and *update*. MH-Diff [Chawathe

and Garcia-Molina, 1997] detects also the *move* and *copy* structural changes, which have been traditionally considered difficult to detect (expensive) operations.

Attribute changes modify the representation of information, e.g., font or color changes. There exist also *type* changes, which are modifications that occur when the HTML tags change: e.g., a `p` tag which becomes a `div`. Type changes can be detected by [Rocco et al., 2003]. This algorithm uses the Page Digest model which provides a mechanism for locating nodes of a particular type.

More sophisticated changes are sometimes mentioned [Yadav et al., 2007], but not analyzed: for instance, *behavioral changes*; they occur in active HTML elements like scripts, embedded applications and multimedia. These new forms of Web content have a big impact on the Web today, but they require a more complex modeling.

Semantic changes [Lim and Ng, 2001] capture the meaning of content that has changed. Another meaning for semantic changes is that of detecting only the changes that are relevant in a Web page, depending on a relevance metric defined over a partition (e.g., blocks) of that page [Pehlivan et al., 2009].

§ 2.2.4. Change Representation

There are various ways to present the difference between two documents. Changes are usually stored in a physical structure generically called *delta* file or tree.

In [Yadav et al., 2007], this structure consists in specialized set of arrays that capture the relationships among the nodes and the changes that occur to them.

Systems for monitoring change like [Jacob et al., 2004; Liu et al., 2000] typically have implemented a user interface to present changes in a graphical way. For instance, *HTMLDiff* merge the input Web page versions into one document that will summarize all the common parts and also the changed ones. The advantage is that the common parts are displayed just once, but on the other hand, the resulting merged HTML can be syntactically or semantically incorrect.

Another choice linked to change presentation is to display only the differences and omit the common parts of the two Web pages. When the documents have a lot of data in common, presenting only the differences could be better, with the drawback that the context is missing. The last approach is to present the differences between the old and new version side by side. These presentation modes are used in combination, rather than being the unique choice for a given system. For example, [Jacob et al., 2004; Liu et al., 2000] are presenting the results of the change monitoring service using a hybrid approach: presentation modes are combined depending on the type of change that is presented.

Changes are sometimes *quantitatively* described rather than in a descriptive manner. In contrast with all the previous ways of describing or visualizing the changes, quantitative approaches estimate the amount of change of a specific type. For instance, approaches similar to CMW [Flesca and Masciari, 2007] do not reconstruct the complete sequence

of changes, but give a numerical value of it. In this case, supposing a threshold for change, one can determine if a page has changed or not, still useful in the majority of applications.

§ 2.3. Statistical: Change Estimation

Estimative models In crawling related applications, the interest is more in whether a Web page has changed or not, in order to know if a new version of a Web page should be downloaded.

From this perspective, a simple estimation of the change frequency is as effective as explicitly computing it, as we show in Section § 2.2. If Web crawlers were more aware of the semantics of data they process, they could clearly benefit of a broader, richer insight into the different facets of a Web page and could develop different strategies related to storage and processing.

An estimation of the change rate, although not describing where the change appeared or its type, is still useful, especially when we can imagine a strategy that combines estimative and comparative methods of deriving dynamics. For this reason, we shortly present some of the existing *statistical approaches*.

Poisson model In the attempt to fairly estimate the Web page change rate, a formal model for predicting the optimal recheck for new content is studied in [Cho and Garcia-Molina, 2000]. An important result of the study is that changes that occur on the Web follow a Poisson process. A Poisson process models a sequence of random events that happen with a fixed rate over time. Supposing a(n ideally complete) change history available, the frequency of change is *estimated*, that is, the statistical method predicts when new content is produced.

The time independence of *homogeneous Poisson* models [Cho and Garcia-Molina, 2000, 2003] does not capture the reality on the Web [Sia et al., 2007]. In the case of dynamic Web content (like blogs), the posting rates vary depending on multiple parameters that are dependent one of another. Hence, [Sia et al., 2007] propose an *inhomogeneous Poisson* model, which do not assume that events happen independently and where learning of posting patterns of Web pages is used. To avoid excessive overload on servers of Web sites that provide timely content (e.g., news), [Sia et al., 2007] proposes an *adaptive* aggregator approach. That approach which uses Web feeds is meant to adapt the crawl based on the estimated change frequency.

The work of [Cho and Garcia-Molina, 2003] optimizes the prediction for some typical use cases in which, by adapting the parameters of the Poisson model to the requirements of the application, a better estimation accuracy can be achieved. In contrast with other approaches that totally ignore this real-world aspect, the situation when there is no complete change history of Web pages is analyzed. Indeed, in a perfect setting, change

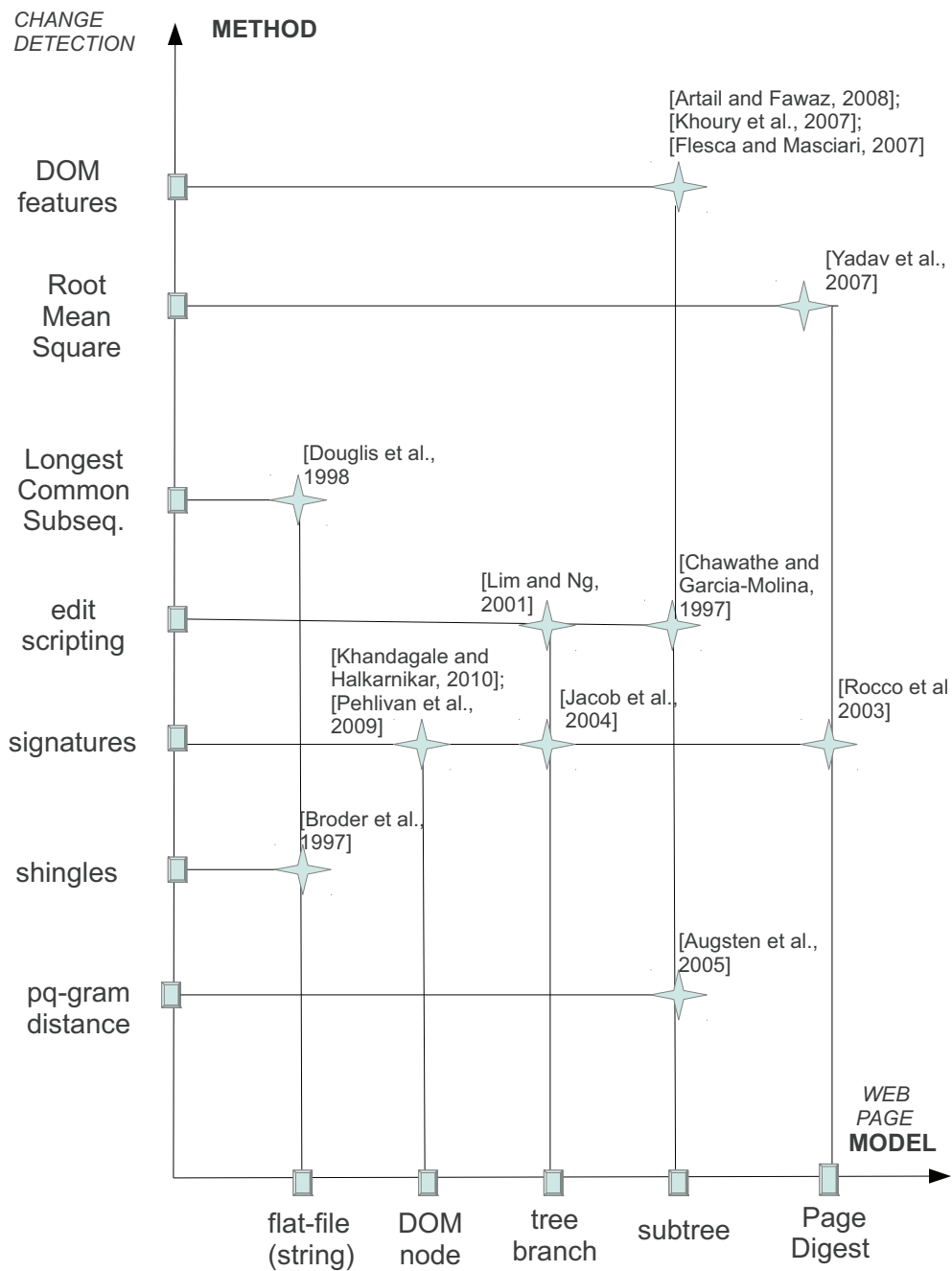


Figure 1.1.: Summary of comparative change detection approaches

patterns can be reliably established only by having a complete change history. However, most of the time, one can only know that a change has occurred or, at best, be aware of the last date of change.

Kalman filters There are tentatives to automatically adapt the crawl to a predicted rate of change: [II et al., 2008] uses a textual vector space model to identify the patterns of the page and to train Kalman filters. In the end, the change is the event that does not match the prediction.

Kalman filters model the evolution of a system and are meant to perform an exact inference in a linear, dynamic system. The possible drawbacks are the hypothesis on the system linearity and the fact that the method uses, for complexity reasons, an incomplete vector space model.

§ 2.4. Towards an Object-Centric Model for Change Detection

All the studied approaches consider that all that is *new* is also *relevant*. This is related to the perspective of change detection, which is rather *document-centric*, than *object-centric*. By *object* we mean here a block or fragment of the Web page that represents meaningful information per se: a news article or blog post, a comment, an image, etc.

Indeed, few techniques make a semantic distinction between changes that appear in different parts of Web pages in order to disregard insignificant ones. Nonetheless, we believe that it is crucial to understand whether changes are relevant, since they may occur because of the dynamic template of a Web page (advertisements, applications, scripts, etc.) and therefore impact these boilerplate parts of the page.

The fact that there exist active components in the Web page that are more or less relevant makes it hard to decide whether the change has been initiated by, for instance, an advertisement or by updating the content of an interesting news article. The idea would be to filter template and other unrelated, but “active” elements before applying any change detection algorithm. The benefit is twofold: on one hand, we avoid running the algorithm over template parts of a Web page, but also we reduce the temporal drift, that is, distorting the Web page frequency of update through the detection of irrelevant changes.

For instance, the open-source *Heritrix* archive crawler [Sigurðsson, 2005] uses regular expressions to filter irrelevant changes. Vi-Diff [Pehlivan et al., 2009] uses the VIPS algorithm [Cai et al., 2003] to get from a Web page an hierarchy of semantic blocks out of a Web page. VIPS uses heuristics on the visual appearance of a Web page to group together content that seems to have similar importance in the page. These semantically related blocks are compared between versions in order to detect structural and content changes.

We have only shortly mentioned in Subsection § 2.1 the use of Web feeds for time-stamping. Next, we analyze more in depth the value of Web feeds in what it concerns change detection.

§ 3. Web Feeds

§ 3.1. Ephemeral Content

Innovation is constantly created on the Web through the collective intelligence of its contributors. Besides the abundance of information, we nowadays face the difficulty to keep track of updated information, as the Web is changing faster than our faculty to perceive it. For this reason, new tools are made available on the Internet in order to remain current with real-time or frequently changing data. To avoid consuming time on information gathering, a generally impacting idea was to bring content from different sources in one place, using streaming techniques. Content is made available by their publishers much easier than before, and users have only to subscribe to it, stating in this way their interest.

Web feeds are about publishing and subscribing to Web content. Quoting [Finkelstein, 2005], Web feeds describe multiple types of Web content: product specifications, opinions on every conceivable subject, software and business tips, press releases (news articles), personal blogs posts, etc. We have encountered in our crawling phase other types of objects that are linked to Web feeds: wiki entries, dictionary terms, forums posts.

Timely content consists of any kind of event information that is linked to a Web feed. This type of content is in expansion on the Web: `syndic8`⁹, that has maintained statistics since 2001, shows the exponential growth of the feed technology. We begin by describing the structure of a Web feed and the meta-elements that we consider relevant in the process of main content identification.

§ 3.2. Feed Files Structure

RSS (acronym for RDF Site Summary, or Really Simple Syndication) and *Atom* are popular feed formats, dialects of XML, that are used to publish frequently updated in a standardized format. A feed can be then parsed using dedicated libraries¹⁰. An RSS or Atom document, also called a *Web feed*, contains a *Web channel* that groups various *items* (RSS) or *entries* (Atom). We will next refer to feed items, but the items and entries represent the same concept.

⁹<http://www.syndic8.com/stats.php>

¹⁰<http://www.davidpashley.com/projects/eddie.w3c99htmlSpec>

In the RSS specification¹¹, each item has three compulsory elements: title, link, and description. While the *title* gives the name of the Web article (usually) as it appears on the referenced Web page through the *link* URL, the *description* is meant to be a short text describing the respective article.

Typically, the *description* of an item often represents the first few lines of the Web article, encoded in HTML for presentation purposes, possibly with a link at the end (like “Read more...”) to the original Web page. In other cases, the description contains only a sentence, which summarizes the article. However, the entities of interest are stated and some reliable keywords of the content of interest can still be extracted. Few are the Web sources that provide the full content of an article through the feed item description. The metadata of a feed item is meant to act like a “teaser” and bring interested people to visualize the full Web page for obvious reasons: creating more traffic to the respective Web site, expose the user to advertisements, etc.

In the Atom standard, the three compulsory elements of an entry (i.e., item) are *id*, *title*, and *updated*. The description, named *summary* in the Atom specification, is only recommended. Though Atom is a more modern standard, it is still less widely deployed than RSS [Oita and Senellart, 2010b].

§ 3.3. Characteristics

Web feeds are a rapidly evolving phenomenon. They synthesize the fresh information that is published by a Web channel, describing the new resources through their *meta-tags*. Web feeds are used for publishers in order to “advertise” Web content, (most of the time, in the form of Web articles) and by user to stay up-to-date. An important characteristic is that feeds are associated with information which is dynamic and informative.

We leverage the two following aspects:

1. item metadata constitute a source of *keywords* and *timestamps* for Web pages;
2. a feed channel naturally group items *of the same type*.

We can then derive the fact that a Web feed groups together structured metadata about the content of interest present in Web pages, pages that are *structurally similar*. This observation is very important in the second part of this chapter, where, similar to wrapper induction techniques, we need an input representing a set of Web pages that share a common HTML template and present structured content of the same type. This kind of sample pages is, nevertheless, often manually collected. There exist algorithms [Chakrabarti and Mehta, 2010; Crescenzi et al., 2005] that structurally cluster Web pages of a Web site. Using the Web feeds, we obtain an automatic approach to collect structurally similar pages.

¹¹<http://cyber.law.harvard.edu/rss/rss.w3c99htmlSpec>

Table 1.1.: Dataset feed types

| Type | Number | Proportion |
|--------------|------------|---------------|
| Atom | 21 | 6.1% |
| RDF | 30 | 8.8% |
| RSS 0.91 | 1 | 0.2% |
| RSS 2.0 | 288 | 84.7% |
| Total | 340 | 100.0% |

§ 3.4. Web Page Change Detection

Web content is dynamic: new information is added or removed at speed rate, and within a short period of time, updates can be very frequent. The lifespan of the Web pages that present this content becomes much smaller than the average, ranging from minutes to days. This causes the content to become “ephemeral”. For instance, when a crawler is not aware of the new content published, and its frequency of crawl is smaller than the content’s rate of change, the respective data is lost and therefore not indexable. In Web archiving for instance, this results in missing snapshots and incoherent versions.

If available, Web feeds can be of use in the analysis of the change detection process. By analyzing the new items of a feed for a period of time, we can identify patterns in the publishing strategy and possibly automatically adapt the crawl of the site having this feed.

We first perform some statistics on the temporal aspects of feeds. A Web feed refers to a primary Web page, the *channel*, which is usually either the home page of the site, or a hub from which we can find links to information presented in the form of Web articles. The rest of the feed describes individual feed items corresponding to new or updated articles.

To state their value in the learning process of a publishing strategy for the *content of interest*, we have crawled over a period of a little more than one month, twice a day, a number of 400 Web feeds with all associated Web pages. We first describe how these feeds were selected and then report some statistics of interest.

Acquisition The set of feeds was collected from the Web by passing in large part through a feed search engine called Search4RSS¹². This search engine returns a number of feeds relevant to a given keyword. We scraped the way the interface returns the results

¹²<http://www.search4rss.com/>

as records to acquire a list of feed URLs.

The keywords chosen in order to probe the search interface are names of domains: *art*, *biology*, *environment*, *medicine*, *science* and *universe*. We have used Wordnet in order to get hyponyms of these terms (for example for art, a hyponym is photography) and constructed a bag of terms representing subdomains. The new terms were used to automatically probe the Search4RSS interface and to construct, for each domain, a list of feed URLs. The purpose of this semi-automatic selection is to get an insight into the diversity of feeds, in terms of formats, update patterns, and types of Web articles. Additionally, to ensure coverage of common blog platforms as well as the more news-oriented results returned by Search4RSS, a number of blog sites were manually selected from a list of “best blogs”¹³.

We thus obtained a list of about 400 sites (this number ensures coverage of the variety of Web feeds, while remaining manageable without any involved archiving infrastructure) that were systematically crawled. At the end of the crawling period, we noticed that some of the feeds had not been updated at all, some others had disappeared, and some could not be parsed. Filtering them out, we obtained an archive of 340 *active* feeds and their associated pages. For each domain, for each followed site, we have stored the feed and the resources associated, mainly the channel page, and the Web pages that were referred to by each item.

Characteristics For the feed analysis, we have used Eddie¹⁴, a feed parsing library for Java, based on a SAX-based parser capable of parsing the ill-formed XML. Eddie supports the standard RSS, Atom, and RDF formats for feeds. The `FeedData` structure returned by this parser can be exploited to extract all kinds of useful information about the channel and its composing items. In particular, for the channel we have metadata like language, tagline, description and title. For an item component of a channel, we encounter the same type of metadata, plus author and categories of that article.

Feed types Let us first look at the types of feed formats that we encountered in our dataset. As shown on Table 1.1, most feeds use the 2.0 dialect of RSS, while a minority use Atom or RDF. RSS 0.91 was only used once among the 340 feeds, and RSS 1.0 never at all, which might suggest the coming obsolescence of these two feed formats. However, it is possible for these numbers to be biased by the use of Search4RSS as our main source for feeds.

Number of items We have looked at the number of items that were presented in a given feed. Indeed, though it is theoretically possible for a feed to refer to all previously published items, this is rarely done in practice so to limit the size of the resulting feed

¹³<http://bloggerschoiceawards.com/>

¹⁴<http://www.davidpashley.com/projects/eddie.w3c99htmlSpec>

file. In effect, most feeds present only the k most recent items, where items that are evolving together with new content on the Web site to which the feed refers to. We show in Figure 1.2 a histogram of the number of items per feed in the dataset. Roughly 75% of the feeds present information about less than 30 items at a time. The other peaks observed in Figure 1.2 are explained by the “magic” values of $k = 50$ and $k = 100$. If a feed only contains 10 items (the most frequent number), it means that by crawling it twice daily, we could capture a maximum of 20 new articles per day. As we shall see, there is a minority of feeds with a higher update frequency than that, for which some of the updates were missed in our crawl suggesting that some feeds needed to be crawled much more often.

Time-related feed metadata In the RSS specification, and similarly in other feed formats, temporal information can be given through the elements `lastBuildDate`, `tTL`, and `updateFrequency` for the channel, and `pubDate` and `lastModified` for items. From our observations, although `pubDate` is an optional element, it is present in the vast majority of feeds. This is not the case for the other types of time-related elements mentioned above, though `lastBuildDate` can be somehow inferred as the publication date of the most recent item. This observation is important as it shows that feeds can be used to determine when new data is added to a channel and of help in the task of change detection.

Update intervals We have collected all publication dates corresponding to the items appearing during the period of experiments; as an item has one publication date, the number of publication dates is equal to that of items. We are interested in the range of *update intervals* between two publications, as well as indications whether a feed has a *regular publishing strategy*.

In Figure 1.3, we present the median update interval between two publications of each feed as a cumulative plot (in green, in the middle). Note that the x-axis has logarithmic scale. Figure 1.3 shows for instance that 20% of the feeds have a median update interval of less than an hour, and that around 10% of the feeds have a median update interval of exactly one day, which corresponds to feeds having regular, automatic, updates, every day. Globally, it is important to note that there is no typical update interval, and that, even disregarding extreme cases, it can range to less than a hour to more than a week. Figure 1.3 also shows other quantities of the update interval of each feed, which helps to see the diversity of update patterns for a given feed: thus, even though 60% of the feeds have a median update interval of a day or less, less than 10% of them always have at least an update per day, and more than 90% of them have at least had a daily update in the period of observation. There is actually a big gap between the median update interval values and minimum and maximum values, which makes more difficult to predict the next update of a given feed, given the specified period of observation.

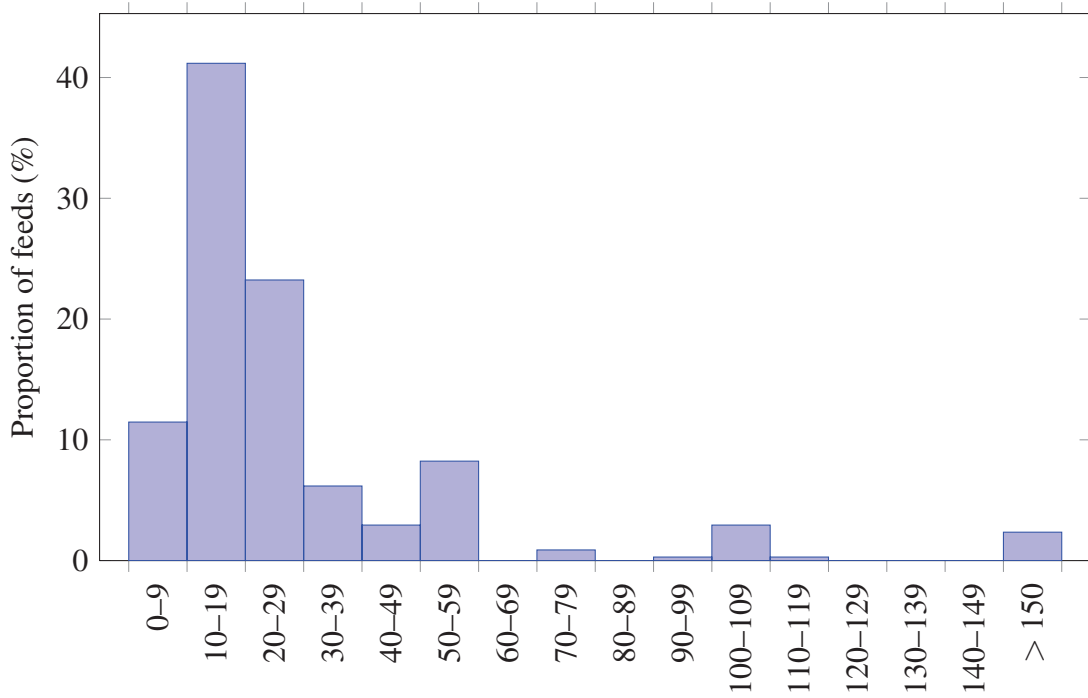


Figure 1.2.: Number of items per Web feed

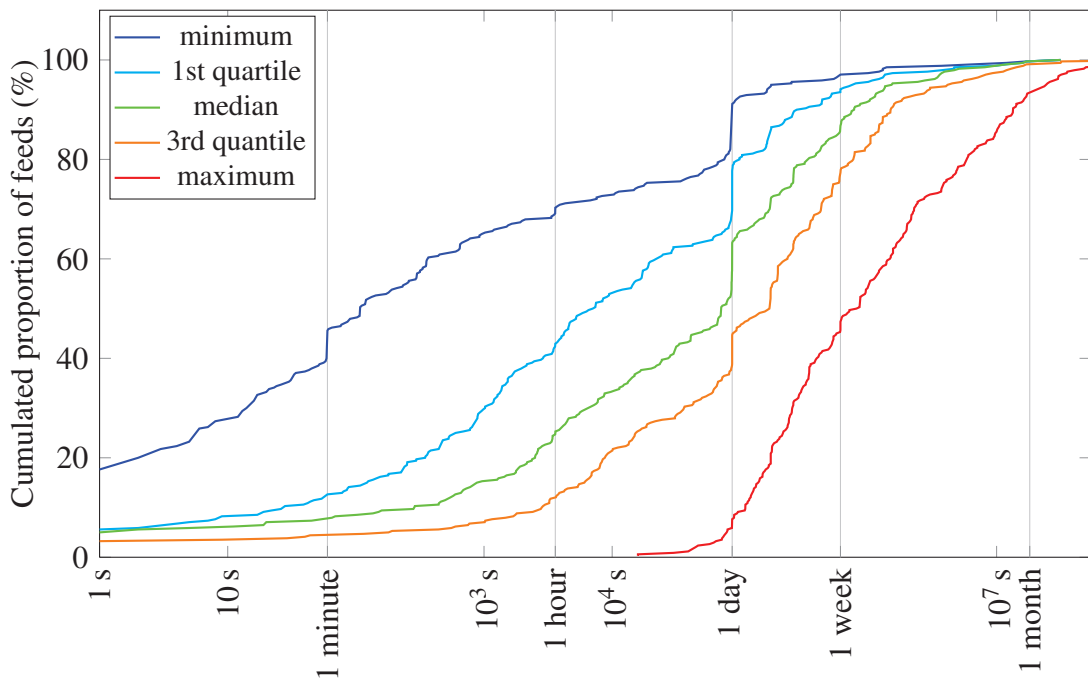


Figure 1.3.: Quartile value of interval between feed updates (logarithmic scale)

Table 1.2.: Feed statistics per domain

| Domain | Number of feeds | Average mean update interval | Pooled standard derivation of update interval |
|--------------|-----------------|----------------------------------|---|
| Art | 87 | 12 days, 14 hours, 12 min | 82 days, 6 hours, 32 min |
| Biology | 80 | 7 days, 13 min | 8 days, 17 hours, 43 min |
| Blogs | 29 | 15 hours, 35 min | 8 hours, 39 min |
| Environment | 7 | 19 hours, 49 min | 4 days, 15 hours, 18 min |
| Medicine | 8 | 3 days, 19 hours, 16 min | 1 day, 22 hours, 43 min |
| Other | 13 | 4 days, 16 hours, 48 min | 4 days, 19 hours, 46 min |
| Science | 112 | 22 days, 12 hours, 45 min | 14 days, 21 hours, 35 min |
| Universe | 4 | 4 hours, 44 min | 7 hours, 5 min |
| Total | 340 | 12 days, 15 hours, 17 min | 37 days, 16 hours, 49 min |

We show in Table 1.2 some other statistics on update intervals at domain level. For each domain, the average mean update interval is given, as is the *pooled standard deviation* of update intervals, which is the statistically-founded way to summarize the deviation of the union of series of numbers. As can be seen, there are huge variations among domains, another indication of the absence of a general, common update interval. We also note here the very high standard deviations in some domains¹⁵, especially since a given domain can represent Web sites of very different types (news items, blogs, wiki entries, etc.). In the *art* domain, for instance, we have observed that there are several sites that publish small articles about paintings or photographs of the order of 100 entries per day.

