MASTER IN COMPUTER SCIENCE

Automatically extracting news articles from the Internet

Jasselette, Arnaud; Vanderwhale, Mathieu

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 2004-2005

Automatically Extracting News Articles from the Internet

Arnaud Jasselette – Mathieu Vanderwhale

Mémoire présenté en vue de l'obtention du grade de Maître en informatique

Abstract

As information of interest is scattered around the World Wide Web, the need for fully automatic extraction processes to fetch relevant data cannot be ignored. Nowadays, five billion pages are available on the Internet and almost two million new pages are being added daily. This thesis aims at defining a comprehensive issue to extract news articles specially, from the early classification of significant pages to the article retrieval properly speaking. We developed News Ripper, a "wrapper" that achieves this Web mining task by clustering similar news pages before comparing their layouts to bring the articles to light.

L'information utile étant éparpillée à travers le Web, le besoin de processus automatiques d'extraction de données pertinentes se fait manifestement ressentir. Actuellement, cinq milliards de pages sont disponibles sur Internet et pratiquement deux millions de pages sont ajoutées chaque jour. Ce mémoire vise à définir en particulier une solution complète pour extraire des articles de presse en ligne, depuis la classification de pages concernant un sujet jusqu'à l'extraction proprement dite des articles. Nous avons développé News Ripper, un "wrapper" qui accomplit cette tâche de Web mining en regroupant les pages similaires avant de comparer leurs contenus pour en extraire les articles.

Foreword

This thesis is based on our internship at the University of Technology in Sydney, Australia.

We wish to thank our promoter, Mrs Noirhomme-Fraiture, and our resident professor abroad, Mr Simeon Simoff, for their help. We also want to especially thank Debbie Zhang, who patiently supervised us during our internship.

Contents

Introduction

This thesis aims at designing a fully automatic process for extracting news articles from the World Wide Web. Such an automatic task is not straightforward because it is only of great interest on a large scale, as the World Wide Web counts to date more than five billion pages. Furthermore, this process must be completely independent of both the page structure and the news topic to be as worthwhile as possible.

In order to take the full extraction process into account, we were led to consider three successive core steps. First and foremost, the *Web classification* pre-classifies both sites and pages of interest. Afterwards, the *conversion of the HTML source code into a given data structure* allows to handle HTML efficiently. Eventually, the *Data extraction* strictly speaking targets at fetching relevant data within Web pages.

Chapter 1: Intuitive approach explains how our work during the internship inspired the contents of this dissertation. The latter is split in two parts: *Part I – State of the Art* discusses the literature about the three steps mentioned above while *Part II – News Ripper* focusses on the implementation of the application we developed.

In Part I, the Web classification is analysed in both *chapter 2: Web sites classification* and *chapter 3: Web pages classification.* Since we are going to work on tree data structures, *chapter 4: Tree structures* goes into the field in greater depth. *Chapter 5: Wrappers, chapter 6: Induction tools, chapter 7: RoadRunner, chapter 8 : NLP-based tools* illustrate tools called "wrappers" which are intended for retrieving *data* of interest from the Internet. They actually cover both second and third steps, since they often create their own data structures before extracting data. As we wanted to concentrate specially on *news articles* retrieval, *chapter 9: News extraction using tree edit distance*, the main inspiration for our application, closes the State of the Art.

News Ripper, as for it, embraces only the two last steps as wrappers do. The *conversion of the HTML source code into a given data structure* concentrates on the particular tree structure we designed: the "layout tree", which is an abstract representation of a Web page layout. The *Data extraction* steered us to the *News Extraction* specifically.
However, we realized the requirement of an extra step between the building of layout trees and the extraction of news articles: this intermediate step is the *clustering of similar pages*. Notice that the clustering is completely different from the classification: the former aims at grouping pages having a common general layout together while the Web classification is meant for fetching pages of interest on the Internet.

Chapter 10 : Context outlines our work experience at the University of Technology in Sydney and the major functions of News Ripper. *Chapter 11 : Smartly building layout trees* explains why we needed to define a suitable data structure, the layout tree, to achieve the comparison of Web pages and the retrieval of news articles. The clustering of similar pages is developed in *chapter 12: News pages clustering*. The extraction of news articles properly speaking is treated in *chapter 13 : Extracting the news articles* at last.

Chapter 1

Intuitive approach

Contents

1.1 Introduction

This chapter aims at introducing intuitively the topic of our dissertation. We shall first present the context and the applications of the news extraction process from the Internet. This task belongs to a wider application domain: the extraction of pertinent data from Web pages.

We shall help the reader to understand why the news extraction process is no sinecure by presenting two trivial examples (section 1.4) that will help the reader realize the objectives we target.

The solution we implemented during our internship is explained in section "A smart issue". This solution comprises three successive steps. These steps will explain the chain of ideas of our work. We shall finally notice that other previous steps are necessary to consider the fully automatic news extraction process as comprehensive. We shall describe the contents of the *State of the Art* by matching its different chapters with the successive steps of the smart issue (section 1.5).

1.2 Context

"The Web poses itself as the largest data repository ever available in the history of human kind. Major efforts have been made in order to provide efficient access to relevant information within this huge repository of data." [1]

This information is generally stored on HTML files. HTML defines the layout of a Web page, by structuring the data in tables, divisions, columns, titles, buttons, etc. This layout is crucial for the display of the information on a screen but is useless when the aim is to fetch the relevant information for a database or a program. In case of a news Web site, there are often an advertisement banner at the top, a menu on the left side, some links at the bottom, various pictures, so that the news article represents just a small part of the source code.

Therefore, the issue is a program that manages to throw away this so called "useless code" defining the layout of the Web page, in order to keep the information of interest.

Our work is to provide these fetched articles, notwithstanding the further use. There are various techniques designed to extract the information of interest but the major challenge is to provide this information quickly. As always, the "easy-to-program" techniques are effective but inefficient. That is the reason why smart and efficient techniques are necessary.

1.3 Applications

News extraction can be useful in many ways. The news articles can be extracted for the purpose of either a user or a program.

To help out users, an obvious solution would be the display of the news article on a PDA screen, where there is not enough space for pictures, menus, banners, etc. On such a device, the layout can be unfair, so that the aim is to be able to resize (zoom) the text in order to make it easily readable. Apart from the display PDA's have a relatively slow Internet access. The efficiency would be better because the PDA's would not have to download all the heavy data such as images or backgrounds and would not have to execute the javascript code. This approach could be extended to mobile phones (WAP, 3rd generation, UMTS). Showing fertile imagination, it could be useful for people working daily with news articles who only need the unformatted text. They want to display several articles of a topic on the same screen so they can compare them easily.

Information can also be used further by mining agents. For instance, a chatterbot[1] that would be able to discuss any news topic. Let us suppose a user wants to talk about the war in Iraq, the chatterbot has got a database with links to Web sites of some famous newspapers, it extracts all the Web pages relating to this topic and then it fetches the articles inside these pages. Once it has got the articles, it can be able to discuss "war in Iraq" by analysing them with the usual chatterbot techniques.

Another use of news extraction is to create a database of structured or semi-structured data where text mining agents can effectively fetch the data available on the Web. In the context of news mining, the bots fetch the news articles, which then are transformed into a structure form and both the structured and unstructured data are stored on the mining base for accessing by the mining agents [1] (figure 1.1).

The news extraction process can actually be applied to other on-line sources, such as virtual communities, company sites and government sites. In the same way, text mining agents can become other specified data mining agents that could find useful information in the mining base (figure 1.2).

[1]a chatterbot is a program designed to converse with a user.

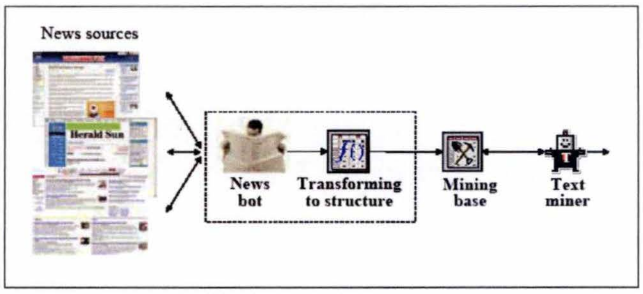


Figure 1.1: *The news mining portion of the system*

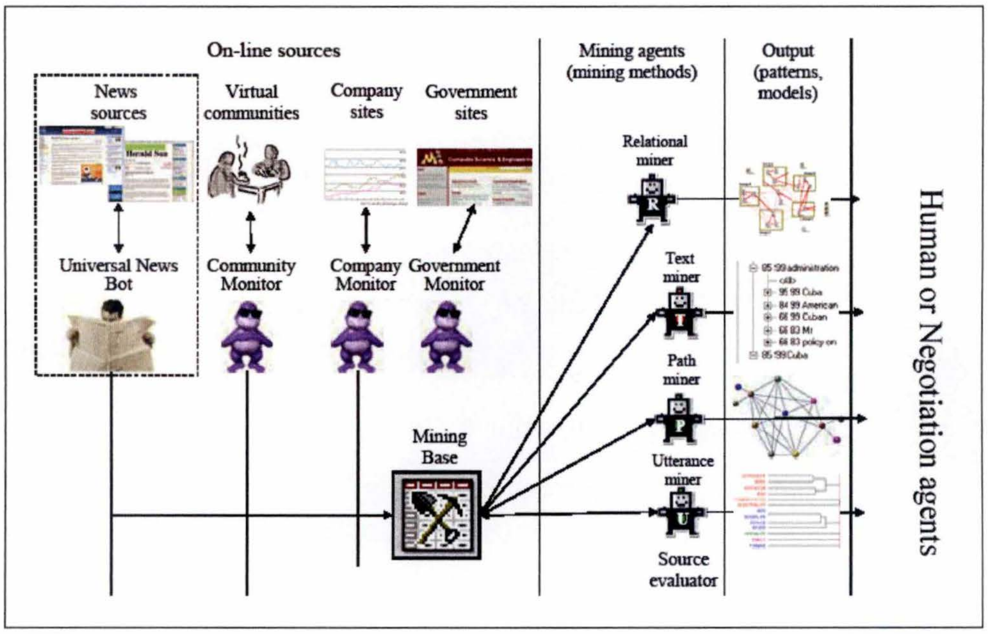


Figure 1.2: *A smart data mining system*

For instance, an obvious text mining agent is a news search engine, that would retrieve only the relevant articles linked to a given keyword. Most of the current search engines treat the information, regardless the HTML structure. When a keyword is found, the Web page is displayed, without taking into account where it has been found. The keyword could appear in a link, a meta tag, the title of the news article, or in the article itself, hence nothing can prove the results are really relevant.

An obvious issue would be therefore the creation of a search engine that would not use the META tags to find the relating pages but that would try to match the keywords with the relevant text only. The relating Web pages found would be more pertinent since the searched keywords are ensured to be at least present in the article, ignoring words that would have been only in a link, an advertisement, a menu, a pop-up, etc. The advantage of this technique is obviously counterbalanced by its slowness.

1.4 Trivial algorithms to extract news articles

Here we present two trivial solutions [1] to extract an article from a news Web site. The goal is to extract the plain text of the article, i.e. without its layout. Every object in the page that does not matter with the news article has to be deleted.

These solutions are imperfect. Anyway, they are an understandable manner to explain the main issues.

1.4.1 Example one: The header and tail filter

This is a trivial algorithm we found in the literature [1].

- Two similar pages are extracted from the same Web site, under the same category. The news sources on the Internet can be found on the Web sites of major newspapers. Another way is to use the news portal provided by search engines such as Google or Yahoo.
- All the HTML tags are removed, so that only the unformatted text is kept.
- Both texts are transformed into arrays. Each cell is a line of text.
- A structure *Header* will contain all the text that is common at the beginning of both files.
- A structure *Tail* will contain all the text that is common at the end of both files.
- The algorithm starts and appends the common cells to Header.
- Once the cells are different, the algorithm goes to the end of both files and starts to append the common cells at the end to Tail.
- At the end, Header and Tail contain the text to be removed by the filter.
- Once the filter has been created, it can be used to delete useless text in similar pages from the one Web site and then to extract the articles (see figure 1.3).

```
Input: two text files from the same web site, each contains a news article
Output: a data structure contains:
        String URL
        String Header
        String Tail
  1.    Remove all the html tags in the files.
  2.    Break down the files into one dimensional arrays (a and b), each cell
        contains a line of text.
  3.    For each cell of the array from beginning
            1.  if a[i] == b[i], append a[i] at the end of Header string
            2.  if a[i] != b[i], break;
  4.    For each cell of the array from the end
            1.  if a[i] == b[i], insert a[i] at the beginning of Tail string
            2.  if a[i] != b[i], break
  5.    Set the URL value to the common part of the URLs of two text file
Return the data structure that contains URL, Header and Tail.
```

Figure 1.3: *The pseudo-code of Header and Tail filter*

However, the common structure between two similar pages can not be restricted to three parts (a header, the text, a tail). The common structure is much more complicated. We have to take the HTML structure into account in order to identify the page format before locating the article.

1.4.2 Example two: Table break-down

This algorithm (see figure 1.4) improves the "header and tail filter" as it deals with the different `<table>` elements. [1]

- An HTML page is broken down into content blocks. The news article is expected to be the content block which is displayed on the *centre* of the page. Therefore, it is reasonable to assume that the biggest block of text is the news article.
- Most Web sites employ HTML tables to divide the page into blocks by using `<table>` tags. The idea is to copy text blocks found between `<table>` and `</table>` tags into an array.
- Once the array is complete, we count the number of words in each cell. The one that contains the maximum number of words is the largest text block, so the

```
Input: HTML file
Output: The largest body of text contained in a table
Begin
  1. Break down the HTML file into a one dimensional array, where
     each cell contains a line of text or an HTML tag
  2. Remove the HTML tags except <table> and </table>
  3. Set table_counter to 0
  4. For each cell in the array:
        a. if <table> tag is encountered, increase table_counter by 1
        b. if <\table> tag is encountered, decrease table_counter by 1
        c. if it is a text element, append it to the end of con-
           tainer[table_counter]
  5. Return container[i] that contains the largest body of text by count-
     ing the number of words.
End
```

Figure 1.4: *The pseudo-code of the break-down algorithm*

article.

Notice that the algorithm handles the tag hierarchy between `<table>` tags. As the array index is increased by 1 when a `<table>` tag is encountered and decreased by 1 when a `</table>` tag is encountered, all the text portions that belong to the same depth in the tags hierarchy are concatenated. That way, we do not understand how the article can be identified since there is maybe more than one text block in each cell of the array. If all the text blocks from the same level of hierarchy are concatenated, a lot of "noisy" text will be added. Finally, the concatenation of all the noisy text blocks from the same level of hierarchy can lead to a cell that contains more words than the article does.

If the array index was increased by 1 each time a `<table>` tag is encountered without being ever decreased, each cell would contain a separate text block. However, in this case, nothing guarantees that the entire article is always stored on a single element `<table>`.

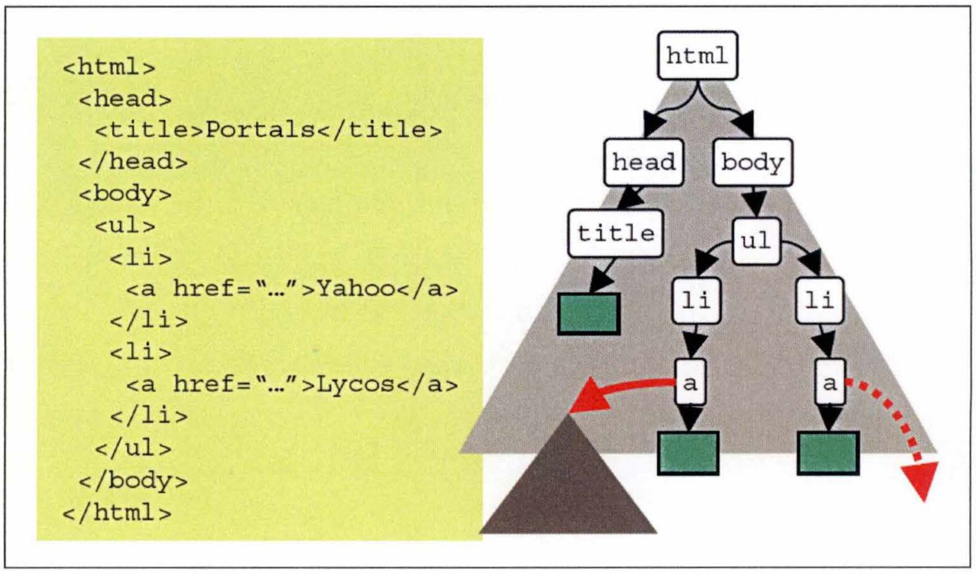
The extraction accuracy can be improved by algorithms that do not rely only on `<table>` tags information but on the whole HTML structure.

1.5 The smart issue

The pages on most news Web sites are automatically generated and filled-up with articles coming from a database. Hence we can use the common format from a set of similar pages to detect the parts that have nothing in common with each other. These distinct parts are likely to be the news articles. Yet, before being capable of comparing similar pages, we need to group them together. For this reason we need to identify why and how pages are similar. We shall use a tree structure that reflects the hierarchy of HTML tags in order to compare their likenesses.

1.5.1 From HTML to a tree structure

The structure of a Web page can be nicely described by a tree (DOM tree)[2]. Tree structures can be useful for both clustering similar pages and improving the efficiency of the algorithms that extract articles. Figure 1.5 shows the transformation from an HTML source to a DOM tree.



```
<html>
 <head>
  <title>Portals</title>
 </head>
 <body>
  <ul>
   <li>
    <a href="...">Yahoo</a>
   </li>
   <li>
    <a href="...">Lycos</a>
   </li>
  </ul>
 </body>
</html>
```

Figure 1.5: *An HTML source and its DOM tree*

The HTML file is analysed by an HTML parser. Building the tree is rather easy because the parsing of an HTML file corresponds to a preorder traversal in a tree. Let us analyse the example in figure 1.5.

A root node is created with the <HTML> tag, the only root of the tree. A child node is created each time an enclosed element is encountered and so on recursively. In this example, <HTML> is the root node. <HEAD>...</HEAD> is an enclosed element in the element <HTML>...</HTML>, so that <HEAD> becomes a child of the <HTML> node. The process goes on and adds the child <TITLE> to the <HEAD> node, then adds the contents of the element <TITLE>...</TITLE> (Portals) to the <TITLE> node. As there is no further enclosed element, the algorithm goes up until it finds a second enclosed element: <BODY> in the example, and so on.

There is a relation between a tree structure and the layout of the displayed Web page: the deeper in the tree, the more specific in the layout. The vertices close to the root represent the main separations in the page, for example the division between tables, menus, and advertisements. In the same way, the vertices close to the leaves represent the formatting of a specific object, for example text.

However, all the HTML tags are not involved in the layout, so that we can remove tags which are "useless" considering our goals. The tree structure we shall use is then a lightened DOM tree that reflects the layout of the page, hence we called it a "layout tree".

1.5.2 Clustering similar pages

We can therefore apply an algorithm that goes from the root to the leaves and compares the identical nodes between two layout trees. As soon as two nodes are different, the subtrees below these nodes are given up. The output is thus the largest common subtree between two layout trees from the top to the bottom. The larger is the common tree, the more similar are the pages layout.

1.5.3 Extracting the news articles

The text containing the article of interest is always located in the leaves of the layout tree. Notice that it is often split between a couple of leaves, according to the complexity of its format. An algorithm is used to compare dissimilarities in the leaves between a set of layout trees. If we assume that pages are similar to each other and that the article is different in each page, we are likely to extract the news articles if we extract the dissimilar leaves. That is the reason why we have to find similar pages before executing the algorithm.

1.6 Contents of Part I – State of the Art

As we went along with the implementation of our tool, we analysed the extraction process from the end to the beginning. We first tried to directly extract the news articles. Then we realized that we needed to cluster similar pages beforehand by comparing layout trees. After, we became conscious of the usefulness of an appropriate HTML parsing, in order to improve the quality of both clustering and news extraction algorithms.

1.6.1 Need for a preliminary Web classification

Despite the clustering step, the fully automatic process is still not complete; it could begin a few steps sooner. Indeed, we assumed that pages have been previously downloaded from well-chosen Web sites and yet we should automate this earlier step as well.

For this reason, we decided to introduce the state of the art with two chapters about the Web classification, a well-developed topic that is too large to be covered by our program. The Web classification is presented as two separate fields: the Web sites classification (chapter 2) and the Web pages classification (chapter 3). The former is used to find a set W of Web sites which contain one or many specific keywords. The latter is used to fetch the pages of interest inside the Web sites $\in W$.

As said in the introduction, it is quite important to distinct the Web classification from the page clustering. The former aims at finding the relevant Web sites and pages while the latter aims at clustering pages that have a similar layout.

1.6.2 No panacea for news extraction

We decided to handle layout trees in order to cluster similar pages. We shall see that other tree structures can be used for achieving this task or other purposes (chapter 4).

The data extraction on the Internet can be done by many ways. We had to present several approaches to extract data from the Internet. Tools that extract data from Web pages are called "wrappers" in the literature. We gathered them up in generic classes thanks to their major features and we shall explain the main functioning of each class (chapter 5). Three classes of wrappers are likely to be useful for news extraction: Induction tools (chapter 6), RoadRunner (chapter 7) and Natural Language Processing-based tools (chapter 8).

In our case, we were inspired by a recent proceeding from the 13th International Conference on World Wide Web (chapter 9) about the news extraction.

Part I

State of the Art

Chapter 2

Web sites classification

Contents

2.1 Introduction

In this chapter we shall present the classification of Web sites of interest it would be useful to do before a Web pages classification. From this perspective, a context briefly outlines the need for a beforehand Web sites classification. Then major classification techniques such as Web directories, classification through metatags and classification by topics are analysed in order to understand the main issues related to this Web sites classification.

2.2 Context

The amount of information available on the World Wide Web is huge and growing each year. For the present, Google searches through more than 5 billion pages with 1.5 million pages being added daily. One reason for this development is the relatively low cost of publishing a Web site: a Web site is quite cheaper than brochures or advertisements in newspapers, it is more interactive for Web users, more easy to update and it reaches millions of people.

Such a growth raises a major problem : the quality of a search query becomes more and more poor unless an effective process of classification is beforehand realized. The need to provide automated assistance to Web users for Web pages classification and categorization is thus well real. This huge growing amount of pages is however not the only reason for automated classification. Two other main reasons are taken into account. The first one is the ever-changing nature of resources available on the Web. It is simply not feasible to keep up with the fast pace of growth and change on the Web through a manual classification effort without expending immense time and effort. The second reason is that classification itself is a subjective activity. Different classification schemes are needed for different applications. No single classification scheme is suitable for all applications. Therefore different types of classification schemes, representing different facets of knowledge, may need to be applied in an ongoing fashion as new applications demand them.

About classification, a significant distinction has to be made. While the Web contains more than 5 billion pages, the total amount of Web sites is clearly less considerable. It is consequently useful to bring out two points : the classification of Web sites and the classification of Web pages. These two domains are not distinct from each other but, on the contrary, complement each other.

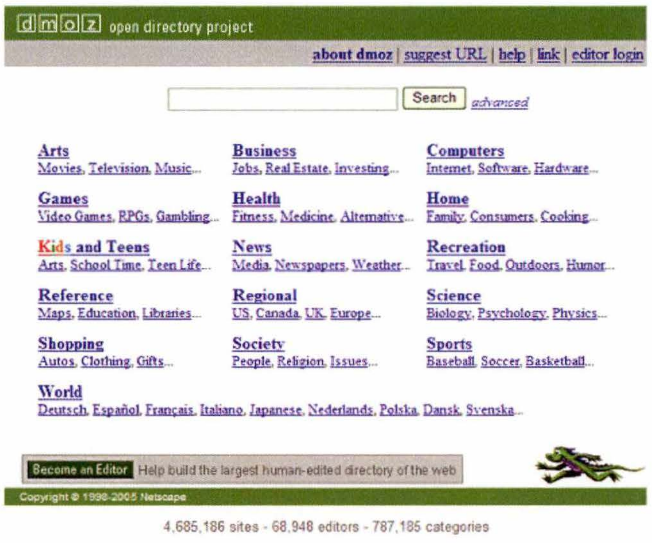
The Web sites classification considered in this chapter is rather not simple because of the ever-changing nature of Web sites and the eclecticism of topics within Web sites. Though such a categorization can reduce the search space dramatically. The problem of spotting complete Web sites of special interest to a user is not handled adequately yet, in spite of its importance for various applications. Various methods have been applied such as the directory services like Yahoo, the metadata records and the topics classification and drew our attention in this thesis.