So the notion of item varies from a specialized article, that might contain a lot of text (as it is the case for news), to an article almost exclusively composed of images or videos. The more structurally homogeneous category of “popular” blogs has a less intriguing deviation of update intervals.

§ 4. Perspectives

Using a technique of temporal analysis on feeds similar to the one presented in the previous section, we can determine the publishing strategy of the feed channel that

¹⁵Note that it is not impossible for a standard deviation to be much greater than a mean value, it just means that there are a few values much greater than the mean in the distribution.

represents the dynamic part of a Web site. Therefore, we could adapt the rate of crawl to the approximated frequency.

As not all changes that occur are significant ones, detecting change at object level is a promising direction. The use of Web feeds for this task is considered next in Chapter 2, which studies relevant article object extraction Web pages. So, besides the fact that Web feeds can help in the extraction of timestamps for obtaining the approximate change rate, they have the advantage of being descriptive enough to ease the extraction of the Web page's relevant part. This improves the quality of change detection when performed as a pre-processing step (before any of the comparative methods that we presented in Subsection § 2.2), by focusing on the changes that occurred on the content of interest, and not on the whole Web page.

2. Web Objects Extraction

Data-intensive Web sites, e.g., blogs or news sites, present pages containing Web articles (a blog post, a news item, etc.). These articles, typically automatically generated by a content management system, use a fixed template and variable content. Unsupervised extraction of their content (excluding the boilerplate of Web pages, i.e., their common template) is of interest in many applications, such as indexing or archiving. Due to the sophistication of the template and the presence of unrelated content in the Web page, a standard approach in unsupervised wrapper induction is not sufficient on its own to identify the content of interest. We present a novel approach for the extraction of Web articles from dynamic Web pages. We present here FOREST and SIGFEED, which both target the zone of the Web page relevant to some (automatically acquired) keywords to obtain structural patterns that may be used to identify the content of interest. We consider two potential source of keywords: Web feeds that may link to the Web page, and terms found (through Tf–Idf) on the Web page itself.

SIGFEED, performing on feeds only, has been published at the IWAW workshop [Oita and Senellart, 2010b], while the more generic system called FOREST is under review to WSDM [Oita and Senellart, 2012].

§ 1. Context

Textual Web content on modern Web sites is, in the overwhelming majority of cases, produced by dedicated content management systems (CMSs). Such software generates Web pages containing textual articles (news items, forum messages, wiki articles, blog posts, tweets, etc.) by filling a template with information fetched from databases. In this process, the original textual or structured content are turned into full-fledged HTML documents, where the Web article is hidden among the markup encoding the site layout [Grumbach and Mecca, 1999].

Some parts of the resulting Web page are thus *meaningful* (they form the Web articles

that are the main content of the Web site), others are *boilerplate* (they just ensure a common layout of the site, or add contextual information, navigation structure, advertisements, comments). Note that boilerplate may change from one page to another and cannot be assumed to be completely static. In addition, boilerplate can actually take up more volume than meaningful information [Gibson et al., 2005]. Distinguishing between main content and boilerplate is a challenging task [Bar-Yossef and Rajagopalan, 2002; Kohlschütter et al., 2010; Pasternack and Roth, 2009; Ramaswamy et al., 2004; Song et al., 2004], with many Web information retrieval and data mining applications: search engines index Web pages based on the informative part of their content; end-users are primarily interested in the main content, and may wish to extract it for readability or accessibility purposes; Web archivists and analysts may wish to archive Web articles independently of the containing Web page [Oita and Senellart, 2010b] to track their evolution irrespectively of changes in layout.

To identify the information of interest, techniques from the literature have considered the extraction of “informative blocks” [Song et al., 2004], pagelets [Bar-Yossef and Rajagopalan, 2002; Caverlee et al., 2004], fragments [Ramaswamy et al., 2004] or articles [Kohlschütter et al., 2010; Pasternack and Roth, 2009] from Web pages. These notions are essentially equivalent: they represent the Web page’s *main content*. A variety of techniques has been used in these works: text-based [Kohlschütter et al., 2010; Pasternack and Roth, 2009], tag-based [Weninger et al., 2010], visual-based [Mehta et al., 2005; Song et al., 2004], or using heuristics on DOM paths [Oita and Senellart, 2010b].

§ 2. Related Work

Relevant content extraction from Web pages is an extensively studied problem [Chang et al., 2006], of use in many applications such as information extraction, data cleaning for data mining and information retrieval, adaptation of Web pages to small devices; it is also the spotlight of many online applications¹. We provide here a brief review of the state of the art.

Automatic wrapper induction An important design choice of modern Web sites, and a consequence of the use of CMSs, is that Web pages from the same site share a common structure, a structure that can be easily traced in the DOM tree of the Web pages [Crescenzi et al., 2005]. To illustrate, Figure § 2 shows an example of two pages from the same Web site presenting variable content within a fixed template. This structural similarity across Web pages of the same site has been leveraged in a number of information extraction techniques to identify data records from data-intensive, somewhat structured Web sites [Arasu and Garcia-Molina, 2003; Buttler et al., 2001; Crescenzi

¹For instance, <http://fivefilters.org/content-only/>



Figure 2.1.: Two sibling news pages (BBC travel blog)

et al., 2001; Liu et al., 2003]. A typical use case is the Deep Web, where given sample response pages that result from the submission of a form (e.g., on e-commerce Web sites), the task is to extract all properties (price, name, availability, etc.) of a given product. On the other hand, these techniques tend not to directly apply to the extraction of Web articles, that are less structured information.

Content extraction has often been formalized as a wrapper induction problem [Kushmerick et al., 1997] for extracting data objects, also known as data records [Arasu and Garcia-Molina, 2003; Liu et al., 2003]. In unsupervised settings, wrapper induction makes use of the common structure of various objects, either at single Web page level [Crescenzi et al., 2001; Liu et al., 2003], or across different Web pages that share the same template [Arasu and Garcia-Molina, 2003; de Castro Reis et al., 2004]. MDR [Liu et al., 2003] and variants [Liu and Zhai, 2005; Zhai and Liu, 2005] compare string paths that have been identified as being under the same *generalized node* that represents the “data region.” A work that compares tag paths as segments rather as strings is [Miao et al., 2009]. Closer to our use case, RTDM [de Castro Reis et al., 2004] uses the *tree edit distance* that leverages the common structure of *news* Web pages in order to identify record patterns. The record identification is performed by, first, detecting the data region using a spectral clustering algorithm, and second, identifying records by subtree intra-clustering. The template tends to incorporate elements that change from page to page (links, ads, etc.): this means the changes in content are not reliable enough to decide that content is informative [Arasu and Garcia-Molina, 2003; Crescenzi et al., 2001]. Also, due to the increasing complexity of HTML pages at DOM level, in the absence of a content

relevance measure, these techniques risk detecting structural patterns that have little of no relevance with respect to the extraction target.

Block-based Web search In the search for relevance, segmentation algorithms partition Web pages into blocks that are further analysed to find meaningful features. Vision-based segmentation is one of the most frequent: VIPS [Cai et al., 2003] or methods such as [Bohunsky and Gatterbauer, 2010; Pehlivan et al., 2009; Zhai and Liu, 2005] have defined a *content relevance measure* using visual clues, that is, heuristics on which humans usually rely upon to identify blocks or portions of Web pages that seem to be more interesting than others. However, as argued in [Weninger et al., 2010], given the fast evolving manners of expressing visual properties of text, visual clues are not always reliable and tend to become obsolete with time. Another drawback in the use of visual cues is that the Web page typically needs to be rendered. Therefore, algorithms that use CAI03VIPS [Bruno et al., 2011; Mehta et al., 2005; Yu et al., 2003] in the pre-processing phase to segment the Web page into blocks that are topically coherent need large memory resources and processing time.

An alternative is to use DOM-based page segmentation. The OMNI [Buttler et al., 2001] system provides an automated way of learning rules that helps identify the *Web object* boundaries. The approach is based on structural features of a tag node (e.g., fanout, size, tag count) combined with heuristics to identify an object separator tag (e.g., repeating tag, standard deviation in the size, etc.). In order to determine whether a Web page block is *informative*, several different measures have been proposed. For instance, [Song et al., 2004] defines a measure of importance for blocks using spatial and content features (e.g., link density or inner text length), but relies on machine learning to identify the best combination of features. We note also [Kao et al., 2005], which uses an entropy measure to determine the importance of page blocks. Entropy is a classical measure for determining the degree of coherence of a system, by measuring its randomness. However, this technique is used independently of the actual relevance of the content on which entropy is applied.

Exploiting keywords Using keywords to extract content of interest is a recent idea in information extraction drawn from information retrieval. Both techniques that we propose, SIGFEED and FOREST are keyword-based.

The query terms occurring in search logs have been proposed as a source of keywords in [Chakrabarti and Mehta, 2010], but with a different goal as ours: to perform an unsupervised structural clustering of Web pages that have been obtained in response to a user query. In order to group pages that are structurally similar, [Chakrabarti and Mehta, 2010] traces the paths of DOM nodes that contain the search logs and performs the clustering only on these paths.

A big advantage of using keywords is computational. In contrast to other tech-

niques [Miao et al., 2009], as not all DOM elements are considered equally important. Filtering only those which are interesting for further analysis drastically reduces computations.

While in FOREST we cluster DOM element types based on their characteristics across different Web pages (and thus essentially use the fact that the paths are not unique), [Chakrabarti and Mehta, 2010] operates only on independent tag paths.

Finally, the technique presented in [Chakrabarti and Mehta, 2010] is only available in the presence of search logs. The access to search logs is however limited either to the Web sites owners or to search engines themselves. In FOREST and SIGFEED, keywords are *automatically* discovered.

Using the Web feeds' semantic clues In SIGFEED we use the Web feeds metadata to extract linguistic clues about the relevant content in a Web page. Using the item's title and description, SIGFEED uses heuristics on the DOM tree to extract the content of the Web page that is referred to by the corresponding item in the feed.

ArchivePress, a blog-archiving project [Pennock and Davis, 2009] has developed a Wordpress plugin that archives posts using Web feeds. The principal drawback is that only the content that can be delivered by the RSS feed is captured. In effect, an RSS feed can have a full coverage of the article and its media files, but this case is rare in practice because a feed is often just a way of advertising content. In contrast, we exploit the feed clues with the goal to extract the full information associated. Additionally, we do not limit ourselves to a blogging platform in particular.

To our knowledge, there is no previous work that leverages feed metadata to extract the relevant part of the Web page, in an automatic manner.

By converting our HTML sample pages in DOM trees in which we look for the occurrence of keywords, we get closer to the rich literature on keyword search over XML documents [Guo et al., 2003; Sun et al., 2007]. Similar to this line of work, our ranking measure of informativeness applies at the granularity of DOM elements, as opposed to other types of ranking operating at document or string level.

However, our goal is different. First, XML keyword search works assume keywords as given, while FOREST and SIGFEED obtain them automatically. Second, the ranking measures significantly differ: the emphasis in [Sun et al., 2007] is on finding the smallest lowest common ancestor (SLCA) that contains all searched keywords, while we are interested in regions that are dense in keywords but which not necessarily contain all of them. We usually also diverge in what it concerns the vision of the ranking measure for the XML elements: for instance, [Guo et al., 2003] uses an adaptation of PageRank.

We compare, in Section § 6.3, FOREST and SIGFEED to a baseline that implements an adapted version of the SLCA [Sun et al., 2007], named COVERAGE.

Text extraction In the task of article extraction, [Cardoso et al., 2011] targets, besides the main text, different components of an article, such as the title, publication date, or author.

By using a technique used primarily in image processing, similar to [Miao et al., 2009], CETR [Weninger et al., 2010] computes, per line of HTML code of the Web page, a tag ratio array. The resulting matrix can be fed to a histogram clustering algorithm which filters extraneous data (i.e., boilerplate).

A number of recent works [Kohlschütter et al., 2010; Pasternack and Roth, 2009] aim at extracting the *textual content* of a Web article in a Web page by relying on the text density in subsequent Web page segments. BOILERPIPE [Kohlschütter et al., 2010] relies on shallow text features and learning to identify the fulltext of a Web article and to filter in this way the boilerplate present in the respective Web page. A current limitation of BOILERPIPE is when a single Web page contains various articles: the text of all articles tends to be taken as a whole if unusual separators are used. Also, when the text of an article is segmented by the use of images (or different kind of contents than text), BOILERPIPE tends to return a partial result. Relying only on shallow text features, without any other clue on what is interesting to extract, may also conduct to the extraction of other portions of the page which are richer in text than the article itself (for instance, the comments). However, while we need some sample pages that share the same template, a plus boilerpipe is that it works at the level of individual Web pages. We use BOILERPIPE as a baseline method: this is a state-of-the-art method used for article extraction. Its extractors have been trained on typical Web articles, which makes it perfectly applicable in our context.

Template removal and change detection Template removal methods need usually as input Web pages that share a uniform layout [Vieira et al., 2006]. Tree matching (e.g., the number of occurrences of some branch in the whole forest of DOM trees) or abstract structural features [Crescenzi et al., 2005] are usually employed as pre-processing before template removal. Most of the time the problem is reduced to that of finding common DOM subtrees [Caverlee et al., 2004] by cross-page clustering for a set of HTML documents. This is also the setting for change detection algorithms [Artail and Fawaz, 2008; Lim and Ng, 2001], which, rather than detecting what has changed, detect what did not change across different pages, and find the answer by exclusion.

Tools and standards Some tools, such as the Reader mode of the *Safari browser* or similar browser plugins, aim at presenting the main content of a Web page in a visually, more readable way. These tools use a combination of heuristics, site-specific parameters, and estimation of text density.

Several recent development of Web technologies go in the direction of adding more semantics to the markup of a Web page to clearly identify the main content of a Web

page. The *HTML 5* working draft introduces the very useful `<article>` tag to denote a Web page's or a section's main content. *HTML 5* is not widely used on the Web at the moment, and it is unclear at this point whether the use of such a tag will be consistent enough from a Web site to another.

Another initiative is the *hAtom*² microformat for syndicated content, which indicates in particular which part of a Web page corresponds to a feed item; the use of *hAtom* on the Web is still marginal, while our approach aims at being generic, without relying on user markup.

§ 3. Signifier-based Methods

§ 3.1. Signifiers

We use the generic term of a signifier to define a linguistic, or, if possible, a semantic clue. Examples include a keyword, *n*-gram, an entity, etc. The goal is acquire a notion of relevance using the identification of signifiers. Signifiers may directly given or come from other sources, possible examples being query logs, HTML metadata, etc. The methods that we develop aim at being fully automatic and self-sufficient; we therefore automatically construct a set of relevant terms for each Web page, by various methods, here detailed. In this thesis, we use the notion of *signifier* in the IR sense, as a linguistic clue. However, the high incidence that, say, a keyword may have in the text of a Web article makes it closer to a conceptual entity (for instance, “Halloween” in our example), although this is not formally stated.


Signifiers are used to spot the relevant zones of a Web page. At DOM level, they help us to determine which DOM elements of the corresponding tag tree are more important than others.

As an example of their usefulness, we show in Figure 2.2 how some keywords like “Halloween”, “past” and “present” may target the article zone in the given Web page.

§ 3.2. Acquisition


Feed-based This feed-based method of acquiring signifiers for Web pages is very straightforward. Without any global analysis on all the textual content of pages, we obtain some reliable clues on the content that is interesting in a Web page. Supposing the existence of a Web feed linked to a Web site, resuming in this way its new entries, we can extract some clues about what is interesting in those pages by parsing the feed items metadata. In particular, we retrieve all textual content from the *title* and *description* meta-tags of each feed item. The HTML code of the description is stripped, only text is

²<http://microformats.org/wiki/hatom>




A Day of the Dead offering in the Nunkini cemetery, in Campeche, Mexico. (Jeffrey Becom/LPI)


Related



Worldwide weird:
Halloween to the extreme
Taking normally odd rituals one step further



Oktoberfest: Then and now
It has not changed that much over time



Ireland
In Ireland, arguably the holiday's birthplace, Halloween is still greeted

On the last night of the autumn harvest, the world changes from the sunny warmth of summer to the cold dark of winter, the land from fertile to barren. The ancient Celts believed this transition gave supernatural forces a chance to break through into the world of the living, and their evil mischief to flourish.

They came to celebrate the night leading into winter as Samhain (meaning "summer's end"), the festival widely considered to be the precursor of Halloween. On Samhain night, the Celts believed, the spirits of people who had died in the past year would walk among the living, so villagers put out food and sweets to pacify these spirits – a ritual that may have preceded trick-or-treating. (There is no hard evidence, however, that Samhain was indeed a festival of the dead, points out historian Nicholas Rogers, in his book Halloween: From Pagan Ritual to Party Night.)

Although Halloween has pagan origins, its name is derived from the Christian holiday "All Hallows Eve", or the evening before All Saints' Day (1 November). The holiday itself was adapted by Christians who hoped to stamp out paganism, and over the years, some of the darker aspects of Halloween have been replaced by more light-hearted, family-friendly festivities. But Halloween's ties with the scary and supernatural still hold strong today, in celebrations all over the world.

Figure 2.2.: Match of keywords in a Web article

kept. The resulting sequence of terms are transformed, using basic NLP or IR, into two kinds of descriptors: keywords and n -grams.

A n -gram represents a sequence of n terms, taken as they appear, from the title and description. The choice for n is a compromise between false positives and false negatives in the extraction process and it is discussed in Section § 5.2.

To obtain the keywords, we tokenize and stem the words, sort the resulting lexemes according to their frequency, seen as a measure of their importance, and keep only outstanding ones (say, top- k). The keywords are dense in the portion of the Web page that contains the information of interest, but can frequently digress the attention to other zones that are also rich in keywords, like the ones containing lists of related articles, comments or categories for a given Web article.

A question that we study in Section § 6.3 is whether the *description* generally contains the fulltext of a Web article. If this is true, then having the feed associated with a Web site solves the article extraction problem. On the other hand, if this is not generally true, the difficulty would be to determine these particular cases on-the-fly, so in the absence of the ground truth.

Tf-Idf Given a set of sample Web pages, we apply tokenization and stop words removal on the text of each. Then, we apply the classic *Tf-Idf* measure to identify a set of relevant keywords for each page. The top- k weighted terms according to this measure are called keywords (or, to be consistent with SIGFEED, signifiers) of the respective page.

In the experiments, we fix the k threshold to 10 (but discuss other settings). More complex processing, such as POS-tagging or semantic analysis is possible, but not required: preliminary experiments suggest little impact on the results.

Having these two *keyword sources*, we investigate in Section § 6.3 their quality and implications on the the performance of FOREST.

§ 4. Preliminary Notions

We first introduce some preliminary notions that are common to the techniques that we will further present in this chapter.

From HTML pages to XML trees From the Web feeds, we gather the items URLs and construct a usable version of the HTML page in the form of a DOM tree. We use for this the HTMLCleaner³ parser, which puts the HTML tag soup in the right order and filters out scripts and other tag nodes (e.g., `<noscript>`) that are neither dealt with, nor necessary in our approach.

³<http://w3c99htmlSpeccleaner.sourceforge.net/>

Significant Nodes For each analyzed document d_k , we extract all leaf nodes, that is, all non-empty textual nodes at the bottom of the DOM tree, that are significant. A *significant node* is a leaf DOM node whose textual content matches at least one signifier.

§ 5. SIGFEED

We have started by studying the problem of object extraction in the context of Web feeds, where the idea of a Web object is very intuitive.

§ 5.1. Intuitions and Implementation

We describe in this subsection the feed-based algorithm for finding the article that corresponds to a Web feed item.

Feed-based extraction We present a bottom-up algorithm that, given a Web feed, identifies the DOM node that wraps the main content of a Web page by matching the signifiers against the textual content of that page. All leaf nodes (i.e., textual nodes with no children) that contain at least one signifier are extracted. We then establish for each resulting leaf DOM node their relevance based on a classic measure of signifier density. Next, we classify all nodes which achieve a positive value for this measure according to the closest ancestor that is a block-level element.

An empirical heuristic that we considered effective is to take the closest ancestor that is a block element (e.g., `div`). We have observed that many current Web pages linked to a Web feed have the object of interest confined in a `div`. This is not surprising since, due to the utilization of CSS, `div` represents the most common type of DOM element used to group content.

After this clustering based on their *lowest common div block ancestor* (LCBA), we are able to tell which of the nodes containing signifiers share the same ancestor. These common ancestors give us the zones of the page that are interesting. In order to chose one of the candidate LCBA nodes, we compute the sum of the signifier densities, for all leaf nodes having in common a particular LCBA, and finally chose the candidate LCBA with the largest value for this sum.

The workflow of the extraction method using Web feed clues is summarized in Algorithm 1.

In our preliminary experiments, we have varied the parameters, allowed the use of either keywords or n -grams. We therefore observed that when matching signifiers that correspond to keywords, the number of DOM nodes to analyze increases, while when using n -grams, the number of candidates decreases and we tend to have a better precision. This happens because, in what it concerns the contents of the article object, n -grams are

Input: a URL of a Web feed *feedUrl*

Output: extracted data objects *dobs*

```

feedData = getFeedDataStructure(feedUrl); items = feedData.getItems();
foreach item in < items > do
    dob.created = item.getPubDate();
    semanticInput = item.getTitle() + cleanHTMLtags(item.getDescription());
    3grams = getSubsequencesOf3Tokens(semanticInput);
    dob.signifiers = 3grams;
    dob.URL = item.getLink();
    domRoot = getCleanedHTMLCodeFrom(dob.URL);
    leafTagNames = domRoot.evaluateXPath("// * [not (*)]/name()");
    foreach leafType in leafTagNames do
        leafNodes = domRoot.getElementsByName(leafType);
        foreach leafNode in leafNodes do
            nbOfSemanticMatches = leafNode.nbMatches(3grams);
            if nbOfMatches ≥ 1 then
                significantNodes.put(leafNode, nbOfMatches);
            end
        end
    end
    blockNode = div, table.tr.td foreach sigNode in significantNodes do
        while sigNode isNotA blockNode do
            lcbAncestor = sigNode.getParent();
        end
        candidateBlocks.add(lcbAncestor);
    end
    foreach lcbAncestor in candidateBlocks do
        nbSigNodes = intersectionSize(ancestor.getDescendants(), significantNodes);
         $relevance[lcbAncestor] = \sum_{n=1}^{nbSigNodes} \frac{sigNode.nbOfSemanticMatches}{sigNode.textualLength}$ ;
    end
    wrapperNode = the ancestor with the biggest value
    for relevance ;
    if wrapperNode != null then
        dob.setFullTextAndRefsFrom(wrapperNode);
        dobs.add(dob);
    end
    else
        try with keywords instead of 3–grams
    end
end
return dobs

```

Algorithm 1: Feed-based object extraction

more significant than keywords, because they normally come from the title and the first lines of the article.

Nevertheless, in the rare cases in which the description is shortly summarizing the article, rather than representing the first lines of it, the n -grams will not be of use. In our experiments, the use of 3-grams gave best results. In the case of 3-grams mismatch, we tried with the *keywords*.

In practice, the LCBA node that represents the wrapper of the main content has to contain a large proportion of signifiers, but not necessarily all. Indeed, the description in particular can introduce noise text by the inclusion of expressions like “read more”, “check out full content”, etc. However, this can be easily filtered with a *Tf-Idf* analysis applied on the corpus of extracted descriptions from the feed items; this will filter recurrent expressions that are not informative.

Exclusion of comments In general, data objects in general (news articles, blog posts) may have comments associated. We would want to make a clear difference between the comments of an article and the article itself because:

1. from a conceptual point of view, the information of interest in an article is not the same as comments about it;
2. the crawl of the article should be separated from the crawl of the comments: if each time a comment is added, the article is considered as changed, a new crawl is necessary (ideally, incremental crawl)..

A reference to comments can be found as a meta element in the feed items, but unfortunately this is not a compulsory standard element in the feed (RSS or Atom) specification. On the other hand, the zone of comments can be easily identified using heuristics because of the *list*-compliant structure in HTML.

Illustration We take the example of a Web page, which is partially presented in Figure 2.3. The Web page is well-formatted using `HtmlCleaner`⁴ parser; this step is necessary in order to reliably select leaf nodes from the DOM tree of the page and keep for further analysis only those that contain at least one signifier.

We observe that the title of the article is present in the `<title>` of the feed item, as well as the first two lines of text, which are encoded in the item’s `<description>`. There is also a timestamp that corresponds to the publication date. The tag `<content:encoded>` is not a compulsory element, therefore we do not make assumptions about its usefulness in our algorithm.

In Figure § 5.1 we observe that the title of the article is also present in the zone of comments. Another possible zone of incidence is the navigational part presenting the

⁴<http://htmlcleaner.sourceforge.net/>

Health & Medicine | Mind & Brain | Technology | Space | Human Origins | Living World | Environment | Physics & Math | Video | Photos | Podcast | RSS

Blogs / Cosmic Variance


« Restrepo
Zozobra »

A study on how to study

by daniel

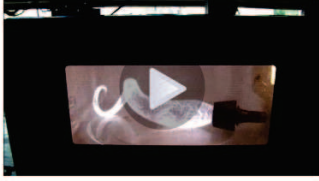
One of the most delightful aspects of being a scientist is that you're always learning. Your colleagues teach you things. Your students teach you things. Journal articles teach you things. You sit quietly at your desk and figure things out. You're perennially a student. But how to be a better student?