2.3 Web directories

Categorization is an important ingredient in the classification process of Web sites. Web directories such as Yahoo![1], Looksmart[2] and the Open Directory Project[3](Figure 2.1) have know an evident popularity these last years.

These directory services can offer useful information but the entries there are often incomplete and out of date due to manual maintenance. Furthermore, these resources have been created by large teams of human editors and represent only one type of classification scheme that, while widely useful, can never be suitable to all applications. Categorization is an intellectual task and it is really not easy to create a system that spots special kinds of Web sites and offers the opportunity to search them while it would turn out to be very useful.

[1]http://www.yahoo.com
[2]http://www.looksmart.com
[3]http://www.dmoz.org


dmoz open directory project

about dmoz | suggest URL | help | link | editor login

Search advanced

Arts
Movies, Television, Music...

Business
Jobs, Real Estate, Investing...

Computers
Internet, Software, Hardware...

Games
Video Games, RPGs, Gambling...

Health
Fitness, Medicine, Alternative...

Home
Family, Consumers, Cooking...

Kids and Teens
Arts, School Time, Teen Life...

News
Media, Newspapers, Weather...

Recreation
Travel, Food, Outdoors, Humor...

Reference
Maps, Education, Libraries...

Regional
US, Canada, UK, Europe...

Science
Biology, Psychology, Physics...

Shopping
Autos, Clothing, Gifts...

Society
People, Religion, Issues...

Sports
Baseball, Soccer, Basketball...

World
Deutsch, Español, Français, Italiano, Japanese, Nederlands, Polska, Dansk, Svenska...

Become an Editor Help build the largest human-edited directory of the web

Copyright © 1998-2005 Netscape

4,685,186 sites - 68,948 editors - 787,185 categories

Figure 2.1: *The Open Directory Project home page*

2.4 Classification through metatags

John M. Pierre [3] from the Linköping University in Sweden gives us an interesting automated classification of Web sites based on the metatags. He analyses the nature of Web contents and metadata in relation to requirements for text features. He describes a system for automatically classifying Web sites into industry categories thanks to targeted spidering including metadata extraction and opportunistic crawling of specific semantic hyperlinks.

The first step in this classification process is the analysis of Web contents and its quality. Therefore Mr Pierre worked on a set of 29.998 Web domains, randomly chosen. He assumed as well that it is common to include a title and possibly a set of keywords and description metatags. One of the more promising sources of text features should be found in Web page metadata.

Within the huge collection of Web domains, he counted the number of words used in the contents attribute of the `<META name="keywords">` and `<META name="description">` tags as well as `<TITLE>` tags and `<BODY>` tags. Table 2.1 shows the result of this inventory.

Tag Type	0 words	1-10 words	11-50 words	51+ words
Title	4%	89%	6%	1%
Meta-Description	68%	8%	21%	3%
Meta-Keyword	66%	5%	19%	10%
Body Text	17%	5%	21%	57%

Table 2.1: *Percentage of Web pages with words in HTML tags*

As we can see, a title generally counts between one and ten words. The body text often counts more than 51 words. But a really interesting result is the lack of Meta-description and Meta-keywords for the Web pages. It is quite surprising as we all know that they play an important role in the ranking and the display of search results given by several major search engines. Indeed metatags can be useful when they exist because they contain text specifically intended for aiding in the identification of Web site's subject areas.

Good text features, that is words that describe a domain, are needed to accurately discriminate between different categories in order to build an automated classification process. According to D. Lewis[4] a feature must be :

1. Relatively few in number
2. Moderate in frequency of assignment
3. Low in redundancy
4. Low in noise
5. Related in semantic scope to the classes to be assigned
6. Relatively unambiguous in meaning

Metatags seem to meet for subject classification those requirements better than other sources of text such as titles and body text.

The second step of the classification is the choice of a classification scheme. In this work, the North American Industrial Classification System (NAICS) has been chosen. The original NAICS is actually not presented as in table 2.2. The full NAICS has six levels of hierarchy and contains several thousand subcategories. But for the experiment, it has been simplified to the main top-level categories. The job is now to arrange the Web sites in this classification system using the metatags as features.

The next step, the third one, is the targeted spidering. Due to the lack of homogeneity in Web contents, the existence of key features can be rather inconsistent. A targeted spidering approach tries to gather as many key features as possible with as little effort as possible. These key features are the metatags but in the case of their absence, other sources of text such as titles and body text were needed to provide adequate coverage of Web sites. The spider begins at the top level page of the Web site and attempts to extract useful text from metatags and titles if they exist, and then follows links for frame sets if they exist. It also follows any hyperlinks that contains key substrings in their anchor text and again looks for metatag contents in those pages. Only if no metatag contents was found did the spider gather the actual body text of the Web page. Finally, all the extracted texts are concatenated to a single document representative of the concerned Web site.

The fourth step of the categorization is the training data. 13,557 domain names among the 29,998 at first had usable text contents and were pre-classified according to one or more industry categories. Two approaches were used. Mr Pierre firstly used among

NAICS Code	NAICS Description
11	Agriculture, Forestry, Fishing and Hunting
21	Mining
22	Utilities
23	Construction
31-33	Manufacturing
42	Wholesale Trade
44-45	Retail Trade
48-49	Transport and Warehousing
51	Information
51	Finance and Insurance
53	Real Estate and Rental and Leasing
54	Professional, Scientific and Technical Services
55	Management of Companies and Enterprises
56	Administrative and Support, Waste Management and Remediation Services
61	Educational Services
62	Health Care and Social Assistance
71	Arts, Entertainment and Recreation
72	Accommodation and Food Services
81	Other Services (except Public Administration)
92	Public Administration
99	Unclassified Establishments

Table 2.2: *NAICS categories*

others a combination of 426 NAICS category labels as training example. The second approach is the more conventional classification by example. He used 3618 pre-classified domain names along with text for each domain obtained using the targeted spider. an information retrieval engine for comparing queries to training examples is applied.

The final step, the classifier algorithm, analyses the results (text documents representative of Web sites) given by the spider. A decision algorithm assigns the Web sites to the NAICS categories. It is based on the K-nearest neighbours algorithm and is capable of producing good results even when the amount of training data is limited. This decision module is also responsible for thresholding and presenting the final set of automatically assigned categories.

To conclude, this type of system have the advantage of being adjustable. It can be applied to any domain specific classification scheme. All that is needed is to define the categories, assemble the training data, and configure the spider to extract the appropriate features. The spider may be constructed to follow specific types of links, or extract sections of Web page contents that are most useful for a given domain.

2.5 Classification by topics

Another interesting approach of Web sites classification has been developed by Martin Ester, Hans-Peter Kriegel and Matthias Schubert[5]. They define the Web sites classification as follows : given a set of site classes C and a new Web site S consisting of a set of pages P, the task of Web sites classification is to determine the element of C which best categorizes the site S. They introduce three approaches of classifying Web sites based on different representations:

- *Classification of superpages*: a Web site is represented as a single virtual Web page consisting of the union of all its pages, i.e. the Web site is represented by a vector of term frequencies.
- *Classification of topic vectors*: a Web site is represented by a vector of topic frequencies. This classification method will not be discussed in this thesis.
- *Classification of Web site trees*: a Web site is represented by a tree of pages with topics.

2.5.1 Classification of superpages

This simple technique is an extension of the methods used for the Web pages classification. The Web site is considered as a graph : the Web site of a domain D is a directed graph $G_D(N, E)$. A node $n \in N$ represents an HTML page whose URL starts with D. A link between n_1 and n_2 with $n_1, n_2 \in N$ is represented by the directed edge $(n_1, n_2) \in E$. Thus, every HTML document under the same domain name is a node in the site graph of the domain and the hyperlinks from and to other pages within the same domain are the connecting edges.

The Web site is downloaded very simply : the starting page (the page whose URL consists of the domain name only) is downloaded and read. An HTML parser is then applied to determine the links to the other pages within the site. Then every link to a page, beginning with the same domain name, is followed. The pages already visited are marked to avoid circles.

An algorithm walks through the nodes of the site graph and counts terms with a vector counting the frequency of terms over all HTML pages of the whole site. The Web site is represented by a single superpage. Afterwards the vector can be classified by any standard data mining package.

This technique is very simple and is not much more complex than the classification of single pages. But it is only valid for small and medium-size business sites where the actual site really stands behind the domain name. The right choice of the key terms is a delicate problem as well. But the worst drawback of this method is the loss of local context : keywords appear anywhere within the site and are aggregated to build up a bag-of-words view of the whole Web site graph. This leads to a no distinction between an appearance within the same sentence, the same page or the same site while the context is a major actor in the classification task.

2.5.2 Classification of Web site trees

In this approach, single key terms are no more used and are exchanged for complete HTML documents, that is, for topics. Topics have the advantage to preserve the local context. In order to summarize the contents of a single Web page, a topic is assigned out of a predefined set of topics (or page classes) to it. This is done by text-classification on terms using the Nave Bayes classifier. If the page doesn't belong to any of the predefined topics, we assign the "other"-category to it.

In their work, Ester, Kriegel and Schubert found that after examining many business sites in several trades, the following categories of pages, although their trades varied widely, are to be found in most classes of business-sites : company, company philosophy, online contact, places and opening hours, products and services, references and partners, employees, directory, vacancies and other. Therefore, it is possible to find a list of recurrent topics for a specific type of site. For the news Web sites, it is easy to find such topics : politics, economy, sports, culture, weather, etc.

Once the topics have been identified within a Web site, the classification task need to build a Web site tree based on the labelled pages of a Web site and on a directed graph of the previous section. The idea is that the structure of most sites is very hierarchical. Sites begin with a unique root node provided by the starting page and commonly have directory-pages that offer an overview of the topics and the links leading to them. Furthermore, the information in most sites begins very general in the area around the starting page and is getting more and more specific with increasing distance.

For building Web site trees, our three researchers use the minimum number of links as a measure of distance between two pages of a site and build up the tree as the set of minimum paths from the starting page to every page in the graph. Therefore, they perform a breadth-first search through the Web site graph and ignore the links to pages already visited. Note, that in the case of two paths to the same page of equal length, the one occurring first is chosen. Figure 2.2 shows an example of a Web site tree for an IT-service provider.

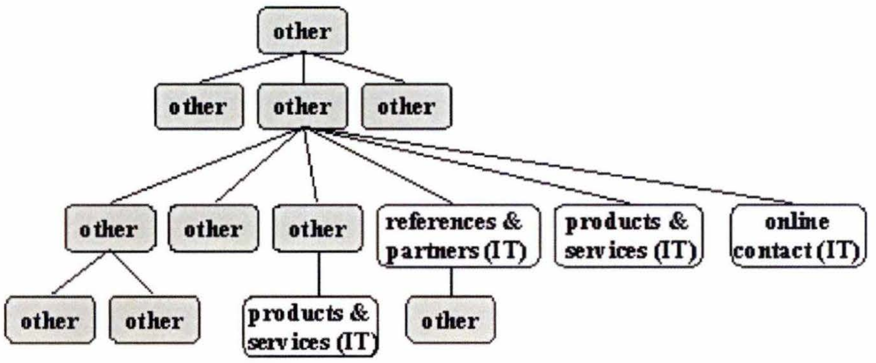


Figure 2.2: *Example of Web site tree for small IT-service provider*

Now a Web site tree has been built, the classification is now possible with this tree. A Web site class is the class that contains all the topics of a class. Thus, to classify this Web site, we define the most likely Web site class as:

$$C = maxarg\ \Pr[C_i|S] = maxarg\ (\Pr[C_i] \cdot \Pr[S|C_i])$$

This definition is based on the Nave Bayes approach and means that the predicted class C of the site S is the class C_i that explains the occurrence of the given site S best. Due to the Bayesian rule the probability $\Pr[C_i|S]$ is the product of the a priori probability $\Pr[C_i]$ and the probability $\Pr[S|C_i]$ that given the model for class C_i the Web site tree S would have been constructed. We therefore estimate $\Pr[C_i]$ as the relative frequency of Web sites in the class C_i.

To calculate $\Pr[S|Ci]$, let us take a simple example for our news Web site. Let us assume that there are three topics in the news Web site class : politics (a), economy (b) and culture (c). Our news Web site tree is represented at the center of figure 2.3.

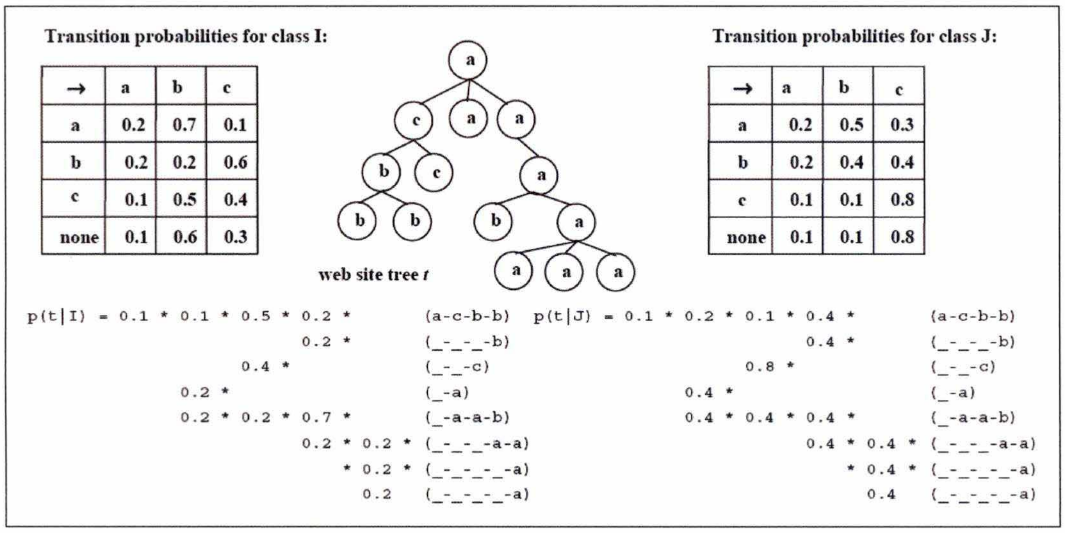
We have now to build a probability table for all possible transitions between the topics of the news site class as shown in figure 2.3.

The left table in figure 2.3 shows all the possible transitions between the topics. The values are given as example. For instance, the probability to transit from the politics topic to the economy topic is 0,7. The probability to stay in the same culture topic is 0,4. The none state indicates the probability to start with a given topic. For all the other Web site classes, a similar table must be build.

The concept of k-order Markov chains is then applied to web site trees using the following procedure. Beginning with the probability for the label of our root node we multiply the probabilities of the transition between the k last nodes and their successors. In the simple case of 1-order Markov chains, for example, the transition probability for the page classes X and Y with respect to site class S is the probability that within a Web site of class S a link from a page of class X leads to a page of class Y. Since there can be more than one successor in the tree, we multiply the transition probabilities for every child node, traversing along every path that a tree contains simultaneously.

To calculate the conditional probability $\Pr[S|C_i]$ of the Web site tree S, let us use the following definition:

$$\Pr[S|C_i] = \prod_{t \in S} \Pr\left[\, l_t \,|\, pre(k,t), pre(k-1,t), \ldots, pre(1,t) \,\right]$$



Transition probabilities for class I:

→	a	b	c
a	0.2	0.7	0.1
b	0.2	0.2	0.6
c	0.1	0.5	0.4
none	0.1	0.6	0.3

Transition probabilities for class J:

→	a	b	c
a	0.2	0.5	0.3
b	0.2	0.4	0.4
c	0.1	0.1	0.8
none	0.1	0.1	0.8

```
p(t|I) = 0.1 * 0.1 * 0.5 * 0.2 *         (a-c-b-b)
                           0.2 *         (_-_-_-b)
                     0.4 *               (_-_-c)
               0.2 *                     (_-a)
               0.2 * 0.2 * 0.7 *         (_-a-a-b)
                           0.2 * 0.2 *   (_-_-_-a-a)
                               * 0.2 *   (_-_-_-_-a)
                                 0.2     (_-_-_-_-a)

p(t|J) = 0.1 * 0.2 * 0.1 * 0.4 *         (a-c-b-b)
                           0.4 *         (_-_-_-b)
                     0.8 *               (_-_-c)
               0.4 *                     (_-a)
               0.4 * 0.4 * 0.4 *         (_-a-a-b)
                           0.4 * 0.4 *   (_-_-_-a-a)
                               * 0.4 *   (_-_-_-_-a)
                                 0.4     (_-_-_-_-a)
```

Figure 2.3: *The calculation of $Pr[S|C_i]$ for two site classes I and J*

where:

- l = topic of a node
- t = node
- S = path
- k = number of predecessors of a node
- $pre(k, t)$ = function which returns the topic of the k^{th} predecessor of node t.

Thus, for every node t in the site tree S the probability that its label l_t occurs after the occurrence of the labels of its k predecessors is multiplied. Figure 2.3 shows the calculation of $\Pr[S|C_i]$ for two site classes I and J.

Finally, we are able to calculate the most likely Web site class $C = maxarg\ (\Pr[C_i] \cdot Pr[S|C_i])$ while we have $\Pr[C_i]$ and $\Pr[S|C_i]$. The considered Web site is thus associated with the topic C.

2.6 Conclusion

This second chapter was all about the Web sites classification. As more than a million Web pages are being added every day on the World Wide Web, it is necessary to classify this huge amount of data. The first step in order to tidy up the Internet is the classification and the categorization of Web sites. The three techniques studied in great detail in this chapter allow such a task. In our case, about the extraction of news articles, it would be very interesting to firstly classify the news Web sites or news portals and secondly classify the Web pages of interest within each of these news sites. For instance, if we are looking for sports news, we could classify the Web sites that only give sports news to restrict the field of research and then classify the relevant news within these sports Web sites. The second step of the Web classification is therefore the Web pages classification considered in chapter 3.

Chapter 3

Web pages classification

Contents

3.1 Introduction

In this third chapter we shall present the classification of relevant Web pages. It is the second step, after the Web sites classification, in the scheme of the news extraction. In order to analyse such a task, we shall go into three pages classification methods in depth. The first one concerns the URLs and table layout classification, the second one the classification through summarization and the third one the Ant-Miner algorithm. We shall finally close this chapter by a conclusion.

3.2 Context

Web pages classification is necessary to help the Internet user to cope with the mass of information on the World Wide Web. Web site classification, argued in the previous section, is the first step of this task but is actually not often used. Even if it is, the second step, the pages classification, has to be done to give the final results of a user query. Various techniques have been suggested but are based on the same idea: analysing the contents of Web pages with keywords. Though some interesting and original approaches have been discussed these two last years: the URLs and table layout classification, the classification through summarization and the ant colony concept.

3.3 URLs and table layout classification

The first method of pages classification concerns the use of URLs and the placement of links to them on a referring page placed in a table layout. Both researchers, Lawrence Kai Shih and David R. Karger, assume that contents providers tend to choose URLs and page layouts that coherently structure their contents according to topic. They want to formalize such intuitions into a general way of algorithmically predicting the properties of the links targets.

3.3.1 URLs hierarchy

While the World Wide Web Consortium **argues** that document URLs should be opaque, L.K. Shih and D.R. Karger[6] noticed that most URLs nowadays have human-oriented meanings and are useful for recommendation problems. On many Web sites, URLs are organized in a hierarchy according to subject. For example, on the CNN news Web site, the articles about space of the last year are placed under the following URL: cnn.com/2004/tech/space. That means that all articles about this topic are placed in a first space subtree, then in a second more general tech subtree, and so on of the CNN tree. The location of an article in the URL tree is therefore suggestive of the user's interest in it. But this technique is also helpful for the Internet user. A good URL structure provides helpful contextual clues for the reader. Browsers use besides this URL hierarchy advantage displaying URLs along with the text description of a link. The user can infer from a URL that a document serves a particular function or relates to a topic. This hierarchical organization is the main advantage of URLs for this work.

Another plus is that they are easy to extract and relatively stable. Each URL maps uniquely to a document, and any fetchable document must have a URL while other Web features, like words or texts, are optional and not unique to a document.

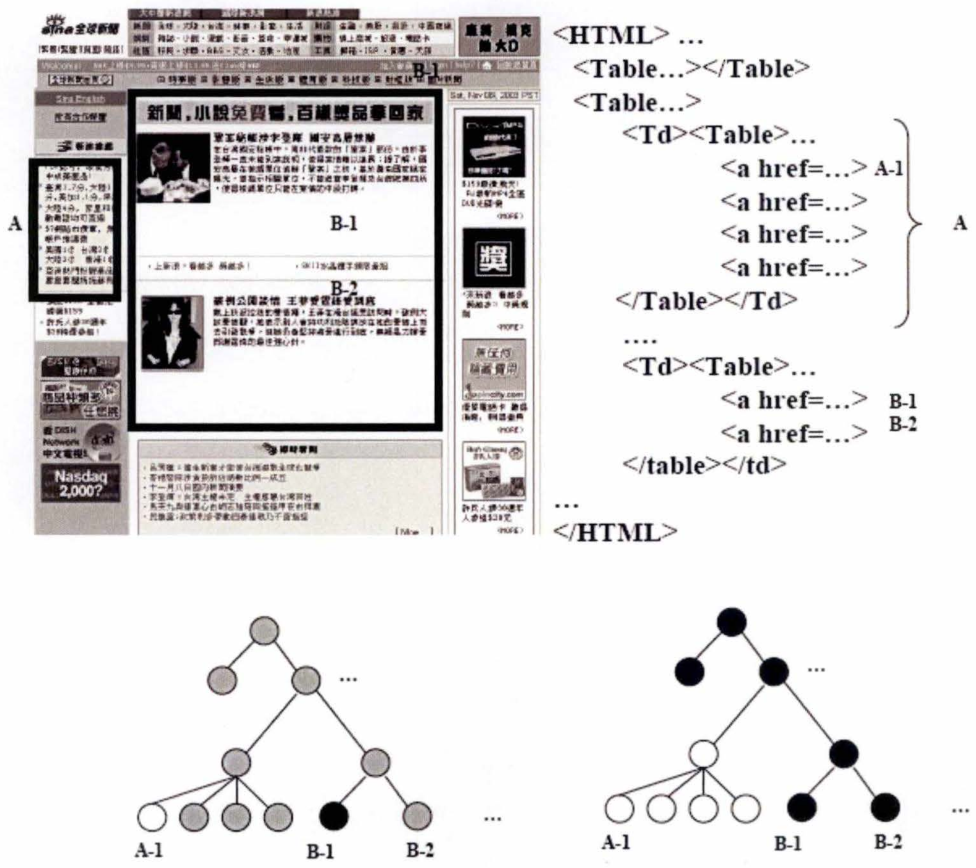
The last asset is that URLs can be read without downloading the target document. This makes to perform Web pages classification more quickly possible.

Technically, to convert a URL hierarchy into a tree structure,L.K. Shih and D.R. Karger tokenized the URL by the characters /, ? and &. The character / is a standard delimiter for directories that was continued into Web directories. The characters ? and & are standard delimiters for passing variables into a script. http: is the root of the tree and successive tokens in the URL become the children of the previous token.

3.3.2 Table layout

Another technique of this classification approach is the use of a table layout within a Web page. This layout, following the example of URLs, is hierarchical as well. For instance, the CNN Web site front page offers a "table of contents" partitioning its news articles under a number of labels such as "U.S.", "World", "Travel" and "Education" representing subject classifications that may well be strong indicators of "interestingness" for a reader. This also allows to help a user understand how to use a site: the layout tends to be templated (most pages will retain a "look and feel") and different articles on one particular topic might appear in the same place on the page day after day.

The table layout is delimited by HTML table tags that have to obey a standardized HTML grammar making the table feature easy to extract. The HTML table tags correspond to rectangular groupings of text, images and links. But this layout often contains much more than the site's navigation. The contents of a site might use tables to group together



```
<HTML> ...
  <Table...></Table>
  <Table...>
     <Td><Table>...
            <a href=...> A-1
            <a href=...>
            <a href=...>
            <a href=...>
     </Table></Td>
     ....
     <Td><Table>...
            <a href=...> B-1
            <a href=...> B-2
     </table></td>
...
</HTML>
```

Figure 3.1: *A table layout, its HTML code and its conversion into a tree structure*

articles by importance (i.e. the headline news section), by subject, or chronologically (newest items typically at the top).

To convert the HTML table structure into a tree-shape, they used a hand-written Perl program that extracted the HTML table tags `<table>` and `<td>`. The root of the tree is the entire page's HTML. The children of a node are the next lower level of table elements. Figure 3.1 shows an example of such a tree structure of a table layout.

3.3.3 Mathematical model

As we can now notice, each of both elements (URLs and table layout) of the Shih and Karger's classification approach have been transformed into a tree structure. From this point, these tree structures, while coming from two different elements, are the same and will be handled indistinctly as Bayes nets.

The underlying idea of the classification problem of the Bayes nets is as follows: imagine that you do not understand the Chinese language. A friend of yours knows this complex language and you ask him to recommend you a good Chinese news article. He gives you the URL of this article (B-1 in figure 3.1) and you find on the same page, just below the news article, what it seems to be another article (B-2 in figure 3.1), even if you do not know the Chinese language. Your mind just guessed and assumed that it was strongly possible that the other series of signs just below the recommended article is a news article as well. It made good guesses about some of its semantic attributes. Another intuition would be that if one person mentioned that the top link in box A (Figure 3.1) was an advertisement, it might be reasonable to guess that everything in box A was an advertisement too. The work of both researchers tries to formalize and build algorithms that automatically make generalizations like these.

The aim of the automated generalization is to work out the class of each node of the tree. Formally, a probability distribution is defined over the possible classes of items in the tree. To accomplish this, the tree holding the items to be classified is considered as a Bayes net. The assumption that the class to be learned is correlated to the tree position is captured in a model based on mutations. Some children of internal nodes may "mutate" into different classes.

There are two classes (0 and 1) and two mutation probabilities (forward: θ and backward: ϕ). The class of a node follows probabilistically from the class of its parent. Suppose node x has parent y in the tree. Let N_x and N_y denote the classes of nodes x and y respectively. Then

$$\Pr[N_x = 1 | N_y = 0] = \theta$$
$$\Pr[N_x = 0 | N_y = 1] = \phi$$

Applying this rule from the root to the leaves, it is possible to assign classes to the nodes. Given the root's class, we can flip coins according to the formulas above to determine the children's classes; given these we can generate the grandchildren's classes, and so on.

To initiate the process, we must choose a value for the root class. In order to minimize the number of parameters, it is useful to choose a root class function with the already existing parameters θ and ϕ. For the root node r, we declare that

$$\Pr[N_r = 0] = \frac{\phi}{\theta + \phi}$$

This formula is useful: if x is a child of the root, then

$$\begin{aligned} \Pr[N_x = 0] &= \Pr[N_x = 0 | N_r = 0] \Pr[N_r = 0] + \Pr[N_x = 0 | N_r = 1] \Pr[N_r = 1] \\ &= (1 - \theta) \frac{\phi}{\theta + \phi} + \phi \frac{\theta}{\theta + \phi} \\ &= \frac{\phi}{\theta + \phi} \end{aligned}$$

In other words, with this root probability, all nodes in the tree have the same probability of being class 0, prior to labelling any of the leaves. Later, as the leaves are labelled, these probabilities will change. If θ and ϕ are small, the model asserts that a node is likely to have the same class as its parent, and thus likely to have the same class as its siblings and other nearby nodes in the tree. As the mutation probabilities increase, the correlation between nearby nodes in the tree decreases. θ and ϕ are usually constrained to be less than 0,5.

The Bayes net is created by setting conditional probabilities for all edges in the tree as follows: for node x with parent y, $\Pr[N_x = 1 | N_y = 0] = \theta$ and $\Pr[N_x = 0 | N_y = 1] = \phi$; and by setting the root prior as follows: $\Pr[N_0 = 1] = \frac{\phi}{\theta + \phi}$.

How do the probabilities in the tree change when leaves are labelled? As we are working on Bayes nets, we must use the concept of evidence. If we use no evidence, each node x will have the probability $\Pr[N_x = 0] = \frac{\phi}{\theta + \phi}$, that is of being class 0. But if we use evidence, the problem is not the same. For example, let us assume that we know the classes of a set of nodes E. This will influence our predictions about other nodes. The previous probability is now conditioned on the evidence E. We can expand this conditional probability as

$$\frac{\Pr[N_x = 0 \cap E]}{\Pr[E]}$$

Let us suppose now that the root r has two children x and y, and let E_x and E_y denote the labelled leaf nodes in the x and y subtrees, respectively. Given the class of the root r, the classes of nodes in the two subtrees are independent of one another, and their probabilities can be multiplied: $\Pr[E|N_r = 0] = \Pr[E_x|N_r = 0] \Pr[E_y|N_r = 0]$ and similarly for the case $N_r = 1$. But to calculate this multiplication, we must, as said above, pick a random class for x, and then assign classes to items below x based on the class of x. That is

$$\begin{aligned}\Pr[E_x|N_r = 0] &= \Pr[E_x|N_x = 0] \Pr[N_x = 0|N_r = 0] + \Pr[E_x|N_x = 1] \Pr[N_x = 1|N_r = 0] \\ &= (1-\theta) \Pr[E_x|N_x = 0] + \theta \Pr[E_x|N_x = 1]\end{aligned}$$

Generalizing that $p_{x0} = \Pr[E_x|N_x = 0]$ and $p_{x1} = \Pr[E_x|N_x = 1]$, the formula above, generalized to an arbitrary number of children, says that for any node x,

$$p_{x0} = \prod_{y \in children(x)} [(1-\theta)\, p_{y0} + \theta\, p_{y1}]$$

$$p_{x1} = \prod_{y \in children(x)} [\theta\, p_{y0} + (1-\theta)\, p_{y1}]$$

Working up from the leaves, two quantities p_{x0} and p_{x1} at each node x in the tree are calculated. At the end of the recursion we have p_{r0} and p_{r1} and the root. At this point we can compute

$$\begin{aligned}\Pr[E] &= p_{r0} \Pr[N_r = 0] + p_{r1} \Pr[N_r = 1] \\ &= p_{r0} \frac{\phi}{\theta + \phi} + p_{r1} \frac{\theta}{\theta + \phi}\end{aligned}$$

Thanks to this classification method based on the Bayes nets concepts, each node of the original tree has been assigned to a class.

3.4 Classification through summarization

While categorization is expected to play an important role in future search services, Dou Shen, Zheng Chen, Qiang Yang, Hua-Jun Zeng, Benyu Zhang, Yuchang Lu and Wei-

Ying Ma [7] are working on a new Web pages classification algorithm based on Web summarization for improving the accuracy. This summarization is made through four methods for each Web page. These four methods give each one a set of sentences associated to a high significance factor. The set of sentences given by the particular summarization methods is the summary. The final result is the sum of the significance factors of the four methods.

3.4.1 S_{Luhn}

The first measurement to get, S_{Luhn}, comes from the Luhn's method. In this extraction-based method, every sentence of a Web page is assigned with a significance factor, and the sentences with the highest significance factor are selected to form the summary. In order to compute the significance factor of a sentence, a significant words pool is created for each category by selecting the words with high frequency after removing the stop words (words that from a non-linguistic point of view do not carry information) in that category. Then a 3-steps method is applied to the Web page:

1. Set a limit L for the distance at which any two significant words could be considered as being significantly related.
2. Find out a portion in the sentence that is bracketed by significant words not more than L non-significant words apart.
3. Count the number of significant words contained in the portion and divide the square of this number by the total number of words within the portion.

The result is the significant factor related to the sentence.

But this method is only available when the category of a Web page is known, i.e. for a training set of Web pages. If we take a testing Web page from the Internet, its category is not yet known. The significant score of each sentence can not be calculated according to the significant words pool corresponding to its category label because we do not know its category. To solve this problem we need to calculate the significant factor for each sentence according to different significant words pools over all categories separately. The significant score for the target sentence will be averaged over all categories and referred to as S_{Luhn}. The summary of this testing page will be made up by the sentences with the highest scores. This S_{Luhn} is the first member of the final summarization equation.

3.4.2 S_{LSA}

S_{LSA} is the second measurement to calculate. LSA stands for Latent Semantics Analysis and represents terms and related concepts as points in a very high dimensional "semantic space". LSA is based on singular value decomposition (SVD), a mathematical matrix decomposition technique that is applicable to text corpora experienced by people. SVD uses among others singular vectors representing patterns and singular values indicating the importance degree of the corresponding pattern within the document. Any sentence of a given Web page containing a word combination pattern will be projected along the singular vector corresponding. The sentence that best represents this pattern will have the largest index value S_{LSA} with this vector. As in the Luhn's method, the sentences with the highest S_{LSA} are selected to form the summary.

3.4.3 S_{CB}

In this third method, the structure of Web pages is taking into account. A Web page contains a lot of noisy blocks (see section 4.4.1) leading to a more difficult summarization. The Function-Based Object Model (FOM) allows to utilize this structure of the Web page. In FOM, objects are classified into a Basic Object (BO) or a Composite Object. A BO is the smallest information body that cannot be further divided, i.e. a jpeg file. HTML speaking, there is no other tag inside the contents of a BO. A CO as for it is a set of Objects (BO or CO) that perform some functions together. Once all the BOs and COs have been identified, they are characterized by a category such as information object, navigation object, decoration object, etc. Then the contents body (CB) of the Web page, that is the main objects conveying important information about the topic of that page, is identified as follows:

1. Consider each selected object as a single document and build the $TF \times IDF$ index for the object.

2. Calculate the similarity between any two objects using cosine similarity computation, and add a link between them if their similarity is greater than a threshold chosen empirically. After processing all pairs of objects, we will obtain a linked graph to connect different objects.
3. In the graph, a core object is defined as the object having the most edges.
4. Extract the CB as the combination of all objects that have edges linked to the core object.

A score S_{CB} is then assigned to each sentence. If the sentence is included in the CB then $S_{CB} = 1.0$ otherwise $S_{CB} = 0.0$. The summary of the Web page is the set of sentences having a $S_{CB} = 1.0$.

3.4.4 S_{Sup}

The last measurement to get is the S_{Sup}. For each Web page we want to summarize, a series of eight features are automatically extracted from the page. These eight features, thanks to a Nave Bayesian classifier, will then be used to identify whether a sentence should be selected into its summary or not. Here are the eight features utilized: Given a set of sentences S_i, $(i = 1 \dots SN)$ in a page,

1. f_{i1} measures the position of a sentence S_i in a certain paragraph.
2. f_{i2} measures the length of a sentence S_i which is the number of words in S_i.
3. $f_{i3} = \sum TF_w \cdot SF_w$. This feature takes into account not only the number of word w into consideration, but also its distribution among sentences. We use it to punish the locally frequent words.
4. f_{i4} is the similarity between S_i and the title. This similarity is calculated as the dot product between the sentence and the title.
5. f_{i5} is the cosine similarity between S_i and all text in the page.
6. f_{i6} is the cosine similarity between S_i and meta-data in the page.
7. f_{i7} is the number of occurrences of word from S_i in special word set. The special word set is built by collecting the words in the Web page that are italic or bold or underlined.

8. f_{i8} is the average font size of the words in S_i. In general, larger font size in a Web page is given higher importance.

where

- PN: the number of paragraphs in a document
- SN: the number of sentences in a document
- PL_k: the number of sentences in a certain paragraph k
- $Para(i)$: the associated paragraph of sentence i
- TF_w: the number of occurrences of word w in a target Web page
- SF_w: the number of sentences including the word w in the page

The Nave Bayesian classifier using the eight features to find the sentences to select for the summary is:

$$\Pr[s \in S | f_1, f_2, \ldots f_8] = \frac{\prod_{j=1}^{8} \Pr[f_j | s \in S] \ \Pr[s \in S]}{\prod_{j=1}^{8} \Pr[f_j]}$$

where

- $\Pr[s \in S]$ stands for the compression rate of the summarizer which can be predefined for different applications
- $\Pr[f_i]$ is the probability of each feature i
- $\Pr[f_i | s \in S)$ is the conditional probability of each feature i

This equation gives a score S_{sup} to each sentence.

3.4.5 The final equation

Finally, we obtain a temporary summary for the Web page we wanted to summarize. This summary is made up of the set of sentences given by the four methods applied on the concerned Web page. The final score of each sentence is calculated by the following sum:

$$S = S_{Luhn} + S_{LSA} + S_{CB} + S_{Sup}$$

The final summary will be made up of the sentences having the highest score S.

3.4.6 Classification process

Now all the summaries of the Web pages we want to classify have been created, we are able to apply once again a Nave Bayesian Classifier. It will allow us to use the joint probabilities of words and categories to estimate the probabilities of categories given a document. The Nave Bayesian Classifier uses the following Bayes' rule:

$$\Pr[c_j|d_i,\hat{\theta}] = \frac{\Pr[c_j|\hat{\theta}] \prod_{k=1}^{n} \Pr[w_k|c_j,\hat{\theta}]^{N(w_k,d_i)}}{\sum_{r=1}^{|C|} \Pr[c_r|\hat{\theta}] \prod_{k=1}^{n} \Pr[w_k|c_r,\hat{\theta}]^{N(w_k,d_i)}}$$

where

- $\Pr[c_j|\hat{\theta}]$ can be calculated by counting the frequency with each category c_j occurring in the training data;
- $|C|$ is the number of categories;
- $\Pr[w_i|c_j]$ stands for probability that word w_i occurs in class c_j which maybe small in training data, so the Laplace smoothing is chosen to estimate it;
- $N(w_k, d_i)$ is the number of occurrences of a word w_k in d_i;
- n is the number of words in the training data.

Experimental results show that this classification approach through summarization can achieve a 12.9% improvement of correct Web pages found compared with a classical search query through the *Yahoo* or *LookSmart* directories.

3.5 Ant-Miner

The last Web pages classification method is based on Ant-Miner, an Ant Colony algorithm for discovering classification rules [8]. This approach is very original and innovative and is an important research direction followed by Nicholas Holden and Alex A. Freitas from the University of Kent in England. As this method uses rules, the goal of the classification task is to discover rules from a set of training data and apply those rules to a set of test data (unseen during training), and hopefully predict the correct class in the test set.

Ant-Miner classification rules are based on words contained in unstructured or semi-structured texts found in Web pages. As everybody knows, there is no program at the moment that can fully understand the meaning of a given Web page. Simple interpretation can only be made. The reason is twofold: first, the natural language is very complicated and second, the number of possible classification rules is exponential on the number of words, so that the search space becomes quickly very large. Therefore N. Holden and A.A. Freitas decided to restrict the scope of the investigations about words. For finding the best words representing a Web page they want to select appropriate words in the summaries or descriptions of the Web page in `<meta>` tags applying linguistics-based text preprocessing techniques. They then use WordNet. WordNet is an electronic lexicon that contains several relationships between words. It is an attempt to match the human understanding of words and their relationships into an electronic database. They use three linguistic resources from this lexicon to preprocess the data:

1. *Removing the words suffixes*: In order to cut down the number of attributes (words) for their Ant-Miner, it is necessary to perform stemming on words. It allows also to find patterns more easily. For instance, instead of selecting the three words *eat*, *eating* and *eater*, they only select the word *eat* having the same stem and the same meaning as the two last.
2. *Identifying all the nouns in the text*: Instead of taking all the important words within a text such as verbs, nouns, adjectives, etc, they use WordNet to identify only the nouns. The idea behind this technique is that nouns are usually the subject of a sentence. Once again, it allows to cut down the number of attributes to add to their list of words.
3. *Capturing the idea of a given word in a more generic form*: If different pages have the same idea behind what they contain, expressed by a series of attributes, WordNet must find the relationship between these words resulting in the selection of only one or two relevant words to restrict again the number of attributes. For example, if one page contains the words: window, roof, and door, and another Web page contains the words chimney, room and brick then WordNet should find the word house. This word generalization is unfortunately risky: if WordNet finds the wrong relationship between the words, it could lead to wrong results of the

Ant-Miner. That is the reason why both researchers apply an algorithm searching for the hypernyms (generalizations) of a pair of words and returning the best one.

Once the number of attributes has been strongly and efficiently cut down, they can be used by the Ant-Miner algorithm. The Ant-Miner algorithm is based on the following Ant Colony paradigm: in nature, ants work on finding food and taking it back to their ant hill. To do that, they create "highways" to and from their food. But these "highways" have not the same length. One ant takes a short path, another a bit longer, another one the longest, etc. Each ant lays down an amount of pheromone and the other ants are attracted to these amounts of pheromone. If an ant follows the shortest path, it will make more trips from and to the ant hill, laying down consequently more and more amounts of pheromone compared with the ant following the longest path. As ants are attracted to pheromone, the ones following longer paths than the shortest will be attracted to the large amounts of pheromone on the shortest path. Finally all of them will follow the shortest "highway".

This Ant Colony paradigm is the basic idea of the Ant-Miner algorithm. Instead of foraging for food the ants in the Ant-Miner algorithm forage for classification rules, and the path they take correspond to a conjunction of attribute-value pairs (terms). A rule consists of an antecedent (a set of attribute values) and a consequent (class): `IF <attrib = value> AND ... AND <attrib = value> THEN <class>`. Figure 3.2 gives a high-level pseudo code of Ant-Miner helping to understand its inner working.

```
TrainSet = {all training cases};
DiscoveredRuleList = [];    /* initialized with empty list */
REPEAT
   Initialize all trails with the same amount of pheromone;
   REPEAT
      An ant incrementally constructs a classification rule;
      Prune the just-constructed rule;
      Update the pheromone of all trails;
   UNTIL (stopping criteria)
   Choose best rule out of all rules constructed by all ants;
   Add the best rule to DiscoveredRuleList;
   TrainSet = TrainSet - {cases correctly covered by best rule};
UNTIL (stopping criteria)
```

Figure 3.2: *High-level pseudo code of Ant-Miner*

The algorithm first starts by initializing the training set to the set of all Web pages and initializing the discovered rule list to an empty list. As we can see, the main part of the pseudo code in figure 3.2 is an outer Repeat-Until loop with a smaller inner one. Each iteration of the outer loop discovers one classification rule. This first step of this loop is to initialize all trails with the same amount of pheromone, which means that all terms have the same probability of being chosen (by the current ant) to incrementally construct the current classification rule. Then, the inner loop constructs an individual rule in three steps.

In the first step, an ant starts with an empty rule and incrementally constructs a classification rule by adding one term at a time to the current rule. A $term_{ij}$ (representing a triple $< Attribute_i = Value_j >$) is added to the current rule depending on whether the product $n_{ij} \cdot t_{ij}(t)$ is high where n_ij is essentially the information gain associated with $term_{ij}$ and $t_{ij}(t)$ is the amount of pheromone currently available in the position i, j of the trail being followed by the current ant and associated with $term_{ij}$ at iteration t. The higher the value of n_{ij} the more relevant for classification $term_{ij}$ is and so the higher its probability of being chosen. The quality of the rule constructed by an ant is evaluated and, as time goes by, the best trail positions to be followed, i.e. the best terms to be added to a rule, will have large amounts of pheromone, increasing their probability of being chosen to construct a rule.

The second step of the inner loop is the deletion of irrelevant terms of the current rule if this operation does not decrease the quality of the rule.

The third step is the increase of the pheromone in the trail followed by the ant, proportionally to the rule's quality. In other words, the higher the quality of the rule, the higher the increase in the pheromone of the terms occurring in the rule antecedent.

Finally the outer loop chooses the highest-quality rule out of all the rules constructed by all the ants in the inner loop, and it adds the chosen rule to the discovered rule list. Then it removes all cases that satisfy the rule antecedent and have the same class as predicted by the rule consequent from the initial Web page set. The next iteration creates a new rule with the smaller set of Web pages and so on.

The output of Ant-Miner is the discovered rule list with classes allowing the classification.

Experimental results on a set of 127 Web pages in three different classes (Education, Technology and Sport) were harvested from the BBC Web site. Ant-Miner produces accuracies that are at worst comparable to the more established $C5.0$ algorithm, a powerful data mining tool; and Ant-Miner discovers knowledge in a much more compact form than $C5.0$, facilitating the interpretation of the knowledge by the user.

3.6 Conclusion

This chapter was about the Web pages classification. Various original and innovative approaches have been treated. All of them have special features that suit specific problems. Experimental results, as well as those in the previous chapter, show that it is important to use them before the news extraction task as the amount of information available on the World Wide Web is huge and growing each year. But it is only the first process of the whole extraction issue. The next chapter we shall present is about the study of data structures that will be utilized for extracting news articles.

Chapter 4

Tree structures

Contents

4.1 Introduction

In this chapter we shall present the various data structures that are utilized to handle HTML, especially the tree structures (Context). Then three tree structures are analysed: the first one is about the Document Object Model tree, the second one about the site style tree and the third one about the tree structure we use in our work: the rooted ordered labelled tree, followed by a conclusion.

4.2 Context

The amount of data available on the Web has been growing explosively during the past few years. Users have now the opportunity to benefit from the available data in many interesting ways. But unfortunately, these ways are not always efficient. And the need for the user is, of course, a fast and efficient search that gives correct and accurate results.

That is the reason why major efforts have been made in order to provide efficient access to relevant information within this huge quantity of data. Two broad views of this problem have recently evolved:

- The first one is characterized by the unstructured view of data. It has developed breakthrough technologies (such as Web search engines) based on information retrieval methods. Browsing and keyword searching are two of these methods. But they are intuitive and present severe limitations. Browsing is not suitable for locating particular items of data because following links is tedious, and it is easy to get lost. Furthermore, browsing is not cost-effective as users have to read the documents to find desired data. Keyword searching is sometimes more efficient than browsing but often returns vast amounts of data, much more than the user can handle. How many times did you enter a keyword in a search engine resulting in an infinite list of Web sites, most of them having nothing to do with what you are looking for?
- The second one is characterized by the structured or semi-structured view of data. It borrows techniques from the database area to provide an efficient managing of the data available on the Web. But these adapted methods are still not spread mostly because of two intrinsic problems: the need for high human intervention and the low quality of the extraction results.

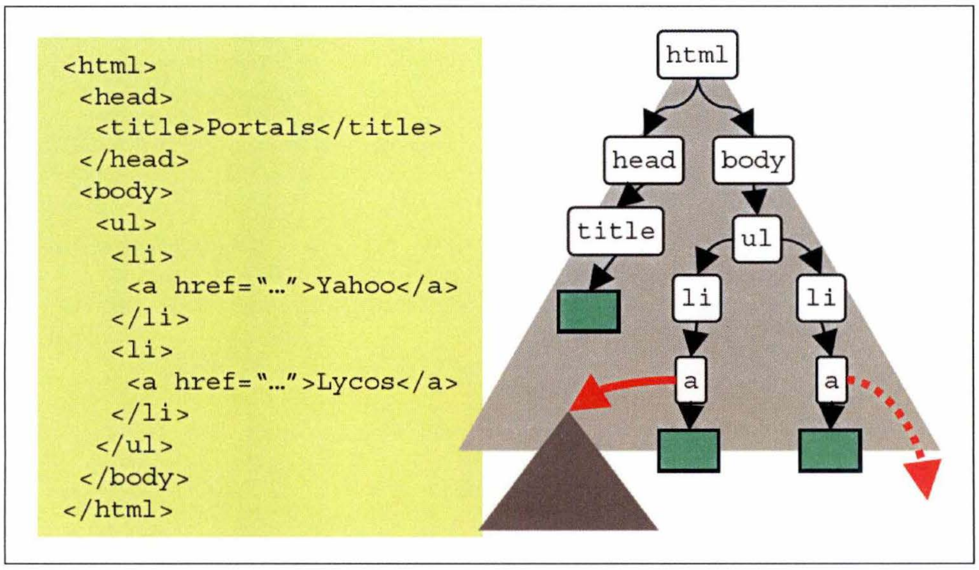
The need of a more structured view of data in order to provide efficient access to pertinent information still remains.

In order to response to these problems about the structured or semi-structured view of data, we have use in our work tree structures. These data structures are very specific and the research into this topic is considerable. Therefore, we shall only linger over the state of the art of these tree structures related to our work.

4.3 Document Object Model tree

The W3C defines the Document Object Model (DOM) as follows: "*The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.*
With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and contents. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model." [2]

This precise definition gives us a broad approach of the use of the Document Object



```
<html>
 <head>
  <title>Portals</title>
 </head>
 <body>
  <ul>
   <li>
    <a href="...">Yahoo</a>
   </li>
   <li>
    <a href="...">Lycos</a>
   </li>
  </ul>
 </body>
</html>
```

Figure 4.1: *From HTML code to the DOM tree representation*

Model. In our work, we will actually only use the logical structure of this description. Indeed, we just use the concept of representation that makes correspond an HTML page with a DOM tree where the tags are internal nodes and the detailed texts, images or hyperlinks are the leaf nodes (see figure 4.1).

The component that reads a text-formatted HTML file or stream and converts it to a DOM tree is called a parser. There are many parsers available that all implement the DOM interface. Some of the parsers are commercial softwares, some are free. The parsers also differ with HTML versions. The parser we use is a free parser adapted by ourselves in order to manipulate HTML (see Part II).