This morning the New York Times has an [article on "study habits"](#). It argues against the conventional wisdom (find a clean, neutral space, and bear down on a single topic), and in favor of what might be called intellectual cross-training: "alternating study environments, mixing content, spacing study sessions, self-testing". The basic philosophy seems to be encapsulated:



woman_asleep_at_desk

New Joe Genius Episode






Figure 2.3.: A typical Web article and its corresponding data item from the feed

```

<item>
<title>A study on how to study</title>
<link>http://feedproxy.google.com/~r/CosmicVarianceBlog/~3/
uatEVOIO0g/</link>
<comments>http://blogs.discovermagazine.com/cosmicvariance/2010/09/07/a-
study-on-how-to-study/#comments </comments>
<pubDate> Wed, 08 Sep 2010 03:16:54 +0000 </pubDate>
<dc:creator> daniel </dc:creator>
<category> <![CDATA[Advice]]> </category>
<guid isPermaLink="false">http://blogs.discovermagazine.com/cosmicvariance/?p=5353
</guid>
<description> <![CDATA[One of the most delightful aspects of being a scientist
is that you&#8217;re always learning. Your colleagues teach you things. Your
students teach you things. Journal articles teach you things. You sit quietly at
your desk and figure things out. You&#8217;re perennially a student. But how
to be a better student? This morning the New York [...]]]> </description>
<content:encoded><![CDATA[<p>One of the most delightful aspects of being a
scientist is that you&#8217;re always learning. .../>]]> </content:encoded>
<wfw:commentRss> http://blogs.discovermagazine.com/cosmicvariance/2010/09/07/a-
study-on-how-to-study/feed/ </wfw:commentRss>
<slash:comments>6 </slash:comments>
<feedburner:origLink> http://blogs.discovermagazine.com/cosmicvariance/2010/09/07/a-
study-on-ow-to-study/ </feedburner:origLink>
</item>

```

DISCOVER
M A G A Z I N E

Health & Medicine | Mind & Brain | Technology | Space | Human Origins | Living World | Env

Cosmic Variance

« Restrepo Zozobra »

A study on how to study

by Daniel Holz

One of the most delightful aspects of being a scientist is that you're always learning. Your colleague teach you things. Your students teach you things. Journal articles teach you things. You sit quietly a your desk and figure things out. You're perennially a student. But how to be a better student?

This morning the New York Times has an [article on "study habits"](#). It

Share 66 Tweet 8 Share 574 +1 0 Share 1

September 7th, 2010 8:16 PM
in Advice, Top Posts | 9 comments | RSS feed | Trackback >

9 Responses to "A study on how to study"

1. Tweets that mention [A study on how to study](#) | Cosmic Variance | Discover Magazine -- Topsy.com Says:
September 7th, 2010 at 8:44 pm

[...] This post was mentioned on Twitter by Ron Simon, Maggie, World Amazing Things, Al Poe, Sains & Teknologi and others. Sains & Teknologi said: [A study on how to study](#) | Cosmic Variance: One of the most delightful aspects of being a scientist is that you're ... <http://bit.ly/auVRjn> [...]

Figure 2.4.: The incidence of signifiers coming from the article title

most recent articles of the Web site. If the title was sufficient, we could just take it from the Web page `title` meta tag, making this method independent of the existence of Web feeds. However, preliminary experiments show that this hypothesis does not work in practice.

Some random examples of n -grams from the description are “delightful aspects”($n=2$), “being a scientist”($n=3$), “This morning the New”($n=4$). In general, the n -grams have a smaller chance to be present in large number in other zones of the Web page. In this particular example, the algorithm identifies the article even using only “This morning the New”. This happens because “This morning the New” is a discriminative n -gram in the Web page. On the other hand, “delightful aspects” or “being a scientist” n -grams appear also in other regions of the page, e.g., in the first comment (Figure § 5.1, `div/ol/li/div/p`).

For simplicity, assume that the process of matching n -grams returns the two subtrees illustrated in Figure § 5.1. The value attached to each leaf node represents the number of signifiers contained in its textual content.

The first subtree corresponds to the Web article, while the second to the comments region. Having these two candidate blocks representing the LCBA for the significant nodes, we rank them by considering the aggregated density measure of their significant nodes, which, in this case, gives as best candidate the first subtree.

§ 5.2. Experiments

In order to prove the validity of our approach for extracting data objects from Web feeds, we have fully implemented the method and performed experiments in order to evaluate it.

The experiments were performed on the dataset of Web feeds collected by scraping the response pages of the Search4RSS⁵ feed meta-search engine. For each feed, we have analyzed the XML structure of the Web channel, and, after an analysis of its items, we applied SIGFEED for all the Web pages referenced by the `url` meta-tag of each item.

We have tested the hypothesis of returning as LCBA of the DOM leaf node containing the title. This method gave poor results, for several reasons: the title may not fully match, but even if we apply flexible matching, it can appear in several different places in the Web page. More importantly, given only the location of the title leaf element, it is not an easy task to identify the limits of the whole article due to HTML block nesting.

We compare now the performance of SIGFEED to BOILERPIPE [Kohlschütter et al., 2010]. BOILERPIPE represents a state-of-the-art method for identifying the text of an article of a Web page by filtering boilerplate content. We stress that, as we use an idea of relevance that BOILERPIPE does not have access to, obtaining a better precision does not diminish the interest of this approach, which is more general.

⁵<http://www.search4rss.com/>

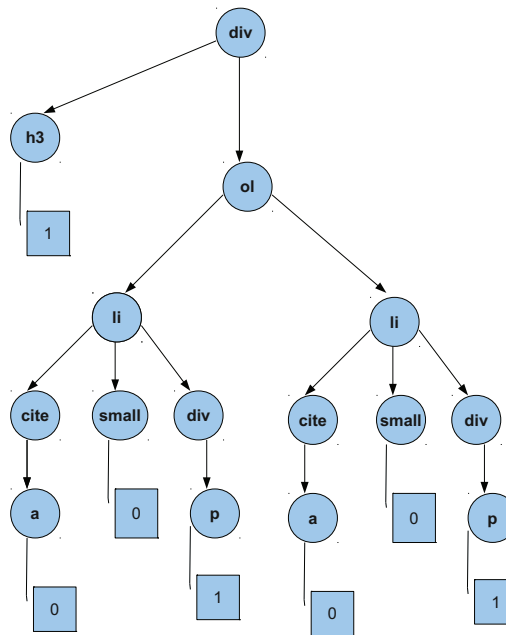
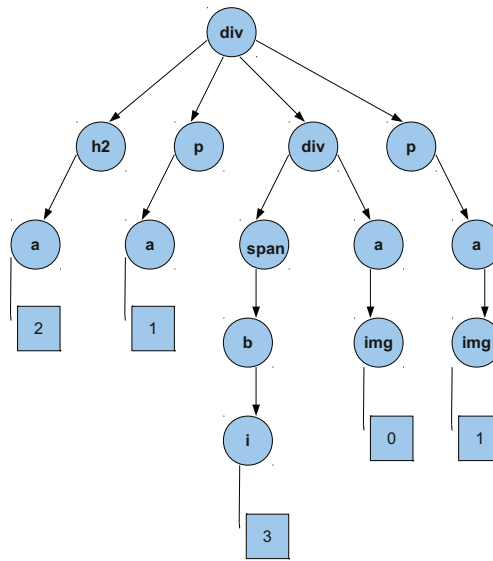


Figure 2.5.: Simplified example of two LCBA

Table 2.1.: SIGFEED experimental results

| Method | Correctly extracted | Percentage |
|------------|---------------------|------------|
| SIGFEED | 1038/1314 | 79.0% |
| BOILERPIPE | 821/1314 | 62.5% |

We do not consider the text density in the Web page, but rank DOM nodes based on a measure of content relevance with respect to the metadata that we get from the feed items. In this way, a DOM node may contain plenty of text, but may be judged as worthless if the text does not contain any of our signifiers. Moreover, there are cases in which a single Web page contains various consecutive articles. Using text density measures and visual features to identify content separators is tricky. From our observations, in these cases, BOILERPIPE takes the textual content of all the articles or just of the densest one.

We perform the experiments for 60 sites containing articles related to the *art* domain, corresponding to a total of 1314 feed items. We manually checked the correct identification of the article on Web pages. We only compare the textual result of the extraction because it is the output of the freely usable BOILERPIPE implementation⁶. However, by operating at DOM level, our method extracts an XML subtree from which we can derive not only the content, but also links, images, etc.

Numerical results are given in Table 2.1. We consider an article as correctly extracted when its title and fulltext are exactly those that we observe on the considered Web page. The notion of correct result does not take into account partial matches (e.g., segments of fulltext only), so note that we are not performing here a standard precision/recall analysis. The overall valid extraction percentage of SIGFEED reaches the satisfactory number of around 79%, compared to the 62% obtained by BOILERPIPE.

Finally, we note some observations regarding the situation of incorrect article identification according to the results validation criteria. SIGFEED tends to identify an ancestor the article wrapper node, that is, it may include more content than necessary (e.g., comments). This is also the case because, at DOM level, the article content and its comments have been logically grouped together. In contrast, BOILERPIPE tends to segment the article before being actually over, therefore to partially extract the content.

We present more extensive results on SIGFEED and BOILERPIPE in the next section, were these two approaches are used as baseline techniques.

Limitations Although being a rather simple and efficient method for the extraction of Web articles, SIGFEED method is not generic, as it works for pages which have a Web feed associated. Also, it is based on a heuristic related to the DOM block elements. To address these two drawbacks, we next present FOREST.

⁶<http://code.google.com/p/Kohlschutter09boilerpipe/>

§ 6. FOREST

§ 6.1. Introduction

We introduce a technique for effective extraction of a Web page’s main content, by taking into account both the content itself, and the repeated structure of Web pages that have been generated by the same software resource. We further call these pages, that represent the input of our algorithm, sample Web pages. The proposed method, called FOREST for “*Focused Object Retrieval by Exploiting Significant Tag paths*”, works in a succession of steps:

1. we automatically acquire some keywords for each Web page in the set of sample pages;
2. we identify, at DOM level, structural patterns that are shared by most of the sample pages;
3. we rank these patterns through a novel relevance measure based on information theory and statistics in order to identify the most informative one;
4. we infer a generic XPath expression which gives the location of the main content in the sample Web page.

We outline the following *contributions* of this work:

- (i) a novel measure for computing the informativeness for the content of a Web page;
- (ii) a technique for wrapper induction at Web site level and automatic identification, using the tag paths of significant DOM elements, of a generic XPath signature of the node that contains the main content of interest;
- (iii) effectiveness experiments showing the high accuracy in terms of precision and recall of FOREST over around 1,000 Web pages acquired from 93 heterogeneous Web sites, with favorable comparisons, for different settings and baselines including some state-of-the-art methods for article text extraction.

§ 6.2. Methodology

In this section, we describe FOREST’s *signifier-aware process* for extracting the Web page block that corresponds to the article object, in a consistent way across different sample pages. The three main steps performed by FOREST are summarized in Algorithm 2, Algorithm 3 and Algorithm 4.

Input: *classOfPages* = a set of sample Web pages

Output: *xmlDocuments*, *keywords*

foreach *webPage* in *classOfPages* **do**

xmlDoc = *htmlCleanerNormalization(htmlPage)*;

xmlDocuments.add(xmlDoc);

 frequency analysis on *terms* of the *xmlDoc*;

 index globally *terms* with respect to the *xmlDocuments*;

end

compute the *Tf-Idf* scores of *terms*; return weighted terms as *keywords*;

Algorithm 2: FOREST: *Tf-Idf* keyword acquisition

§ 6.2.1. Structural Patterns

Enforcing structure We add to every node in the tag tree a *dfs* attribute that records the order of browsing of a DFS (depth-first search) walk starting from the root. These *dfs* values serve to identify nodes in the case when they do not have any unique combination of attributes (e.g., *id*, *class*), but also to keep track of possible positions of nodes in the tag tree. This enforced structure is serialized into an *XML document*. We will denote all such documents coming from a single Web feed channel d_k , $1 \leq k \leq n$ where n is the number of items in the feed (typically, 10 [Oita and Senellart, 2010b]).

Input: *classOfPages*, *keywords*

Output: *elementTypes*

foreach *xmlDoc* in *xmlDocuments* **do**

 compute the *significantPath* of *xmlDoc* based on the acquired *keywords* (cfm. Definition 1);

foreach *domNode* in a *significantPath* **do**

elementId := $\langle \textit{elementClue}, \textit{dfs} \rangle$ (cfm. Definition 2);

structuralPattern := $\langle \textit{elementId}, \textit{level} \rangle$ (cfm. Definition 3);

 index the *elementId* depending on the *level* in which it occurs;

 compute the *informativeness* of the *domNode*'s content (cfm. formula 2.4);

infoSum+ = *informativeness(structuralPattern)*;

end

end

return *elementTypes* := $\langle \textit{structuralPattern}, \textit{infoSum} \rangle$;

Algorithm 3: FOREST: structural clustering of significant DOM elements

Significant XPath For each significant node, we construct its terminal path. A *tag path* is the sequence of node identifiers from the root to a node in the tag tree.

Definition 1 A *significant terminal path* is a tag path where the leaf node is a significant.

By following the path of signifiers in the DOM, we have an idea about the location of leaf nodes that are significant in a Web page. However, due to the nested structure of elements at DOM level, this does not say (a priori) anything about the boundaries [Buttler et al., 2001] of the article object.

In our method, we first analyze all the composing DOM elements of the significant paths with the goal of finding structural similarities across different sample pages. Next, we study, with a novel measure of informativeness, DOM elements clustered based on their structural similarities from the relevance of content point of view.

Since the leaf node that represents the source of a significant tag path contains keywords, all its ancestors are textual nodes containing keywords. The higher in the DOM hierarchy, the more keywords a node tends to contain; in the same time, it also tends to contain more *non-significant terms*. Note that the number of significant paths that have to decompose can be small if the number of keywords is small, or if these keywords are consistently pinpointing the same terminal paths.

Element identification A DOM element typically has a tag name and a list of attributes. However, not all elements have attributes that can make them unique. Paragraphs, table elements, etc. are rarely determined by a unique combination of attributes. Luckily, due to the pre-processing phase, each node has, besides its tag name, at least one *dfs* attribute.

To be able to reuse the element type from one document to the other, we add a refinement that deals with the fact that, contrarily to what one would expect, *id* and *class* names generated by CMSs may slightly vary from one page to another. In particular, it is common to find, say, a `div` tag with a class attribute of “post wrapper-09” in one document, and of “post wrapper-02” in another, knowing that empirically they denote the same type of element.

We found that a practical way of abstracting out these small differences is to simply keep the first token of the value of an attribute and filter out possible numbers. Although this is not compulsory to achieve better results than the baselines, it improves them in practice. We next refer to the stemmed value of an attribute as its *tolerant form*. For instance, the tolerant form of the attribute value “post wrapper-02” (occurring in page two) and “post wrapper-09” (occurring in page nine) is simply “post”. For gaining more insight into the process, we show in Figure 2.6 some manifestation of the structural patterns.

Definition 2 The *element type* of a DOM element is defined as the XPath expression constructed based on $\langle t, atts \rangle$, where t is the tag name and $atts$ is a set of key-value pairs as follows:

- (i) if the DOM element has attributes other than *dfs* (e.g., *id*, *class*) then *atts* is set to the collection of attribute key-value pairs, where the values have been reduced to a tolerant form;
- (ii) otherwise, *atts* is set to $\{dfs = d\}$, where *d* is *dfs* index of the respective element.

Examples of simple element types are `//div[@id="container" and @class="post"]` or `//p[@dfs=24]`.

Element description There are various methods to operate on the DOM model of a document [Lim and Ng, 2001; Yadav et al., 2007]. This has been especially studied for change detection algorithms. We define next structural patterns of DOM elements in a slightly different manner.

Definition 3 A *structural pattern* is defined by the combination of an *element type* and the *level* on which a significant element occurs in the tag tree (i.e., its index in the significant path).

Structural patterns are used to identify similarities between DOM elements across various pages. Indeed, the values of element clues and level can be common to nodes belonging to different Web pages, as long as they share the same generation source. The *dfs* position of an element is either present in its type or added to the structural pattern in order to uniquely identify a DOM element in a particular Web page. Figure 2.7 gives more insights on structural patterns, that have been clustered based on the level on which they occur. Although the elements having the *dfs* equal to 21 and 29 respectively share the same type, they are counted separately because their reside on different levels.

The pos vectors in Figure 2.6 are generated by the *dfs* sequence for the composing elements of significant paths. We observe that they trace the hierarchy similarities across sample Web pages.

In general, besides the hierarchy similarity, the sample documents share a global pool of *element types*, for us structural patterns. This idea stands at the root of wrapper induction techniques; for instance, [Arasu and Garcia-Molina, 2003] differently formalizes the element types through *equivalence classes*.

§ 6.2.2. Informativeness Measure

We now introduce the measure of relevance for ranking these structural patterns, based on the fact that have been formed by DOM elements that are significant across various d_k .

We fix an node element e_i in an XML document d_k . Let x be the number of signifiers in e_i 's text, counted with their *multiplicity*. Then all other terms represent non-signifiers, let y be their number. Then $N = x + y$ is the number of terms in the text of e_i . We analogously denote the number of signifiers and non-signifiers in the *whole* d_k in which e_i occurs, that , as X and Y respectively.

| | |
|------------------------|--|
| <i>sd</i> ₁ | |
| | 1 – 19 – 20 – 21 – 25 – 78 – 140 1 – 19 – 20 – 21 – 25 – 78 – 82 – 83 – 86 – 92 1 – 19 – 20 – 21 – 25 – 78 – 98 1 – 19 – 20 – 21 – 25 – 78 – 133 1 – 19 – 20 – 21 – 25 – 78 – 136 |
| <i>sd</i> ₂ | |
| | 1 – 19 – 20 – 21 – 29 – 175 – 181 – 208 – 209 – 210 – 216 1 – 19 – 20 – 21 – 29 – 175 – 181 – 197 – 198 – 199 – 206 1 – 19 – 20 – 21 – 29 – 82 – 107 1 – 19 – 20 – 21 – 29 – 82 – 85 – 86 |
| <i>sd</i> ₃ | |
| | 1 – 19 – 20 – 21 – 29 – 83 – 99 |

Figure 2.6.: The *pos* vectors across three sample pages

| <i>level</i> | Identifying clue | <i>pos</i> | <i>nbOcc</i> |
|--------------|-----------------------------------|------------|--------------|
| 1 | //body | 1 | 10 |
| 2 | //div[@id='container'] | 19 | 10 |
| 3 | //div[@class='maincntnr'] | 20 | 10 |
| 4 | //div[@class='idem'] | 21 | 10 |
| 5 | //div[@class='idem'] | 25,29 | 10 |
| 6 | //div[@class='story'] | 78,82,83 | 8 |
| 6 | //p[@style='text-align:justify;'] | 175 | 2 |
| 7 | //p[@dfs=98] | 98 | 1 |
| 7 | //p[@dfs=107] | 107 | 1 |

Figure 2.7.: Example of DOM element types occurring in terminal paths

Statistical signifier density One of the most natural ways to determine whether a node is highly significant is to compute its density in signifiers, i.e., $\frac{x}{N}$. However, when N is small, this density might be imprecise, due to lack of observations (a node formed of a single signifier is likely not to be the most significant node of the document). In such contexts, we can use *Jeffrey's add-half rule* [Krichevsky and Trofimov, 1981] as a better statistical estimator of the proportion of signifier terms, yielding $\frac{x+1/2}{N+1}$.

Furthermore, when sampling N elements from a potentially larger set, we have a margin of error on the semantic density. With f the frequency given by the estimator above, the standard deviation is $\sqrt{\frac{f(1-f)}{N}}$ [Freedman et al., 1998]. Assuming 1 standard deviation to obtain a confidence interval of $\approx 70\%$ [Freedman et al., 1998] and combining with the aforementioned estimate, we obtain the following interval for the semantic density of a node:

$$\frac{x+1/2}{N+1} \pm \frac{1}{N+1} \sqrt{\frac{(x+1/2) \times (y+1/2)}{N}} \quad (2.1)$$

We now define as Jeffrey's statistical density, J , the lower value of this interval, i.e., the worst-case estimator at 70% confidence of the semantic density; if this value is less than 0 (because the sample is not large enough), we fix it to 0:

$$J = \max \left(0, \frac{1}{N+1} \left(x+1/2 - \sqrt{\frac{(x+1/2) \times (y+1/2)}{N}} \right) \right) \quad (2.2)$$

As an example, if $x = 8$ and $y = 20$, $J(x, y)$ is comparable to that of a node with $x = 3$ signifiers and $y = 5$ non-signifiers: the lower proportion ($\frac{2}{5}$ compared with $\frac{3}{5}$) is compensated by the smaller number of observations. Even more interestingly, when $x = 1$ and $y = 0$ we have a $J \approx 0.32$, to be compared to the naive density of 1: this element is indeed dense, but due to the low (zero here) number of non-signifiers, we cannot be very sure of its importance. As expected, this density measure tends to favor rather specific nodes, that is, nodes that appear lower in the DOM hierarchy.

Unexpectedness We derive another approach to significance of a node from the notion of *unexpectedness*, coming from the cognitive model of *simplicity theory* [Dimulescu and Dessalles, 2009] and information theory in general: this measure relies on the observation that humans tend to find a situation interesting when they perceive a discrepancy in complexity.

A situation is unexpected if it is simpler to describe than to generate. Assume a computation model given (say, Turing machine encodings for a given universal Turing machine). Given an object, we consider its generation complexity C_w (i.e., the size of the program that has generated it) and its description complexity C (i.e., its Kolmogorov complexity, the minimum size of a program that describes it); then the unexpectedness

of this object is the difference between the two (note that we always have $C \leq C_w$). We apply this to the simple setting of non-uniform binomial distributions, that corresponds to our context.

Specifically, for each significant node in the DOM tree, we consider its unexpectedness with respect to the number of signifiers and non-signifiers contained in the subtree defined by its location in the DOM. The generation complexity corresponds (up to an additive constant) to the logarithm of the number of ways to draw $x + y$ elements out of a set of $X + Y$ elements: $C_w = \log(X + Y)^{x+y} = (x + y) \log(X + Y)$. The description complexity, on the other hand, represents the complexity of describing the content of the textual node, knowing that x terms are signifiers: it is the logarithm of the number of ways of choosing exactly x signifiers and y non-signifiers, that is: $C = x \log X + y \log Y$. Finally, the unexpectedness is the difference between these two complexities:

$$U = (x + y) \log(X + Y) - x \log X - y \log Y \quad (2.3)$$

As a typical example, for a Web page with a total of 20 signifiers and 100 non-signifiers, a node with 10 signifiers and 26 non-signifiers will have an unexpectedness of 23 bits, which is definitely higher than a node with 3 signifiers and 1 non-signifier: 6 bits.

More intuitively, our preliminary experiments show that unexpectedness favors elements with a large amount of text content that is richer in signifiers than the typical distribution of signifiers on the Web page as a whole. This turns out to be complementary to the statistical density J , which favors nodes poor in non-signifiers.

Informativeness

$$I(sp_i, d_k) = J(sp_i, d_k) \times U(sp_i, d_k) \quad (2.4)$$

This measure characterizes the informativeness of a structural pattern $sp_i, i \in 1 : m$, where m is the total number of structural patterns that are shared by our sample pages, as the sum of the products between the unexpectedness and the statistical signifier density of a DOM element that has the structural pattern sp_i in document d_k .

§ 6.2.3. Combining Structure and Relevance

A global measure of relevance for a structural pattern combines the informativeness of it (Section § 6.2.2) with its number of occurrence and a decay factor given by its level (Section § 6.2.1).

In terms of the number p of significant terminal paths where sp_i occurs on *level*, we have:

$$R^{\text{FOREST}}[sp_i] = \sum_{k=0}^p I(sp_i, d_k) \times p \times \text{level}(sp_i) \quad (2.5)$$

There exists a single sp_i (element type on a certain level), so the number of occurrence of sp_i in the sample pages is p , $p \leq n$, where n is the total number of documents. The role of the p factor is clear, since we want to give a bigger weight to structural patterns that are not only informative, but also very frequent. In addition, the *level* factor is a heuristic favoring nodes that are deeper in the DOM tree. The primary reason for this addition is that elements that are too high in the hierarchy (e.g. <body>) are more unlikely to effectively identify the target article object because they are not discriminative. The idea of a decay factor has been also introduced in other works [Guo et al., 2003; Lim and Ng, 2001], under different forms. For instance, in the ranking formula of [Guo et al., 2003] the decay factor is a value in the range 0 to 1.

We rank the structural patterns $sp_i, i \in 1 : m$ using this relevance measure. In the end, we simply derive the XPath clue of the best ranked structural pattern, which at this point fully identifies a target subtree across various documents. For clarity, following the reasoning on Figure 2.7, the path of the generic wrapper will be: `//div[contains(@class, 'story') and @dfs='78' or @dfs='82' or @dfs='83']`.

By applying the generic element type as a *XPath query* expression over the $d_k, k \in 1 : n$, we are most likely to find a node element that satisfies these structural conditions and whose content is highly informative.

Input: *elementTypes*

Output: *infoBlock* := the most informative block of a Web page

foreach *possible level* **do**

 get the *structuralPatterns* occurring on this *level*;

foreach *elementClue* of *structuralPattern* **do**

 count the *nbofOccurrences* of *elementClue*;

 group all its *dfs* positions across *classOfPages*;

end

fullXPathClue = *elementClue* completed with its *dfs* positions across *classOfPages*;

 get *infoSum* associated with *structuralPattern*;

 compute the *relevance* using *informativeness*, *nbofOccurrences* and *level* (cfm. formula 2.7);

 construct a *genericWrapper* := $\langle fullXPathClue, relevance \rangle$;

candidates.add(genericWrapper);

end

sort *candidates* based on their *relevance*;

return *infoBlock* := the top ranked element from *candidates*;

Algorithm 4: FOREST: ranking of structural patterns

Next, we describe the COVERAGE baseline.

Coverage Intuitively, the DOM node which contains the main content of an article should be defined as the smallest, lowest common ancestor (SLCA [Sun et al., 2007]) node in the hierarchy that has a maximal coverage. For a structural pattern $sp_i, i \in 1 : m$, the *coverage* of it represents the sum of normalized *Tf-Idf* weights of signifiers occurring in the text of a node that has this structural pattern sp_i in a document d_k .

$$Cov(sp_i, d_k) = \frac{\sum_{j=0}^{nbSigs(node(sp_i, d_k))} weight(signifier_j)}{totalNbOfSignifiers} \quad (2.6)$$

We implement this as a baseline, with the aim to show what can be achieved by making use of the bag of signifiers, in comparison with the more sophisticated measures used by FOREST. For this setting, COVERAGE baseline uses the formula of coverage 2.6 defined previously to select a structural pattern that is best covered in terms of signifiers:

$$R^{COVERAGE}[sp_i] = \sum_{k=0}^p Cov(sp_i, d_k) \times p \times level(sp_i) \quad (2.7)$$

§ 6.3. Experiments

We test FOREST using two sources of signifiers: keywords extracted from the Web pages themselves using the *Tf-Idf* measure, and keywords from the Web feeds associated to the dataset.