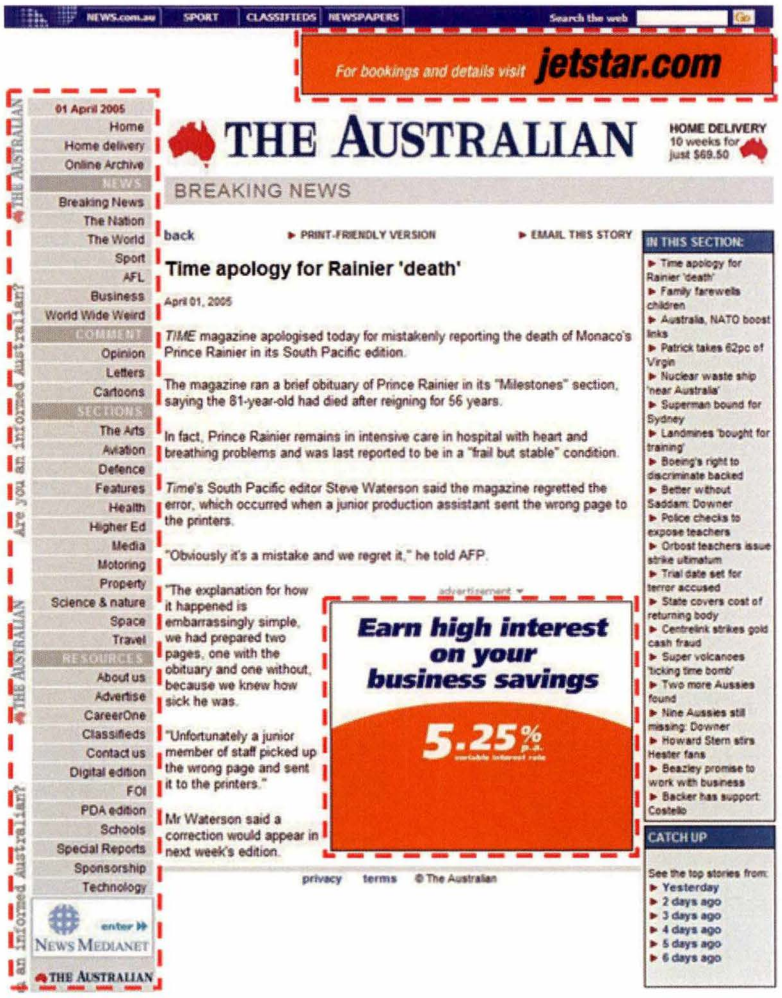
4.4 Style tree

The main part of the work when you want to extract the news article from a news page is to locate the article within the page, ignoring navigation panels, advertisements, links, etc. Although these information items are useful for human viewers and even necessary for the Web site owners, they are useless for the extracting task [9]. The need to detect and eliminate this useless information is important and an effective way to do that is to use the style tree structure.

4.4.1 Noisy blocks

We can divide a typical Web page in two parts: a part containing the main content blocks where the useful information is located (in our case, the news article) and another part containing blocks of useless information such as banner advertisements, navigational guides, decoration pictures, etc. We call these blocks that are not the main content blocks of the page the noisy blocks or the local noises.

Eliminating these noises is thus of great importance because they seriously harm the accuracy of data mining. The research of Lan Yi, Bing Liu and Xiaoli Li in this topic is of great interest. They propose a pre-processing step called Web page cleaning that eliminates the local noises within a Web page. This first task before mining becomes critical for improving the data mining results.



Time apology for Rainier 'death'

April 01, 2005

TIME magazine apologised today for mistakenly reporting the death of Monaco's Prince Rainier in its South Pacific edition.

The magazine ran a brief obituary of Prince Rainier in its "Milestones" section, saying the 81-year-old had died after reigning for 56 years.

In fact, Prince Rainier remains in intensive care in hospital with heart and breathing problems and was last reported to be in a "frail but stable" condition.

Time's South Pacific editor Steve Waterson said the magazine regretted the error, which occurred when a junior production assistant sent the wrong page to the printers.

"Obviously it's a mistake and we regret it," he told AFP.

"The explanation for how it happened is embarrassingly simple, we had prepared two pages, one with the obituary and one without, because we knew how sick he was.

"Unfortunately a junior member of staff picked up the wrong page and sent it to the printers."

Mr Waterson said a correction would appear in next week's edition.

- Time apology for Rainier 'death'
- Family farewells children
- Australia, NATO boost links
- Patrick takes 62pc of Virgin
- Nuclear waste ship 'near Australia'
- Superman bound for Sydney
- Landmines 'bought for training'
- Boeing's right to discriminate backed
- Better without Saddam: Downer
- Police checks to expose teachers
- Orbost teachers issue strike ultimatum
- Trial date set for terror accused
- State covers cost of returning body
- Centrelink strikes gold cash fraud
- Super volcanoes 'ticking time bomb'
- Two more Aussies found
- Nine Aussies still missing: Downer
- Howard Stern stirs Hester fans
- Beazley promise to work with business
- Backer has support: Costello

See the top stories from:
- Yesterday
- 2 days ago
- 3 days ago
- 4 days ago
- 5 days ago
- 6 days ago

Figure 4.2: *Noisy blocks in a news Web page from The Australian*

Figure 4.2 is an example of noisy blocks within a news page of the Web site of the famous newspaper "The Australian".

The cleaning of the page is based on the following observation: "*In a typical commercial Web site, Web pages tend to follow some fixed layouts or presentation styles as most pages are generated automatically. Those parts of a page whose layouts and actual contents (i.e., texts, images, links, etc) also appear in many other pages in the site are more likely to be noises, and those parts of a page whose layouts or actual contents are quite different from other pages are usually the main contents of the page.*"

4.4.2 The style tree

Basing oneself on this observation, the three research workers introduce a new kind of data structure: the style tree [9]. This one is based on the DOM tree structure. As seen in the Document Object Model section, a DOM tree is the representation of an HTML page where the tags are internal nodes and the detailed texts, images or hyperlinks are the leaf nodes. But this data structure is too poor to study both the presentation style and real contents of the Web pages.

The style tree, as for it, captures the common layouts (or presentation styles) and the actual contents of the pages in a Web site. This structure is able to compress the common presentation styles of a set of related Web pages.

Let us illustrate this new concept with an example (see figure 4.3).

In this example d_1 and d_2 are two DOM trees. We can observe that, except for the tags

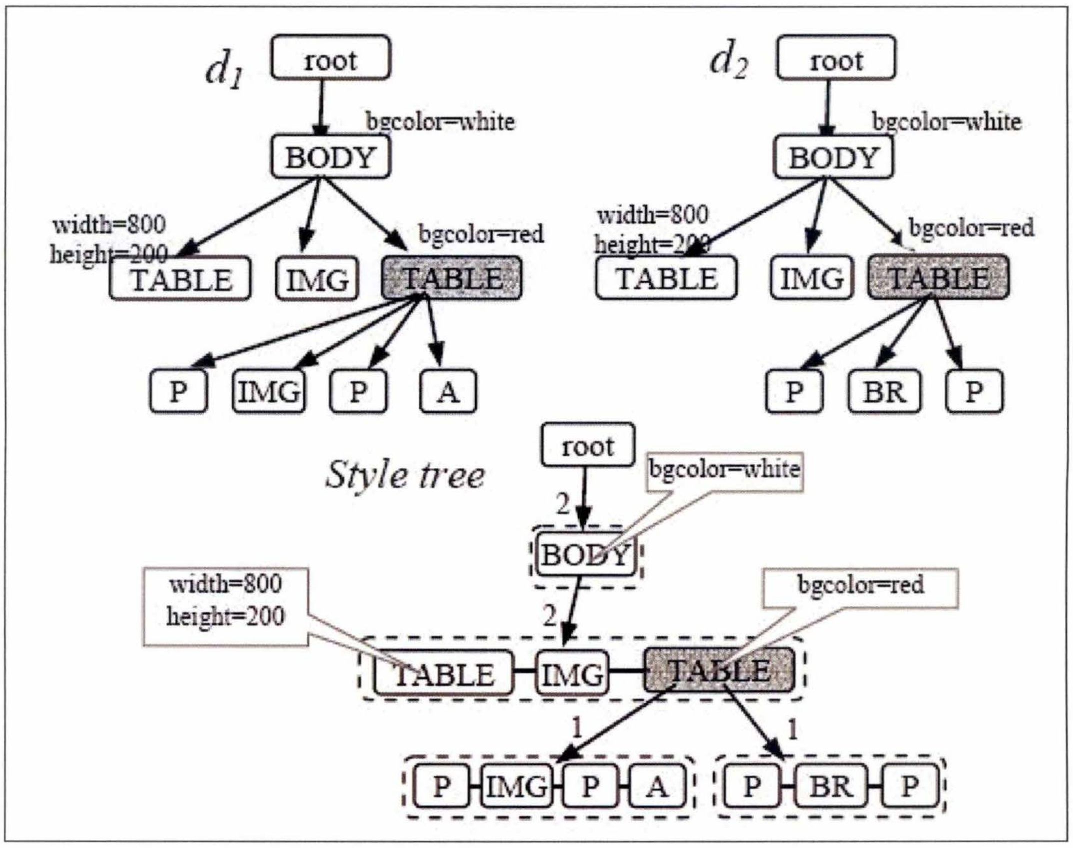


Figure 4.3: *A style tree based on two DOM trees*

at the bottom level, all the tags in d_1 have their corresponding tags in d_2. Thus, d_1 and d_2 can be compressed. A count is used to indicate how many pages have the same style at the same level of the style tree. For instance, we can see that the BODY tag is in d_1 and in d_2. BODY has thus a count of 2. The same phenomenon appears for the TABLE-IMG-TABLE tags. This sequence of identical tags is called a style node that represents a particular style at this level. A style node is thus a sequence tags node in a DOM tree. In this style node, the tag nodes are called element nodes, distinguishing them from the tag nodes in the DOM tree.

Below the right most TABLE element node, we can see that d_1 and d_2 diverge. We have two different style nodes: P-IMG-P-A and P-BR-P. The count is set to 1 for each different style node.

Clearly, the style tree is a compressed representation of both DOM trees. It enables us to see which parts of the DOM trees are common and which parts are different.

Thanks to this example we can now construct a formal definition of a style tree: a style tree consists of two types of nodes, namely, style nodes and element nodes.

- *A style node (S)* represents a layout or presentation style, which has two components, denoted by (Es, n), where Es is a sequence of element nodes (see below), and n is the number of pages that has this particular style at this node level.
  In figure 4.3, the style node P-IMG-P-A has 4 element nodes, P, IMG, P and A, and $n = 1$.
- *An element node (E)* has three components, denoted by $(TAG, Attr, Ss)$, where TAG is the tag name, $Attr$ is the set of display attributes of TAG and Ss is a set of style nodes below E.
  On Figure 4.3, "TABLE" is a tag name, the display attribute of TABLE is bgcolor $=$ RED and Ss are the styles nodes below TABLE.

4.4.3 The site style tree

The next step in this approach of the style tree is to build a site style tree (SST) for the pages of a whole Web site [9]. For this, a DOM tree for each page is built and then merged into a style tree in a top-down fashion. At a particular element node E in the

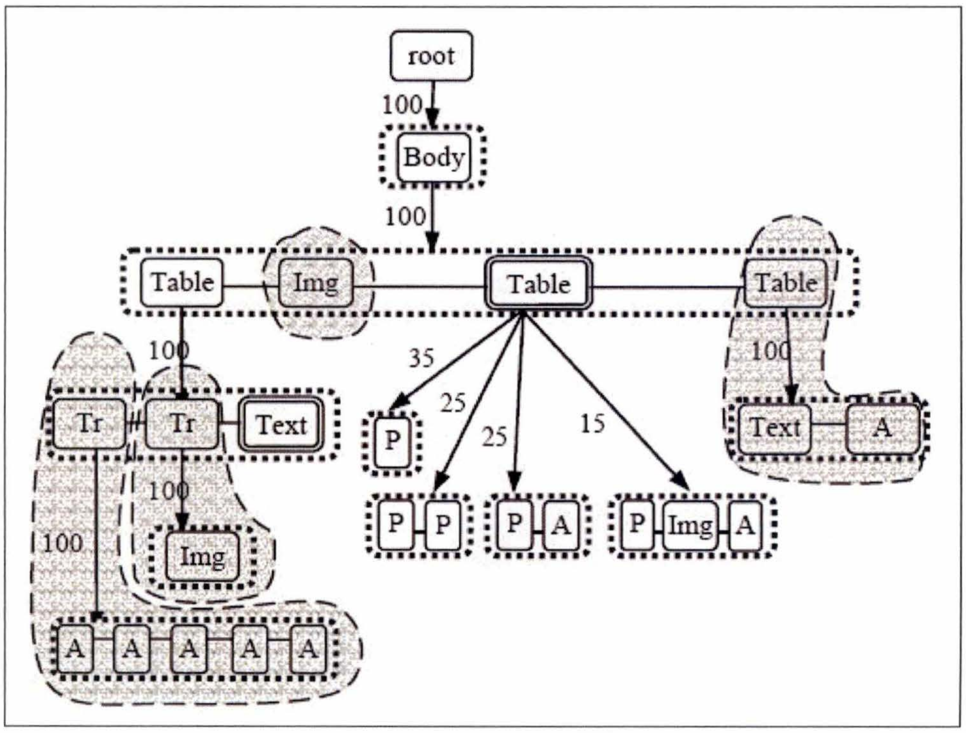


Figure 4.4: *An example of site style tree*

style tree, which has the corresponding tag node T in the DOM tree, we check whether the sequence of child tag nodes of T in the DOM tree is the same as the sequence of element nodes in a style node S below E (in the style tree). If the answer is yes, we simply increment the page count of the style node S, and then go down the style tree and the DOM tree to merge the rest of the nodes. If the answer is no, a new style node is created below the element node E in the style tree. The sub-tree of the tag node T in the DOM tree is copied to the style tree after converted to style nodes and element nodes of the style tree.

4.4.4 Presentation importance and composite importance

The definition of noise is based on the following assumptions:

- The more presentation styles that an element node has, the more important it is, and vice versa.
- The more diverse that the actual contents of an element node are, the more

important the element node is, and vice versa.

The importance of an element node is given by combining its presentation importance and contents importance. The greater the combined importance of an element node is, the more likely it is the main contents of the pages. A metric is needed to measure the importance of a presentation style:

Node importance: for an element node E in the SST, let m be the number of pages containing E and l be the number of child style nodes of E (i.e. $l = |E.Ss|$), the node importance of E, denoted by $NodeImp(E)$, is defined by

$$NodeImp(E) = \begin{cases} -\sum_{i=1}^{l} p_i \, log_m \, p_i & if \ m > 1 \\ 1 & if \ m = 1 \end{cases}$$

where p_i is the probability that a Web page uses the i^{th} style node in $E.Ss$.

Intuitively, if the number of child style nodes (l) is small, the possibility that E is presented in different styles is small. Hence the value of NodeImp(E) is small. If E contains many presentation styles, then the value of NodeImp(E) is large. In the example of figure 4.4, the importance of the element node BODY is 0 ($l \, log_{100} \, l = 0$) since $l = 1$. That is, below BODY, there is only one presentation style Table-Img-Table-Table. The importance of the double-lined TABLE is $-0.35 \, log_{100} \, 0.35 - 2 * 0.25 \, log_{100} \, 0.25 - 0.15 \, log_{100} \, 0.15 = 0.292 > 0$

But in this calculation, we do not consider the descendents of the element nodes. For example, we cannot say that BODY is a noisy element by considering only its node importance. We thus need another measure that takes into account the importance of an element node and its descendents. For this, we must add to the node importance, the average of the node importance of its descendents, multiplied by an attenuating factor. This sum is called the *composite importance* of a node.

While the presentation of node is important, we also need its contents importance. That is the reason why the composite importance of a leaf element node based on the information is defined in its actual contents (i.e. texts, images, links, etc.). Here is a more formal definition:

Composite importance: for a leaf element node E in the SST, let l be the number of features (i.e. words, image files, link references, etc.) appeared in E and let m be the number of pages containing E, the composite importance of E is defined by

$$CompImp(E) = \begin{cases} 1 - \frac{\sum_{i=1}^{l} H(a_i)}{l} & if\ m > 1 \\ 1 & if\ m = 1 \end{cases}$$

where a_i is an actual feature of the contents in E. $H(a_i)$ is the information entropy of a_i within the context of E,

$$H(a_i) = -\sum_{j=1}^{m} p_{ij}\, log_m\, p_{ij}$$

where p_{ij} is the probability that a_i appears in E of page j.

4.4.5 Noise definition

Now we have given definitions for node importance and composite importance, we are able to give a definition of *noise*: For an element node E in the SST, if all of its descendents and itself have composite importance less than a specified threshold t, then we say element node E is *noisy*.

Another definition is important, the one of meaningful: If an element node E in the SST does not contain any noisy descendent, we say that E is *meaningful*.

4.4.6 Web page cleaning

All the definitions having been given, the cleaning process can be launched (see figure 4.5). Given a Web site, the system first randomly crawls a number of Web pages from the Web site and builds the site style tree based on these pages. Sometimes it is impossible to crawl whole sites because they are too large. That is why the process does it by a random way. Once done, the process calculates the composite importance of each element node in the SST and finds the maximal noisy nodes and maximal meaningful nodes. Then it maps the DOM tree of the page to the SST, and depending on where

```
1:  Randomly crawl $k$ pages from the given Web site $S$
2:  Set null SST with virtual root $E$ (representing the root);
3:  for each page $W$ in the $k$ pages do
4:      BuildPST(W);
5:      BuildSST($E$, $E_w$)
6:  end for
7:  CalcCompImp($E$);
8:  MarkNoise($E$);
9:  MarkMeaningful($E$);
10: for each target Web page $P$ do
11:     $E_p$ = BuildPST($P$)        /* representing the root */
12:     MapSST($E$, $E_p$)
13: end for
```

Figure 4.5: *The whole process of the Web page cleaning*

each part of the DOM tree is mapped to the site style tree, we can find whether the part is meaningful or noisy by checking if the corresponding element node in the SST is meaningful or noisy. If the corresponding element node is neither noisy nor meaningful, we simply go down to the lower level nodes. The process finally gives as result the main contents of the page after cleaning, in other words and in our case, the news article.

4.5 Rooted, ordered, labelled trees

The two previous types of tree have inspired us in our work. As beforehand said, we will use the DOM tree representation to stand for our trees.

More than a representation, the following mathematical properties of the trees will be used in order to handle quickly and efficiently these data structures:

- *Tree*: a tree is a connected undirected graph with no simple circuits.
- *Rooted tree*: a rooted tree is a tree in which one node has been designated the root, in which case the edges have a natural orientation, towards or away from the root.

- *Labelled tree*: a labelled tree is a tree where labels have been added to each node. In our case, each node will be labeled with an HTML tag name.
- *Ordered tree*: an ordered tree is a tree where the children of each internal vertex are ordered.

From now on, we will use rooted, ordered, labelled trees [10]. But to simplify, we will only use the terminology tree to speak about rooted, ordered, labelled trees.

We shall also often speak about preorder traversal. A traversal algorithm is a procedure for systematically visiting every vertex of an ordered tree. A preorder traversal is defined recursively:

- Visit root.
- Visit left subtree in preorder.
- Visit right subtree in preorder.

Other traversals like inorder and postorder are also used but we will not apply them.

Figure 4.6 is an example of these different traversals.

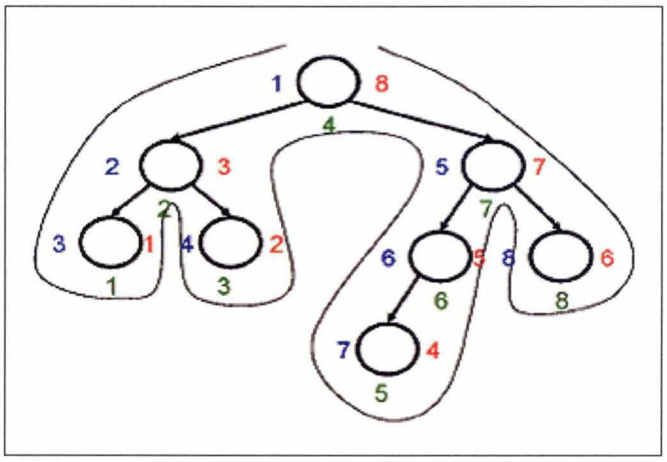


Figure 4.6: *Preorder (blue), inorder (green), and postorder (red) traversals*

4.6 Conclusion

This chapter was about the various tree structures used to handle HTML. The DOM is a complex data structure and only its logical structure was pertinent for our work. The site style tree is another sophisticated tree structure and introduces an important notion for our research: the noisy blocks. Finally we conclude this chapter by giving a definition of rooted ordered labelled tree. Now we have been briefed with these kind of data structures, we are ready to attentively examine wrappers (which sometimes use tree structures) in chapter 5.

Chapter 5

Wrappers

Contents

5.1 Introduction

We shall present from here and during the three following chapters the last step of the long development through the news extraction process. Wrappers are tools that extract data from the Internet. This chapter will introduce the most famous wrappers available in the literature. We shall begin with a set of definitions to explain what a wrapper is. As there are many wrappers in the literature, we classified them considering their common features (section 5.3). Five classes and an unclassifiable wrapper (RoadRunner) have been found. We shall present each class and its usefulness for our topic briefly. Since some tools are more suitable for news extraction, we decided to present them in separate chapters.

5.2 Definition

"Wrappers are specialized programs that identify data of interest and map them to some suitable format".[11]

"A wrapper is a piece of software that enables a semi-structured Web source to be queried as if it were a database".[12]

Let us take a random example: we could imagine a wrapper that goes everyday on a weather forecast Web site to extract the temperatures and store them on a database. The wrapper knows exactly where the temperature is encoded, for example at line 65, after the `<a>` tag. As the site is automatically generated from a database everyday, the wrapper can hope the temperature will always be present at the same place, so the Web site can be queried as if it were a database.
However, Web sites change and they change often.Nowadays, wrappers must be capable to adapt themselves to those changes; they must verify the retrieval data and automatically correct themselves if this data is erroneous.

The problem of generating a wrapper for Web data extraction can be stated as follows. Given a Web page S containing a set of implicit objects, determine a mapping W that populates a data repository R with the objects in S. The mapping W must also be able of recognizing and extracting data from any other page S' similar to S. A wrapper is a program that executes W. [11]

Notice that "similar pages" generally means pages from the same Web site, with a common layout. We shall study some wrapper generation tools whose goal is to be highly accurate, robust and as automatic as possible.

The extraction of a news article is clearly more complicated that the one of a temperature since we have to extract a big part of the whole page contents instead of a field. Furthermore, the objective is not necessarily to extract data in order to store it on a database. However, techniques utilized by wrappers to extract data can be very useful to our topic.

5.3 Classifying Web data extraction tools

There are many tools to develop wrappers available in the literature [11]. Some tools have common features allowing classifying them. Five classes have been created, based on the main technique used to generate a wrapper. Unfortunately, RoadRunner is too specific to be clustered into one of those classes.

Table 5.1 is a summary of the main features belonging to each class.

Class of Wrapper	Main features
Languages for Wrapper Development	They are not really wrappers but languages designed to assist in constructing wrappers.
Wrapper Induction tools	The wrapper is derived from a given set of examples. These tools rely on formatting features that delineate the structure of the data to extract.
HTML-tree based tools	They create a parsing tree that reflects the HTML tag hierarchy of the document in order to extract data semi-automatically.
NLP-based tools	They extract relevant information thanks to techniques from natural language processing.
Structure-based tools	They decompose (portions of) a Web page to find its inherent structure, discover nested elements and then extract semi-structured data.
RoadRunner	Unclassifiable. The only known wrapper that can be considered as fully automatic.

Table 5.1: *Classes of Wrappers*

5.3.1 Languages for wrapper development

These are languages specially designed to assist users in constructing wrappers (survey). Instead of manipulating the HTML code with languages such as Java or Perl, they proposed techniques to simplify the coding of a wrapper. Be aware that these tools do not output wrappers; they are only languages to help writing them. The best known tools are Minerva, TSIMMIS and Web-QQL.

For example,**Minerva** parses HTML files using a grammar in EBNF style. For each document, a set of productions is defined. Following, the user writes the code to access the target data in procedural programming.

Web-QQL [13] is aimed at performing SQL-like queries over the Web. It is capable of locating selected pieces of data in HTML pages. It creates a hypertree (an abstract HTML syntax tree representing the document) whose can be queried using the syntax of the language and it outputs the results in a suitable format.

Those tools are still "manual" because the user must examine the document and find the HTML tags that separate the objects of interest. In other words, the process of discovering objects boundaries is carried out manually. This is the reason why this tool can not be clustered with the tree-based tools, which are semi-automatic.

Useful for our topic ?

Those softwares are not suitable for the extraction of a news article, considering our requirements. They are designed to extract specific data, which can be accessed at a precise location. If the goal is to retrieve a single data field, for example the current temperature, Web-QQL can easily find it by a query (for instance, go to the third child at the fifth level of depth). Yet, an article is spread over several vertices in an HTML tree context; it is also spread over several productions in a syntax grammar context, so it could not be easily reconstitute by simple queries. Moreover, we want the process to be fully automatic; the user should not have to look at the HTML code to extract an article.

5.3.2 Wrapper induction tools

The wrapper induction tools generate delimiter-based extraction rules derived from a given set of training examples. They rely on formatting features that implicitly delineate the structure of the pieces of data found. These tools are really suitable for HTML documents. This approach is used by tools such as WIEN and STALKER.

WIEN is a pioneer wrapper induction tool. The user gives a set of pages where data of interest is labelled to serve as examples and WIEN returns a wrapper that works with

```
TITLE
  Begin Rule
  SkipTo(" colid " value = " " >
         <font size = + 1 > <b>)
  End Rule
  SkipTo(</b> </font> <br> by
         <a href = " /)
```

Figure 5.1: *Example of delimiter-based rules to extract the title of a book*

each labelled page. The pages are assumed to have a pre-defined structure and induction heuristics are used to generate wrappers. For example, if a page has structure HLRT (a head, a body with two colums (left, right), a tail), a HLRT wrapper is generated. These wrappers do not deal with nested structures and are not able to adapt themselves to variations.

STALKER [12] is similar to WIEN but can deal with hierarchical data extraction. It takes as input examples where data of interest has been labelled by the user. It creates a sequence of tokens representing the surrounding of the data to be extracted. STALKER generates an extraction rule that covers as many as possible of the given examples. When an example can not be covered, it generates a "disjunctive rule". The solution is a set of disjunctive rules. STALKER can handle hierarchy and nested objects. It can also verify the data extracted and automatically repair the wrapper in case of incorrect results. We shall study in chapter 6 how STALKER works and how induction tools can be useful for our topic.

5.3.3 HTML-tree based Tools

These are tools that rely on inherent structural features of HTML documents for accomplishing data extraction. These tools create a parsing tree that reflects the tag hierarchy of the document (see chapter 4). Following, extraction rules are applied to the tree to perform the extraction process. Those rules can be either automatic or semi-automatic. The most famous tools are W4F and XWRAP.

W4F (World Wide Web Wrapper Factory) is a toolkit to build wrappers:

- First the user describes how to access the document (URL). He can choose some rules to set how he wants the document to be extracted from the Web.
- Then, the document goes to a parser that constructs a DOM tree (Document Object Model).
- Second the user describes what pieces of data to extract. He can choose extraction rules for locating data into the parsing tree.
- Third, he declares which target structure to use for storing the data extracted (String list, database, etc.).

W4F features a graphic user interface and a wizard that can return a canonical path expression for a piece of information selected by the user. Apart of the creation of a tree, W4F is different from Web-QQL since the user is assisted to construct the extraction rules instead of having to write code. These rules are defined by the HTML Extraction Language (HEL). As the wizard cannot deal with a collection of items, the user who is interested in various items of the same type must manually write extraction rules that generalize the path expression provided by the wizard. This program is then considered as semi-automatic, it is not as manual as Web-QQL but it could not be classified as automatic.

XWRAP [14] is another semi-automatic tool to construct wrappers. It also has a graphic user-friendly interface and features a component library that provides basic building blocks for wrappers. Before the extraction the tool cleans up bad HTML tags and syntactical errors and then turns the document into a parsing tree. The tool operates by leading the user through a number of steps, selecting in each step proper components of its library. The user may try among six data extraction heuristics to locate data objects of interest; those heuristics are predefined to deal with several types of structuring HTML mark-ups. In the case of Web pages that match very well to the heuristics, XWRAP can extract data very efficiently. By the way, if the user is not satisfied with the extraction results, he can refine the process by specifying data types for the elements. XWRAP outputs a wrapper coded in Java for a specific source.

Useful for our topic?

As presented in chapter 1, we shall use tree structures to cluster similar pages and extract the news article, so that our program would belong to this class. However, we want our program to be fully automatic and completely transparent to the user, so that there will be no wizard neither extraction rules in our application.

5.3.4 NLP-based Tools

Natural Language Processing (NLP) is used to find and extract relevant data existing in natural language documents. These tools usually apply techniques such as filtering, part-of-speech tagging and semantic tagging to build relationships between phrases and sentences elements, so that extraction rules can be derived. Such rules are based on syntactic and semantic constraints that help to identify the relevant information within a document.

These tools are more suitable for pages with grammatical text (job listings, rental advertisements, announcements, etc). Famous tools based on this approach are RAPIER, SRV and WHISK [15]. We shall study them in chapter 8.

Useful for our topic?

These techniques are actually based on free text analysis. They can run on HTML files as well since HTML can be consider as text but this approach is completely different from the usual tactics of data retrieval from Web sites. However, these techniques must be mentioned as they are a very interesting alternative for news extraction.

5.3.5 Structure-based tools

NODOSE [16] (Northwestern Document Structure Extractor) is an interactive tool for semi-automatically determining the structure of documents that contain semistructured information and then extracting their data. The user has a graphical user interface to help him decomposing the document hierarchically; he must outline the interesting regions and describe their semantic. The decomposition process occurs in levels. For each level of decomposition, the user builds an object with a complex structure, and then decomposes it in other objects with a more simple structure.

Following, once NODOSE knows how to construct some objects, he can automatically identify other objects in the document (a mining component attempts to infer the grammar of the document from objects constructed by the user).

DEBYE [17] (Data Extraction By Example) is an interactive tool that receives as input a set of example objects taken from a sample Web page and generates extraction patterns that allow extracting objects from other similar pages. Given a target structure for objects of interest, it tries to locate in Web pages portions of data that implicitly conform to that structure.
Trough a GUI, the user takes data from the sample page and assemble what is called "nested tables". The tables assembled are examples of the object to be identified on the target pages. DEBYE generates object extraction patterns (OEP) to represent the structure and the textual surrounding of the objects to be extracted.
The extraction on a target page is done by a bottom-up algorithm, which first identifies the atomic values in the page a then tries to assemble complex objects using the structure of the OEP as a guide.

Useful for our topic ?

Once again we want our program to be fully automatic and completely transparent for the user, so that there will be no wizard neither extraction rules. Moreover, these tools are designed to understand the structure of documents, to handle and extract complex or semi-structured objects. We do not need these features to extract some text from a page; the hierarchy in the document does not matter.

5.4 RoadRunner

[18] RoadRunner could not be classified in one of the five classes presented above. RoadRunner is the only known wrapper which does not request a user intervention. It works by comparing the HTML structure of two or more given sample pages. The sample

pages must belong to the same "page class" (which means similar pages from a Web site). It generates as a result a schema for the data contained in the sample pages. A grammar is inferred from this schema, this grammar is capable of recognizing in a random Web page instances of the attributes identified in the sample pages. All the extraction process is based on an algorithm that compares the tag structure of the sample pages and generates regular expressions that handle structural mismatches found between both structures. We shall explain in detail how this algorithm works in chapter 7.

5.5 Conclusion

We saw different techniques to extract data of interest from the Internet. Most of these techniques are designed to extract data as if it were a field coming from a database, so that they are not suitable for news extraction. No one of the presented Wrappers seems to be the obvious issue to extract news articles but we could find in some classes ideas to help implement it.

Two classes and RoadRunner deserve to be analysed in details as they present features that could help us to extract news articles: We shall explain the induction tools, especially STALKER, in the next chapter. RoadRunner and its basic algorithm will be analysed in chapter 7. Finally, the NLP-based tools are introduced in greater detail in chapter 8.

Chapter 6

Induction tools

Contents

6.1 Introduction

As we saw in the previous chapter, an induction tool generates delimiter-based extraction rules derived from a given set of training examples. It does not rely on the inherent structural features of HTML documents (nested data patterns) but rather on the formatting features that delineate the data to extract. [19]

Notice that theses techniques based on the detection of the formatting features surrounding the target data assume to handle documents that contain semi-structured data, in order to extract fields as if it were a database.

We shall see that the extraction of data belongs to a larger process, offering the verification of the extracted data and the automatic repair of the wrapper in case of wrong results (section 6.2). Then we shall introduce the features of extraction rules before showing how the most famous induction tool, STALKER (section 6.4) [12], creates those rules.

We shall conclude by showing whether induction tools are suitable or not for news extraction (section 6.7).

6.2 Lifecycle of an induction wrapper

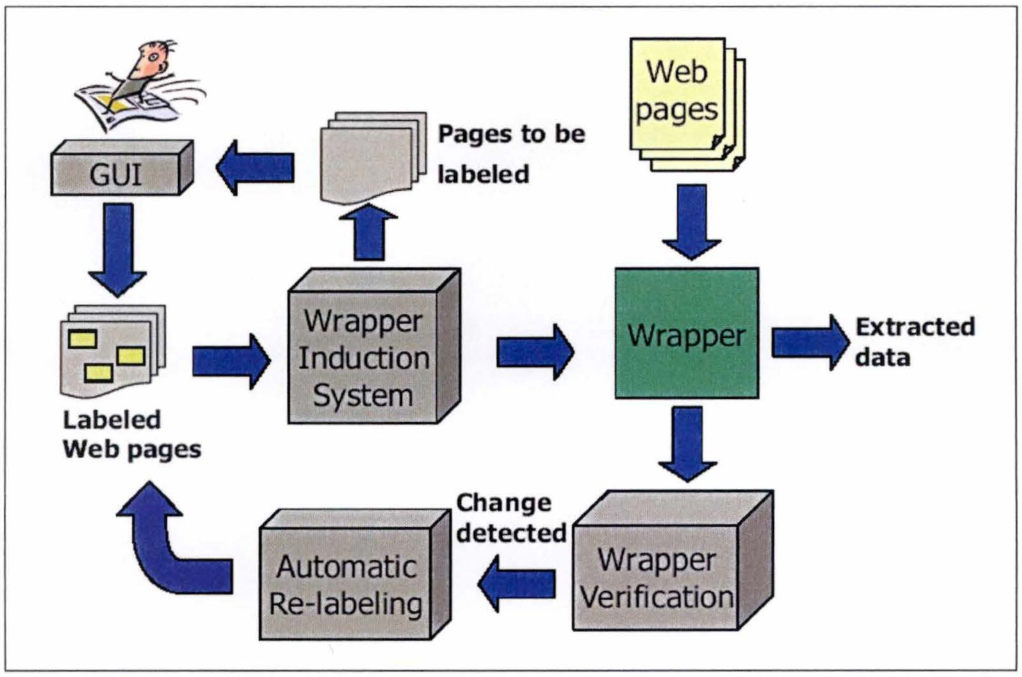


Figure 6.1: *Lifecycle of a wrapper*

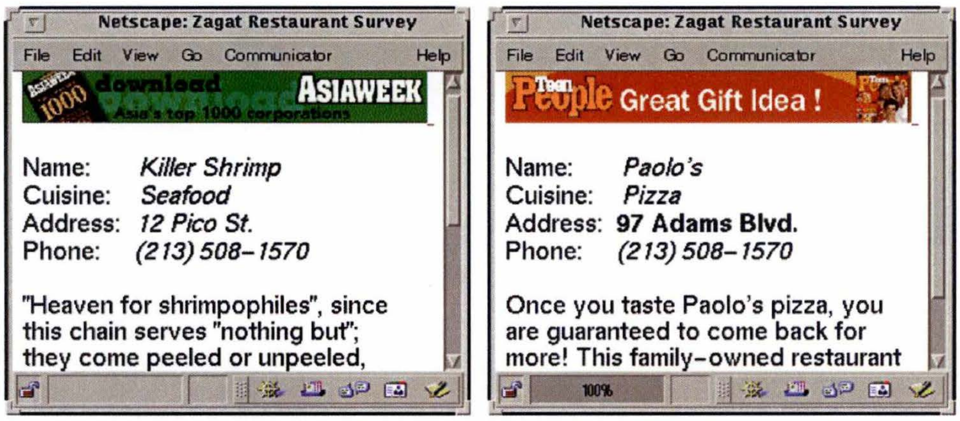
The figure 6.1 illustrates the lifecycle of a wrapper [12]:

- First the user provides the initial labelled examples using the GUI.
- The system can suggest the user to label extra pages in order to improve the accuracy of the wrapper.
- The wrapper induction system takes a set of pages labelled with examples of the data to be extracted.

- The output of the wrapper induction system is a wrapper that contains extraction rules that describe how to locate the desired information on a Web page.
- Web pages can be given to the wrapper as input and it extracts the desired data.
- A wrapper verification system uses the functioning wrapper to learn patterns that describe the new data that has just been extracted.
- If a change is detected, the system automatically repairs the wrapper: it uses those new patterns to locate examples on the changed pages (automatic re-labelling).
- The system re-runs the wrapper induction system with the new labelled pages.
- An improved wrapper is outputted.

6.3 How to locate the information on the page

Let us consider the figure 6.2 presenting restaurants from the same Web site.


Netscape: Zagat Restaurant Survey
File Edit View Go Communicator Help
download ASIAWEEK Asia's top 1000 corporations
Netscape: Zagat Restaurant Survey
File Edit View Go Communicator Help
People Great Gift Idea !
100%

Name: *Killer Shrimp*
Cuisine: *Seafood*
Address: *12 Pico St.*
Phone: *(213) 508-1570*

"Heaven for shrimpophiles", since this chain serves "nothing but"; they come peeled or unpeeled,

Name: *Paolo's*
Cuisine: *Pizza*
Address: **97 Adams Blvd.**
Phone: *(213) 508-1570*

Once you taste Paolo's pizza, you are guaranteed to come back for more! This family-owned restaurant

Figure 6.2: *Two sample pages of restaurants*

Assume that the address on the right example is in bold style because the restaurant is close to a specified address.

The HTML code of the left example contains somewhere :

```
E1:...Cuisine:<i>Seafood</i><p>Address:<i>12PicoSt.</i><p>Phone:<i>...
```

And the code of the right example looks like :

```
E4:...Cuisine:<i>Pizza</i><p>Address:<b>97 Adams Blvd.</b><p>Phone:<i>...
```

We add two others examples close to the first one:

```
E2:  ...Cuisine:<i>Thai</i><p>Address:<i>512 Oak Blvd.</i><p>Phone:<i>...
E3:  ...Cuisine:<i>Burgers</i><p>Address:<i>416 Main St.</i><p>Phone:<i>..
```

6.3.1 Extraction rules

For any given item to be extracted from a page, we need an extraction rule to locate both the beginning and the end of that item. Each HTML document is analyzed as a sequence of tokens (words, numbers, HTML tags, etc.). So we need to find the first and the last token of an item. The issue is to create a set of extraction rules that work for all of the pages in the source.

The extraction rule is based on "landmarks" (i.e. groups of consecutive tokens) that enable the wrapper to locate the start and the end of an item within a page.

Let us consider the three restaurant descriptions E1, E2, E3 presented below figure 6.2. In order to identify the beginning of the address, we can use the rule:

```
R1 = SkipTo(Address) SkipTo(<i>)
```

This means that the wrapper will start from the beginning of the document and skip each token until "Address" and ignore everything until the landmark <i>. This rule is called a start rule because it identifies the beginning of the field "Address".

The rule:

```
R2 = SkipTo (Address :  <i>)
```

is suitable as well. R2 uses a 3-tokens landmark that precedes the beginning of the address in examples E1, E2 and E3.

Let us now assume E4, in that case, the address appears in bold style because the restaurant is within one mile from a current location. We need to create an extraction rule that allows the use of disjunctions:

either `SkipTo (Address : <i>)`

or `SkipTo (Address : <b>)`

Disjunctive rules are ordered lists of individual disjuncts (i.e. decision lists). The wrapper successively applies each disjunct in the list in a straightforward process until it finds one that matches.

Notice that in this case, one could have a non disjunctive rule:

```
SkipTo (Address : _HtmlTag_)
```

6.3.2 Working with both start and end rules

All the rules presented above are forward rules (they start at the beginning of the document and stop when there is a matching between the rule and the document). One can use backward rules as well in order to improve the spotting of fields. A backward rule starts at the end of the page and goes towards the beginning. In the example the address can be found by using the following backward rules:

```
R11 = BackTo(Phone) BackTo(_Number_)
R12 = BackTo(Phone:<i>) BackTo(_Number_)
```

This technique is called co-testing: the user labelled a few examples and the system learns both a forward and a backward rule. Thanks to this technique, the wrapper can detect mistakes more efficiently and then asks the user to label new examples.

6.4 STALKER

STALKER [12] is a wrapper induction algorithm that learns extraction rules based on examples labelled by the user. A graphic user interface allows users to mark up several pages on a site, and then the system generates a set of extraction rules that accurately extract the required information. The approach uses a greedy-covering inductive learning algorithm which incrementally builds the extraction rules from the examples.

The number of examples required is rarely above ten, in many cases, when sites have been generated, based on a fixed template, two examples are enough.

STALKER can deal with complex examples because it uses the hierarchical structure of the source to decompose one difficult problem into a series of simpler ones. For instance, instead of using one complex rule to extract all restaurants names, addresses and phone numbers, STALKER decomposes the problem:

- It applies a rule that extracts the whole list of restaurants;
- it uses another rule to break the list into tuples corresponding to each restaurant;
- the name, address, etc. is finally extracted from each tuple.

> *STALKER is a sequential covering algorithm that, given the set of training examples E, tries to learn a minimal number of perfect disjuncts that cover all examples in E.* [12]

A perfect disjunct is a rule that:

- covers at least one training example;
- produces the correct result on any example where the rule matches.

Once a perfect disjunct P has been found, STALKER removes from E all examples on which P is correct, and the whole process is repeated until there are no more training examples in E.

To generate a perfect disjunct, STALKER first creates an initial set of candidate-rules C and then repeatedly applies the following three steps:

- select the most promising candidate from C;
- refine that candidate;
- add the resulting refinements to C.

STALKER uses two types of refinements:

- Landmark refinement : the rule is made more specific by adding a token to one of the existing landmarks.
- Topology refinement : adds a new 1-token landmark to the rule.

6.4.1 Example

Let us assume that we want to learn a start rule for the address at figure 6.2. STALKER selects an example to guide the search, for instance E4. It generates a set of initial candidates, which are rules of a single 1-token landmark. These landmarks are chosen so that they match the token that immediately precedes the beginning of the address. We have two initial candidates:

```
R5 = SkipTo(<b>)
R6 = SkipTo(_HtmlTag_)
```

R5 does not match with E1, E2, E3 as the `<b>` token appears only in E4. On the other hand, R6 matches in all four examples, even though it matches too early (R6 stops as soon as it finds an HTML tag, which happens in all four examples before the beginning of the address). Because R6 has a better generalization potential, STALKER selects R6 for further refinements.

R6 must be refined; STALKER creates other candidates by using a landmark refinement (a token is added to the landmark in R6)...

```
R7 = SkipTo(:_HtmlTag_)
R8 = SkipTo(_Punctuation_ _HtmlTag_)
```

...and creates other candidates by topology refinement (a new landmark is added to R6):

```
R9 = SkipTo(:)  SkipTo(_HtmlTag_)
R10 = SkipTo(Address) SkipTo(_HtmlTag_)
```

As R10 works correctly on E1, E2, E3 and E4, STALKER stops the learning process and returns R10.

6.5 Verifying the extracted data

Sites change and they change often. The wrapper must be capable of checking if the extracted data is correct. Machine learning techniques are applied to learn a set of patterns that describe the information extracted in a field labelled by the user. After having extracted data, the system can verify it by comparing its patterns to the learned patterns.

The learned patterns represent the format of the field as a sequence of words or wildcards. Wildcards are syntactic categories to which words belong (alphabetic, numeric, capitalized, etc.). In our example with addresses (12 Pico St., 512 Oak Blvd., 416 Main St. and 97 Adams Blvd), all the fields start with a pattern (_Number_ _Capitalized_) and end with (Blvd.) or (St.). The starting and ending patterns together are called the data prototype of the field. Complex algorithms (that it would be inopportune to cover in this dissertation) are designed to construct these data prototypes.

Notice that to improve the efficiency, the verification can be done randomly or every x extractions instead of checking every single data field.

6.6 Automatically repairing the wrapper

We assume here that we are coping with minor formatting changes or slight reorganizations in a page. We shall use two techniques presented above to allow the wrapper to be repaired automatically: the co-testing and the data prototype.

Let us consider figure 6.3 on the next page to help understand the process.The wrapper is used to extract four fields (Author, Title, Price, Availability). We examine the extraction rule for the "title" field: this rule uses co-testing: both forward and backward rules are applied. On the first example, the title is in bold style, the rules are defined to surround a field between the tags `<b>` and `</b>`.

On the second example, the source has changed and the title has become yellow.Thanks to the technique of co-testing, it is highly unlikely to find another field that would match both rules. Even if it occurs, the data prototype analysis should discover the mistake. In the example, both rules do not match any field in the page and the wrapper outputs "NIL".

Once the error is detected, the system takes the set of training examples (labelled by the user) and a set of new pages from the same source to identify the data field on the new pages. It learns all the starting and ending patterns that describe the field in the training examples (the data prototype). Next, each new page is scanned to identify all segments that begin with one of the starting patterns and end with one of the ending patterns. Those segments, which are called candidates, are handled as follows:

- The candidates that are significantly longer or shorter than the training examples are eliminated from the set.
- The candidates are then clustered to identify subgroups that share common features. The features of the candidate that are considered are its position on the page, its adjacent landmarks and whether it is visible or not for the user.
- Each cluster gets a score based on how similar it is to the training examples and the highest score is expected to contain the correct examples of the data field.

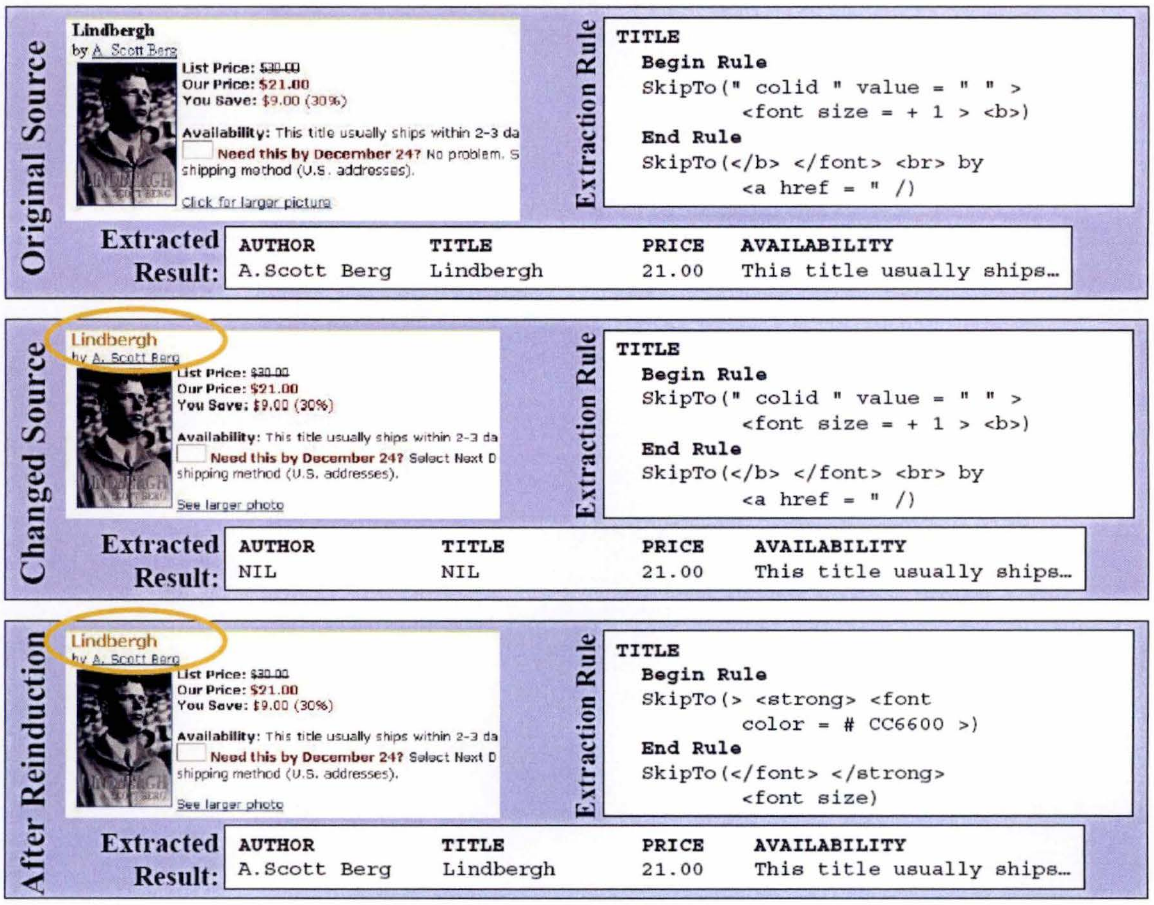
On figure 6.3, we can see the result of this automatic reinduction process, the extraction rule has been updated automatically.