Dataset construction Next, we describe the *RED* (for *RSS-based Experimental Dataset*) dataset used to evaluate all techniques discussed in this section. Note that the existence of Web feeds is not a condition for FOREST, which only needs some sample pages that may represent *per se* the source of keywords. We have used Web feeds not only as a potential, alternative source of keywords, but also because the feed items of a Web channel refer to Web pages that typically share the same template.

The motivation for construction of this dataset is the current impossibility to test FOREST directly on existing Web article content extraction datasets: first, they operate at Web page level [Kohlschütter et al., 2010]; second, the datasets that are using the setting of sample pages for the main article extraction are not online [de Castro Reis et al., 2004]; finally, datasets using sample pages may exist, but are used instead in the context of the deep Web for response record extraction. In addition, one of the source of signifiers that is common to FOREST, but also to the SIGFEED baseline is Web feeds, and there exists, at our best knowledge, no other feed-based dataset for Web article content extraction.

Feeds of Web sites are acquired in an automatic manner by scraping the results of a *feed meta-search engine*, Search4RSS⁷. The condition of selection of feeds is to be parseable and to point to Web articles (and not tweets, for instance). Both feed and reference Web pages have been crawled at a given point in time, for 90 Web sites, with 3 exposing two slightly different templates. We have thus accumulated 93 types in total and 1,010 sample Web pages. Note that the annotation process is particularly time-consuming since more than 1,000 Web pages need to be annotated by hand.

As mentioned, feed URLs were given by Search4RSS in response to a topic query (keyword-based). For this reason, *RED* is quite heterogeneous: it includes various types of Web articles that exists on the Web on a particular subject (e.g., poetry), such as blog posts, news pages, professional or personal Web pages, etc. There exist various particularities of Web articles that we have observed during the annotation phase; for instance, we have found main article content scripted, spreading across different pages, or mainly composed of images or videos. Content is represented in new ways, which increases the difficulty of the extraction task. The gold standard is further discussed and available at the first author's Web page.⁸

Gold standard Remember that the target of the extraction is a Web article, so the goal is to retrieve the *title*, *fulltext*, and metadata like author(s), publication date, image captions, categories and tags, if they exist.

The gold standard for our dataset has been manually annotated. We have also annotated multiple, random Web pages corresponding to feed items. This is useful in the analysis of the number of pages that are necessary to reach a top efficiency. On the other hand, not all Web feeds have the same number of items in their channel, so we have annotated between 2 and 20 Web pages per feed, knowing that the typical number of items in a feed is 10 [Oita and Senellart, 2010b]. After a round of quality assurance to check that the guidelines were well understood, the annotation task is intuitive enough to reach a high-level of inter-annotator agreement; the precision from one annotator to another was 97%.

Baselines We compare the two variants of FOREST (when keywords come from feeds, or from the tf-idf analysis) to four different baselines.

The first is BOILERPIPE [Kohlschütter et al., 2010], already introduced in Section § 2. As a state-of-the-art method in content extraction, BOILERPIPE uses quantitative linguistics (features like average word length, absolute number of words, and the like) mingled with some heuristics on the DOM tree and semi-supervised learning to identify fragmented, short text in blocks of a document as *boilerplate*, and filter it in order to obtain the main article content. Unlike FOREST, BOILERPIPE needs to be trained for

⁷<http://www.search4rss.com/>

⁸<http://perso.telecom-paristech.fr/~oita/research.html>

specific data, but a pre-trained extractor (i.e., *ArticleExtractor*) that we believe to be the best adapted for the articles in *RED*, is publicly available in the author’s implementation.⁹ No significant differences were observed for other provided extractors.

Another baseline, that we have previously developed, is SIGFEED [Oita and Senellart, 2010b]. This technique selects, at the level of a single Web page (similarly to BOILERPIPE), the smallest, lowest `<div>` block ancestor in the DOM hierarchy that is the most dense in keywords obtained from the feed description (similarly to FOREST (feeds)).

The COVERAGE (see (2.6)) baseline is one intuitive technique that takes into account the tf-idf weighted signifiers that we automatically acquire in the presence of multiple sample pages. This heuristic is useful to test whether our elaborate informativeness measure adds value.

Finally, the DESCRIPTION heuristic simply takes, as the main content of the Web page, the title and description of an item as it appears in the Web feed (with some processing on the description to eliminate the possible HTML encoding). This hypothesis is important to be tested in the case of Web feeds, because there are cases in which feeds contain the whole title and fulltext of an article.

Performance metrics The result of our technique is a *generic tag path* which returns, for each Web page of studied channel, a DOM subtree in the form of an *XML document*. This is useful to get any media resource that is typically incorporated in an article which contributes to its object view. In spite of that, we make the evaluation on the extracted textual content, to compare it with our baselines (in particular, because the output of BOILERPIPE and DESCRIPTION is plain text). We also want the comparison measure to be insensitive to different amounts of whitespace extracted by various methods. After a typical normalization, we compute the set S of 2-grams (two consecutive words) in the output of all methods, and estimate classical *precision* and *recall* measures by comparing it to the set G of 2-grams of the gold standard, as:

$$\text{Precision}(G, S) = \frac{|G \cap S|}{|S|}, \quad \text{Recall}(G, S) = \frac{|G \cap S|}{|G|}.$$

Precision and recall are then summarized by their harmonic mean, the F_1 measure. Note that the precision we compute is exactly the ROUGE-N [Lin, 2004] measure used for comparing the performance of text summarization techniques.

Main results We show in Table 2.2 the mean precision, mean recall, and corresponding F_1 measure of the different methods tested over the whole dataset. We note that, since we have a sample of 90 independent sites and values of the order of 90%, the confidence interval at 95% probability (1.96 standard deviation) [Freedman et al., 1998] is ± 0.06 .

⁹<http://code.google.com/p/Boilerpipe/>

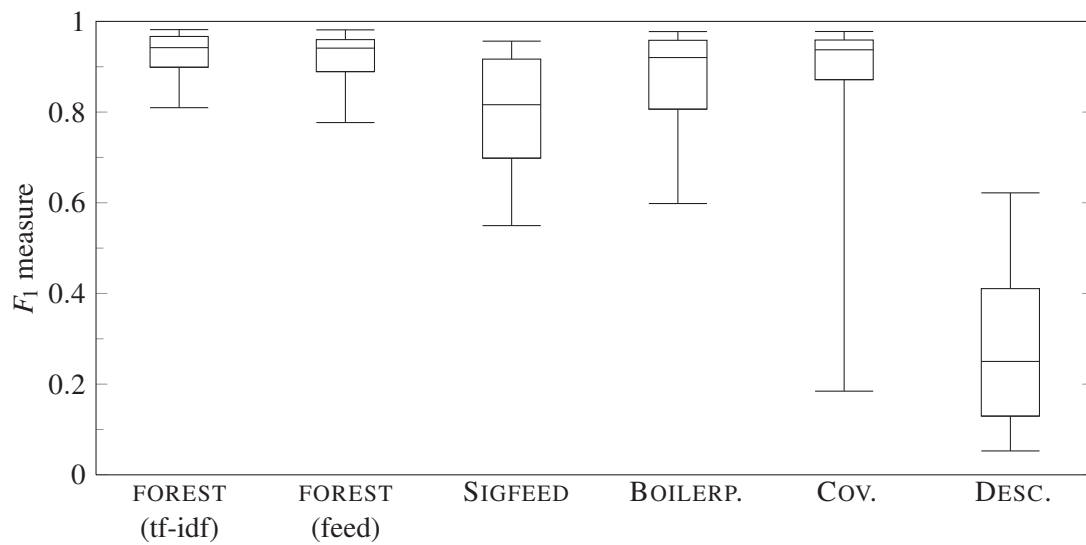


Figure 2.8.: **Article extraction: effectiveness results for all the considered techniques**

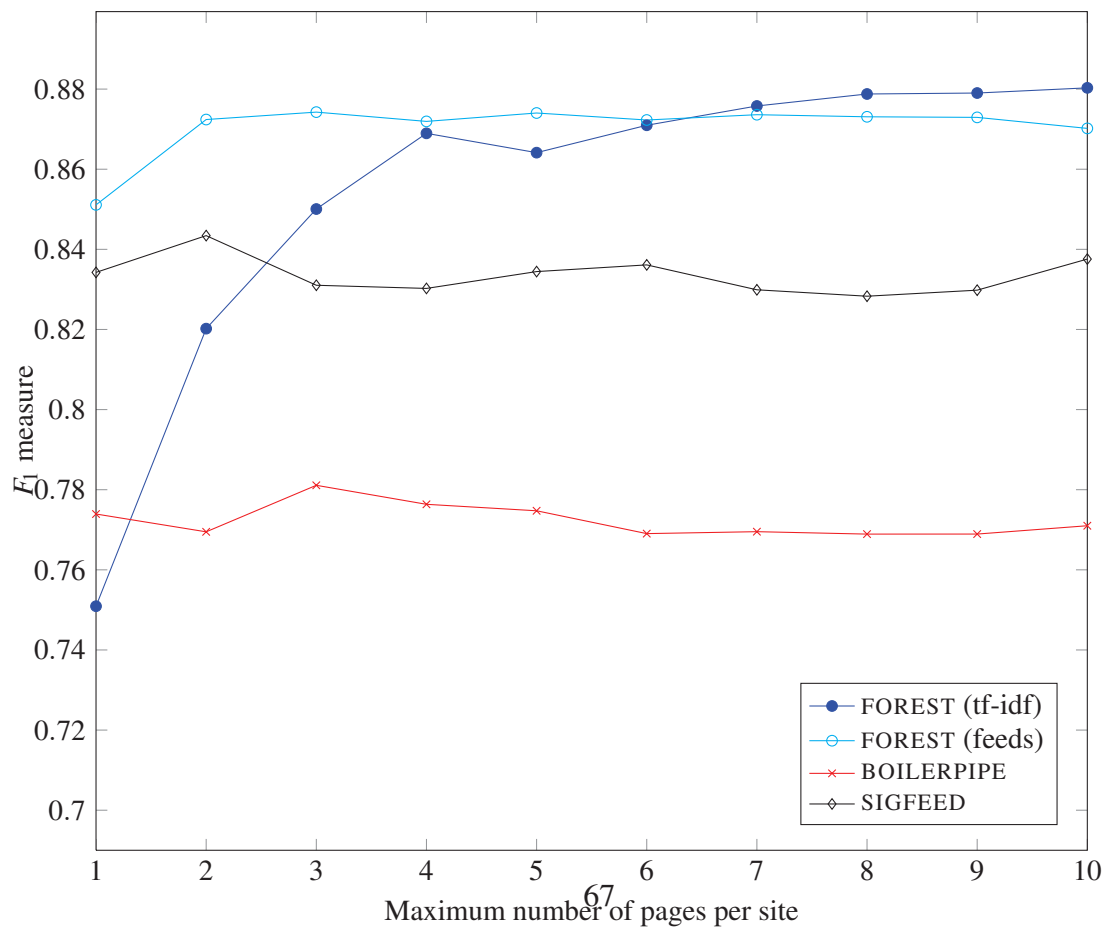


Figure 2.9.: **Evolution of the F_1 measure in function of the (maximum) number of pages considered, for each Web source, to share a similar layout**

| | Prec. (%) | Rec. (%) | F_1 (%) |
|-----------------|-----------|-------------|-------------|
| FOREST (tf-idf) | 93 | 92.8 | 92.1 |
| FOREST (feeds) | 92 | 90.5 | 89.6 |
| BOILERPIPE | 79.5 | 84.0 | 81.7 |
| SIGFEED | 88 | 83.9 | 84 |
| COVERAGE | 89.7 | 83 | 82.9 |
| DESCRIPTION | 84.4 | 22 | 30.3 |

Table 2.2.: FOREST: mean precision, recall, and their corresponding F_1 -measure

To investigate more precisely the shape of the distribution of results for each method, Figure 2.8 presents the F_1 measure of the different methods. We show for each method its 9th and 91th percentile (whiskers), its first and third quartile (box) and its median (horizontal rule).

Both variants of FOREST significantly outperform the baselines, with a global F_1 measure of, respectively 92.1% and 89.6%. These results were obtained for the whole dataset, that is, 1006 Web pages.

BOILERPIPE achieves a relatively low score here, despite the fact that the Web pages of our dataset (blog posts, news articles) match the kind of Web pages the *ArticleExtractor* has been trained on. We observe in practice that BOILERPIPE has the following shortcomings: when a Web page contains various small Web articles, the text of all is taken as a whole. Also, when the text of an article is segmented by the use of images (or different kind of content than text), BOILERPIPE considers them as separators, giving in this case a partial result. At the same time, BOILERPIPE can be applied directly at the level of a single Web page, which is a more independent setting than that of FOREST, in which at least two pages that share the same template are needed.

The intuitive COVERAGE approach that uses a relevance measure based on weighted keywords and the SIGFEED [Oita and Senellart, 2010b] heuristic-based method manages a higher level of F_1 measure than BOILERPIPE. We infer from this that, wherever their source, keywords are globally useful in the task of content extraction. SIGFEED is however outperformed by FOREST: the simple `div` block heuristic that works in many cases, fails to fully extract the article when complex HTML element nesting is involved.

Precision of the DESCRIPTION baseline is low, suggesting first, that feed items also contain 2-grams that do not appear in the main content (an example of that are dedicated links to go to the unabridged version), and second, judging by the abysmal recall, that feed items are often incomplete versions of the main content of a Web page. We also found out that, for practical (the article can be very long) or commercial purposes (to attract site visitors), many feed generators just cut the description to a couple of lines [Finkelstein, 2005].

To look more carefully into these results, turn now to Figure 2.8. This graph shows in particular that in addition of having better performance in average, the two variants of FOREST are also more robust: on 90% of the corpus (resp., 75%), the F_1 measure is greater than 84% (resp., 91%), to compare with 55% and 73% for BOILERPIPE.

Another interesting feature shown in Figure 2.8 is that both SIGFEED and COVERAGE have quite a high median, which means they will work well on most sources, but have a F_1 -measure less than, respectively, 55% and 9%, on 10% of the corpus. As already noted, DESCRIPTION performs very poorly, with a F_1 score greater than 50% on less than 25% of the corpus.

The similar performance of FOREST(tf-idf) and FOREST(feeds) suggests that keywords extracted from Web sites themselves are as useful as keywords from more trusted sources like Web feeds items. However, FOREST(tf-idf) has the advantage of not depending on the presence of Web feeds.

Influence of the number of pages To understand the impact of the number of pages with the same layout available to FOREST, we plot in Figure 2.9 the obtained F_1 measure of different methods with respect to the number of pages sharing the same template. Obviously, since neither SIGFEED nor BOILERPIPE make use of the repeated structure, the variation of their F_1 measure here is not significant: it is just due to the somewhat fluctuating performance behavior on the collection of Web pages of a given site.

FOREST(feeds) is already at the same effectiveness level as the SIGFEED baseline that does a comparable job, and is even slightly better, perhaps thanks to the measure of informativeness used.

FOREST requires at least two pages sharing the same layout: this is helpful not only for the acquisition of discriminative keywords using *Tf-Idf*, but also allows the exploitation of the repeated Web page structure for pattern identification. As soon as there are at least two sample pages, FOREST reaches an F_1 score that is already above that of BOILERPIPE.

When the signifiers are given by a *Tf-Idf* analysis on the Web pages themselves, FOREST cannot be applied with good precision on a single page: this is expected, since it means that the *IDF* measure is here constant and cannot serve to distinguish between page-specific and terms which are common to the Web site as a whole. In addition, there is no repeated structure that can serve to add relevance to tag paths. FOREST keeps improving as the number of Web pages increases, to reach a plateau around 8–10 pages. In any case, a small number of pages is enough to get better results than the baselines, which broadens the applicability of the method.

Influence of the number of keywords Another parameter that can be modified is the number of signifiers kept for a given Web page. From our experiments (see Figure 2.10), as long as the number of signifiers exceeds 5, the quality of the extraction is not overly affected, though we do observe a slight reduction in effectiveness when too many

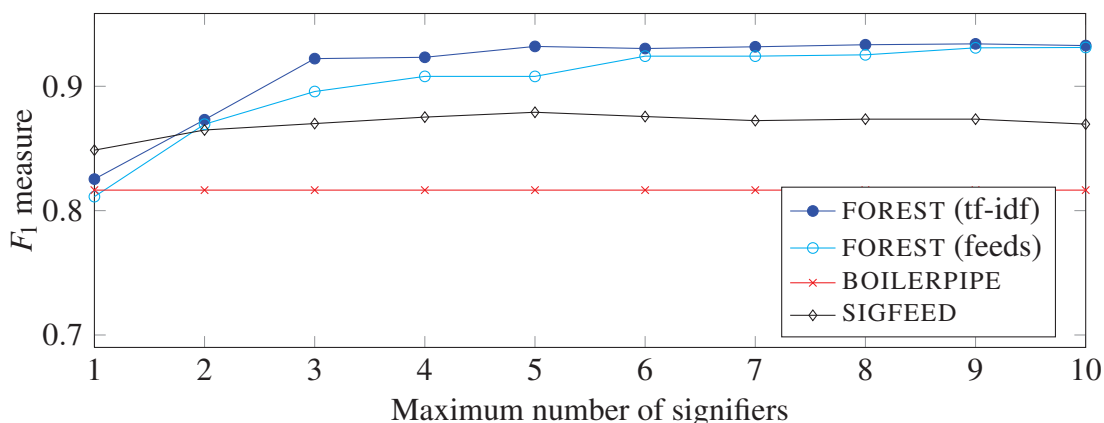


Figure 2.10.: Evolution of the F_1 measure in function of the (maximum) number of signifiers used in the process of DOM path selection

signifiers are considered.

When the number of signifiers falls below 5, the resulting few terminal paths are not giving enough insight on the informativeness of their element patterns, and the precision of results gets lower.

BOILERPIPE does not use signifiers, so the variation of its F_1 measure is zero. On the contrary, SIGFEED does, and we observe a surprisingly high F_1 stability, with a high score, even when using a single keyword. This could be explained by the `div` heuristic that is employed in SIGFEED: the idea that significant nodes are simply clustered based on their first (i.e., lowest in the hierarchy) `div` ancestor. Obtaining low variations of the efficiency when less signifiers are used to get significant nodes could mean either that these significant nodes have in common the same `div` block. As the sources of signifiers for SIGFEED are the title and description of an item, the feed signifiers tend to concentrate at the beginning of the Web article, at the same relative location. So having many other keywords in this case does not change the lowest common block ancestor that is chosen as wrapper for the target article.

Miscellaneous We report briefly on additional variations of the settings for FOREST.

We have tested U and J separately in the beginning to find a suitable measure. See Figure 2.11, where we have experimented with a maximum of 10 sample pages per feed source. The common pattern is that, both for FOREST (tf-idf) and FOREST (feeds), J and U alone give reasonable F_1 -measure scores, though lower than the combination of $J \times U$. In particular, J tends to have a higher precision than U and a lower recall, which makes the combination of the two a good compromise.

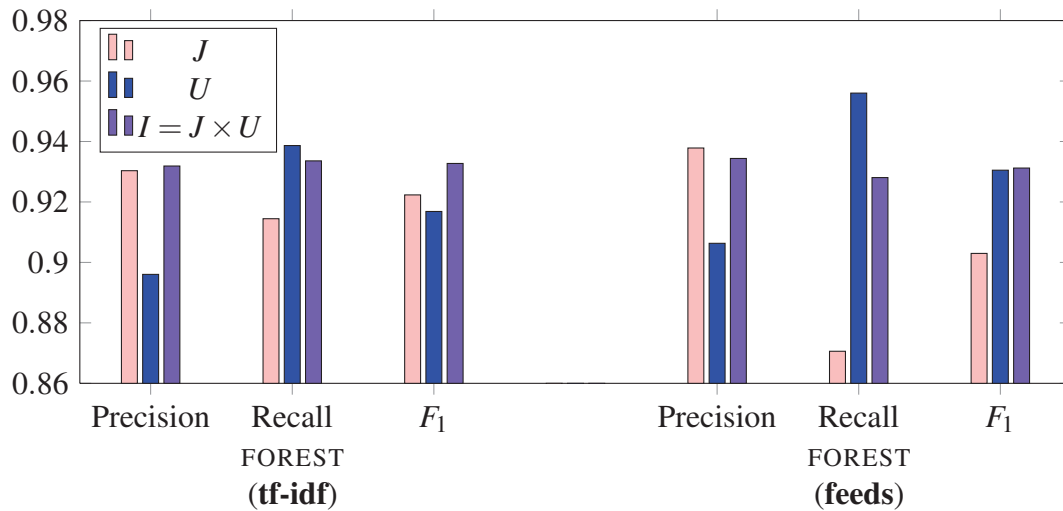


Figure 2.11.: Influence of J and U on mean precision, recall, and the F_1 -measure

Challenges Even if the DOM block is correctly identified, FOREST efficiency can be lowered in some cases by the fact the extraction may also contain comments or related links. The cause is simple: the signifiers can also pinpoint comments or links; when comments or links are integrated together with the main content in a DOM block without a proper logic segmentation, the common block is taken as result.

We have annotated the gold standard regardless of the actual relevance that comments or related links may have to signifiers, with the aim to extract only the main content. It is however difficult to decide whether this type of related content can not be useful for a user that issues a keyword query.

We have made experiments to filter out first, the lists of anchors from the DOM of the document in the pre-processing phase, and second, significant paths that have the keyword “comments” in their signature. However, these heuristics barely improve the results: 1% for the first and 2% for the latter. The reason that we observe in practice is that either the lists of anchors is part of the gold standard for some articles, or the heuristics do not work because the anchors or comments are encoded in multiple, subsequent divs rather than in list items in `ol`, `ul`, etc typical DOM lists types. The improvement is also minimal for other heuristic choices like the use of the tolerant form of a DOM node attribute in definition 2 (1%), and biasing FOREST in favor of deep nodes by a decay level factor in relevance formula (2.7) (2%).

Discussion

We have presented a novel unsupervised technique that mingles wrapper induction with content analysis, for Web pages that share the same HTML template. The algorithm has the originality of using keywords to trace locations of interesting blocks in the Web pages. We filter out significant tag paths and define a measure of relevance at the level of DOM elements. This measure takes into account not only the structural patterns of the elements, but also the content-based informativeness. This approach produces a single, generic tag path that is used to extract the data of interest across the various pages. Being tested in comparison with state-of-the-art approaches for content extraction, for a diversified dataset of Web pages, FOREST demonstrates to be an efficient technique for Web article extraction.

3. Discovering the Semantics of Objects

In order to automatically explore the mass of information existent on the deep Web, many current techniques assume the existence of domain knowledge, which is costly to create and maintain. In this chapter, we present a new perspective on the understanding of this type of structured data source that does not require any domain-specific knowledge. Unlike previous approaches, we do not independently perform the various analysis steps (e.g., form interface parsing, information extraction on response page, object attributes labeling), but integrate them to gain more insight based on their complementarity. Through data extraction, and using the form itself for validating mapping hypothesis, we reconcile input and output schemas in a labeled graph which is further aligned with a generic ontology.

This work has been established in collaboration with Antoine Amarilli, during his internship concerning the improvement of PARIS ontology alignment algorithm by processing joins.

For clarity, we note that this article describes a vision, prototyped for preliminary testing, rather than a full-fledged system. Its worth saying that all advances regarding the individual analysis modules combined in our framework, including the alignment algorithm, can only improve the feasibility and effectiveness of our vision at large scale.

The contents of this chapter have been published at the VLDS workshop [Oita et al., 2012].

§ 1. The Deep Web

The *deep Web* consists of dynamically-generated Web pages that are reachable by issuing queries through HTML forms. A form is a section of a document with special *control*

elements (e.g., checkboxes, text inputs) and associated labels. Users generally interact with a form by modifying its controls (entering text, selecting menu items) before submitting it to a Web server for processing.

Forms are primarily designed for human beings, but they must also be understood by automated agents for various applications such as general-purpose indexing of response pages [Madhavan et al., 2006], focused crawling [Fang et al., 2007; Kumar et al., 2011], extensional crawling strategies (e.g., Web archiving), automatic construction of ontologies that guide the task of Web data extraction [Furche et al., 2011b; Wu et al., 2005b], etc.

However, most existing approaches to automatically explore and classify [Hedley et al., 2006] the deep Web crucially rely on domain knowledge [Furche et al., 2011a; He et al., 2004; Yuan et al., 2009] to guide form understanding [Khare et al., 2010]. Moreover, they tend to separate the steps of form interface understanding and information extraction from result pages, although both contribute [Wang et al., 2004] to a more *authentic* vision on the backend database schema. The form interface exposes in the input schema some attributes describing the query object, while response pages present this object instantiated in Web records that outline the form output schema.

In this paper, we determine a mapping between the input and output schemas which associates the data types corresponding to form elements in the input schema to instances aligned in the output schema. A harder challenge is to understand the *semantics* of these data types and how they relate to the object of the form.

The input–output schema mapping may give us hints, such as the input schema labels, but this information cannot suffice by itself. This has been addressed in related work using heuristics [Wang and Lochovsky, 2003] or an assumed domain knowledge [Senellart et al., 2008] which is either manually crafted or obtained by merging different form interface schemas belonging to the same domain. Domain knowledge is, however, not only hard to build and maintain, but also often restricted to a choice of popular domain topics, which may lead to biased exploration of the deep Web. Another option is to perform attribute labeling for response records, a typical step in IE. However, attribute labeling is a difficult task for which heuristics or, again, a source of semantics (e.g., a gazetteer) were typically used. Heuristics are usually applied, data attributes being matched to form element labels or “voluntary labels” found in the neighborhood.

We present a new way to deal with this challenge: we initially probe the form in a domain-agnostic manner and transform the information extracted from response pages into a labeled graph. This graph is then aligned with a *general-domain* ontology, PARIS [Suchanek et al., 2007], using the PARIS ontology alignment system [Suchanek et al., 2011]. This allows us to infer the semantics of the deep Web source, to obtain new, representative query terms from PARIS for the probing of form fields, and to possibly enrich PARIS with new facts. We then spread the acquired knowledge using the input–output schema mapping to the form interface. As a result, we obtain a full wrapper

for the deep Web source, and we can extract its data to enrich the generic ontology with new entities and relationships. We can also use the ontology to guide the form probing by suggesting attribute values which are compatible with our understanding of the input schema. Hence, the reference ontology and the deep Web source can be said to cross-fertilize. Thus, we know how to probe the deep Web source, and we can use PARIS to guide the probing by suggesting attribute values which are compatible with our understanding of the input schema.