6.7 How could induction tools be useful for news retrieval?

It is obvious that tools such as STALKER are not designed to retrieve a news article from a Web page. The original goal is to extract fields from a structured Web page to store them on a database. However, we can wonder if the techniques could be adapted to our topic.

The major drawback is that induction tools assume that pages have a high degree of similarity: the news articles must belong to the one Web site and must be presented with the same layout. We could imagine for example an induction wrapper that goes everyday on a definite portal to extract the headlines.

Another drawback is that it seems impossible to handle the concept of data prototype with a news article. How could we identify patterns that describe the article? This makes the difference between the extraction of a database field and a relevant text. Unfortunately, without being able of identifying a data prototype, the application is also unable to deal with data verification and automatic repairing of the wrapper.



```
TITLE
  Begin Rule
  SkipTo(" colid " value = " " >
          <font size = + 1 > <b>)
  End Rule
  SkipTo(</b> </font> <br> by
          <a href = " /)
```

Extracted Result:

AUTHOR	TITLE	PRICE	AVAILABILITY
A.Scott Berg	Lindbergh	21.00	This title usually ships...

```
TITLE
  Begin Rule
  SkipTo(" colid " value = " " >
          <font size = + 1 > <b>)
  End Rule
  SkipTo(</b> </font> <br> by
          <a href = " /)
```

Extracted Result:

AUTHOR	TITLE	PRICE	AVAILABILITY
NIL	NIL	21.00	This title usually ships...

```
TITLE
  Begin Rule
  SkipTo(> <strong> <font
          color = # CC6600 >)
  End Rule
  SkipTo(</font> </strong>
          <font size)
```

Extracted Result:

AUTHOR	TITLE	PRICE	AVAILABILITY
A.Scott Berg	Lindbergh	21.00	This title usually ships...

Figure 6.3: *An example of the reinduction process*

Positively, the programming of the extraction rules is rather easy and the processing is efficient. Though, if we define extraction rules that fetch the contents between the tag just before the beginning of the news article and the tag following the end of the article, we shall be likely to extract some HTML code that characterizes the layout of the article. We should then clean up the article by removing the remaining HTML tags after the extraction.

Finally, we want our process to be fully automatic and independent from the source. We saw that induction tools need examples labelled by the user and work only on similar pages, so that they are unfortunately unconvincing considering our topic.

6.8 Conclusion

We showed how to extract data thanks to induction tools, specially STALKER. We saw that such tools are unsuitable for extracting large pieces of text since they are designed to fetch fields as if the Web page were a database. Moreover, the techniques concerning the output verification and the automatic repair that were very attractive are not apt to work in the case of our topic.

The next chapter will introduce RoadRunner, a wrapper that seems to be more appropriate for extracting news articles.

Chapter 7

RoadRunner

Contents

7.1 Introduction

RoadRunner [18] is one of the most powerful wrappers. In the literature, it is presented as the only known wrapper to be fully automatic [11]. Since we want our program to be fully automatic, it is interesting to briefly examine how RoadRunner works.

RoadRunner finds a regular grammar that represents the HTML code for a set of HTML pages (samples) and then uses this grammar to parse other pages and extract pieces of data.

We shall present why RoadRunner is considered as unique in the literature. Then we shall have to introduce some definitions and concepts in order to explain the algorithm. The presented algorithm is actually a simplified version of the original one. It is comprehensive enough to understand how RoadRunner works. Finally, we shall see how RoadRunner could be useful for the news extraction.

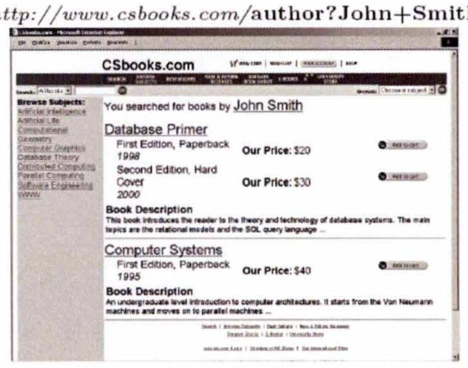
7.2 Why is RoadRunner better?

- First, the authors' goal is that of *"fully automating the extraction process, in such a way that it does not rely on any a priori knowledge about the target pages and their contents"* [18]. It does not need user-specified examples neither any interaction with the user during the wrapper generation process.
- Second, it is usually assumed that a wrapper induction system has some a priori knowledge about the page organization. Most wrappers assume that the sample pages contain a collection of flat records. In other cases, when a wrapper works by searching nested data, it needs to know how the data are nested and the attributes to extract. RoadRunner has no a priori knowledge about the page contents, which means it does not know how the data is organized in the HTML page. Moreover, RoadRunner is not restricted to flat records or nested structures but can handle both.
- Finally, most of tools generate a wrapper by examining one HTML page at a time. RoadRunner works with two pages at a time. The discovery of a common pattern is based on the study of similarities and dissimilarities between the pages; mismatches are used to identify relevant structures.

7.3 Context, definitions and assumptions

The main intuition is that the site generation can be seen as an *encoding* process of database contents into strings of HTML code and the data extraction can be seen as the *decoding* process. Pages in data intensive sites are usually automatically generated. These pages are produced by programs which query databases to generate a source dataset; that is serialized into HTML code to produce the actual pages (notwithstanding the introduction of banners, images, links, etc.). A source dataset is thus *a set of tuples of a possibly nested type that will be published in the site pages.*

*http://www.csbooks.com/*author?John+Smith



CSbooks.com

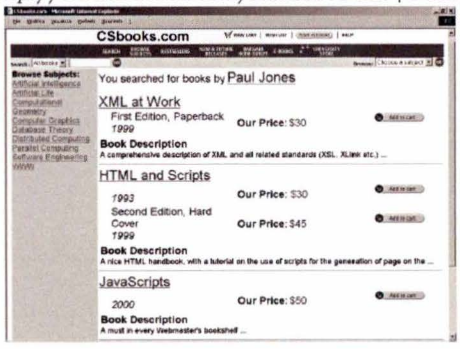
Browse Subjects:
Artificial Intelligence
Artificial Life
Computational Geometry
Computer Graphics
Database Theory
Distributed Computing
Parallel Computing
Software Engineering
WWW

You searched for books by John Smith

Database Primer
First Edition, Paperback *1998* Our Price: $20
Second Edition, Hard Cover *2000* Our Price: $30
Book Description
This book introduces the reader to the theory and technology of database systems. The main topics are the relational models and the SQL query language ...

Computer Systems
First Edition, Paperback *1995* Our Price: $40
Book Description
An undergraduate level introduction to computer architectures. It starts from the Von Neumann machines and moves on to parallel machines ...

*http://www.csbooks.com/*author?Paul+Jones



CSbooks.com

Browse Subjects:
Artificial Intelligence
Artificial Life
Computational Geometry
Computer Graphics
Database Theory
Distributed Computing
Parallel Computing
Software Engineering
WWW

You searched for books by Paul Jones

XML at Work
First Edition, Paperback *1999* Our Price: $30
Book Description
A comprehensive description of XML and all related standards (XSL, XLink etc.) ...

HTML and Scripts
1993 Our Price: $30
Second Edition, Hard Cover *1999* Our Price: $45
Book Description
A nice HTML handbook, with a tutorial on the use of scripts for the generation of page on the ...

JavaScripts
2000 Our Price: $50
Book Description
A must in every Webmaster's bookshelf ...

Figure 7.1: *Examples of automatically generated web pages*

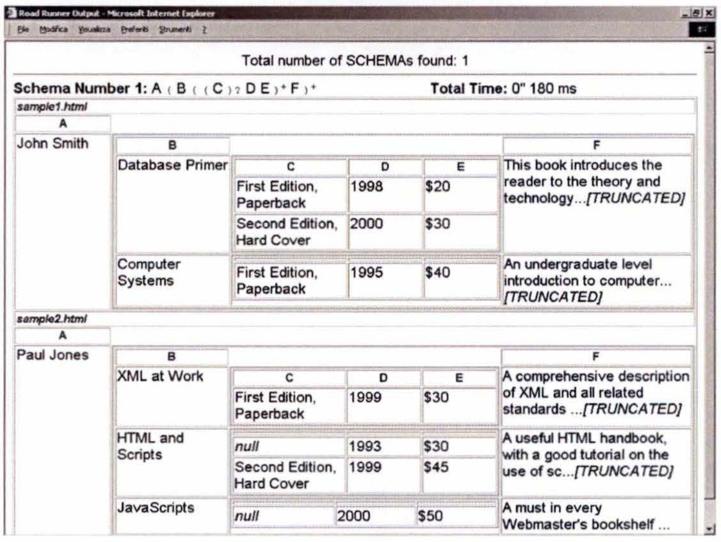
The problem can be summarized as follows: given a set of similar pages as sample, find the nested type of the source dataset and extract it. For example, let us consider figure 7.1 from CSBooks.com. The pages have been generated from a script that queries a database to produce a nested dataset in which each tuple contains data about one author, his list of books and for each book a list of editions.

Roadrunner will compare the HTML code of the two pages, infer a common structure and a wrapper and use that to extract the source dataset. Figure 7.2 presents the extracted dataset in an HTML page to help understanding but is more likely to be useful in a database. We can see the nested structure from the pages. The fields are anonymous (here labelled A, B, C, D, etc.); they must be renamed manually after the dataset has been extracted.

7.4 Data extraction in RoadRunner

7.4.1 Introduction

RoadRunner is a very complex application. We shall only show the main ideas of the process that creates the grammar used to extract the data. The example below seems to be too easy and is actually designed on purpose to explain the major concepts. For an extraction on real web pages, RoadRunner uses powerful algorithms that can handle situations far more complicated than the ones in the example. This explanation aims



Total number of SCHEMAs found: 1

Schema Number 1: $A\ (\ B\ (\ (\ C\ )?\ D\ E\ )^+\ F\ )^+$ **Total Time:** 0" 180 ms

sample1.html

A	B	C	D	E	F
John Smith	Database Primer	First Edition, Paperback	1998	$20	This book introduces the reader to the theory and technology...*[TRUNCATED]*
		Second Edition, Hard Cover	2000	$30	
	Computer Systems	First Edition, Paperback	1995	$40	An undergraduate level introduction to computer... *[TRUNCATED]*

sample2.html

A	B	C	D	E	F
Paul Jones	XML at Work	First Edition, Paperback	1999	$30	A comprehensive description of XML and all related standards ...*[TRUNCATED]*
	HTML and Scripts	*null*	1993	$30	A useful HTML handbook, with a good tutorial on the use of sc...*[TRUNCATED]*
		Second Edition, Hard Cover	1999	$45	
	JavaScripts	*null*	2000	$50	A must in every Webmaster's bookshelf ...

Figure 7.2: *Output of RoadRunner*

at giving a notion of how an HTML-aware wrapper (based on a grammar and fully automatic) works.

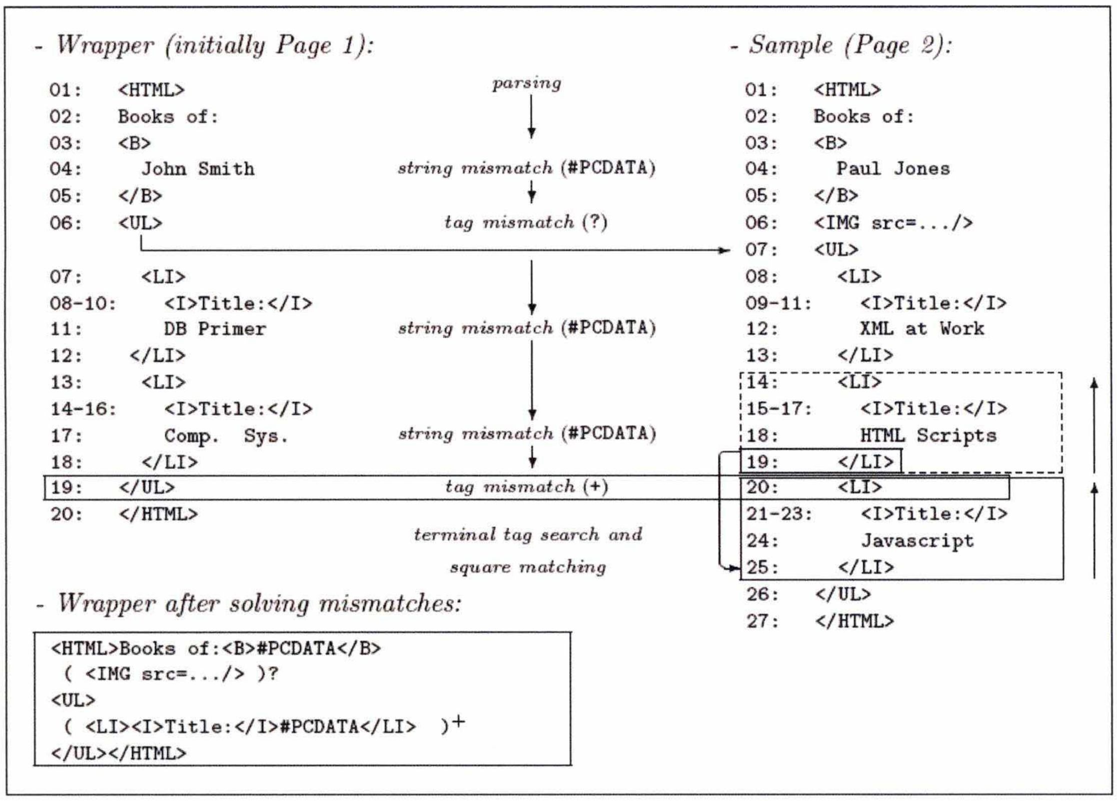
7.4.2 Before executing the algorithm

To avoid errors in the sources due to missing tags, the HTML file is transformed into an XHTML file (a restrictive variant of HTML where all tags have to be properly closed and nested). Several tools are available on the Internet to do it. Sources must also be pre-processed by a lexical analyzer to transform them into lists of tokens, each token is either a HTML tag or a string value.

An initial version of the wrapper is defined from the first page (see figure 7.3, left column). At this step, the wrapper is exactly the same as the list of tokens representing the first page (normal since the wrapper defines the regular grammar of one page only).

7.4.3 ACME algorithm

The algorithm is based on a matching technique called ACME, for Align, Collapse under Mismatch, and Extract.



```
<HTML>Books of:<B>#PCDATA</B>
 ( <IMG src=.../> )?
<UL>
 ( <LI><I>Title:</I>#PCDATA</LI> )+
</UL></HTML>
```

Figure 7.3: *ACME Algorithm*

The algorithm works on two objects at a time: the wrapper and the sample (another similar HTML page transformed into a list of tokens). The wrapper is progressively refined: the algorithm tries to find a common regular expression for both pages by solving *mismatches* between the wrapper and the sample.

The algorithm parses the sample using the wrapper, a mismatch happens when some token in the sample does not comply with the grammar specified by the wrapper. Mismatches are important because they generalize the wrapper when they can be solved. The algorithm succeeds if a common wrapper can be generated by solving all mismatches encountered during the parsing. As seen above, the wrapper is thus a grammar that represents the HTML code for a set of HTML pages.

During the parsing, two kinds of mismatches can occur: "String mismatch" or "Tag mismatch".

String mismatch

A String mismatch means the discovery of a field. Since the pages are similar (belong to the same site), string mismatches may be due only to different values of a database field. In figure 7.3, a string mismatch occurs between the names of the authors at line 04. The wrapper which initially equals Page 1 is generalized by replacing the string 'John Smith' by the regular expression #PCDATA. The same thing occurs between the titles of the books (DB Primer - XML at Work). #PCDATA is thus a code that means a possible field in the original database has been found.

Tag mismatch

Two options are possible when a tag mismatch occurs. The algorithm can discover an optional tag, a tag that is present only in one of both pages, in figure 7.3 an image of the author is only available on page 2 (line 6). The algorithm can also discover an *iterator*. An iterator is a repeated pattern, in the example, the author on page 1 has two books while the one on page 2 has three books, the tag mismatch is thus due to an extra instance of the nested structure "book".

Let us suppose first that the search for an iterator has failed, we may assume that the tag mismatch is due to the presence of optionals. This means that, either on the wrapper or on the sample we have a piece of HTML code that is not present on the other page. By skipping this piece of code, the parsing can be resumed. This is done in two steps.

1. Assuming the sample has the optional expression, a cross search is done through the wrapper to be sure that the expression is not present elsewhere.
2. The wrapper is generalized by introducing one pattern "( )?". In our example the line `(<IMG src=.../>)?` is added to the wrapper.

The parsing is resumed, taking into account the difference due to the optional tag. In the example, token 6 in the wrapper will be compared to token 7 in the sample and so on.

Let us now consider the iterator mismatch: A tag mismatch occurs between lines 19 and 20 because a third book appears in the sample. The pattern `<LI><I>Title:</I>-`

`#PCDATA</LI>` is repeated. The algorithm needs to identify these patterns that we shall call squares to generalize the wrapper accordingly. This is done in three steps:

1. *Square location by Terminal-Tag Search*: We know that both the wrapper and the sample contain at least one occurrence of the square (otherwise it would be an optional pattern). Let us call o_w and o_s the number of occurrences of the square (2 and 3 in the example). Before encountering the mismatch, the occurrences have matched each other $\min(o_w, o_s)$ times. So we can identify the last token of the square as the token just before the mismatch position. Now we have the terminal tag of the square, we must find the initial tag. When the mismatch occurs (line 19 on the wrapper), we know that it means the end of the list of squares on one sample and the beginning of a new square on the other one but we do not know which one of the samples has got the longest list. We take the tag that follows the terminal tag on both samples, so we have two possibilities for the initial tag: `</U>` or `<LI>` in the example, which means a square of the form `</UL>...</LI>` or `<LI>...</LI>`. To discover the good square, we look forward on the wrapper and the sample to find an occurrence of the terminal tag `</LI>`. It fails on the wrapper and succeeds on the sample, so there is one candidate occurrence of the square at tokens 20 to 25.

2. *Square matching*: To check if the candidate really represents the square, we try to match it with some upward portion in the sample. The process is done backwards; it starts to match tokens 25 and 19, then 24 and 18 and so on. The search succeeds since there is a correct matching between the square (lines 14 to 19) and the candidate (lines 20 to 25)

3. *Wrapper generalization*: The wrapper can now be generalized by replacing the repeated occurrences of the square by (square)$^+$. As you can see at the bottom of figure 7.3, the square book is noted

   `(<LI><I>Title:</I>#PCDATA</LI>)`$^+$

7.5 Useful for our topic ?

Once again, the algorithm has been intended for extracting fields between HTML tags instead of a large and structured text. The algorithm is fully automatic but it is obvious that it works quite better if the pages given as input are similar. If the pages are coming from different sources, it is likely to produce a useless wrapper after having solved mismatches.

A possible application would be, after having clustered the Web sites to get similar pages, to run the RoadRunner wrapper and to fetch all the #PCDATA fields. As seen above, a #PCDATA field means the discovery of a "string mismatch".
Then we could assume that the longest fields contain the text corresponding to the news article. By comparing large fields close to each others, we could reconstruct the article.

Notice that this issue is smarter than to remove all the HTML tags and fetch all the longest parts of text to reconstruct the article because in that case, we do not take into account the contents between HTML tags that is common to both analysed pages. To our opinion, we can easily assume that two news articles are not likely to have common sentences, especially if they come from randomly chosen pages.

Once the first two news articles have been extracted, the wrapper can be improved by locating in the grammar the set of #PCDATA fields containing the article, so that all the #PCDATA fields before the beginning and after the end of this set can be ignored.

The major drawback is the necessity of analysing a lot of similar pages before the grammar becomes fair enough to extract the text correctly. The creation of the grammar is a long process that is better as the number of samples grows. RoadRunner is designed to be useful and efficient when the goal is to extract data from a large amount of similar pages.

Hence, this technique could be useful only if we face a large set of similar pages. Since our process must be fully automatic, a clustering of the different Web pages is required before the extraction process starts. The latter could be adapted to the size of each cluster, so that the extraction from large clusters could be improved by using tools analogous to RoadRunner. Anyway, we chose to apply the same extraction process notwithstanding the amount of Web pages to handle or the size of the clusters.

7.6 Conclusion

Although RoadRunner is considered as fully automatic, we saw that it is likely to output relevant results only if the Web pages given as input are similar. We can conclude that there is an obvious necessity for a clustering of similar pages before executing almost any wrappers available in the literature. One single class of wrappers could work without that constraint: the Natural Language Processing-based tools. We shall study three famous wrappers belonging to this class in the next chapter.

Chapter 8

NLP-based methodologies

Contents

8.1 Introduction

In this chapter we shall study another technique to extract relevant data : the NLP-based method. A context briefs on its interest in our work and is followed by the information extraction section. Then the natural language processing and its relation with the artificial intelligence is mentioned. An NLP-based tool (Autoslog [15]) working on grammatical text is analysed and three others (Rapier, SRV, WHISK) working on ungrammatical text as well. We close the chapter by a short conclusion.

8.2 Information extraction

Information extraction [20] [21] is the mapping of natural language texts (such as newswire reports, newspaper and journal articles, electronic mail, World Wide Web pages, any textual database, etc.) into predefined, structured representation, or templates, which, when filled, represent an extract of key information from the original text. The information concerns entities of interest in the application domain (e.g. companies or persons), or relations between such entities, usually in the form of events in which the entities take part (e.g. company takeovers, management successions etc.). Once extracted, the information can then be stored in databases to be queried, data mined, summarised in natural language, etc.

A trivial example of information extraction is to find the perpetrator of a terrorist attack reported in the newspapers :

The Parliament was bombed by the guerrillas

The aim of the information extraction is to find the perpetrator (the guerrillas) giving this sentence from a news article. Or to identify the targets of this terrorist attack (the Parliament), etc.

The identification of such entities is done by the set of extraction patterns (or extraction rules) that is used to extract from each document the information relevant to a particular extraction task. But the use of such rules is a difficult and time-consuming task. A lot of various techniques are used in field IE because information extraction is actually a real hard task. A single technique is only suitable for a few defined specialised problems in IE. In general, there is no common solution for the total problem fields of information extraction. Currently, researchers try to use almost all artificial intelligent methods and machine learning algorithms to achieve high performance and automatic information extraction from documents. Advanced methods and algorithms such as bayesian model, Hidden Markov Model (HMM), Decision Tree, etc. use the NLP technology because it is one of the most basic techniques.

8.3 Natural Language Processing

NLP was initially used for machine translation, speech recognition and also knowledge representation. The basic idea of using NLP in information extraction is analysing the grammatical structure of a sentence and then constructing grammatical rules for some useful information within this sentence. The rules based on syntactic and semantic constraints are then applied to identify the relevant information to extract. Relationship between phrases and sentence elements are built by various techniques such as automated filtering, part-of-speech tagging and lexical semantic tagging to derive the extraction rules.

"*NLP techniques can be considered as an automated generalised indexing procedure that extracts from the full textual contents of the document linguistically significant structures.*"

With NLP, the text (e.g. a news article) is broken into tokens. It is then possible to identify sentences. Within the sentences we can determine context of words and phrases using various dictionaries and domain specific lexicons. Actually, NLP techniques are more suitable for information written in grammatical text or in telegraphic style such as job listings, apartment rental advertisements, etc.

The two following sections present various applications of NLP. The first one shall explain the different techniques used to identify useful information within a grammatical text. The second one shall examine various representative tools applied in online documents, mostly in the form of ungrammatical text.

8.4 Grammatical text

In this section we assume that the document on which the NLP tool works contains a grammatical text, that is a plain text.

Various NLP tools have been created in this case : AutoSlog, LIEP, PALKA, CRYSTAL, CRYSTAL + Webfoot and HASTEN. Describing precisely all these applications is beyond the scope of this thesis but we shall nevertheless present AutoSlog to show the main ideas of these NLP tools.