Next, in Section § 2, we describe some important related work. In Section § 3, we present our vision for deep Web analysis. Section § 4 discusses preliminary results of the ontology alignment process.

§ 2. Related Work

[Madhavan et al., 2006] introduces the Google’s *surfacing* approach to deep Web content: Web pages are indexed by simulating form submissions, retrieving answer pages and putting them into the index. Until now, no other strategies have been implemented at large scale, because the technical difficulties encountered with the deep Web are related to the absence of domain-independent, reliable ways to acquire semantics. Specific domain knowledge-based solutions exist, but they do not scale in practice.

Form understanding [Raghavan and Garcia-Molina, 2001] presents the general design for a deep Web crawler and addresses the problem of fully understanding of a form, and more specifically recognizing and responding to simple dependencies between form elements (e.g., city and state; the city must be inside the state specified) We address this question through the use of ontology relations (e.g., *locatedIn*).

[Zhang et al., 2004] writes about the query interface as guided by a hypothetical syntax, which connects semantics to presentation. [Furche et al., 2012a] are taking further this hypothesis and infers a domain *logic*.

Merging input schemas of deep Web interfaces has been used to perform schema integration [Wu et al., 2005a], automatically bootstrap specific domain ontologies [Wu et al., 2005b], Web database sampling [Hedley et al., 2006; Ipeirotis and Gravano, 2002] and classification [Wang and Lochovsky, 2003]. Schema matching has been seen as a crucial step in deep Web data integration because the resulting common query interface (i.e., virtual schema) gives uniform access to multiple sources. As the virtual schema can mediate queries, possible applications are query translation [Liang et al., 2008] and routing [An et al., 2007]

[Wu et al., 2005b] matches different deep Web interfaces belonging to the same domain in order to bootstrap a domain knowledge. This domain is further exploited to *train* concept classifiers with the aim to identify zones in the response pages that contain concepts and instances. [Quattoni et al., 2007] uses conditional random fields, a

classical discriminative model for classification problems. To match attributes between schemas, many other different types of structural, linguistic and semantic features have been proposed. For instance [He et al., 2008] combines multiple matchers based on the Dempster-Shafer theory of evidence.

The main drawback of these approaches is that data integration dramatically relies on the interface schema, whose shallow features (the form structure and labels) are neither complete, nor representative enough for the actual backend database schema. Different sources support different query capabilities. Therefore, the analysis of response pages before assessing a deep Web source as relevant to a certain topic query or schema has been considered necessary [Balakrishnan and Kambhampati, 2011; Fang et al., 2007].

Interface probing To obtain response pages, the form has to be filled in and submitted first. Most approaches described in the literature are domain-specific and use dictionary instances [Senellart et al., 2008]. *Domain-agnostic probing* approaches are more powerful because they do not make such assumptions and incrementally build knowledge that tends to improve the probing and the quality of response pages. However, existing domain-agnostic techniques do not aim at understanding the intensional purpose of the form, but at extensional crawling [Barbosa and Freire, 2004].

Record data extraction and semantic annotation Deep Web response pages are an extremely rich source of semi-structured information. Works dealing with response pages assume the form probing mechanism understood and focus on information extraction (IE) from *Web records*.

We note that a complete IE analysis includes the following three steps:

1. record identification [Caverlee et al., 2004];
2. attribute alignment [Zhu et al., 2006] ;
3. attribute labeling [Wang and Lochovsky, 2003].

Nevertheless, very few works in the large literature on this subject simultaneously deal with these steps or try to benefit from their combination. For instance, [Zhu et al., 2006] performs the first two steps using a hierarchical Conditional Random Fields (CRF). While solutions to record identification and attribute alignment are extensively studied and therefore quite mature, the existing solutions for attribute labeling are typically based on label assignment heuristics.

While a non-trivial *semantic analysis* of records is needed, heuristics [Wang and Lochovsky, 2003] perform the annotation of data object attributes by matching them to form element labels or other “voluntary labels” found in the neighborhood of the input schema, supposing that the input schema could be linked to a domain knowledge.

An empirical observation on a search interface is that if an accurate query is sent to a Web database, the returned result will contain more accurate results, but also less data records. Many works aim at composing high-quality queries that can guarantee accurate results. However, high-quality queries can be issued a priori only if the form is understood. For this reason, [Tiezheng et al., 2007] first classifies form interfaces in specific domains and selects depending on these domains typical instances as query terms.

In comparison with the *input* schema of a form interface that can be more or less assumed as known, the *output* schema is obtained by probing the form and analyzing the response pages. As a rule, extracting the schema of the records [Nestorov et al., 1998] is a clustering process that groups together record features that are structurally similar, on the same response page or across various pages.

Automatically extracting instances from the deep Web with the aim of enriching a domain-specific ontology can improve Web search by reducing the access to external resources such as the Web. Another important motivation is enriching existent ontologies: quoting [An et al., 2008], “the success of the Semantic Web depends on the existence of sufficient amount of high quality semantics contained in ontologies”.

The data extracted from deep Web sources through IE processing can be used to build and/or enrich ontologies [An et al., 2008; Davulcu et al., 2004; McDowell and Cafarella, 2006; Su et al., 2009; Thiam et al., 2008], gazetteers [Furche et al., 2012b] or to expand sets of entities [Wang and Cohen, 2007]. [McDowell and Cafarella, 2006] uses *learning* to effectuate an alignment of deep Web instances to a domain-specific ontology. Instance learning is based on named entity recognition (NER) and comparison with concept labels or named entities in a domain ontology. ODE [Su et al., 2009] in particular gets closer to our work by its holistic approach, but still needs a domain ontology built by matching different deep Web interfaces. A more important difference appears in the annotation of the extracted data from response pages using heuristic rules for label assignment, similar to [Wang and Lochovsky, 2003]. Comparatively, we use an ontology alignment algorithm.

To estimate the semantic similarity between term candidates (i.e., Web labels) and concepts in the ontology in order to perform their alignment, many approaches are available.

[Thiam et al., 2009] proposes an alignment using the Taxomap tool¹, exploiting only syntactic-based similarity measures as term inclusion or *n*-gram similarity. A typical method is the comparison of class and property names of concepts in ontologies using a string distance metric [Stoilos et al., 2005]. Unsupervised NER systems are based on lexical resources either based on an ontology as WordNet [Reynaud and Safar, 2007] or DbPedia [Bizer et al., 2009], or on the Web itself [Cimiano et al., 2005].

The next step is the discovery of the semantic relationships between the entity of the

¹<http://www.lri.fr/~hamdi/TaxoMap/TaxoMap.html>

form and the record attributes; for this, several techniques are proposed in the literature. Typically, statistical and rule-based methods use the instances in a *textual context* in order to infer the relation between them [Chen et al., 2008; Cimiano et al., 2005]. These techniques use thus instance terms to make requests to a search engine, retrieve results, process the snippets using NLP analysis, and extract sentences that are meant to give a context for the relation existing between the respective instances. Another option [Stoilos et al., 2005] is to match the terminology of a given term with a *known* concept using semantic resources such as DBpedia or WordNet [Reynaud and Safar, 2007]. Yet another trend is to use classifiers that can predict specific relations (e.g., *subClassOf*) given enough training and test data [Beisswanger, 2010]. [Suchanek et al., 2006] combines linguistic and statistical analysis to extract relations from Web documents. The closest work to ours may be [Limaye et al., 2010], an approach relying on supervised learning that uses a generic ontology to infer types and relations among the data in a Web table. We deal with the more general setting of deep Web interfaces here, and we propose a fully automatic approach that does not require human supervision. It could be adapted in our deep Web context in the following way: we represent the lists of instances obtained through attribute alignment in response records as *feature vectors*. The aggregation of such vectors for a given deep Web source results in a “Web table”, from which the proposed technique can be applied. However, the method relies on machine learning to identify types, instances and relationships. We propose a fully automated approach instead.

§ 3. Envisioned Approach

We now present our vision of a holistic deep Web semantic understanding and ontology enrichment process, which is summarized in Figure 3.1: a Web form is analyzed and probed, record attribute values are extracted from result pages, and their types are mapped to input fields. While these steps are rather standard and we follow the well-established best practices, they have never been analyzed in a holistic manner without the assumption of domain knowledge that describes the form interface. The novelty of studying these steps in connection comes from their contribution to the formation of a labeled graph which encompasses data values of unknown types and *implicit* semantic relations. This graph is further aligned with a generic ontology for knowledge discovery using PARIS.

§ 3.1. Form Analysis and Probing

The form interface is presented as an *input schema* which gives a prescriptive description of the object that the user can query through the form. The input schema is the ordered list of labels [Ding and Li, 2011] corresponding to form elements, possibly together with constraints and possible values (for drop-down lists and other non-textual input fields).

Probing is the process of submitting a form and retrieving response pages. Important data constraints or properties of the backend Web database can be discovered through well-designed probing and response page analysis. Some may be precious for a crawler that interacts with the form: Are stop words indexed? Which Boolean connectors are used (conjunctive or disjunctive)? Is the search affected by spelling errors?

We perform form probing in an agnostic manner (i.e., without domain knowledge) following [Barbosa and Freire, 2004; Madhavan et al., 2006]. We try to set non-textual input elements or to fill in a textual input field with stop words or with *contextual terms* extracted from non-textual input controls (e.g., drop-down list entries) or surrounding text (e.g., indications to the user). We rely on the fact that many sites provide a generous index (i.e., a response page can be obtained by inputting a single letter). A more elaborate idea is to use AJAX auto-completion facilities.

§ 3.2. Record Identification

If the form has been filled in correctly, we obtain a result page. Otherwise, to identify possible error pages, our method infers a characteristic XPath expression by submitting the form with a nonsense word and tracing its location in the DOM of the response page. This approach uses the fact that the nonsense word will usually be repeated in the error page to present the erroneous input to the user. If not, techniques such as those of [Senellart et al., 2008] can be applied.

If the probing yields a response page which does not contain the error pattern, then we determine in the DOM of the response page, the generic XPath location of Web records using [Oita and Senellart, 2012].

§ 3.3. Output Schema Construction

A way to build the output schema is to use the reflection of a given domain knowledge in response pages [Tiezheng et al., 2007; Wang et al., 2004]. In contrast, we perform *attribute alignment* [Alvarez et al., 2008] for records obtained from different pages.

Since Web records represent subtrees which are structurally similar at DOM level, we extract the values of their textual leaf nodes and cluster them based on their DOM path. The rationale is that the values found under the same record internal path are attributes of the same type. For instance, “Great Expectations” and “David Copperfield” in Figure 3.1 both represent literals of the *title* attribute of a *book* and have a common location pattern.

We define a *record feature* across various response pages as the association between a relevant record internal path and its cumulated bag of instances. The *output schema* for a response page is then defined by the ordered sequence of record features. In practice, we remove uninformative record features from the output schema by restricting ourselves to paths which contain *different* instances across various response pages.

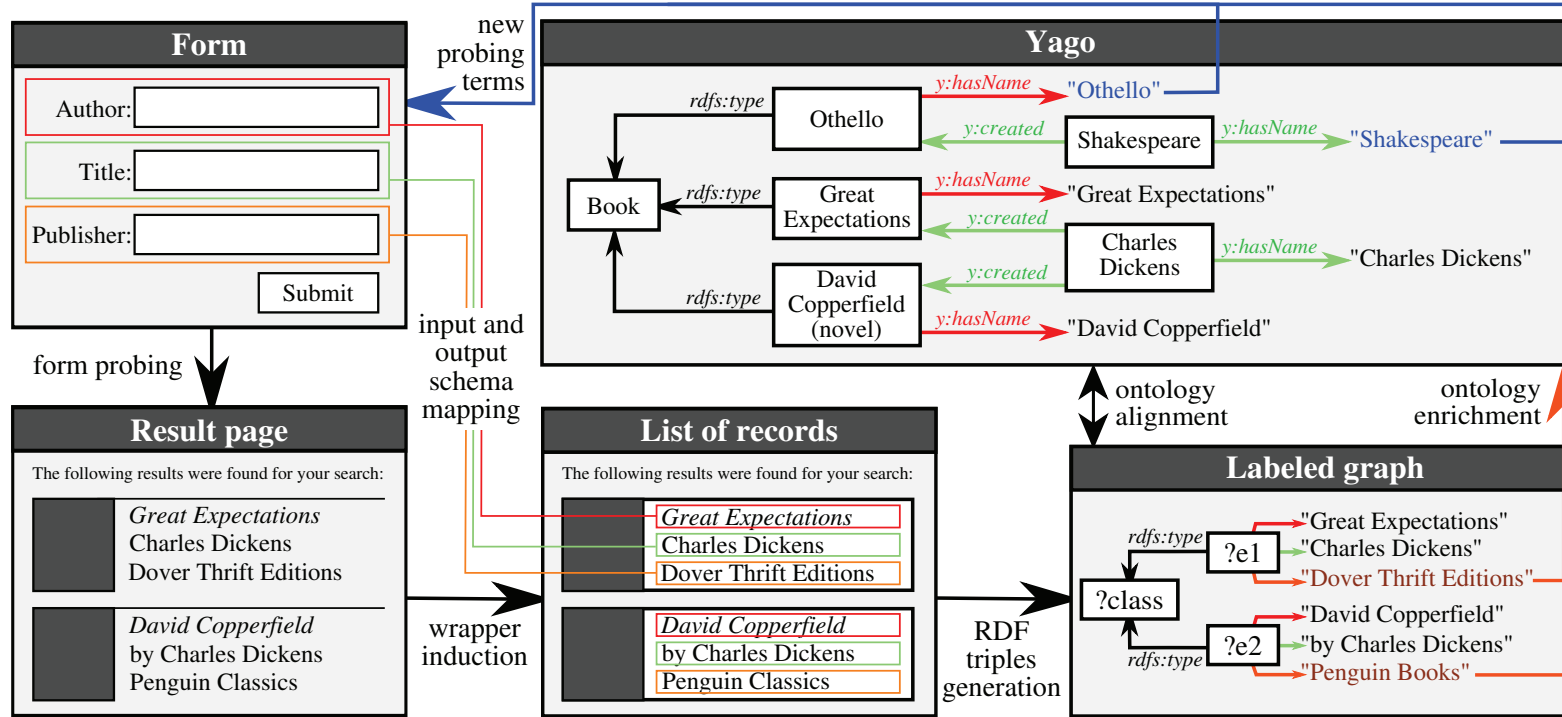


Figure 3.1.: Deep Web form understanding and ontology enrichment: overview of the envisioned approach

§ 3.4. Input and Output Schema Mapping

We align input fields of the form with record features of the result pages in the following fashion. For non-textual form elements such as drop-down lists, we check if their values do not trivially match one of the record features of the output schema. For textual form elements, we use a more elaborate idea. Due to binding patterns, query instances which appear at a certain record internal path should appear again at the same location when they are submitted in the “right” input field for this path. If we submit them in an unrelated field, however, we should obtain an error page or unsuitable results.

Formally, given a record feature f of the output schema, we can see if it maps to a textual input t by filling in t with one of the initial instances of f (say i) and submitting the form. Either we obtain an error page, which means f and t should not be mapped, or we obtain a result page in which we can use f 's record internal path to extract a new bag I of instances for f . In this case, we say that t and f are mapped if all instances in I are equal to i or contain it as a substring (i.e., i appears again at f 's location pattern). We obtain the mapping by performing these steps for all couples (f, t) .

Most of the time, the input–output schemas do not match exactly. The attributes that cannot be matched are usually explicit in the input schema (e.g., given by non-textual inputs, like drop-down lists), or only present in the output schema (e.g., the price of a book).

§ 3.5. Labeled Graph Generation

We represent the data extracted from the Web records as RDF triples [Recommendation, 2012] in the following manner:

1. each record is represented as an *entity*;
2. all records are of the same *class*, stated using *rdf:type*;
3. the attribute values of records are viewed as *literals*;
4. each record links to its attribute values through the relation (i.e., *predicate*) that corresponds to the record internal path of the attribute type in the response page;

These triples describe the entity of the form, for each response record in a star representation 3.2, similar to [Herzig and Tran, 2012]. The aggregation of all these representations, based on implicit, common *rdf:type* relations between attributes, creates a directed, labeled graph. Since the labels are not fully semantically described, and many of them are rather logical before the alignment, it is possible to add much more information to the representation, provided that we have the means to extract it. An idea would be to include a more detailed representation of a record by following the hyperlinks that we identify in its attribute values and replacing them in the original response page with the

DOM tree of the linked page. In this way, the extraction can be done on a more complete representation of the backend database. We can also add complementary data from various sources, e.g., Web services or other Web forms belonging to the same domain.

§ 3.6. Ontology Alignment using PARIS

The ontology that we compile from the result pages is aligned with a large reference ontology. We use YAGO [Suchanek et al., 2007], though our approach can be applied to any reference ontology. We use PARIS [Suchanek et al., 2011] to perform the ontology alignment. Unlike most other systems, PARIS is able to align both entities and relations. It does so by bootstrapping an alignment from the matching literals and propagating evidence based on *relation functionalities*. Through the alignment, we discover the class of entities, the meaning of record attributes and the actual relation that exists between them.

PARIS can be expected to perform better with a larger sample of result page data to align, and annotates the alignments it produces with confidence scores: hence, we can perform the alignment incrementally, starting on a small sample and performing more probing to extend the sample until we obtain an alignment with satisfactory confidence. Two main adaptations are needed to use PARIS in the deep Web data alignment process.

First, extracted literals usually differ from those of YAGO because of alternate spellings or surrounding stop words. A typical case on Amazon is the addition of related terms, e.g., “Hamlet (French Edition)” instead of just “Hamlet”. To mitigate this problem we normalize the literals, eliminate punctuation and stop words. Pattern identification in the data values of the same type could increase the probability of extracting cleaner values. We are working on a way to index YAGO literals in a manner that is resilient to the small differences we wish to ignore. A promising approach to do this is *shingling* [Broder et al., 1997].

Second, an entity-to-literal relation in the labeled graph may not necessarily correspond to a single edge in the reference ontology, but to a sequence of edges. This amounts to a *join* of the involved relations; a typical case in our prototype is the “author” attribute which is linked to a record entity through a two-step YAGO path “y:created y:hasPreferredName”. To ensure that the alignment with joins, typically costly, can be performed in practice, we limit the maximal length of joins. A consequence is that PARIS will explore a smaller fraction of YAGO in the search for relations relevant to the data of our labeled graph.

In addition to the use of record attribute values as literals, PARIS could use the form labels (through the input–output mappings) to guide the alignment and favor YAGO relations with a similar name. Some record instances do not align with any literal of the ontology. The cause is that they represent information which is unknown to YAGO.

§ 3.7. Form Understanding and Ontology Enrichment

Thanks to the ontology alignment, we obtain knowledge about the data types, the domains and ranges of record attributes, and their relation to the object of the form (in our case, a book). The propagation of this knowledge to the input schema through the input–output mapping (for the form elements that have been successfully mapped) results in a better understanding of the form interface. On the one hand, we can infer that a given field of the Amazon advanced search form expects author names, and leverage YAGO to obtain representative author names to fill in the form. This is useful in intensional or extensional automatic crawl strategies of deep Web sources. On the other hand, we can generate *new* result pages for which data location patterns are already known and enrich YAGO through the alignment that we once determined.

There are three main possibilities to enrich the ontology. First, we can add to the ontology the *instances* that did not align. For instance, we can use the Amazon book search results to add to YAGO the books for which it has no coverage. Second, we can add *facts* (triples) that were missing in YAGO. Third, we can add the *relation types* that did not align. For instance, we can add information about the publisher of a book to YAGO. This latter direction is more challenging, because we need to determine if the relation types contain valuable information. One safe way to deal with this *relevance problem* is to require attribute values to be mapped to a form element in the input schema. We can then use the *label* of the element to annotate them.

Facts themselves can be annotated with a probability depending on its generation confidence from result pages. We did not need to use this information for the moment, however, but it may be useful for future developments. In the former case, the non-alignment is even beneficial, because it can avoid semantic drift. as we have multiple instances of the same type, if even a single one aligns to the ontology, then the discovered type propagates to all sibling instances. That means that the ontology can be enriched with new instances and new facts.

Using the alignment, we can remember that a certain relation type can be obtained by querying the form in a certain way and extract the information using the appropriate existing data wrapper (i.e., the record feature path). Having discovered the type of a record feature and its corresponding relation with the entity using PARIS, Having understood the probing, more precise response pages can be further obtained. New structured data extracted from record pages (using the known location patterns) are automatically recognized from the semantic point of view using the mappings of PARIS.

The generated triples can be directly added to the ontology can improve the process of recognition. For record features for which PARIS could not align any instance, not only is the type of instance unknown, but also the relation. If we consider that they are valuable instances then a way to do this is to verify using the input–output mapping if their record feature maps to an element of the input schema. For example, knowing the “Publisher” label in correspondence with instances of a “Publisher”, we can create a new Amazon

relation “amazon:isPublishedBy” of a YAGO book and add the respective instances to YAGO.

The reason can be that these instances represent junk text that happens to appear in a structured fashion in the response record (e.g., “In Stock.”) or that the related entities are unknown to YAGO, whose source of facts is Wikipedia. In the former case, the non-alignment is even beneficial, because it can avoid semantic drift. In the latter, as we have multiple instances of the same type, if even a single one aligns to the ontology, then the discovered type propagates to all sibling instances. That means that the ontology can be enriched with new instances and new facts.

Using the alignment, we can remember that a certain relation type can be obtained by querying the form in a certain way and extract the information using the appropriate existing data wrapper (i.e., the record feature path). Having discovered the type of a record feature and its corresponding relation with the entity using PARIS,

Having understood the probing, more precise response pages can be further obtained. New structured data extracted from record pages (using the known location patterns) are automatically recognized from the semantic point of view using the mappings of PARIS.

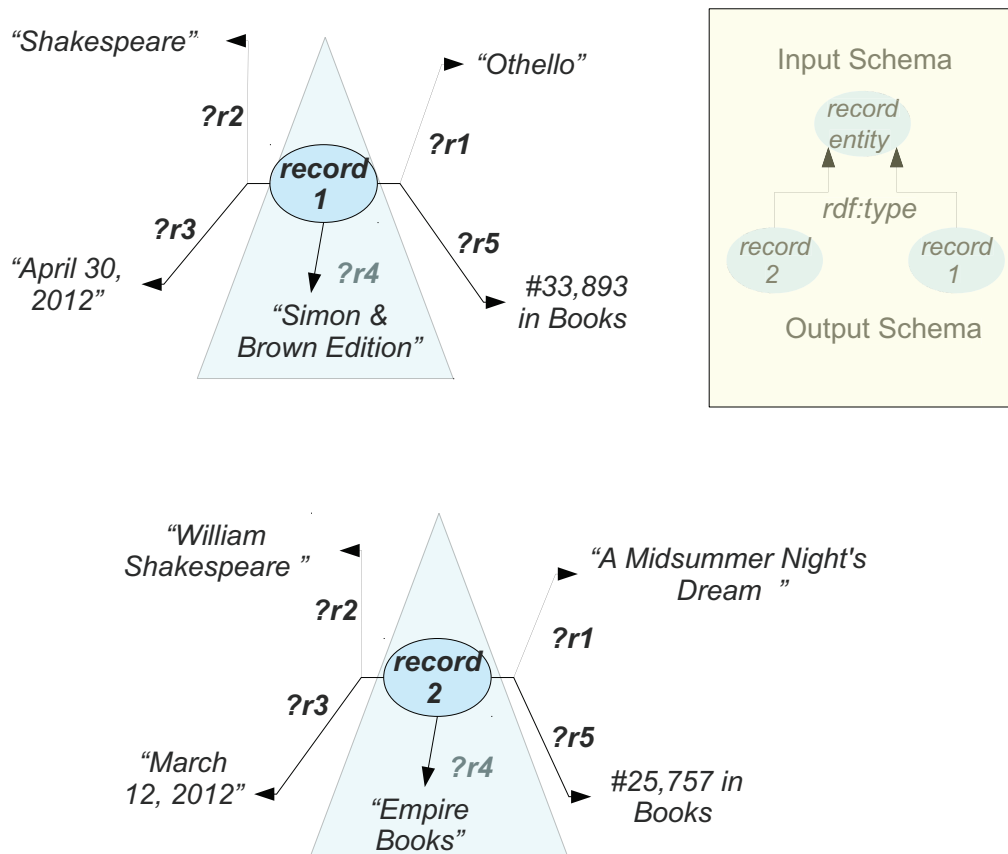
The generated triples can be directly added to the ontology. For record features for which PARIS could not align any instance, not only is the type of instance unknown, but also the relation. If we consider that they are valuable instances to be added to the ontology, then a way to do this is to verify using the input–output mapping if their record feature maps to an element of the input schema. If so, we use the label of the corresponding form element to guide the alignment for the discovery of a relation (say, find if there is a “Publisher” class in YAGO and check its instances) or we transform the record internal path of this feature in a new relation using this label. For example, knowing the “Publisher” label in correspondence with instances of a “Publisher”, we can create a new Amazon relation “amazon:isPublishedBy” of a YAGO book and add the respective instances to YAGO.

PARIS can be expected to perform better with a larger sample of result page data to align, and annotates the alignments it produces with confidence scores: hence, we can perform the alignment incrementally, starting on a small sample and performing more probing to extend the sample until we obtain an alignment with satisfactory confidence.

§ 4. Preliminary Experiments

We have prototyped this approach for the Amazon book advanced search form². Obviously, we cannot claim any statistical significance of the results we report here, but we believe that the approach, because it is generic, can be successfully applied to other sources of the deep Web.

²<http://www.amazon.com/gp/browse.w3c99htmlSpec?node=241582011>



?r_ = type of relation constructed based on the record internal path

Figure 3.2.: Triples generation using deep Web IE results

Our preliminary implementation performed agnostic probing of the form, wrapper induction, and mapping of input–output schemas. It generated a labeled graph with 93 entities and 10 relation types out of which 2 (title and author) are recognized by YAGO. Literals underwent a semi-heuristic normalization process (lowercasing, removal of parenthesized substrings). We then replaced each extracted literal with a similar literal in YAGO, if the similarity (in terms of the number of common 2-grams) was higher than an arbitrary threshold.

We aligned this graph with YAGO by running PARIS for 15 iterations, i.e., a run time of 7 minutes (most of it was spent loading YAGO, the proper computation took 20 seconds). Though the vast majority of the books from the dataset were not present in YAGO, the 6 entity alignments with best confidence were books that had been correctly aligned through their title and author. To limit the effect of noise on relation alignment, we recomputed relation alignments on the entity alignments with highest confidence; the system was thus able to properly align the title and author relations with “y:hasPreferredName” and “y:created y:hasPreferredName”, respectively. These relations were associated to the record internal paths of the output schema attributes and propagated to form input fields.

Discussion Our vision is that of a holistic system for deep Web understanding and ontology enrichment, where each stage of the process (form analysis, information extraction, schema matching, ontology alignment, etc.) would benefit of every other part. Many challenges remain to be tackled, and we detail more about this in the next part.

Part III.
Discussion

§ 1. Conclusions

The outline of the next thesis part concerning the final discussion is as follows: we give some global conclusions over the work that we have performed until now, we continue with the presentation of some applications that are particularly relevant to our contributions, and, finally, we discuss further work.

Timestamping and change detection We have first performed an aggregate analysis of different state-of-the-art strategies for change detection [Oita and Senellart, 2011] and then focused more in detail on an approach based on Web feeds. We have also presented some statistics concerning the temporal aspects of feeds that leverage the temporal metadata in the change detection process [Oita and Senellart, 2010b]. In the cases where Web pages are linked to a Web feed, one application that we have presented in [Oita and Senellart, 2010a] is the reconstruction of Web pages from their main content, together with other properties extracted from other sources (e.g., the feed itself) and a common template.

SIGFEED We have presented SIGFEED, a feed-based algorithm [Oita and Senellart, 2010b] for finding the data objects that correspond to Web feed items. These items refer to the interesting content in a Web page, but also present useful metadata about it. This top-down algorithm has achieved a good accuracy and is fast. The overall idea is simple: extract the DOM node of the article of interest as the lowest common block ancestor of multiple significant nodes. The relevance of a DOM node is computed based on the keywords that we get from the title and description of a feed item.

FOREST We got further with the keyword-based strategy, in a more general context. As a result, we have developed a more mature algorithm named FOREST. This is an unsupervised technique that mingles wrapper induction with content analysis, for main content extraction from Web pages. FOREST [Oita and Senellart, 2012], similar to SIGFEED, leverages keywords to trace locations of interesting blocks in sample Web pages, but can acquire these keywords independently of the existence of Web feeds associated with those pages. FOREST successfully exploits both potential sources of keywords: Web feed items and frequent, discriminative terms occurring on Web pages, obtained through a *Tf-Idf* analysis. This algorithm obtains state-of-the-art results for a large spectrum of Web page types, and it is based on principles that are easy adaptable to many contexts e.g., CMSs, deep Web.

Deep Web and semantic annotation of objects We have used FOREST for the extraction of data objects (i.e., response records) from deep Web sources. Here, the signifiers came from the keywords used during the form interface submission, and sample pages

sharing the same template were simply the response pages that we had obtained by submitting the form.

The general aim is, after data object identification using FOREST, to further perform a semantic annotation of objects [Oita et al., 2012]. We have thus presented our vision concerning a holistic framework for deep Web understanding and ontology enrichment, where each stage of the process (form analysis, information extraction, schema matching, ontology alignment, etc.) would benefit of every other part. This is an ambitious project, but our current prototype already exhibits promising results.

§ 2. Further Research

We next highlight some further work plans and other long-term challenges related to the contributions that we have presented in this thesis.

Web Archiving

Introduction Data on the Web has an ephemeral nature: new pages are added, old ones are deleted, other are evolving [Adar et al., 2009]. For this reason, a heritage preservation mission has been started by various Web archiving actors³.

Web archiving is the process of continuously collecting the content of the World Wide Web to ensure its preservation and to allow access to information even after it disappears from the Web [Masanès, 2006]. Many organisations actively run Web crawls and efforts are done to converge into a unified, global archive collection⁴.

An archiving crawler follows the same basic steps as a search engine crawler does to construct indexes for Web pages. However, it does not drop indexes to obsolete Web pages when new ones are discovered, but stores and references versions in time. The final result is a collection of Web pages that can be browsed off-line and, in ideal settings, temporally and semantically queried. Moreover, using data mining and visualization, versioned Web content is also used to detect trends and evolution on a certain topic of interest.

Perenniality of Web archives Time is a particularly important factor of influence also for interpreting crawled content. While data might remain intact, the way we understand and represent it changes, mostly due to the fact that the language itself, the culture and the technology evolve.

One of the most serious challenges encountered in Web archiving is when the format of the archived content becomes obsolete. The possible solutions are software or hardware

³<http://internetmemory.org/en/>

⁴<http://www.mementoweb.org/>

emulation, content migration, and the inclusion of a proxy that incorporates format translation capabilities [Hunter and Choudhury, 2003; Strodl et al., 2007; Swaney et al., 2005].

In order to enable *format translation*, rather than emulation or migration, we advocate for the encapsulation of the main information of the crawled Web page, and the storage of Web data objects independently of their original encoding format. Being decoupled and better described, data objects can be *directly and dynamically linked* with other sources on the Web: e.g., Semantic Web, Social Web, etc.

Content presentation and user interactivity Extracted in a way that captures various content dimensions (or features), Web objects can improve the data organisation, incremental crawl efficiency and the isolation of the presentation from content. This will enhance the possibility of rendering the content in new ways, by adapting it, for instance, to user preferences. One promising direction in this sense is the use of mashups, in which the data objects, user presentation preferences and new features from existing applications (e.g., Google Maps) can be integrated.

Semantic exploitation of collections The extraction of data objects is, in our opinion, already necessary in order to provide value-added services, e.g., the semantic analysis of content in time, trends and evolution, complex user queries, etc. To add new interacting applications or allow adaptation to new specifications, we need a semantic layer that *connects* the data to meaningful structures. A Web archive that contains semantically annotated data objects (possibly, through techniques presented in Chapter 3) can be used more effectively by analysts than just a collection of Web pages.

Another advantage is isolation of features or content through views; for instance, a linguist could focus on the new terms that appear in the main content of multiple Web articles, without bothering about the terms that appear in whole, topically heterogeneous Web pages or their template.

Efficient storage and processing On the long term, the archiving of Web objects with reference to their template could optimize not only the storage, but also the processing. While it can be argued that, for Web archiving in particular, the original Web page is necessary, this is more related to the requirement of well placing the Web content in its precise context. By default, this context is a Web page. However, we believe that a content's context is not only changing, but is also related to much more than its original Web page.

Incremental Web crawling Remember that Web pages can have very lively dynamics, in some cases with update intervals of the order of the minute. In these conditions, it becomes unreasonable to try and capture every successive version of the corresponding

channel Web page. Segmentation of pages and better filtering of irrelevant changes seem to be the best method to deal with the speed rate at which Web content evolves. Web feeds are typically associated with this type of dynamic pages. We believe that investing little effort in order to exploit their specificities in the crawling process can make a difference, as Web feeds become commonplace.

Web content timestamping With the large number of sources of timestamping hints, it should indeed be possible to effectively estimate the freshness of a Web page. Experimental studies on the reliability of these sources, perhaps in more specific contexts (a given Web server software, a given CMS, etc.), which could provide more insight into optimal strategies for timestamping Web pages, are still to be carried out. An interesting line of research would be to complete the existing ideas on the *Temporal Web*, where objects (i.e., complex, structured entities) are reliably timestamped and put in a global network which is time-aware in which there exist events linked together, and we can observe the hidden factors of influence of each.

Main Content Extraction

SIGFEED Although being a good heuristic method in the case of Web articles generated by CMSs and having a Web feed associated, SIGFEED suffers from several drawbacks. First, this method is obviously possible only for Web pages which are linked to a Web feed. Second, the use of the `div` block heuristic; this manner of grouping HTML content can change in the future, which can make the method obsolete, unless another type of block element can be substituted to it. Third, this technique does not make use of the common template shared by Web pages for a given Web channel. We believe that using several sample Web pages in the analysis is not a constraint, but an advantage. Due to their availability, their explicit structural similarities and implicit semantic relations concerning the types of information across different Web pages, exploiting the context created by sample pages can only be beneficial.

FOREST has been designed to address the drawbacks of SIGFEED in terms of genericity. FOREST has been tested on a dataset that is based on a feed crawl. This is an original way of acquiring Web pages that share the same template, but is also adapted to our extraction target: a Web article. As further research, we would want to test FOREST on other types of datasets, namely on sample Web pages that contain response records as objects. Datasets that group pages sharing the same template for object extraction exist, but have been, at our best knowledge, considered only in the context of *response records* generated from the submission of a Web form. We have successfully tested FOREST for the sample pages generated by the Amazon advanced search form. We therefore believe that FOREST ideas can be used on a broader range of settings.

From another perspective, other possible sources of keywords that can be tested are *anchor text*, terms occurring in search engine *query logs*, semantic annotations associated to the Web page (e.g., RDFa, microformats), etc.

Deep Web and Ontology Enrichment

As we have seen from the last chapter, our vision on the deep Web comprehension is based on objects semantics discovery. This needs a complex orchestration of approaches belonging to information extraction, information integration, possibly Ajax processing, etc.

Many challenges remain to be tackled in this domain, and concerning the solution that we have proposed, we mention:

1. test the vision approach against formal datasets of deep Web form interfaces. Although we have prototyped the approach, and it exhibits promising results, further steps have to be done in order to transform it into a mature project.
2. demonstrate the effectiveness of the ontology alignment technique for a *variety of subjects*. In our prototype, the domain of the Amazon form is books. More tests should be performed in order to verify the feasibility of the alignment with PARIS when the domain of the form object is not well covered, and what can be done when it is not covered at all in YAGO.
3. proper management of the confidence in the results of each automatic task, especially when they are used as the input of another task. For instance, imperfect record data extraction would propagate to literal matching. One document may contain both structured and unstructured parts. Except for named entities, instances are often drowned in text, so they are not easily dissociable. Having a proper way of dealing with this cases, we could make the system more resilient to outliers and noise.
4. the identification of new relations of interest among those extracted from a deep Web source and aligned with the ontology. This is a difficult task because the description of the relation would not exist a priori, so it would need to be automatically inferred from the context.
5. integration of information contained in several different deep Web sources of the same domain for the construction of a virtual domain schema. For broad Web search on deep Web content, better defining the boundaries of a domain knowledge through richer, reachable descriptions (from sources like the Semantic Web, Web services, peer communities through question answering, etc.) is still work in progress.

6. the switch to structured from keyword-based queries needs the identification of the relevant domain *of a query*; an interesting further work would be to infer the domain of the query using the alignment of deep Web data with YAGO. For this, we expect complex join processing to be necessary.

Part IV.

Résumé en français

Inférer des Objets Sémantiques du Web Structuré

§ 1. Contexte de recherche

Contenu Web éphémère L'ensemble des contenus du Web est devenu une source d'informations extrêmement précieuse. Afin d'accéder facilement à l'information enrichie chaque jour, les moteurs de recherches utilisent des robots de crawl pour indexer les ressources hyperliées du Web. Les technologies de crawl ont également une utilité importante dans le domaine de l'archivage du Web. Le crawl de versions des pages Web doit être effectué régulièrement, de façon incrémentale, afin d'assurer l'accès continu à l'information, malgré la nature éphémère du contenu numérique.

Trouver des moyens pour gérer l'explosion de l'information tout en triant ce qui est pertinent est un axe important de recherche, dont l'importance augmente avec la facilité de chacun de produire des nouvelles données en utilisant les systèmes de gestion de contenu (typiquement utilisés pour les blogs, wikis, réseaux sociaux, etc.). Dans ce contexte, les flux Web ont un grand potentiel et une direction de cette thèse est de montrer leur utilité dans la surveillance, le filtrage et l'extraction de contenu Web pertinent.

Le Web structuré La croissance continue du Web est due à la multiplication des sources, mais aussi à l'automatisation des moyens de génération de contenu. Le contenu est en effet de plus en plus dynamique et difficile à « saisir », mais, en même temps, présente encore plus de motifs structurels qui facilitent en pratique la tâche d'extraction de contenu pertinent et son annotation sémantique. Avoir un moyen automatique d'exploiter les motifs du Web structuré pour l'enrichissement du Web sémantique et, inversement, découvrir la sémantique des motifs structuraux associés aux données du Web, aurait un impact important sur la façon dont l'information est traitée aujourd'hui sur le Web.

La nature dynamique du Web est visible à travers son contenu : des données sont constamment ajoutées, supprimées ou modifiées. Comme tous les changements ne sont pas importants, une attention particulière doit être accordée au type de contenu qui est affecté par le changement, de manière à éviter un traitement et un stockage inutiles.

Du point de vue de la cohérence sémantique, une page Web peut inclure divers sujets dans des parties différentes de la page ou, au contraire, un seul sujet peut s'étaler sur plusieurs pages du même site Web. En dépit de l'hétérogénéité évidente du contenu,

même au niveau d'une seule page Web, les systèmes de recherche d'information actuels fonctionnent encore indivisiblement au niveau d'une page Web, cette dernière étant considérée comme l'unité d'information de base retournée en réponse d'une requête utilisateur.

Notre objectif tout au long de cette thèse est de présenter des méthodes automatiques qui visent à obtenir une unité d'information de base plus épurée, plus cohérente et qui peut être donc mieux décrite de manière sémantique (par conséquent, facilement fiable au Web sémantique) pour améliorer les recherches sur le Web ou l'archivage du Web. Nous appelons une telle unité d'information de base un objet de données du Web, qui représente un fragment logique d'une page Web (ou même qui s'étend à travers plusieurs pages Web), qui présente une structure bien définie et un contenu sémantiquement cohérent. Nous étudions ici les moyens de repérer automatiquement ces objets dans les pages Web et d'inférer leur différentes propriétés du point de vue temporel ou sémantique.

Le Web sémantique et le Web caché Du point de vue de l'homme, le Web représente la plus grande base de connaissances jamais construite. Toutefois, pour les agents automatisés tels que les robots Web, la sémantique que les humains reconnaissent n'est pas explicite : elle doit être ou bien construite *a priori* et utilisée comme composant externe, ou bien directement extraite au cours du traitement des pages Web. Cette dernière possibilité fait usage du contexte des pages traitées et utilise l'analyse de plusieurs dimensions des données (dont une ontologie, si nécessaire), mais peut avoir l'inconvénient d'être plus coûteuse. Typiquement, les ontologies, qui sont semi-automatiquement ou manuellement construites, aident les programmes pour la découverte de sémantique.

Une grande partie de l'information sur le Web se trouve sur le Web caché, qui fournit des points d'entrées aux bases de données accessibles via des formulaires HTML ou des services Web. Ces pages sont générées à partir des données et des scripts à l'interaction d'un utilisateur qui soumet une requête en remplissant le formulaire avec des paramètres. Un crawler qui doit faire la soumission automatiquement n'a pas l'intelligence de reconnaître le schéma et la sémantique du formulaire. Pour cela, la plupart des approches pour la compréhension du Web caché sont basées sur un domaine de connaissance spécifique au formulaire qui est donné en entrée au crawler. En raison de la nécessité de construire ces bases de connaissances pour un domaine précis, ces approches ne sont pas adaptées pour l'usage sur le Web à grande échelle : un schéma virtuel et universel d'une très bonne qualité comme ces approches l'envisagent n'est pas réaliste vue l'hétérogénéité du Web. Il faudrait donc gérer l'incertitude et exploiter les liens sémantiques implicites.

Dans cette thèse, nous plaidons pour l'exploration du *contexte* créé par des objets Web générées par la même source (ce qui est le cas pour le Web caché, mais plus généralement, pour le Web structuré). Grâce aux motifs structurels identifiés au niveau de l'arborescence DOM des pages Web qui sont partagés par tous les objets existants sur ces pages (schéma de sortie), nous pouvons former un schéma contextuel en interaction avec le schéma

d'entrée (obtenu en faisant une analyse du formulaire) et une ontologie générique. La correspondance obtenue à l'aide d'un alignement d'ontologies, même partielle, peut être suffisante, car les motifs structurels sont associés à une analogie de type (p. ex., *rdf:type*) qui diffuse la sémantique obtenue, par exemple, pour une des instances à toutes les autres qui partagent le même motif structurel. De cette manière, les pièces manquantes en ce qui concerne le sens canonique des données (p. ex., concepts, relations) peuvent être découvertes.

§ 2. Description des contributions

Le chapitre 1 examine les principales approches existantes qui sont utilisées pour inférer la dynamique de pages Web, par l'extraction ou l'estimation de leurs propriétés temporelles. Nous concentrons notre attention sur les techniques et les systèmes qui ont été proposés au cours des dix dernières années et nous les analysons pour obtenir un aperçu des solutions pratiques et les meilleures pratiques disponibles. Nous visons à fournir une vue analytique de la gamme de méthodes utilisées pour évaluer la dynamique de pages Web, et les distinguer en plusieurs types, en fonction de leur nature statique ou dynamique. Pour les méthodes dynamiques reposant sur la comparaison des versions successives d'une page, nous détaillons la modélisation de la page Web et les métriques de similarité utilisées dans le processus de comparaison.

Nous continuons avec une étude approfondie sur les flux du Web, considérés comme possibles instruments qui, lorsqu'ils sont disponibles, peuvent être utiles sur plusieurs plans. Nous faisons aussi quelques statistiques générales sur un corpus des flux que nous avons obtenus par un crawl continu, plusieurs fois par jour, pendant un mois, afin d'observer la dynamique des pages associées. Nous découvrons par la suite que les flux, grâce à leur nature descriptive, aident aussi à l'identification et l'extraction des objets Web, ce qui s'avère être un point important pour l'efficacité de la détection des changements.

Le chapitre 2 commence par la présentation d'une technique qui utilise des indices linguistiques existantes dans les éléments d'un flux afin d'acquérir un ensemble de mots-clés que nous considérons pertinents pour le contenu intéressant des pages Web liées au flux analysé. Le but final est d'identifier et d'extraire ce contenu d'intérêt. Par la suite, nous utilisons les intuitions obtenues par l'étude du problème d'extraction d'objets à l'aide des flux dans un cadre plus général, indépendamment de l'existence de ceux-ci.

La principale contribution de cette thèse est liée à l'identification, l'extraction et la description sémantique des objets du Web à partir des pages qui partagent des motifs structurels.

Les techniques non supervisées pour l'extraction de données à partir des objets du Web utilisent des similitudes structurelles d'objets pour exclure le modèle qui est en

général commun à tous (c.-à-d., leur disposition statique). L'extraction non-supervisée de ce contenu (en excluant la mise en page et les parties qui ne sont pas pertinentes) est d'intérêt dans des nombreuses applications, telles que l'indexation et l'archivage. Toutefois, en raison de la sophistication des modèles et de leur évolution vers une inclusion d'éléments dynamiques tels que des applications, mais aussi du contenu qui n'est pas sémantiquement lié (p. ex., publicité), une approche standard n'est plus suffisante en soi pour identifier le contenu d'intérêt. La difficulté est donc accrue par l'hétérogénéité des pages Web et leur structure plus complexe. Ce défi a été la motivation de nombreuses techniques qui visent à segmenter une page Web afin d'obtenir des pièces cohérentes du point de vue sémantique.

De plus en plus présents sur le Web, les flux Web (en format RSS ou Atom) sont des documents XML qui caractérisent la partie dynamique d'un site Web. En révélant un bref résumé des pages Web qui viennent d'apparaître sur un site donné, les flux Web sont créés principalement pour aider les lecteurs du contenu Web à gérer la masse d'information. Nous présentons une stratégie (SIGFEED) qui extrait, en utilisant les mots-clés qui sont inférés pour chaque objet à partir des méta-données du flux, le contenu intéressant des pages Web associées à ce flux. Comme les articles présents dans ces pages sont le plus souvent uniquement résumés, l'intérêt est de l'extraire dans sa forme complète.

Le deuxième algorithme, FOREST, qui est indépendant de l'existence des flux, nécessite plusieurs pages Web (au moins deux) qui sont générées par la même source (c.-à-d., au même système de gestion de contenu). Cela est le cas pour la plupart des pages de blogs ou d'actualités, et plus généralement, pour les pages du même site qui partagent le même modèle structurel (ce qui revient à faire en pratique une étape préalable de classification structurelle pour les pages de n'importe quel site Web).

Les systèmes de gestion de contenu aident généralement les sites Web qui ont une forte intensité de données. Grâce au modèle de structure fixe qui reçoit du contenu variable (ayant comme source une base de données), des objets de données sont formés. Ils ont la propriété d'être minimaux en termes de structure, tout en exprimant le maximum de contenu informatif. Une vaste source d'objets est aussi le *Web profond*. Dans ce cadre, les objets se trouvent dans des pages Web « cachées » derrière des formulaires (interfaces Web), et qui sont générées dynamiquement en réponse à une requête utilisateur.

Le chapitre 3 traite de l'extraction d'objets Web à partir du Web profond, mais surtout de leur annotation sémantique à travers l'alignement d'ontologies. Similaires aux pages générées par un système de gestion de contenu, les pages réponse du Web profond présentent une similitude structurelle claire mais, par contre, ont la tendance de présenter les divers objets obtenus en réponse à une requête utilisateur sous forme d'enregistrements listés sur la même page. Nous avons appliqué l'algorithme de FOREST pour l'identification des enregistrements en utilisant comme mots-clés les termes qui sont introduits par l'utilisateur (ou par un programme) lors de la soumission du formulaire.

On a réalisé le prototype d'une application qui effectue la description sémantique des objets Web obtenus par la soumission du formulaire de recherche avancée d'Amazon⁵. Les étapes suivantes sont effectuées dans notre prototype :

1. identification des objets ;
2. classification des attributs des objets en fonction de leurs chemins DOM communs, à l'intérieur du sous-arbre défini par FOREST ;
3. construction d'un schéma initial, incomplet des objets (en utilisant les pages réponse et le formulaire) ;
4. construction d'un graphe étiqueté en utilisant les relation de type (c.-à-d., *rdf:type*) ; pour les attributs de chaque classe et en considérant tous les enregistrements du même type ;
5. étiquetage des types et relations inconnus par un alignement avec YAGO.

L'identification de la sémantique des objets de données est réalisée par la construction d'un graphe sémantique, qui décrit en termes canoniques par des triplets RDF le type d'un objet et de ses valeurs possibles, aussi que les relations qui existent entre l'objet du formulaire et ses attributs.

Aperçu des principales contributions de cette thèse

- une vue d'ensemble des stratégies et des meilleures pratiques pour dériver les *aspects temporels* des pages Web, ainsi que des flux du Web pour ce but particulier (chapitre 1) ;
- deux techniques basées sur des mots-clés pour l'extraction du contenu structuré pertinent (qu'on structure autour du terme *objet du Web*) existant à travers des pages Web générées par la même source, avec un modèle similaire. Les mots-clés sont soit automatiquement acquis, soit extraits à partir des méta-données présentes dans les éléments des flux du Web, et guident le processus d'identification d'objets (chapitre 2) ;
- un cadre générique pour découvrir le modèle sémantique d'un objet du Web (profond, en l'occurrence) en représentant les similitudes implicites entre les attributs des objets et entre les relations comme un graphe étiqueté. Ce graphe est aligné avec une ontologie générique comme YAGO pour la découverte des types et des relations inconnues (Chapitre 3).

Le contenu de cette thèse est basé sur des travaux publiés ou soumis, et des détails plus précis sont donnés au début de chaque chapitre.

⁵http://www.amazon.com/Advanced-Search-Books/b?ie=UTF8&*Version*=1&node=241582011&*entries*=0

§ 3. Extraction du contenu pertinent

On décrit par la suite des techniques de recherche d'information qui modélisent d'une manière nouvelle la notion de pertinence, par rapport à des mots-clés (automatiquement acquis), pour l'identification des objets du Web.

Le contenu textuel des sites Web modernes est, dans la très grande majorité des cas, produit par un système dédié de gestion de contenu. Un tel logiciel génère des pages Web en remplissant un modèle prédéfini avec les informations récupérées à partir des bases de données. Dans ce processus, le contenu d'origine est transformé dans un véritable document HTML, où le texte est caché parmi un modèle (ou schéma) [Grumbach and Mecca, 1999] de balisage HTML, partagé par plusieurs pages d'un même site Web.

Certaines parties d'une page Web sont donc *significatives*, tandis que d'autres sont là pour renforcer la mise en page et ajouter des informations contextuelles, structures de navigation, de la publicité ou commentaires, etc. Les éléments qui font partie du modèle de la page peuvent pourtant changer d'une page à l'autre, donc ne peuvent pas être supposés être entièrement statiques. En outre, ce type d'information additionnelle peut même prendre plus de volume que les informations utiles [Gibson et al., 2005]. Identifier et extraire le contenu principal d'une page Web est une tâche difficile [Bar-Yossef and Rajagopalan, 2002; Caverlee et al., 2004; Kohlschütter et al., 2010; Pasternack and Roth, 2009; Ramaswamy et al., 2004; Song et al., 2004], avec de nombreuses applications dans le domaine d'exploration de données : les moteurs de recherche indexent les pages Web pour leurs parties informatives ; les utilisateurs mêmes sont principalement intéressés par le contenu principal, ou souhaiteraient l'extraire pour améliorer la lisibilité ou l'accessibilité ; les archivistes et analystes du Web souhaitent étudier l'évolution du contenu indépendamment des variations de la mise en page.

Un choix important dans la conception des sites Web modernes, et une conséquence de l'utilisation de systèmes de gestion de contenu, est que les pages Web partagent des motifs structurels. Ceux-ci peuvent être facilement retrouvés dans l'arborescence DOM [Crescenzi et al., 2005]. Cette similitude structurelle entre les pages Web d'un même site est utilisée par les techniques d'extraction d'information pour identifier les objets (ou enregistrements) de données [Arasu and Garcia-Molina, 2003; Buttler et al., 2001; Crescenzi et al., 2001; Liu et al., 2003] pour les sites qui génèrent ces objets à partir d'une requête utilisateur. L'utilisation typique est le Web profond, où des pages réponse résultent de la soumission d'un formulaire Web. Par exemple, sur les sites Web de commerce électronique, la tâche est d'extraire toutes les propriétés (prix, nom, disponibilité, etc.) d'un objet produit. Ces techniques qui opèrent sur les enregistrements de données et qui utilisent des méthodes d'induction d'un extracteur à partir d'un ensemble de pages Web ne peuvent pas s'appliquer directement à l'extraction d'articles du Web, qui représentent des informations moins structurées. Aussi, en raison d'un modèle de la page qui devient de plus en plus complexe par la présence du contenu dynamique (applications, multimédia), une approche standard pour l'induction non-

supervisée d'un extracteur du contenu intéressant n'est plus suffisante en elle-même.