8.4.1 Autoslog

This NLP tool builds a dictionary of extraction patterns that are called *concepts*. To each concept is associated a conceptual anchor, actually a triggering word. A set of enabling conditions represent constraints on the concepts. Let us take our trivial example about a terrorist attack on the Parliament :

The Parliament was bombed by the guerrillas

To extract the target of the terrorist attack (the Parliament), the triggering word is the verb *bombed* and forms with the *linguistic pattern "subject—passive-verb"* the concept we are analysing. Now we have thus defined *an extraction pattern* (or *extraction rule*). Applying this rule gives us the possibility to activate the concept because the pattern contains the trigger *bombed* and thanks to the linguistic pattern, the subject of the sentence is found as the target of the terrorist attack. Figure 8.1 is the representation of this example.

CONCEPT NODE:

Name:	target-subject-passive-verb-bombed
Trigger:	bombed
Variable Slots:	(target (*S* 1))
Constraints:	(class phys-target *S*)
Constant Slots:	(type bombing)
Enabling Conditions:	((passive))

Figure 8.1: *An AutoSlog extraction task*

The Name slot is a human readable description of the concept, the *Trigger* is the conceptual anchor and the *Variable slot* is what we are looking for. The subject must be a physical target (*Constraint slot*) and the verb must be used with its passive form (*Enabling conditions*).

Thanks to this example, we can see that AutoSlog applies syntactical and semantic rules to find the relevant information within a plain text. But AutoSlog determines only the syntactic field that contains the target phrase while other NPL tools identify the exact phrase of interest. AutoSlog is single-slot, that is, it can only find one information of interest at a time, while others are multi-slot and can find, in our trivial instance, the target and the perpetrator. AutoSlog is one of these NLP-based tools having their advantages and disadvantages. As a result, it is very important to analyse the problem beforehand and to apply the NPL tool suiting to this particular problem.

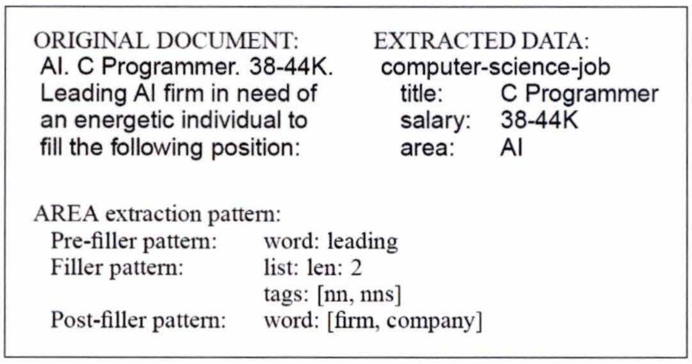
8.5 Ungrammatical text

In the previous section, we were interested in using NLP-based tools on plain texts. Let us see now the application of NLP tools on ungrammatical texts, especially those coming from the World Wide Web. The information we read on many Web pages are often presented in a mixture of grammatical, telegraphic, and/or ungrammatical texts. For example, job postings, apartment rentals, etc. are usually written in informal text and so it is quite difficult to syntactically and semantically parse the text. Moreover, the various techniques used in the previous paragraph do not fit for these online documents. Syntactic and semantic constraints are still applied but *delimiters* are introduced to bound the text to be extracted. Particular NLP methods such as RAPIER, SRV and WHISK have been created to extract correctly this special kind of information.

8.5.1 RAPIER

Robust Automated Production of Information Extraction Rules (RAPIER) [15] takes as input a document and a filled template, used to learn extraction pattern, indicating the data to be extracted. The RAPIER system uses three distinct slots : Pre-filler, filler and Post-filler patterns. The Pre and Post play the role of left and right delimiters, respectively, while the filler describes the structure of the data to be extracted. Figure 8.2 shows an intuitive example of extraction.

In this example, the Pre-filler pattern means that the information to be extracted is immediately preceded by the word *leading* and immediately followed by the words *firm* or *company* (Post-filler pattern). The filler pattern imposes constraints on the structure of the data to be extracted.

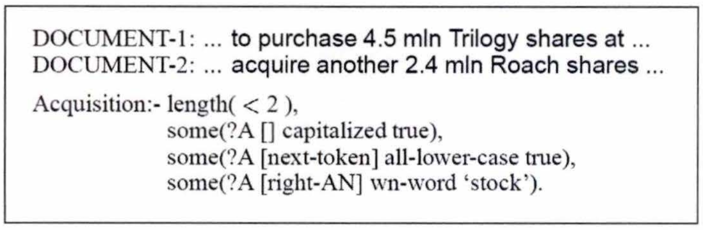


```
ORIGINAL DOCUMENT:              EXTRACTED DATA:
 AI. C Programmer. 38-44K.       computer-science-job
 Leading AI firm in need of        title:     C Programmer
 an energetic individual to        salary:    38-44K
 fill the following position:      area:      AI

AREA extraction pattern:
 Pre-filler pattern:     word: leading
 Filler pattern:         list: len: 2
                         tags: [nn, nns]
 Post-filler pattern:    word: [firm, company]
```

Figure 8.2: *A RAPIER extraction task*

8.5.2 SRV

This NLP tool takes as input a set of tagged documents, and a set of features that control generalization, and produces rules that describe how to extract information from novel documents. It generates extraction patterns that are based on attribute-value tests and the relational structure of the documents [22]. Figure 8.3 gives an instance of extraction task.



```
DOCUMENT-1: ... to purchase 4.5 mln Trilogy shares at ...
DOCUMENT-2: ... acquire another 2.4 mln Roach shares ...
Acquisition:- length( < 2 ),
              some(?A [] capitalized true),
              some(?A [next-token] all-lower-case true),
              some(?A [right-AN] wn-word 'stock').
```

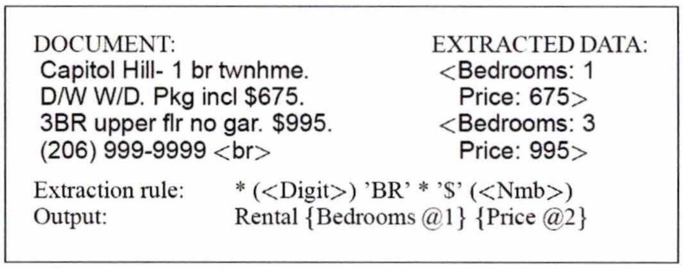
Figure 8.3: *A SRV extraction task*

This example shows how to extract the name of a company that was the target of an acquisition process. The first two predicates of the extraction rule mean that the company name consists of a single and capitalized word while the third predicate means that the company name is followed by a lower-case word. The last predicate rules means that the company name is followed by a word associated with *stock*.

8.5.3 WHISK

WHISK [15] fits for online documents as well as for plain texts. It is a general rule extraction system which learns regular expressions as extraction patterns. These patterns are a special type of regular expressions that have two components: one that describes the context that makes a phrase relevant, and one that specifies the exact delimiters of the phrase to be extracted.

Figure 8.4 shows a sample extraction task:



```
DOCUMENT:                          EXTRACTED DATA:
 Capitol Hill- 1 br twnhme.         <Bedrooms: 1
 D/W W/D. Pkg incl $675.              Price: 675>
 3BR upper flr no gar. $995.        <Bedrooms: 3
 (206) 999-9999 <br>                  Price: 995>

Extraction rule:    * (<Digit>) 'BR' * '$' (<Nmb>)
Output:             Rental {Bedrooms @1} {Price @2}
```

Figure 8.4: *A WHISK extraction task*

As we can see in this instance, the original document about an apartment rental coming from a Web page is written in the form of ungrammatical construction but, however, human readable. The semantic class *bedroom* can be defined as follows :

```
Bedroom ::== ( br; brs; bdrm; bedrooms; bedroom )
```

This allows to find various forms of telegraphic styles for the same word. The extraction rule means : ignore all the characters in the text until you find a digit followed by the "br" string; extract that digit and fill the first extraction slot with it (i.e. "Bedrooms"). Then ignore again all the remaining characters until you reach a dollar sign immediately followed by a number. Extract the number and fill the "Price" slot with it. WHISK is more powerful than the two previous tools because it is multi-slots, that is, it is capable of extracting several records from a document.

8.6 Conclusion

NLP-based methodologies of this eighth chapter is another way to extract data of interest. The NLP tools learn extraction rules to mine out relevant data existing in natural language documents. They can work on grammatical texts, that is plain texts, as well as on ungrammatical texts such as those coming from the World Wide Web. But these techniques are very complex and are related to the huge issue of the natural language processing, a field of the artificial intelligence. Now that all the major automated extraction tools have been considered, we can focus in chapter nine on the one that inspired our work in Australia: the Web news extraction through a mapping process.

Chapter 9

News extraction using tree edit distance

Contents

9.1 Introduction

In this chapter we shall present two major notions for our work: the tree edit distance and the top-down mapping. An introduction summarizes the main difficulties encountered in the previous chapters leading to a danger for the extracting task. Then we shall tackle the extracting task strictly speaking. The first part is about the notions of tree edit distance and mapping. The next part is about the top-down mapping. The third part presents the Web news extraction through the article which we went by for our work followed by its application in the fourth part before finally concluding.

9.2 Context

While important works have been made in order to provide efficient access to relevant information on the Web, it is quite difficult to create generic methods that extract these Web data. The reasons of such a difficult task are that the Web is very heterogeneous and there are no rigid guidelines on how to build HTML pages and how to declare the implicit structure of the Web pages [23]. This problem, raised in chapter 4 is one of our main concerns.

Moreover, data structures are significant as well. Unstructured data where browsing and keyword searching are applied do not allow to use efficient methods to extract these data. Semi-structured or structured data that borrow techniques from the database area have also internal problems.

In order to develop effective methods for extracting data on the Web in a precise and automatic way, we need to take into account specific characteristics of the domain of interest. Ours is on-line newspapers and news portals on the Web, which have become one of the most important sources of up-to-date information. Indeed, there are thousands of sites that provide daily news in very distinct formats and there is a growing need for tools that will allow users to access and keep track of this information in an automatic manner.

In addition to these difficulties, this step is at the end of the whole extraction process, after the parsing and the clustering process (see chapter 1). This means that it can carry out wrong results that are maybe not coming from the extracting task itself because these previous processes can give bad results to the extracting job. In consequence, we must be vigilant and very critical towards the final results we shall analyse.

9.3 News extraction using tree edit distance

One of the most interesting approaches of the news extraction has been developed by Davi de Castro Reis, Paulo B. Golgher, Altigran S. da Silva and Alberto H.F. Laender-four [23], four Brazilian research workers. Their work influenced a lot our dissertation in the area of extraction news. They present a domain-oriented approach to Web data extraction and discuss its application to automatically extracting news from Web sites. This method is based on a highly efficient tree structure analysis that produces very effective results: the tree edit distance. It allows not only the extraction of relevant text passages from the pages of a given Web site, but also the fetching of the entire Web site contents, the identification of the pages of interest (the pages that actually present the news) and the extraction of the relevant text passages discarding non-useful material such as banners, menus, and links (noisy blocks).

This method is thus an all-in-one method that first clusters the pages, generates an extraction pattern, matches the data and finally labels these data.

9.4 Tree edit distance and mapping

This concept is based on the analysis of the structure of Web pages. As seen before, these Web pages can be transformed into tree structures. The tree edit distance allows to evaluate the structural similarities between different pages. The pages with similar structures are putted together in a same group called cluster. The clustered pages are then analysed to find a generic representation of the structure of the pages within a cluster.

Intuitively, the tree edit distance between two trees T_A and T_B is the cost associated with the minimal set of operations needed to transform T_A into T_B. These operations are: *vertex replacement* (or non-identical substitution), *vertex insertion* and *vertex removal*. They are operated on rooted, ordered, labelled trees. To evaluate the tree edit distance, a cost is assigned to each operation. The problem is resolved when we find the minimum cost to transform a tree into another.

Another way to understand the problem is the mapping approach. A mapping is a description of how a sequence of edit operations (replacement, insertion and removal) transforms a tree into another, ignoring the order in which these operations are applied. The discovering of a mapping with minimum cost between two trees will thus solve the problem. Here is a formal definition of this important concept of mapping:

Mapping: Let T_x be a tree and let $T_x[i]$ be the i-ism vertex of tree T_x in a preorder walk of the tree. A mapping between a tree T_1 of size n_1 and a tree T_2 of size n_2 is a set M of ordered pairs (i, j), satisfying the following conditions $\forall (i_1, j_1); (i_2, j_2) \in M$

$$\begin{cases} i_1 = i_2 \text{ iff } j_1 = j_2 \\ T_1[i_1] \text{ is on the left of } T_1[i_2] \text{ iff } T_2[j_1] \text{ is on the left of } T_2[j_2] \\ T_1[i_1] \text{ is an ancestor of } T_1[i_2] \text{ iff } T_2[j_1] \text{ is an ancestor of } T_2[j_2] \end{cases}$$

The first condition establishes that each vertex can appear no more than once in a mapping, the second one enforces order preservation between sibling nodes and the third one enforces the hierarchical relation between the nodes in the trees. Figure 9.1 illustrates a mapping between two trees.

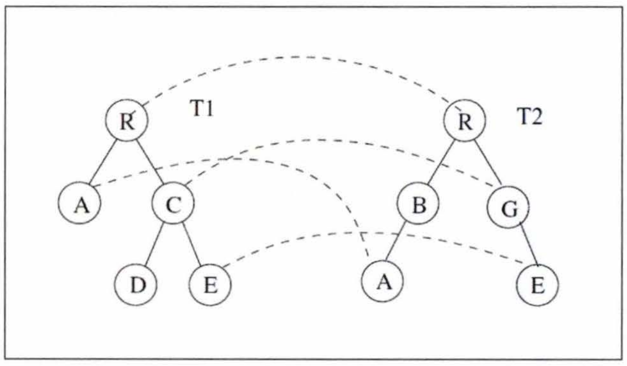


Figure 9.1: *A mapping between two trees*

As estimating the tree edit distance is equivalent to finding the minimum cost mapping, we can now define the mapping cost:

Mapping cost: Let M be a mapping between tree T_1 and tree T_2, let S be a subset of pairs $(i, j) \in M$ with distinct labels, let D be the set of nodes in T_1 that do not occur in any $(i, j) \in M$ and let I be the set of nodes in T_2 that do not occur in any $(i, j) \in M$. The mapping cost is given by $c = S_p + I_q + D_r$, where p, q and r are the costs assigned to the replacement (or non-identical substitution), insertion, and removal operations, respectively. It is common to associate a unit cost to all operations, however, specific applications may require the assignment of distinct costs to each type of operation. Let us apply these two definitions on a simple example (see figure9.2).

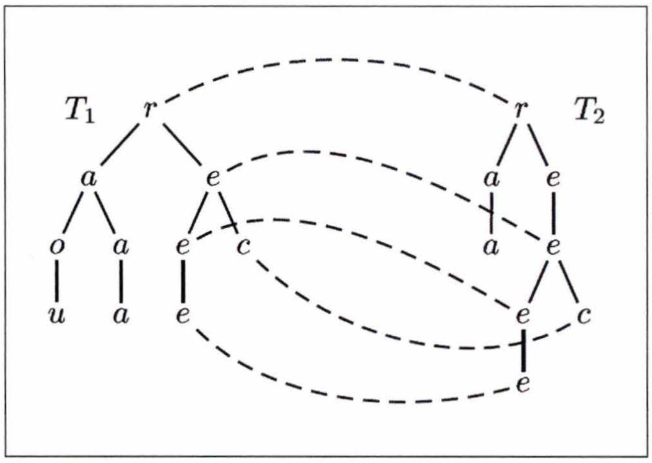


Figure 9.2: *A mapping example*

In this mapping M from tree T_1 to tree T_2, nodes $T_1[1]$, $T_1[7]$, $T_1[8]$, $T_1[9]$, $T_1[10]$ (r, e, e, e, c respectively) are mapped to nodes $T_2[1]$, $T_2[5]$, $T_2[6]$, $T_2[7]$, $T_2[8]$ (r, e, e, e, c respectively), nodes $T_1[2]$, $T_1[3]$, $T_1[4]$, $T_1[5]$, $T_1[6]$ (a, o, u, a, a respectively) are deleted from T_1 and nodes $T_2[2]$, $T_2[3]$, $T_2[4]$ (a, a, e respectively) are inserted into T_2. About the mapping cost, there are 0 replacements, 3 insertions and 5 removals. The cost of mapping $M = (0 \times 1) + (3 \times 1) + (5 \times 1) = 8$ assuming that the cost of edit operations is set to 1.

The tree edit distance is a difficult problem and while several algorithms have been proposed, their complexity is above quadratic. Further, it has been proved that, if the trees are not ordered, the problem is NP-complete. The first algorithm of this problem had a complexity of $O(n_1\, n_2\, h_1\, h_2)$, where n_1 and n_2 are the sizes of the trees and h_1 and h_2 are their heights. The best known complexity of this problem is $O(n_1\, n_2 + {l_1}^2 + {l_1}^{2.5}\, l_2)$ where l_1 and l_2 are the number of leaves in each tree. But despite the inherent complexity of the mapping, several practical applications can be modelled using restricted formulations. For instance, it is possible to impose conditions on the three edit operations. One of these formulations, the top-down distance is significant for our work. The top-down distance is used in the Brazilians' approach. That is the reason why it is now important to write about this topic.

9.5 Top-down mapping

Informally, a top-down mapping restricts the removal and insertion operations to take place only in the leaves of the trees. Moreover, in a top-down mapping, the parents of nodes in the mapping are also in the mapping. Here is a formal definition followed by an example (see figure 9.3):

Top-down mapping: A mapping M between a tree T_1 and a tree T_2 is said to be top-down only if for every pair $(i_1, i_2) \in M$ there is also a pair $(parent(i_1), parent(i_2)) \in M$, where i_1 and i_2 are non-root nodes of T_1 and T_2 respectively.

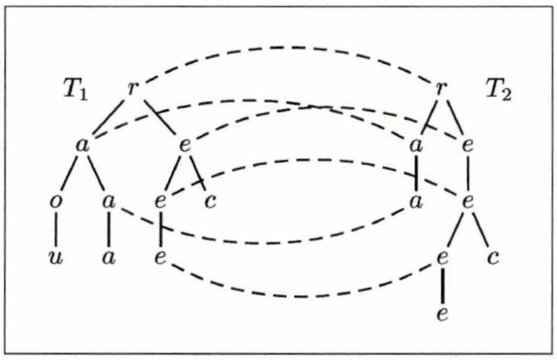


Figure 9.3: *A top-down mapping example*

Several well known algorithms for this top-down mapping have a complexity of $O(n_1\, n_2)$. They have been successfully applied to many Web related applications such as the clustering of XML documents. But in this case we are interested in the problem of evaluating the similarity between Web pages. The problem will be resolved by applying a top-down mapping between two trees that represent these Web pages and, as a consequence, the tree edit distance between them.

This top-down is a new kind of mapping called the Restricted Top-Down Mapping or RTDM. Intuitively, besides the insertion and removal operations, the replacement operation of different vertices is also restricted to the leaves of the tree. Once again, here is a more formal definition:

Restricted Top-Down Mapping (RTDM): A top-down mapping M between a tree T_1

and a tree T_2 is said to be restricted top-down only if for every pair $(i_1, i_2) \in M$, such that $T_1[i_1] \neq T_2[i_2]$, there is no descendent of i_1 or i_2 in M, where i_1 and i_2 are non-root nodes of T_1 and T_2 respectively.

Related to this mapping, we can define the restricted top-down edit distance between two trees T_1 and T_2 as the cost of the restricted top-down mapping between the two trees.

The analysis of the algorithm of this mapping is beyond the scope of this thesis. But this RTDM will be used in the different steps of the Brazilian approach, so it is important to understand the main idea behind this mapping: the algorithm takes as input the two trees to map and gives as output the minimal restricted top-down mapping between these two trees.

9.6 Web news extraction

Now we have explained the important notions of tree edit distance and restricted top-down mapping, we are able to discuss an automatic Web news extraction. This automatic process will identify relevant text pages containing news and their components by crawling Web news portals and extract the news from these collected pages discarding the noisy blocks.

The crawling of the pages will actually not be discussed here. This point is examined in chapters 2 and 3. We shall thus assume that we can directly access to these needed downloaded news Web pages.

The extraction task, as for it, is divided into four steps: page clustering, extraction pattern generation, data matching and data labelling (see figure 9.4). Three of them (clustering, extraction and matching) are based on the RTDM algorithm.

News site contents can be divided in groups that share common format and layout characteristics. These common format and layout features are called templates:

Template: A template is the set of common layout and format features that appear in a set of HTML pages produced by a single program or script that dynamically generates the HTML page contents.

Each field of a template (e.g. a news title) is called a data-rich object. Ideally, the extractors generated by the Brazilian approach should be able to identify each one of these data-rich objects, and discover, among them, which ones correspond to the title and the body of the news article. Let us see now how this process actually works.

9.6.1 Page clustering

The clustering takes as input a previously crawled set of pages (a training set) and generates clusters of pages that share the same template. The technique used for clustering is a classical one that takes as input the result of the RTDM algorithm with a cost model having a cost unit for the edit operations. An arbitrary threshold of 80% determines the similarity of the pages within a same cluster. The output of this step is a set of page clusters sharing the same template.

9.6.2 Extraction pattern generation

This second step refines the notion of cluster. We shall now use the term *node extraction pattern (ne-pattern)*:

Node extraction pattern (ne-pattern): Let a pair of sibling sub-trees be a pair of sub-trees rooted at sibling vertices. A node extraction pattern is a rooted ordered labelled tree that can contain special vertices called wildcards. Every wildcard must be a leaf in the tree, and each wildcard can be of one of the following types:

- SINGLE ($\diamond$): A wildcard that captures one sub-tree and must be consumed.
- PLUS ($+$): A wildcard that captures sibling sub-trees and must be consumed.
- OPTION ($?$): A wildcard that captures one sub-tree and may be discarded.
- KLEENE ($*$): A wildcard that captures sibling sub-trees and may be discarded.

A wildcard is defined by every vertex in the tree that can match any symbol (any label) with its associated type.

In view of this definition, each wildcard corresponds to a data-rich object in the template. SINGLE and PLUS wildcards should correspond to required objects, such as the title of a

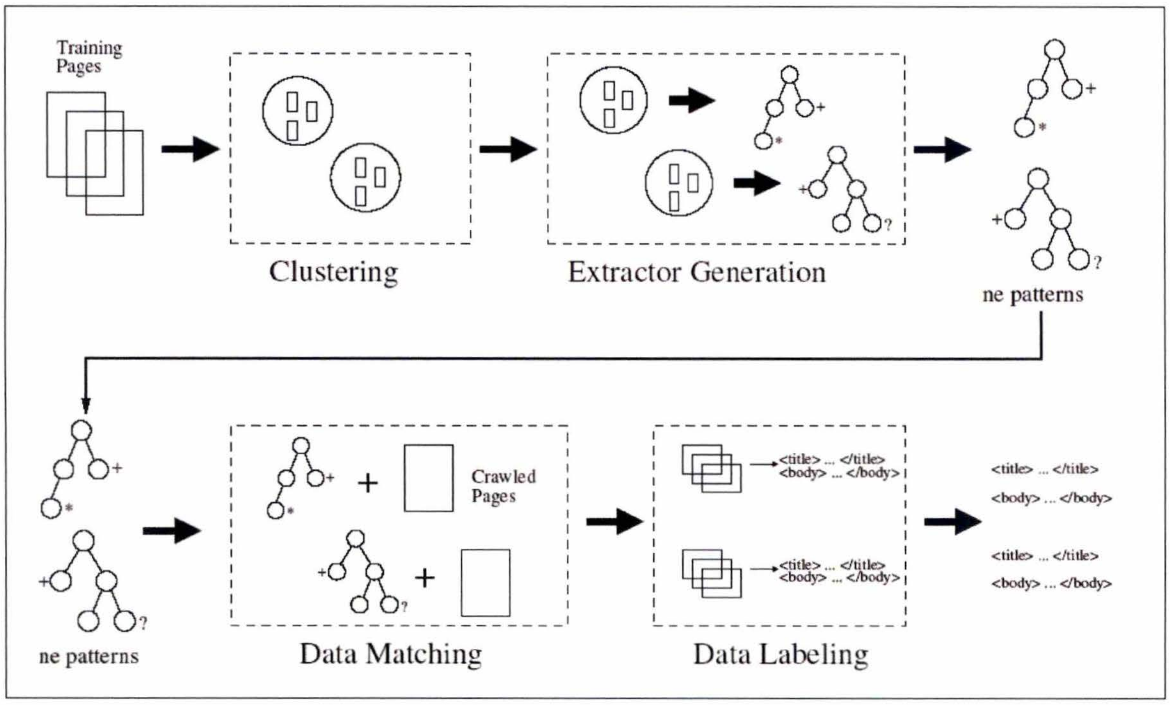


Figure 9.4: *The four extraction steps*

news, and OPTION and KLEENE wildcards should correspond to optional objects, such as related news lists.