Pour cibler l'information d'intérêt dans des pages Web, les techniques issues de la littérature ont considéré l'extraction des blocs d'information [Song et al., 2004], des « pagelets » [Bar-Yossef and Rajagopalan, 2002; Caverlee et al., 2004], des fragments [Ramaswamy et al., 2004] ou articles [Kohlschütter et al., 2010; Pasternack and Roth, 2009]. Ces notions sont essentiellement équivalentes : elles représentent le *contenu principal* d'une page Web. Une variété de techniques ont été utilisées dans ces travaux : basées sur la densité textuelle [Kohlschütter et al., 2010; Pasternack and Roth, 2009], ou celle des balises HTML [Weninger et al., 2010], des approches visuelles [Mehta et al., 2005; Song et al., 2004], ou en utilisant des heuristiques en ce qui concerne les chemins dans le DOM [Oita and Senellart, 2010b]. Cependant, contrairement aux techniques d'induction d'un extracteur à partir d'un ensemble de pages Web, ces méthodes fonctionnent à l'échelle d'une seule page Web. Par conséquent, ces travaux ignorent une caractéristique intéressante de pages appartenant au même site : leur uniformité structurelle.

Nous présentons une nouvelle approche pour l'extraction d'articles du Web, pour le cas courant sur le Web des pages générées dynamiquement par un système de gestion de contenu. Notre algorithme, FOREST, cible la zone de la page Web pertinente pour certains *mots-clés*, automatiquement acquis, pour obtenir des motifs structurels pour l'identification du contenu d'intérêt. Ces motifs seront étudiés du point de vue de leur caractère informatif à travers plusieurs pages.

Méthodes basées sur des mots-clés

Pour repérer la zone du contenu principal d'une page Web, et pour déterminer plus précisément quels sont les éléments de l'arborescence DOM qui sont plus importants que d'autres, nous construisons automatiquement un ensemble de termes pertinents pour chaque page. Nous utilisons la notion de mot-clé comme un indice linguistique. Cependant, la forte incidence qu'un mot-clé peut avoir dans le texte d'un article Web le rend plus proche d'une entité conceptuelle. Nous considérons deux possible sources de mots-clés : des flux du Web qui sont liés aux pages Web, et les termes trouvés par une analyse *Tf-Idf* sur les pages Web mêmes.

L'acquisition de mots-clés

En utilisant les flux du Web En utilisant les flux du Web, qui sont des documents descriptifs, on obtient, sans aucune analyse globale sur le contenu textuel des pages qui sont liées à un flux donné, des indices plutôt fiables sur le contenu qui est intéressant dans une page Web.

Nous commençons par récupérer tout le contenu textuel du titre et de la description d'un élément d'un flux Web : le code HTML de la description est dépouillé, seul le texte est conservé. Après une analyse typique de traitement du langage naturel on obtient des

séquences de lexèmes ; on retient des indices linguistiques pour une page Web sous deux formes : mots-clés et n -grammes. Les n -grammes représentent des séquences de n mots, pris tels qu'ils apparaissent dans la description d'un élément du flux. Les mots-clés sont denses dans la partie de la page Web qui contient l'information d'intérêt, mais peuvent aussi se trouver dans les régions contenant des listes d'articles connexes, des commentaires ou des catégories pour l'article de la page donnée.

En utilisant une analyse *Tf-Idf* Étant donné un ensemble de pages Web qui partagent des motifs structurels, nous appliquons l'analyse classique de traitement du langage naturel sur le texte de chaque page. Ensuite, nous utilisons la mesure *Tf-Idf* qui permet d'identifier l'ensemble de mots-clés pertinents pour chaque page. Dans les expériences, nous prenons les k meilleurs termes où $k=10$ (mais nous discutons d'autres paramètres). Pour mieux se référer à l'idée d'un indice linguistique qui aide dans la recherche de la pertinence, nous utilisons simplement le terme de *signifiant*, en sachant qu'il peut être un mot-clé, mais aussi une n -gramme.

Ayant ces deux sources de signifiants, nous étudions dans la section § 6.3 leur incidence sur la performance de la technique que nous développons par la suite.

Notions préliminaires

Nous nous basons sur les flux pour construire de manière automatique un jeu de données ayant, pour chaque source (site) du Web analysée, plusieurs pages qui sont similaires de point de vue structurel (c.-à-d., des pages échantillon).

Documents XML Ayant acquis un flux Web, nous recueillons les URL des pages référencées dans les éléments de ces flux et nous construisons une version utilisable sous la forme d'un arbre DOM. Cette structure renforcée est sérialisée sous la forme d'un *document XML*. Pour cela, l'analyseur HTMLCleaner⁶ met le code HTML dans le bon ordre et filtre les scripts. Nous notons tous les documents provenant des pages échantillon d_k , $1 \leq k \leq n$ où n est typiquement 10 [Oita and Senellart, 2010b].

Nœuds significatifs Pour chaque document analysé d_k , nous extrayons tous les nœuds DOM qui représentent des nœuds terminaux (c.-à-d., nœuds textuels tout en bas de l'arbre DOM) qui sont significatifs, c.-à-d., leur texte contient au moins un signifiant.

⁶<http://w3c99htmlSpeccleaner.sourceforge.net/>

§ 4. FOREST

Introduction

Nous introduisons une technique efficace d'extraction du contenu principal d'une page Web, en prenant en compte à la fois le contenu lui-même, et la structure répétée de pages Web. La méthode proposée, appelée FOREST pour « *Extraction ciblée d'objets en exploitant les chemins significatifs* », présente les étapes suivantes :

1. nous acquérons des mots-clés pour chaque document (ces mots-clés peuvent également provenir d'autres sources comme les flux, les termes d'une requête Web introduits par un utilisateur, ou des méta-données HTML) ;
2. nous identifions, au niveau DOM de tous les documents, des motifs structurels qui sont significatifs ;
3. nous classons ces motifs grâce à une nouvelle mesure de pertinence basée sur la théorie de l'information et statistique afin d'identifier le motif structurel le plus informatif ;
4. nous déduisons une expression générique de type XPath qui donne l'emplacement du contenu intéressant, d'une façon cohérente pour tous les documents étudiés.

Nous soulignons les contributions de ce travail :

- (i) une nouvelle mesure pour calculer le caractère informatif du contenu d'une page Web ;
- (ii) une technique automatique d'induction d'un chemin générique qui mène au nœud qui contient le contenu d'intérêt à travers plusieurs pages échantillon ;
- (iii) une efficacité démontrée par des expériences en termes de précision et de rappel pour plus de 1.000 pages Web acquises auprès de 93 sites Web hétérogènes, avec des comparaisons favorables, pour différents paramètres et méthodes de référence.

Méthode

Renforcement de la structure d'une page Web D'abord, nous ajoutons une position à chaque nœud dans l'arborescence DOM : ce nouvel attribut nommé *dfs* enregistre l'ordre de navigation de type profondeur d'abord à partir de la racine. Ces positions servent à identifier les nœuds dans le cas où les valeurs des attributs pour un nœud ne donnent pas une combinaison qui identifie d'une manière unique ce nœud dans l'arborescence DOM. Cet ajout revient à un parcours en profondeur de l'arbre DOM et c'est utile typiquement quand l'*id* et l'attribut *class* manquent, pour garder une trace des positions possibles d'un certain type de nœud à travers plusieurs pages échantillon.

Expressions XPath Un *chemin significatif* est un chemin où le nœud feuille contient au moins un signifiant. Le chemin dans ce cas est donné par la séquence d'identificateurs de nœud, à partir de ce nœud jusqu'à la racine.

Nous analysons par la suite juste les éléments DOM qui composent les chemins significatifs. Comme le nœud feuille qui représente la source d'un chemin significatif contient des mots-clés, alors tous ses ancêtres sont des nœuds ayant un texte contenant des mots-clés. Plus nous sommes haut dans la hiérarchie DOM, plus les nœuds ont la tendance de contenir des mots clé ; au même temps, ils contiennent aussi plus de termes non-significatifs. Veuillez noter que le nombre de chemins significatifs que nous décomposons pour analyser les nœuds peut être faible si le nombre de mots-clés est faible, ou si ces mots-clés ont des emplacements identiques.

L'identification des éléments Un élément DOM est généralement identifié par un nom de balise et sa liste d'attributs. Cependant, tous les éléments n'ont pas des attributs qui peuvent les rendre uniques. Les paragraphes, les éléments d'une table par exemple sont rarement déterminés par une combinaison unique d'attributs. Heureusement, en raison de la phase de pré-traitement, chaque nœud possède, outre son nom de la balise, au moins une valeur de position *dfs*.

Definition 4 Le *type* d'un élément DOM est défini comme l'expression XPath construite à partir de $\langle t, atts \rangle$, où t est le nom de la balise et $atts$ est l'ensemble de paires clé-valeur des attributs, comme suit :

- (i) si l'élément DOM possède des attributs autres que *dfs* (par exemple, *id*, *class*) alors $atts$ est formé par les paires clé-valeur des ces attributs ;
- (ii) sinon, $atts$ est défini seulement par $dfs = d$, où d est la valeur *dfs* de l'élément concerné.

Des exemples de types d'éléments simples sont `//div[@id="contenant" and @class="poste"]` ou `//p[@dfs=24]`.

Le type d'éléments

Definition 5 Un *motif structural* est défini par la combinaison d'un *type* et du *niveau* sur lequel l'élément se trouve (c.-à-d., son index dans le chemin significatif).

Les modèles structurels sont utilisés pour identifier les similitudes entre les éléments DOM à travers les pages échantillon. En effet, l'expression XPath et le niveau des éléments peuvent être les mêmes pour des nœuds appartenant à de pages Web différentes, tandis que leur contenu est le plus souvent différent.

La mesure du caractère informatif

Nous introduisons maintenant la mesure de pertinence qui est utilisée pour le classement des motifs structurels.

Nous fixons un élément e_i dans un document XML d_k . Soit x le nombre de signifiants dans le texte de e_i , comptés avec leur *multiplicité*. Tous les autres termes représentent des non-signifiants, soit y leur nombre. Alors $N = x + y$ est le nombre total de termes dans le texte de e_i . Nous désignons de manière analogue le nombre de signifiants et de non-signifiants au niveau d'un document d_k où e_i se trouve, comme X , Y et N respectivement.

La densité statistique de signifiants Une des façons les plus naturelles pour déterminer si un nœud est informatif est de calculer sa densité en signifiants, c'est-à-dire $\frac{x}{N}$. Toutefois, lorsque N est petit, cette densité peut être imprécise, en raison d'un manque d'observations (un nœud formé d'un seul signifiant est susceptible de ne pas être le nœud le plus significatif de ce document). Dans de tels contextes, on peut utiliser la règle de Jeffrey [Krichevsky and Trofimov, 1981] comme un estimateur statistique qui exprime mieux la densité de signifiants, qui devient $\frac{x+1/2}{N+1}$. En outre, lors de l'échantillonnage de N éléments à partir d'un ensemble potentiellement plus grand, nous avons une marge d'erreur sur la densité sémantique. Avec f la fréquence donnée par l'estimateur ci-dessus, l'écart-type est $\sqrt{\frac{f(1-f)}{N}}$ [Freedman et al., 1998]. En supposant un écart-type pour obtenir un intervalle de confiance d'environ 70% [Freedman et al., 1998] et en le combinant avec l'estimation ci-dessus, nous obtenons l'intervalle suivant de la densité sémantique :

$$\frac{x+1/2}{N+1} \pm \frac{1}{N+1} \sqrt{\frac{(x+1/2) \times (y+1/2)}{N}} \quad (3.1)$$

Nous définissons maintenant la densité statistique J comme la valeur inférieure de cet intervalle, c'est à dire, l'estimation au pire des cas à 70% comme confiance de la densité sémantique, et si cette valeur est inférieure à 0 (parce que l'échantillon n'est pas assez grand), nous la fixons à 0 :

$$J = \max \left(0, \frac{1}{n+1} \left(x+1/2 - \sqrt{\frac{(x+1/2) \times (y+1/2)}{N}} \right) \right) \quad (3.2)$$

Cette mesure de la densité tend à favoriser les nœuds avec beaucoup de signifiants et peu de non-signifiants, donc, pour la plupart des cas, des nœuds qui apparaissent plus bas dans la hiérarchie du DOM.

L'inattendu Nous examinons aussi une autre approche pour mesurer la pertinence d'un type d'élément DOM basée sur la théorie de l'information. La notion d'*inattendu*, provient du modèle cognitif de la théorie de la simplicité [Dimulescu and Dessalles,

2009]. Cette mesure repose sur l'observation que les humains ont la tendance de trouver une situation intéressante surtout quand ils perçoivent une différence de complexité. Une situation est inattendue si elle est plus simple à décrire que de la générer. Supposons un modèle de calcul donné (par exemple, un encodage d'une machine de Turing pour une machine de Turing universelle donnée). Étant donné un objet, on considère la complexité de la génération C_w (c.-à-d., la taille du programme nécessaire pour le générer) et sa complexité de description C (c.-à-d., la complexité de Kolmogorov, la taille minimale d'un programme pour le décrire); l'inattendu de cet objet est la différence entre les deux (à noter que nous avons toujours $C \leq C_w$).

Nous appliquons ce cadre à la distribution binomiale simple et non-uniforme, qui correspond le mieux à notre contexte. Plus précisément, pour chaque nœud pertinent dans l'arborescence DOM, nous considérons son caractère inattendu par rapport au nombre de signifiants et non-signifiants contenus dans le sous-arbre défini par son emplacement dans le DOM. La complexité de génération correspond (à une constante additive près) au logarithme du nombre des moyens de tirer $x + y$ éléments hors d'un ensemble de $X + Y$ éléments : $C_w = \log(X + Y)^{x+y} = (x + y) \log(X + Y)$.

D'autre part, la complexité de description représente la complexité nécessaire pour décrire le contenu du nœud textuel : le logarithme du nombre de façons de choisir exactement x signifiants et y non-signifiants, qui est : $C = x \log X + y \log Y$. Enfin, l'imprévu est la différence entre ces deux difficultés :

$$U = (x + y) \log(X + Y) - x \log X - y \log Y \quad (3.3)$$

Plus intuitivement, nos expériences préliminaires montrent que l'inattendu favorise les éléments avec une grande quantité de contenu textuel qui est plus riche en signifiants que la répartition typique des signifiants, sur la page Web dans son ensemble. Cela s'avère être complémentaire à la densité statistique J .

La mesure du caractère informatif

$$I(sp_i, d_k) = J(sp_i, d_k) \times U(sp_i, d_k) \quad (3.4)$$

La pertinence d'un élément DOM représente la somme des produits entre l'inattendu et la densité statistique en signifiants pour cet élément dans le document d_k .

Cette mesure permet de caractériser le caractère informatif d'un motif structurel $sp_i, i \in 1 : m$, où m est le nombre total de motifs structuraux qui sont partagés par nos pages échantillons.

La combinaison de la structure et du contenu

Une mesure globale de la pertinence d'un modèle structurel combine le caractère informatif de celui-ci avec son nombre d'occurrence et un facteur de décroissance donné par

son niveau. En termes du nombre p des chemins significatifs pour lesquels sp_i se produit sur un niveau, nous avons :

$$R^{\text{FOREST}}[sp_i] = \sum_{k=0}^p I(sp_i, d_k) \times p \times \text{niveau}(sp_i) \quad (3.5)$$

Il existe un seul type d'élément sp_i à un certain niveau, donc le nombre d'occurrence de sp_i dans les pages échantillon est p , $p \leq n$, n étant le nombre total de documents. Le rôle du facteur p est clair, puisque nous voulons donner un plus grand poids aux motifs structurels qui sont pas seulement informatifs, mais aussi très fréquents. En outre, le facteur niveau est une heuristique favorisant les nœuds qui sont plus profondes dans l'arborescence DOM. La principale raison de cet ajout est que les éléments qui sont trop élevés dans la hiérarchie (par exemple, *body*) ne identifient pas l'article ciblé parce qu'ils ne sont pas discriminatoires.

L'idée d'un facteur de décroissance a été également mis en place dans d'autres œuvres [Guo et al., 2003; Lim and Ng, 2001], sous des formes différentes. Par exemple, dans la formule de classement de [Guo et al., 2003], le facteur de décroissance est une valeur comprise entre 0 à 1.

Nous classons les motifs structurels $sp_i, i \in 1 : m$ en utilisant cette mesure de pertinence. En appliquant le type d'élément générique en tant qu'une expression XPath sur l'ensemble de documents $d_k, k \in 1 : n$, nous trouvons un nœud DOM qui satisfait ces conditions structurelles et dont le contenu est très informatif.

Couverture Intuitivement, le nœud DOM qui contient l'essentiel du contenu d'un article devrait être défini comme le plus petit (le plus bas) ancêtre commun nœud dans la hiérarchie qui a une couverture maximale. Pour un motif structurel $sp_i, i \in 1 : m$, sa couverture représente la somme des poids *Tf-Idf* normalisé des signifiants qui se apparaissent dans le texte d'un nœud avec le motif structurel sp_i dans un document d_k .

$$Cov(sp_i, d_k) = \frac{\sum_{j=0}^{nbSigs(node(sp_i, d_k))} poids(signifier_j)}{totalNbOfSignifiers} \quad (3.6)$$

Nous mettons en œuvre cette méthode intuitive comme base de référence (COVERAGE), dans le but de montrer ce qui peut être réalisé en faisant usage de l'ensemble des signifiants, par rapport aux mesures plus sophistiquées utilisées par la FOREST. Pour cela, COVERAGE utilise la formule de base de la couverture défini précédemment pour sélectionner un motif structural qui est le mieux couvert en termes de signifiants :

$$R^{\text{COVERAGE}}[sp_i] = \sum_{k=0}^p Cov(sp_i, d_k) \times p \times \text{niveau}(sp_i) \quad (3.7)$$

Expériences

Nous testons FOREST en utilisant deux sources de signifiants : mots-clés extraits des pages Web mêmes en utilisant la mesure $Tf-Idf$, et des mots-clés obtenu du flux Web qui est associé à ces pages échantillon.

La construction du jeu de données Nous décrivons maintenant *RED* (jeu de données basé sur les flux RSS) qui est utilisé pour évaluer toutes les techniques abordées dans cette section. Notez que l'existence de flux Web n'est pas une condition pour FOREST, qui n'exige que quelques exemples de pages qui peuvent représenter la source de mots-clés. Nous utilisons des flux du Web non seulement comme une source alternative de mots-clés pour FOREST, mais aussi parce que les éléments qui composent le flux se réfèrent à des pages Web qui partagent le même modèle structurel. Une alternative serait d'effectuer un regroupement structurel des pages d'un site Web dans la phase de pré-traitement.

De plus, la motivation pour la construction de *RED* est aussi l'impossibilité de tester FOREST directement sur des jeux de données pour l'extraction d'article existants, parce que ou bien ils fonctionnent au niveau d'une seule page Web [Kohlschütter et al., 2010; Weninger et al., 2010], ou bien les pages d'échantillons sont utilisés dans le contexte du Web profond, où l'objet est un enregistrement de données plutôt qu'un article.

Des flux Web sont acquis de manière automatique en utilisant les résultats d'un moteur de recherche, Search4RSS⁷. Les pages Web accumulées proviennent des 90 sites Web, avec 3 sites qui exposent des modèles légèrement différents. Nous avons ainsi accumulé 93 sites et au total 1006 exemples de pages Web qui ont été annotées à la main.

Les URL de flux ont été donnés par Search4RSS en réponse à une requête par mot-clé. Pour cette raison, *RED* est très hétérogène : il comprend divers types d'articles qui existent sur le Web, pour un sujet particulier (par exemple, la poésie), tels que messages de blog, pages d'actualités, pages Web personnelles ou professionnelles, etc.

Annotation de référence La cible de l'extraction étant un article Web, nous annotons deux types possibles de résultats d'extraction : en premier lieu, ne contenant que le titre et le texte intégral ; deuxièmement, l'annotation contient en plus des méta-données de l'article.

L'annotation de référence pour notre jeu de données a été annoté manuellement, pour 2 à 20 pages Web par site. Nous présentons ici les résultats d'un problème d'extraction où le but est de récupérer le *title*, le *texte intégral* et éventuellement l'auteur, la date de publication, la légende des images, des catégories, etc. Aucune différence significative dans les résultats n'a pas été obtenue en testant un problème d'extraction où le but est l'extraction du texte intégral seulement, peut-être parce que les méta-données de l'article ne sont pas toujours présentes.

⁷<http://www.search4rss.com/>

L'annotation de référence est disponible sur la page Web de l'auteur⁸.

Méthodes de comparaison Nous comparons FOREST à quatre méthodes de base différentes.

La première est BOILERPIPE [Kohlschütter et al., 2010], déjà présentée dans la section ???. En tant que méthode de l'état-de-l'art pour l'extraction de contenu textuel d'un article d'une page Web, BOILERPIPE utilise la linguistique quantitative (analyse des caractéristiques telles comme la longueur moyenne des mots, le nombre de mots, etc) mêlé à des heuristiques sur l'arbre DOM et apprentissage semi-supervisé pour identifier les régions d'un document qui sont très segmentés. Ces régions sont filtrées pour obtenir le contenu de l'article. Contrairement à FOREST, BOILERPIPE a besoin d'être entraîné pour des données spécifiques, mais des extracteurs typiques comme *ArticleExtractor* sont mis à disposition du public⁹. Nous utilisons cet extracteur, qui est le mieux adapté pour les articles de *RED*, mais aucune différence significative n'a pas été observée pour d'autres extracteurs.

Une autre méthode de base que nous avons développé précédemment [Oita and Senellart, 2010b] est SIGFEED. Cette technique sélectionne, au niveau d'une seule page Web le plus petit, le plus bas ancêtre de type bloc (p. ex., <div>) dans la hiérarchie du DOM qui est le plus dense en mots-clés.

La méthode COVERAGE prend en compte les poids *Tf-Idf* des signifiants. Ceci est utile de vérifier si notre mesure plus élaborée ajoute de la valeur.

Enfin, l'heuristique DESCRIPTION prend simplement, comme contenu intéressant d'une page Web le titre et la description d'un élément d'un flux tel qu'il apparaît dans les méta-données de l'élément (avec un traitement sur la description afin d'éliminer l'encodage HTML possible). C'est important de tester cette hypothèse, parce que dans le cas des flux Web, il existe la possibilité que les éléments des flux contiennent le texte intégral de l'article dans leur description.

Les mesures de performance Le résultat de FOREST est une expression générique XPath qui renvoie, pour chaque page Web échantillon, un sous-arbre sous la forme d'un *document XML*. Ceci est utile pour obtenir des éventuelles ressources multimédia qui sont généralement incorporées dans un article Web, ainsi contribuant à sa vue d'objet. En dépit du fait qu'on obtient comme résultat un sous-arbre XML, nous faisons bien l'évaluation sur l'extrait textuel de ceci, en particulier, car la sortie du BOILERPIPE et DESCRIPTION est juste du texte.

Nous voulons aussi que la mesure de comparaison soit flexible, et en particulier fasse abstraction de la quantité d'espaces qui peuvent exister dans un contenu textuel et pas

⁸<http://perso.telecom-paristech.fr/oita/research.html>

⁹<http://code.google.com/p/Boilerpipe/>

dans un autre. Pour cela, on utilise des 2-grammes et on calcule les estimations classiques de *précision* et *rappel* comme :

$$\text{Précision}(G, S) = \frac{|G \cap S|}{|S|}, \quad \text{Rappel}(G, S) = \frac{|G \cap S|}{|G|}.$$

Précision et rappel sont ensuite résumés par leur moyenne harmonique, la mesure F_1 . Notez que la précision que nous calculons est exactement la mesure ROUGE-N [Lin, 2004] utilisée pour comparer les performances des techniques de résumé de texte.

Principaux résultats Nous montrons dans le tableau 2.2 la précision et le rappel moyennes, et la mesure correspondants F_1 pour les différentes méthodes testées sur l'ensemble des pages Web. Nous notons que, puisque nous avons 93 sites indépendants et des valeurs de l'ordre de 90%, l'intervalle de confiance à 95% de probabilité (1,96 écart type) [Freedman et al., 1998] est $\pm 0,06$. La figure 2.8 montre plus en détail la forme de la distribution des résultats pour chaque méthode.

Les deux variantes de FOREST, en utilisant les signifiants obtenus à partir des pages Web mêmes par une analyse *Tf-Idf*, ou à partir des éléments des flux, dépassent notablement les méthodes de base, avec une mesure F_1 de 92% et 89% respectivement. Ces résultats ont été obtenus pour la configuration la plus courante, dans lequel nous avons le nombre de signifiants fixés à 10 et pris en considération pour l'analyse globale un maximum de 10 pages pour chaque site Web (pour les autres paramètres, voir plus loin).

BOILERPIPE réalise un score relativement faible ici, malgré le fait que les pages Web correspondent plutôt bien aux genre de pages Web auquel *ArticleExtractor* a été formé. En même temps, BOILERPIPE n'a pas besoin d'exemples de pages multiples pour l'identification de la zone d'intérêt.

L'approche intuitive COVERAGE atteint un niveau légèrement plus élevé de F_1 que BOILERPIPE, ainsi que SIGFEED. D'après nos observations, SIGFEED ne parvient pas à extraire l'intégralité de l'article lorsque les blocs de la page Web sont très imbriqués.

La précision de DESCRIPTION est faible, ce qui suggère d'abord que les éléments de flux contiennent également des 2-grammes qui ne figurent pas dans le contenu principal (un exemple de ce que sont dédiés liens pour accéder à la version intégrale), et le second, à en juger par le rappel catastrophique, que les éléments d'un flux sont souvent des versions incomplètes du contenu principal d'une page Web. Nous avons également constaté que, pour des raisons pratiques (l'article peut être très long) ou à des fins commerciales (le but principal d'un élément est de donner *un aperçu* de la teneur réelle qui existe actuellement sur un site Web, afin que les visiteurs du site peuvent être attirés), la description est souvent coupée à quelques lignes [Finkelstein, 2005].

Comme déjà indiqué, DESCRIPTION fonctionne très mal, avec un score F_1 supérieur à 50% sur moins de 25% du corpus.

Pour examiner plus attentivement ces résultats, la figure 2.8 présente la répartition des scores F_1 sur le corpus, pour toutes les méthodes étudiées. Ce graphique montre

en particulier que, en plus d'avoir de meilleures performances en moyenne, les deux variantes de FOREST sont aussi plus robustes : sur 90% du corpus (resp., 75%), la mesure F_1 est supérieure à 84% (respectivement, 91%), à comparer avec 65% et 85% pour BOILERPIPE. Une autre caractéristique intéressante montre la figure 2.8 est que SIGFEED et COVERAGE ont une médiane élevée, ce qui signifie qu'ils fonctionnent bien sur la plupart des sources, mais ont une mesure F_1 moins que 55% et 9% respectivement, sur 10% du corpus.

Les résultats de FOREST(tf-idf) suggère que cette approche d'acquisition de signifiants est plus robuste que l'utilisation des flux Web dans FOREST(feed) puisque, dans certains cas, les éléments d'un flux peuvent contenir des descriptions très courts. Aussi, les signifiants de FOREST(feed) ont la tendance d'être localisés au même endroit (le début de l'article).

Influence du nombre de pages Pour comprendre l'impact du nombre de pages avec la même mise en page sur FOREST, nous traçons dans la figure 2.9 la mesure F_1 des différentes méthodes par rapport à ce paramètre.

Évidemment, étant donné que ni SIGFEED, ni BOILERPIPE font usage de la structure répétée, la variation de leur F_1 est presque nulle, mais existante (donnée par la performance relativement fluctuante de comportement sur l'ensemble des pages Web d'un site donné).

FOREST(tf-idf) nécessite au moins deux pages partageant le même modèle : cela est nécessaire non seulement pour l'acquisition de mots-clés discriminants, mais permet aussi l'exploitation de la structure répétée des pages échantillon pour l'identification des motifs structurels. Dès qu'il y a au moins deux pages Web, FOREST atteint un score F_1 qui est déjà supérieur à celui de BOILERPIPE.

FOREST(tf-idf) continue de s'améliorer quand le nombre de pages augmente, pour atteindre un plateau autour de 8 à 10 pages, donnant alors de meilleurs résultats que FOREST(feed).

Influence du nombre de mots-clés Un autre paramètre qui peut être modifié est le nombre de signifiants conservés pour une page Web donnée. De nos expériences, tant que le nombre de signifiants dépassent 5, la qualité de l'extraction n'est pas trop touchée, bien que nous observons une légère diminution de l'efficacité lorsque nous prenons en considération trop de signifiants.

Divers Nous rapportons brièvement sur les variations supplémentaires des paramètres de FOREST. Nous avons testé J et U séparément pour trouver la mesure appropriée. Pour FOREST(tf-idf), J seul atteint une précision de 90%, U seul 89%, et pour JU nous obtenons une mesure F_1 de 92%. Pour FOREST(feed), l'influence de J et U est encore plus visible au niveau de la précision et du rappel (voir figure 2.11) : J alors donne

d'assez bons résultats en matière de précision, tandis que *U* améliore considérablement le rappel.

§ 5. Autres directions de recherche étudiées

Nous avons abordé au long de cette thèse autres deux sujets : l'analyse temporelle des pages Web (Chapitre 1), l'extraction de contenu principal des pages Web à l'aide des techniques basées sur des mots-clés (Chapitre 2), et la découverte de la sémantique des objets en suivant le plan de la vision décrite dans le Chapitre 3. En dehors de la contribution principale que nous avons décrite précédemment, nous résumons très brièvement les deux autres travaux effectués dans le cadre de cette thèse :

Détection des changements des pages Web Nous avons procédé à une analyse globale des différents stratégies de l'état-de-l'art pour la détection des changements [Oita and Senellart, 2011]. Aussi, on a étudié plus en détail l'approche basée sur les flux du Web, qui concerne l'exploitation des méta-données temporelles dans le processus de détection de changement pour les pages Web associées à des flux [Oita and Senellart, 2010b]. Dans les cas où les pages Web sont liées à un flux Web, une application que nous avons présenté dans [Oita and Senellart, 2010a] est la reconstruction de pages Web en se basant sur leur contenu principal (filtrant donc toute autre information considérée redondante, par rapport à notre source de mots-clé), en enrichissant ce contenu avec d'autres méta-données extraites depuis les flux mêmes.

Annotation sémantique d'objets Nous avons utilisé FOREST pour l'extraction d'objets de données type réponse à une requête utilisateur à partir de sources Web profondes. Ici, les signifiants provenaient des mots-clés utilisés lors de la soumission du formulaire, et l'échantillon de pages partageant le même modèle étaient tout simplement les pages de réponse que nous avons obtenus en soumettant ce formulaire. En obtenant avec FOREST l'identification et l'extraction des objets de données « livre » d'Amazon, l'objectif a été d'effectuer une annotation sémantique de ces objets en utilisant une ontologie générique. Notre vision d'un cadre holistique et générique pour la compréhension du Web profond a été présenté dans [Oita et al., 2012], où chaque étape (analyse du formulaire, l'extraction des valeurs des objets, la mise en correspondance des schéma d'entrée et de sortie, l'alignement d'ontologies) bénéficie des résultats combinés obtenus pour les autres parties. Il s'agit d'un projet ambitieux qui aime inférer le schéma des objets et leur description sémantique automatiquement, mais notre prototype actuel présente déjà des résultats prometteurs.

Bibliography

- Serge Abiteboul. Issues in monitoring Web data. In *Proc. DEXA*, 2002.
- Eytar Adar, Jaime Teevan, Susan T. Dumais, and Jonathan L. Elsas. The Web changes everything: Understanding the dynamics of Web content. In *Proc. WSDM*, 2009.
- Manuel Alvarez, Alberto Pan, Juan Raposo, Fernando Bellas, and Fidel Cacheda. Extracting lists of data records from semi-structured Web pages. *Art. Data and Knowledge Engineering*, 64(2), 2008.
- Yoo Jung An, James Geller, Yi-Ta Wu, and Soon Ae Chun. Semantic deep Web: automatic attribute extraction from the deep Web data sources. In *Proc. SAC*, 2007.
- Yoo Jung An, Soon Ae Chun, Kuo-Chuan Huang, and James Geller. Enriching ontology for deep Web search. In *Proc. DEXA*, 2008.
- Arvind Arasu and Hector Garcia-Molina. Extracting structured data from Web pages. In *Proc. SIGMOD*, 2003.
- Hassan Artail and Kassem Fawaz. A fast HTML Web change detection approach based on hashing and reducing the number of similarity computations. *Art. Data and Knowledge Engineering*, 66(2), 2008.
- Nikolaus Augsten, Michael Böhlen, and Johann Gamper. Approximate matching of hierarchical data using pq-grams. In *Proc. VLDB*, 2005.
- Ricardo Baeza-Yates, Carlos Castillo, and Felipe Saint-Jean. Web dynamics, structure, and page quality. In Mark Levene and Alexandra Poulovassilis, editors, *Web Dynamics*. Springer, 2004.
- Raju Balakrishnan and Subbarao Kambhampati. SourceRank: Relevance and trust assessment for deep Web sources based on inter-source agreement. In *Proc. WWW*, 2011.
- Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proc. WWW*, 2002.
- Luciano Barbosa and Juliana Freire. Siphoning hidden-Web data through keyword-based interfaces. *Art. J. Information and Data Management*, 1(1), 2004.

BIBLIOGRAPHY

- David T. Barnard, Gwen Clarke, and Nicolas Duncan. Tree-to-tree correction for document trees. Technical Report 95-372, Queen's University, 1995.
- Elena Beisswanger. Exploiting relation extraction for ontology alignment. In *Proc. ISWC*, 2010.
- Dimitri P. Bertsekas and David A. Castañón. Parallel asynchronous Hungarian methods for the assignment problem. *Art. INFORMS J. Computing*, 5(3), 1993.
- Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the Web of data. *Art. Web Semant.*, 7(3), 2009.
- Paul Bohunsky and Wolfgang Gatterbauer. Visual structure-based Web page clustering and retrieval. In *Proc. WWW*, 2010.
- Andrei Z. Broder. On the resemblance and containment of documents. In *Proc. SEQUENCES*, 1997.
- Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. *Art. Computer Networks*, 29(8-13), 1997.
- Emmanuel Bruno, Nicolas Faessel, Hervé Glotin, Jacques Le Maitre, and Michel Scholl. Indexing and querying segmented web pages: the block Web model. *Art. World Wide Web*, 14(5-6), 2011.
- David Buttler. A short survey of document structure similarity algorithms. In *Proc. International Conference on Internet Computing*, 2004.
- David Buttler, Ling Liu, and Calton Pu. A fully automated object extraction system for the World Wide Web. In *Proc. Distributed Computing Systems*, 2001.
- Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. VIPS: a vision-based page segmentation algorithm. Technical Report MSR-TR-2003-79, Microsoft, 2003.
- Eduardo Teixeira Cardoso, Iam Vita Jabour, Eduardo Sany Laber, Rogério Rodrigues, and Pedro Cardoso. An efficient language-independent method to extract content from news Webpages. In *Proc. DocEng*, 2011.
- James Caverlee, Ling Liu, and David Buttler. Probe, cluster, and discover: focused extraction of QA-pagelets from the deep Web. In *Proc. ICDE*, 2004.
- Deepayan Chakrabarti and Rupesh R. Mehta. The paths more taken: Matching dom trees to search logs for accurate Web page clustering. In *Proc. WWW*, 2010.

BIBLIOGRAPHY

- Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled F. Shaalan. A survey of Web information extraction systems. *Art. IEEE Trans. on Knowl. and Data Eng.*, 18(10), 2006.
- S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proc. SIGMOD*, 1997.
- Sudarshan S. Chawathe, Arnand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proc. ACM*, 1996.
- Miao Chen, Xiaozhong Liu, and Jian Qin. Semantic relation extraction from socially-generated tags: a methodology for metadata generation. In *Proc. DC*, 2008.
- Junghoo Cho and Hector Garcia-Molina. The evolution of the Web and implications for an incremental crawler. In *Proc. VLDB*, 2000.
- Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *Art. ACM TOIT*, 3(3), 2003.
- Philipp Cimiano, Günter Ladwig, and Steffen Staab. Gimme' the context: Context-driven automatic semantic annotation with C-PANKOW. In *Proc. WWW*, 2005.
- Lars R. Clausen. Concerning Etags and datestamps. In *Proc. IAWW*, 2004.
- Grégory Cobéna and Talel Abdesslem. A comparative study of XML change detection algorithms. In *Service and Business Computing Solutions with XML*. IGI Global, 2009.
- Grégory Cobéna, Serge Abiteboul, and Amélie Marian. Detecting changes in XML documents. In *Proc. ICDE*, 2002.
- Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large Web sites. In *Proc. VLDB*, 2001.
- Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering Web pages based on their structure. *Art. Data and Knowledge Engineering*, 54(3), 2005.
- Hasan Davulcu, Srinivas Vadrevu, and Saravanakumar Nagarajan. OntoMiner: Bootstrapping and populating ontologies from domain specific Websites. In *Proc. WWW*, 2004.
- Davi de Castro Reis, Paulo B. Golgher, Altigran S. da Silva, and Alberto H. F. Laender. Automatic Web news extraction using tree edit distance. In *Proc. WWW*, 2004.
- Adrian Dimulescu and Jean-Louis Dessalles. Understanding narrative interest: Some evidence on the role of unexpectedness. In *Proc. CogSci*, 2009.

BIBLIOGRAPHY

- Yanhui Ding and Qingzhong Li. Building the schema of a Web entity dynamically. *Art. Journal of Computational Information Systems*, 2011.
- Fred Douglass, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *Proc. USITS*, 1997.
- Fred Douglass, Thomas Ball, Yih-Farn Chen, and Eleftherios Koutsofios. The AT&T Internet difference engine: Tracking and viewing changes on the Web. *World Wide Web*, 1(1), 1998.
- Wei Fang, Zhiming Cui, and Pengpeng Zhao. Ontology-based focused crawling of deep Web sources. In *Proc. KSEM*, 2007.
- Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. A large-scale study of the evolution of Web pages. In *Proc. WWW*, 2003.
- Ellen Finkelstein. *Syndicating Web Sites with RSS Feeds for Dummies*. Wiley Publishing, Inc., 2005.
- S. Flesca and E. Masciari. Efficient and effective Web page change detection. *Art. Data and Knowledge Engineering*, 46(2), 2007.
- David Freedman, Robert Pisani, and Roger Purves. *Statistics*. W. W. Norton, 1998.
- Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, and Christian Schallhart. Real understanding of real estate forms. In *Proc. WIMS*, 2011a.
- Tim Furche, Georg Gottlob, Xiaonan Guo, Christian Schallhart, Andrew Sellers, and Cheng Wang. How the Minotaur turned into Ariadne: ontologies in Web data extraction. In *Proc. ICWE*, 2011b.
- Tim Furche, Georg Gottlob, Xiaonan Guo, Giorgio Orsi, and Christian Schallhart. Forms form patterns: reusable form understanding. In *Proc. WWW*, 2012a.
- Tim Furche, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Automatically learning gazetteers from the deep Web. In *Proc. WWW*, 2012b.
- David Gibson, Kunal Punera, and Andrew Tomkins. The volume and evolution of Web page templates. In *Proc. WWW*, 2005.
- Stéphane Grumbach and Giansalvatore Mecca. In search of the lost schema. In *Proc. ICDT*, 1999.
- Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: ranked keyword search over XML documents. In *Proc. SIGMOD*, 2003.

BIBLIOGRAPHY

- Bin He, Kevin Chen-Chuan Chang, and Jiawei Han. Discovering complex matchings across Web query interfaces: A correlation mining approach. In *Proc. KDD*, 2004.
- Zhongtian He, Jun Hong, and David Bell. Schema matching across query interfaces on the deep Web. In *Proc. BNCOD*, 2008.
- Yih-Ling Hedley, Muhammad Younas, Anne James, and Mark Sanderson. Sampling, information extraction and summarisation of hidden Web databases. *Art. Data and Knowledge Engineering*, 59, 2006.
- Daniel M. Herzig and Thanh Tran. Heterogeneous Web data search using relevance-based on the fly data integration. In *Proc. WWW*, 2012.
- Jane Hunter and Sharmin Choudhury. Implementing preservation strategies for complex multimedia objects. In *Proc. ECDL*, 2003.
- Paul Logasa Bogen II, Joshua Johnson, Unmil P. Karadkar, Richard Furuta, and Frank Shipman. Application of Kalman filters to identify unexpected change in blogs. In *Proc. JC DL*, 2008.
- Panagiotis G. Ipeirotis and Luis Gravano. Distributed search over the hidden Web: hierarchical database sampling and selection. In *Proc. VLDB*, 2002.
- Jyoti Jacob, Anoop Sanka, Naveen Pandrangi, and Sharma Chakravarthy. WebVigiL: An approach to just-in-time information propagation in large network-centric environments. In Mark Levene and Alexandra Poulouvasilis, editors, *Web Dynamics*. Springer, 2004.
- Jyoti Jacob, Alpa Sachde, and Sharma Chakravarthy. CX-DIFF: a change detection algorithm for xml content and change visualization for WebVigiL. *Art. Data Knowl. Eng.*, 52(2), 2005.
- Adam Jatowt, Yukiko Kawai, and Katsumi Tanaka. Detecting age of page content. In *Proc. WIDM*, 2007.
- Hung-Yu Kao, Jan-Ming Ho, and Ming-Syan Chen. WISDOM: Web intrapage informative structure mining based on document object model. *Art. IEEE Transactions on Knowledge and Data Engineering*, 17(5):614–627, 2005.
- H. P. Khandagale and P. P. Halkarnikar. A novel approach for Web page change detection system. *Art. Intl. J. Comput. Theory Eng.*, 2(3), 2010.
- Ritu Khare, Yuan An, and Il-Yeol Song. Understanding deep Web search interfaces: a survey. *Proc. SIGMOD*, 39, 2010.

BIBLIOGRAPHY

- Imad Khoury, Rami M. El-Mawas, Oussama El-Rawas, Elias F. Mounayar, and Hassan Artail. An efficient Web page change detection system based on an optimized Hungarian algorithm. *Art. IEEE TKDE*, 19(5), 2007.
- Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proc. WSDM*, 2010.
- Raphail E. Krichevsky and Victor K. Trofimov. The performance of universal encoding. *Art. IEEE Transactions on Information Theory*, 27(2):199–206, 1981.
- Suneet Kumar, Anuj Kumar Yadav, Rakesh Bharti, and Rani Choudhary. Accurate and efficient crawling the deep Web: Surfacing hidden value. *Art. International J. Computer Science and Information Security*, 9(5), 2011.
- Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proc. IJCAI*, 1997.
- Kyong-Ho Lee, Yoon-Chul Choy, and Sung-Bae Cho. An efficient algorithm to compute differences between structured documents. *IEEE Trans. on Knowl. and Data Eng.*, 16(8), 2004.
- Hao Liang, Fei Ren, Wanli Zuo, and Fengling He. Ontology based automatic attributes extracting and queries translating for deep Web. *Art. J. Software*, 5, 2008.
- Seung-Jin Lim and Yiu-Kai Ng. An automated change-detection algorithm for HTML documents based on semantic hierarchies. In *Proc. ICDE*, 2001.
- Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching Web tables using entities, types and relationships. *Proc. VLDB*, 3(1), 2010.
- Chin-Yew Lin. ROUGE: a package for automatic evaluation of summaries. In *Proc. Workshop on Text Summarization Branches Out (WAS)*, 2004.
- Bing Liu and Yanhong Zhai. NET - a system for extracting Web data from flat and nested data records. In *Proc. WISE*, 2005.
- Bing Liu, Robert Grossman, and Yanhong Zhai. Mining data records in Web pages. In *Proc. KDD*, 2003.
- Ling Liu, Calton Pu, and Wei Tang. WebCQ: Detecting and delivering information changes on the Web. In *Proc. CIKM*, 2000.
- Jayant Madhavan, Alon Halevy, Shirley Cohen, Xin (Luna) Dong, Shawn R. Jeffery, David Ko, Cong Yu, and Google Inc. Structured data meets the Web: A few observations. *Art. Bull. IEEE Computer Society Technical Committee on Data Engineering*, 2006.

BIBLIOGRAPHY

- Georgiana Marşic. *Temporal Processing of News: Annotation of Temporal Expressions, Verbal Events and Temporal Relations*. PhD thesis, Wolverhampton, UK, 2011. URL <http://clg.wlv.ac.uk/papers/marsic-thesis.pdf>.
- Julien Masanès, editor. *Web Archiving*. Springer-Verlag, 2006.
- Luke K. McDowell and Michael Cafarella. Ontology-driven information extraction with OntoSyphon. In *Proc. ISWC*, 2006.
- Rupesh R. Mehta, Pabitra Mitra, and Harish Karnick. Extracting semantic structure of Web documents using content and visual information. In *Proc. WWW*, 2005.
- Genxin Miao, Junichi Tatemura, Wang-Pin Hsiung, Arsany Sawires, and Louise E. Moser. Extracting data records from the Web using tag path clustering. In *Proc. WWW*, 2009.
- Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1), 2001.
- Svetlozar Nestorov, Serge Abiteboul, and Rajeev Motwani. Extracting schema from semistructured data. In *Proc. SIGMOD*, 1998.
- Netcraft. January 2011 Web survey, 2011.
- Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. What’s new on the Web? the evolution of the Web from a search engine perspective. In *Proc. WWW*, 2004.
- Sérgio Nunes, Cristina Ribeiro, and Gabriel David. Using neighbors to date Web documents. In *Proc. WIDM*, 2007.
- Marilena Oita and Pierre Senellart. Archivage du contenu éphémère du Web à l’aide des flux Web. In *Proc. BDA*, Toulouse, France, 2010a. Conference without formal proceedings. (Demonstration).
- Marilena Oita and Pierre Senellart. Archiving data objects using Web feeds. In *Proc. IAWW*, 2010b.
- Marilena Oita and Pierre Senellart. Deriving dynamics of Web pages: A survey. In *Proc. TAWW*, 2011.
- Marilena Oita and Pierre Senellart. FOREST: Focused object retrieval by exploiting significant tag paths. In *Proc. in submission to WSDM*, 2012.
- Marilena Oita, Antoine Amarilli, and Pierre Senellart. Cross-fertilizing deep Web analysis and ontology enrichment. In *Proc. VLDS*, 2012.

BIBLIOGRAPHY

- Jeff Pasternack and Dan Roth. Extracting article text from the Web with maximum subsequence segmentation. In *Proc. WWW*, 2009.
- Zeynep Pehlivan, Myriam Ben Saad, and Stéphane Gançarski. A novel Web archiving approach based on visual pages analysis. In *Proc. IAWW*, 2009.
- Maureen Pennock and Richard Davis. ArchivePress: A really simple solution to archiving blog content. In *Proc. iPRES*, 2009.
- A. Quattoni, S. Wang, L.-P. Morency, M. Collins, and T. Darrell. Hidden-state Conditional Random Fields. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.
- Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden Web. In *Proc. VLDB*, 2001.
- Lakshmesh Ramaswamy, Arun Iyengar, Ling Liu, and Fred Douglis. Automatic detection of fragments in dynamically generated Web pages. In *Proc. WWW*, 2004.
- W3C Recommendation. Resource Description Framework (RDF): Concepts and abstract syntax. <http://www.w3.org/TR/2004/REC-w3c12rdf-concepts-20040210/>, 2012.
- Chantal Reynaud and Brigitte Safar. Exploiting WordNet as background knowledge. In *Proc. ISWC Ontology Matching (OM-07) Workshop*, 2007.
- Daniel Rocco, David Buttler, and Ling Liu. Page Digest for large-scale Web services. In *Proc. CEC*, 2003.
- S. Cenk Sahinalp and Andrey Utis. Hardness of string similarity search and other indexing problems. In *Proc. ICALP*, 2004.
- Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-Web sources with domain knowledge. In *Proc. WIDM*, 2008.
- Ka Cheung Sia, Junghoo Cho, and Hyun-Kyu Cho. Efficient monitoring algorithm for fast news alerts. *Art. Knowledge and Data Engineering*, 19, 2007.
- Kristinn Sigurðsson. Incremental crawling with Heritrix. In *Proc. IAWW*, 2005.
- sitemaps.org. Sitemaps XML format. <http://www.sitemaps.org/protocol.php>, 2008.
- Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma. Learning block importance models for Web pages. In *Proc. WWW*, 2004.

BIBLIOGRAPHY

- Marc Spaniol, Dimitar Denev, Arturas Mazeika, and Gerhard Weikum. Catch me if you can. Temporal coherence of Web archives. In *Proc. IAWW*, 2008.
- Giorgos Stoilos, Giorgos B. Stamou, and Stefanos D. Kollias. A string metric for ontology alignment. In *Proc. ISWC*, 2005.
- Stephan Strodl, Christoph Becker, Robert Neumayer, and Andreas Rauber. How to choose a digital preservation strategy: evaluating a preservation planning procedure. In *Proc. JCDL*, 2007.
- Weifeng Su, Jiyang Wang, and Frederick H. Lochovsky. ODE: Ontology-assisted data extraction. *ACM Trans. Database Syst.*, 34(2), 2009.
- Fabian M. Suchanek, Georgiana Ifrim, and Gerhard Weikum. Combining linguistic and statistical analysis to extract relations from Web documents. In *Proc. KDD*, 2006.
- Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A core of semantic knowledge unifying WordNet and Wikipedia". In *Proc. WWW*, 2007.
- Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. PARIS: Probabilistic alignment of relations, instances, and schema. *Proc. VLDB Endow.*, 5(3), 2011.
- Chong Sun, Chee-Yong Chan, and Amit K. Goenka. Multiway SLCA-based keyword search in XML data. In *Proc. WWW*, 2007.
- Daniel S. Swaney, Frank Mccown, and Michael L. Nelson. Dynamic Web file format transformations with Grace. In *Proc. IAWW*, 2005.
- Mouhamadou Thiam, Nathalie Pernelle, and Nacéra Bennacer. Contextual and metadata-based approach for the semantic annotation of heterogeneous documents. In *Proc. SeMMA*, 2008.
- Mouhamadou Thiam, Nacéra Bennacer, Nathalie Pernelle, and Moussa Lo. Incremental ontology-based extraction and alignment in semi-structured documents. In *Proc. DEXA*, 2009.
- Nie Tiezheng, Yu Ge, Shen Derong, Kou Yue, and Liu Wei. Extracting result schema based on query instances in the deep Web. *Art. Wuhan University J. Natural Sciences*, 12(5), 2007.
- Karane Vieira, Altigran S. da Silva, and Nick Pinto. A fast and robust method for Web page template detection and removal. In *Proc. CIKM*, 2006.
- W3C. HTML 4.01 specification. <http://www.w3.org/TR/REC-w3c99htmlSpec40/>, 1999.

BIBLIOGRAPHY

- Jiying Wang and Fred H. Lochovsky. Data extraction and label assignment for Web databases. In *Proc. WWW*, 2003.
- Jiying Wang, Ji-Rong Wen, Fred Lochovsky, and Wei-Ying Ma. Instance-based schema matching for Web databases by domain-specific query probing. In *Proc. VLDB*, 2004.
- Richard C. Wang and William W. Cohen. Language-independent set expansion of named entities using the Web. In *Proc. ICDM*, 2007.
- Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-Diff: An effective change detection algorithm for XML documents. In *Proc. ICDE*, 2003.
- Tim Weninger, William H. Hsu, and Jiawei Han. Content extraction via tag ratios. In *Proc. WWW*, 2010.
- Wensheng Wu, AnHai Doan, and Clement Yu. Merging interface schemas on the deep Web via clustering aggregation. In *Proc. Data Mining*, 2005a.
- Wensheng Wu, Anhai Doan, Clement Yu, and Weiyi Meng. Bootstrapping domain ontology for semantic Web services from source Web sites. In *Proc. VLDB Workshop on Technologies for E-Services*, 2005b.
- Divakar Yadav, A.K. Sharma, and J.P. Gupta. Change detection in Web pages. In *Proc. ICIT*, 2007.
- Divakar Yadav, A. K. Sharma, and J. P. Gupta. Parallel crawler architecture and Web page change detection. *Art. WSEAS Transactions on Computers*, 7(7), 2008.
- Shipeng Yu, Deng Cai, Ji-Rong Wen, and Wei-Ying Ma. Improving pseudo-relevance feedback in Web information retrieval using Web page segmentation. In *Proc. WWW*, 2003.
- XiaoJie Yuan, HuiBin Zhang, Zong-Yun Yang, and YanLong Wen. Understanding the search interfaces of the deep Web based on domain model. In *Proc. ICIS*, 2009.
- Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proc. WWW*, 2005.
- Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Understanding Web query interfaces: best-effort parsing with hidden syntax. In *Proc. SIGMOD*, 2004.
- Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. Simultaneous record detection and attribute labeling in Web data extraction. In *Proc. KDD*, 2006.