The goal of this step of the extraction task is, taking as input a page cluster, to generate an ne-pattern that accepts all the pages in this cluster. The wildcards represent therefore the contents differences between the pages in the same cluster. To reach this objective, the RTDM algorithm is once again applied. First it is said that vertices a and b of an ne-pattern are equal if and only if:

- a and b are wildcards and both are of the same type;
- a and b are not wildcards and the labels associated with a and b are equal.

Given two ne-patterns ${T_1}^x$ and ${T_2}^x$, the RTDM algorithm finds a mapping between these two ne-patterns ($M_{{T_1}^x \to {T_2}^x}$). From this mapping, a composite ne-pattern is created ${T_3}^x = {T_1}^x \circ {T_2}^x$ using the following rules:

- if a is not in the mapping, then add a' to ${T_3}^x$ where $a' = f(a, ?)$;
- if a maps to b then add a' to ${T_3}^x$ where $a' = f(a, b)$;

- and $f(a, b)$ is defined as:

$$\begin{array}{lcl|lcl|lcl}
f(*,*) & = & * & f(+,+) & = & + & f(\diamond,\diamond) & = & \diamond \\
f(*,+) & = & * & f(+,\diamond) & = & + & f(\diamond,?) & = & ? \\
f(*,?) & = & * & f(+,?) & = & * & f(\diamond,n) & = & \diamond \\
f(*,\diamond) & = & * & f(+,n) & = & + & f(?,?) & = & ? \\
f(*,n) & = & * & & & & f(?,n) & = & ?
\end{array}$$

$f(n_1, n_2) = n_1$ if n_1 and n_2 have identical labels

$f(n_1, n_2) = \diamond$ if n_1 and n_2 have different labels

where n,n_1,n_2 are non-wildcard vertices and the parameter order is not relevant.

The first tree of the cluster is considered being the basic ne-pattern. This first ne-pattern is then compared with the next tree applying the rules above, a new ne-pattern resulting from this comparison. This new one is compared to the next tree and so on till the last tree of the cluster. The result is an ne-pattern that accepts all the pages in that cluster. Here is an example of this complex process (see figure 9.5).

9.6.3 Data Matching

Once the ne-patterns have been determined within each cluster, the data matching task will find the most appropriate ne-pattern to a crawled HTML page. This process is also based on the RTDM algorithm. An appropriate cost model for the three edit operations (replacement, insertion and removal) will give us the cost of the mapping called in this step a *match*. For that, we speak about the consumption of vertices: in a given mapping, if one wildcard vertex in the ne-pattern maps to a vertex in the target HTML tree, then the wildcard consumes the vertex. The data matching is defined as follows:

Match: a match between an ne-pattern and a target tree is a mapping such that the following rules are satisfied in this order:

1. Every non-wildcard vertex in the ne-pattern must map to an identical vertex in the target tree.
2. Every vertex in the target tree must map to an identical non wildcard vertex in the ne-pattern or be consumed by a wildcard.

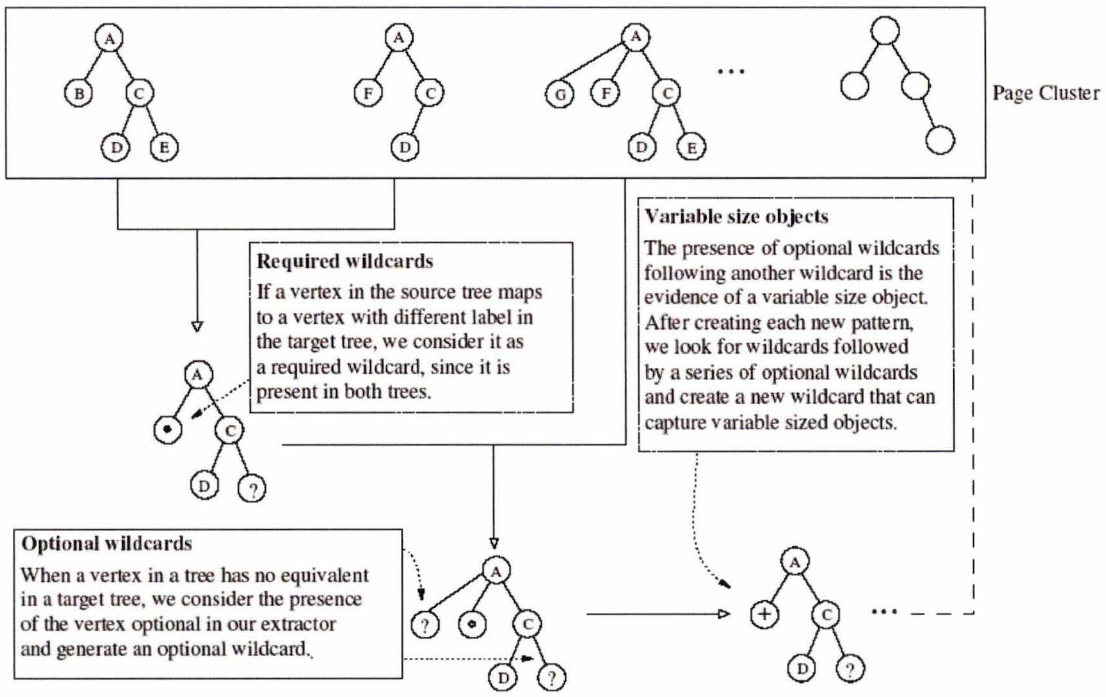

Page Cluster
Required wildcards
If a vertex in the source tree maps to a vertex with different label in the target tree, we consider it as a required wildcard, since it is present in both trees.
Variable size objects
The presence of optional wildcards following another wildcard is the evidence of a variable size object. After creating each new pattern, we look for wildcards followed by a series of optional wildcards and create a new wildcard that can capture variable sized objects.
Optional wildcards
When a vertex in a tree has no equivalent in a target tree, we consider the presence of the vertex optional in our extractor and generate an optional wildcard.

Figure 9.5: *Creation of an ne-pattern accepted by all the pages from the same cluster*

3. Single wildcards ($\diamond$) must consume one sub-tree of the target tree.
4. Plus wildcards ($+$) must consume at least one sub-tree of the target tree.
5. Option wildcards ($?$) must consume one sub-tree of the target tree, if it is possible.
6. Kleene wildcards ($*$) must consume at least one sub-tree of the target tree, if it is possible.

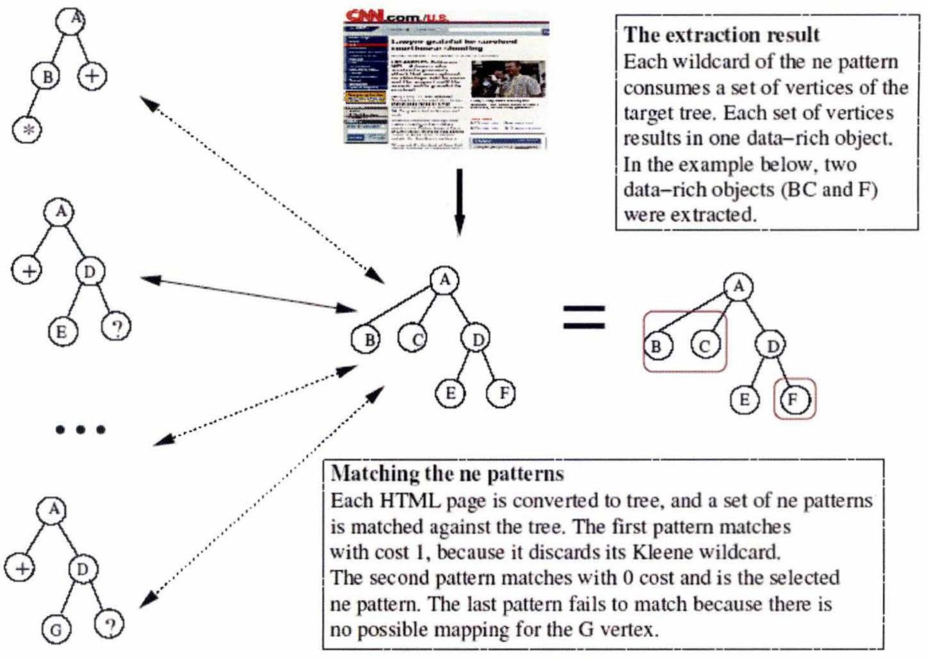
The HTML pages in the clusters transformed into trees are matched with the different ne-patterns. For each data matching, these six rules will lead to one of the three edit operations associated with a cost.

For the **replacement operation**, if the compared vertices are both non-wildcards and have the same label, the cost will be 0. Or if the vertex in the ne-pattern is a wildcard, the cost will be 0 as well. But if the compared nodes are both non-wildcards and have not the same label, the cost will be infinite, meaning that the matching between these two trees fails.

For the **insertion operation**, if there is an ancestor of the current vertex in the target tree (the HTML page) that can be consumed by a wildcard, the cost is 0. Or if the left sibling of this current vertex is consumed by a KLEENE wildcard (*) or a PLUS wildcard (+) the cost is also 0. Otherwise the cost is infinite.

Finally for the **removal operation**, if the wildcard of the current ne-pattern is an OPTION (?) or a KLEENE (*) then the cost is 1. If not, it is infinite.

If the final matching cost is not infinite, the ne-pattern accepts the target page. In this case, the ne-pattern and the HTML page are traversed in pre-order and for each wildcard found in the ne-pattern, the text passage in the vertices consumed by the wildcard is extracted from the HTML page. Figure 9.6 shows a data matching example example.



The extraction result
Each wildcard of the ne pattern consumes a set of vertices of the target tree. Each set of vertices results in one data–rich object. In the example below, two data–rich objects (BC and F) were extracted.

Matching the ne patterns
Each HTML page is converted to tree, and a set of ne patterns is matched against the tree. The first pattern matches with cost 1, because it discards its Kleene wildcard. The second pattern matches with 0 cost and is the selected ne pattern. The last pattern fails to match because there is no possible mapping for the G vertex.

Figure 9.6: *How ne-patterns are matched with Web pages*

9.6.4 Data labelling

This step is the only one that does not use the RTDM algorithm. Its objective is to select from a set of ordered text passages (i.e. the output of the data matching), the passages t_i and t_j that correspond to the title and the body of the news being extracted from the Web page. To achieve this, simple heuristics are applied to this set of text passages. They can be defined as a set of $T = (t_1, p_1), (t_2, p_2), \ldots, (t_n, p_n)$ where each t_i is a text passage retrieved by a wildcard and p_i is the vertex position of this wildcard by a pre-order traversal of the ne-pattern.

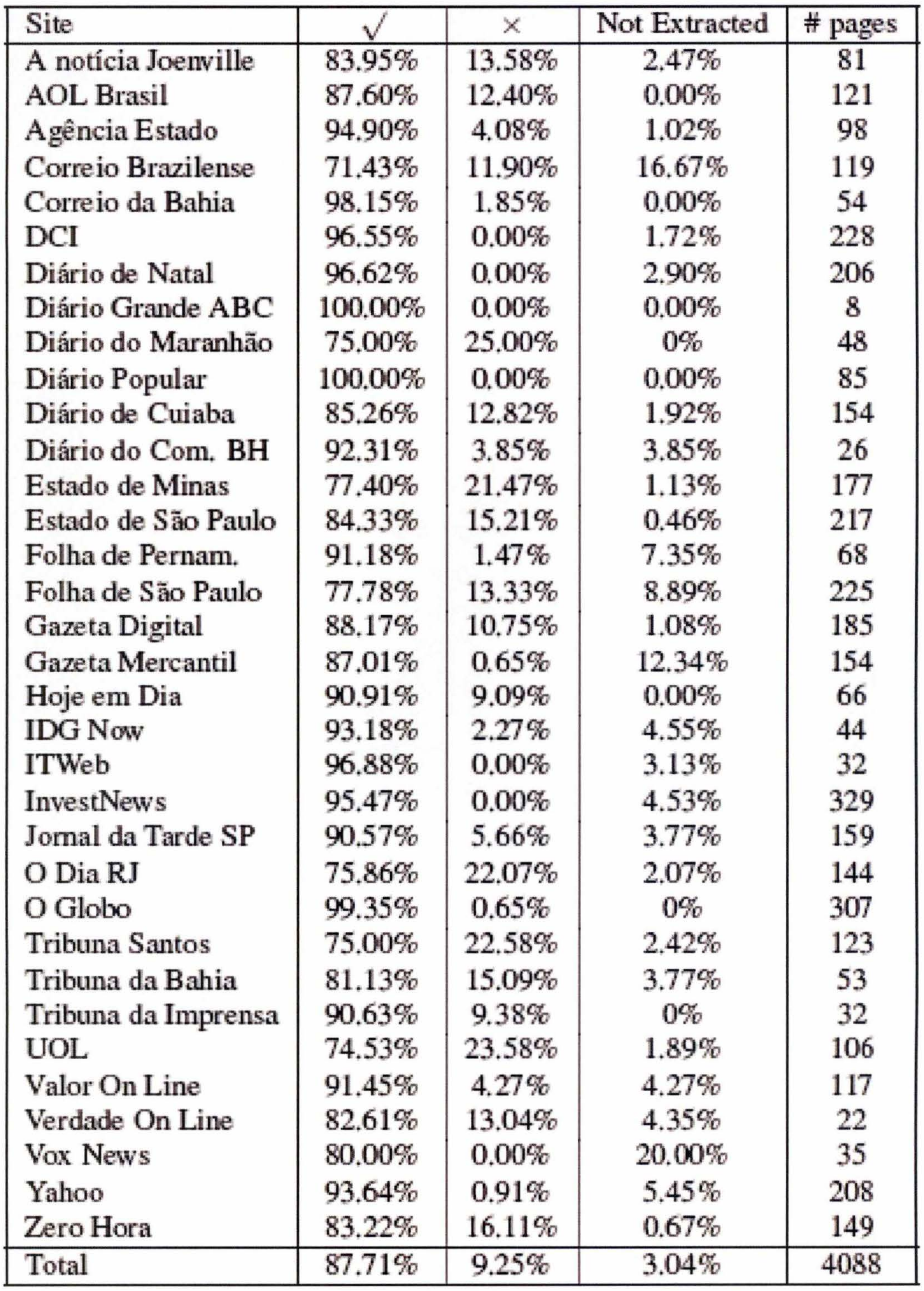
The heuristics are rather basic: about the text, the passage elected to be the body of the news is the longest one with more than 100 words. Further, the passage selected to be the title is one that has ranges from 1 to 20 words, has a maximum intersection with a body passage, and is the closest one to the body. The intuition behind the title selection is that most of the time the title is placed near the body and its terms usually appear in the news body. These heuristics are here formally stated:

For a given T:

- $length(t_i)$ is the number of terms (words) in passage t_i;
- $|t_k \cap t_i|$ is the number of terms that occur in passages t_k and t_i;
- t_i is a news body iff $length(t_i) > length(t_k)\ \forall k : 1 < k < n,\ k \neq i$ and $length(t_k) > 100$;
- t_j is a news title iff $1 \leq length(t_j) \leq 20$ and $\frac{|t_j \cap t_i|}{p_j - p_i} \geq \frac{|t_k \cap t_i|}{p_k - p_i}\ \forall k : 1 < k < l,\ k \neq j$

9.7 Applications

This complex Web news extraction process has been applied in a large experimentation. 4088 HTML pages collected from 35 Brazilian news Web sites have been analysed. Each output of these pages, i.e. the news extracted, has been manually compared with the original HTML page. 87.71% of the news were correctly extracted, while 9.25% were erroneously extracted and 3.04% were not extracted. Figure 9.7 shows the results of this application.



Site	√	×	Not Extracted	# pages
A notícia Joenville	83.95%	13.58%	2.47%	81
AOL Brasil	87.60%	12.40%	0.00%	121
Agência Estado	94.90%	4.08%	1.02%	98
Correio Brazilense	71.43%	11.90%	16.67%	119
Correio da Bahia	98.15%	1.85%	0.00%	54
DCI	96.55%	0.00%	1.72%	228
Diário de Natal	96.62%	0.00%	2.90%	206
Diário Grande ABC	100.00%	0.00%	0.00%	8
Diário do Maranhão	75.00%	25.00%	0%	48
Diário Popular	100.00%	0.00%	0.00%	85
Diário de Cuiaba	85.26%	12.82%	1.92%	154
Diário do Com. BH	92.31%	3.85%	3.85%	26
Estado de Minas	77.40%	21.47%	1.13%	177
Estado de São Paulo	84.33%	15.21%	0.46%	217
Folha de Pernam.	91.18%	1.47%	7.35%	68
Folha de São Paulo	77.78%	13.33%	8.89%	225
Gazeta Digital	88.17%	10.75%	1.08%	185
Gazeta Mercantil	87.01%	0.65%	12.34%	154
Hoje em Dia	90.91%	9.09%	0.00%	66
IDG Now	93.18%	2.27%	4.55%	44
ITWeb	96.88%	0.00%	3.13%	32
InvestNews	95.47%	0.00%	4.53%	329
Jornal da Tarde SP	90.57%	5.66%	3.77%	159
O Dia RJ	75.86%	22.07%	2.07%	144
O Globo	99.35%	0.65%	0%	307
Tribuna Santos	75.00%	22.58%	2.42%	123
Tribuna da Bahia	81.13%	15.09%	3.77%	53
Tribuna da Imprensa	90.63%	9.38%	0%	32
UOL	74.53%	23.58%	1.89%	106
Valor On Line	91.45%	4.27%	4.27%	117
Verdade On Line	82.61%	13.04%	4.35%	22
Vox News	80.00%	0.00%	20.00%	35
Yahoo	93.64%	0.91%	5.45%	208
Zero Hora	83.22%	16.11%	0.67%	149
Total	87.71%	9.25%	3.04%	4088

Figure 9.7: *Results for the news extraction process*

9.8 Conclusion

In this chapter we presented the notions of tree edit distance and mapping used by four Brazilian research workers that inspired our thesis. Then their uses were explained in the third section through the Web news extraction task. Experimental results have been shown in the applications section. This task is the final one in the whole process of data extraction. It has to be attentively considered because wrong results coming from the outputs of the previous tasks have perhaps piled up along the complete process. The extracted data are now ready for further use by mining agents. This chapter closes the state of the art.

Part II

News Ripper

Chapter 10

Context

Contents

10.1 Introduction

We presented in the state of the art different steps that allow extracting information from the Internet as automatically as possible. Obviously, we could not write a program that would have covered the whole process. Moreover, the intentions of Debbie Zhang, the person in charge of us changed during our internship.

We shall first explain how our work has evolved during the internship and where our project stands in the three steps described in the state of the art. Then we will introduce the major functions of our program *News Ripper* and their goals.

The information on how to use the program is at Appendix 1.

10.2 Our internship

At the beginning, we were asked to program an "HTML to DOM tree" parser and a bottom-up algorithm. The goal was to show with a user-friendly graphic interface the results of a bottom-up mapping between two DOM trees. A bottom-up mapping between two trees consists to find identical nodes in both trees from the leaves to the root.

One month later, we were asked to give up the extraction of news articles using the bottom-up algorithm, in order to concentrate on a "top-down" algorithm that would be useful to cluster pages before applying the bottom-up mapping. In the meanwhile, as the trees were really large, we adapted the parser to lighten the DOM trees (i.e. to reduce their size) in order to improve both effectiveness and efficiency of the clustering algorithm.

We spent a lot of time to design the graphic user interface. As for the bottom-up mapping, we had to graphically show the results of the top-down algorithm.

10.3 Major functions

As written above, we do not cover the whole process presented in the state of the art. We assume the first step has been done, i.e. that pages have been downloaded from relevant Web sites. Both first chapters presented in the state of the art, i.e. the web sites classification and the web pages classification are not covered by *News Ripper*.

Our program actually operates on the following steps. It features the transforming of an HTML source to a layout tree with several options, the clustering of similar pages and the extraction of the news article.

Our program is both useful and educational. It allows a user to cluster pages without any graphic results, so that the running is as fast as possible. On the other hand, if a user aims at understanding the results, he can easily see on the layout trees how the algorithms have been applied.

At the end of our internship, the major functions of our program can be summarized as follows:

- A graphic user interface allowing the management of HTML files. HTML files can be opened, edited, saved, closed,etc. A project (a set of all opened HTML files and the results of their clustering) can be opened and saved. The purpose is to make the handling of numerous files easier and to store the results of a clustering. (Appendix 1)

- A graphic user interface allowing to download the source code of a page directly from the Internet. (Appendix 1)
- A graphic user interface to show layout trees. (Appendix 1)
- An improved HTML to layout tree parser which manage to remove useless tags. (Chapter 11)
- A graphic user interface to select the tags to be considered during the parsing (Chapter 11)
- A layout tree to HTML parser. (Chapter 11)
- A top-down algorithm used to cluster similar pages. (Chapter 12)
- A user friendly interface to show on layout trees the results of a top-down mapping. (Chapter 12)
- A graphic user interface to show the results of a clustering. (Chapter 12)
- An algorithm for extracting the news articles of pages from the one cluster. (Chapter 13)
- A user friendly interface to show the results of the news extraction. (Chapter 13)

10.4 Contents of Part II

As we explain in the first chapter, an intuitive approach, we shall introduce the three successive steps that allow to extract news articles from Web pages:

- The parsing of an HTML source in order to build a tree that represents the layout of the HTML page (Chapter 11).
- The clustering of similar pages (i.e. similar layout trees) thanks to a top-down mapping algorithm (Chapter 12).
- The extraction of news articles thanks to a an algorithm which compares the leaves of similar layout trees inside a cluster. (Chapter 13)

Chapter 11

Smartly building layout trees

Contents

11.1 Introduction

Once the relevant pages have been extracted from the Internet, we need to build a tree structure (see chapter 4) from the source code in order to be able to cluster similar pages. The parsing must be done as quickly as possible since there are lots of trees to build.

We shall see why parsing HTML is not so easy and how the parsing can be improved by removing useless information on the Web pages. Actually, if the pages are first "cleaned up", the size of the generated trees is reduced. Then, the algorithms that are applied on the trees to cluster the pages give better and faster results. We shall see why some HTML information is useless and the consequences of its removal.

11.2 Why parsing HTML is hard

11.2.1 Different versions of HTML

Parsing an HTML file is not an easy task [24]. First of all, there are many versions of HTML defined by the W3C (HTML 2.0, HTML 3.2, HTML 4.0, HTML 4.01, XHTML 1.0, XHTML 1.1, XHTML 2.0,).

Our parser has been designed to handle HTML 4.01 documents because it is the most spread format. The parser can recognize the 90 different tags defined in the HTML 4.01 specification. In most cases, it works on other HTML versions and on XHTML as well.

11.2.2 "Badly" formatted text

The second issue is the invalid or badly formatted text. Lots of pages on the Internet contain structural errors that usual Web browsers can handle. The parser should manage to cope with those errors as well; it should "repair" the invalid pages before transforming them into a tree. Our parser can deal with several structural mistakes but we can not guarantee that every malformed page that is readable on a Web browser could be parsed without any errors.

Another issue is the possibility of optional end tags in the HTML code. Some tags such as `<P>`, `<LI>`, `<TD>` can be used with or without the `</P>`, `</LI>`, `</TD>` end tags. Moreover, some tags may or may not overlap. As this is allowed by the W3C, we had to get by with these complications. The parsing of an HTML file is then harder than the one of an XML file. With an XML file, the structure definition is very stringent, so that the file can not be badly formatted. Some famous programs like JTidy are designed to transform a badly formatted HTML file into an XHTML file, a format between HTML and XML where all tags have to be properly closed. As we wanted our parser to be as efficient as possible, it would have been a waste of time to "fix" each page containing optional end tags before constructing the tree, we needed to parse and repair the file simultaneously.

11.2.3 HTML parsers available on the Internet

A few libraries to help program HTML parsers are proposed on the Internet. Most of them have bugs or restricting features that we have faced before finding the good one. As we spent a lot of time to experiment various parsers, we think it can be useful to explain the issues we had with each tested parser.

JavaCC HTML Parser by Quiotix Corporation

Can be found at *http://www.quiotix.com/downloads/html-parser/*.

This parser does not support document structure, which means it can not break down an HTML page into blocks to extract its elements[1]. The parser recognizes start and end tags and calls the functions defined by the user to handle them. So the user has to write functions to cope with the discovery of a start tag or an end tag. That way, the parser can not check if the document is badly structured: if tags are missing, the program written by the user above the parser must handle them itself.

Moreover, a bug has been discovered. The attributes of an HTML field, may or may not be quoted, and if they are quoted, either single or double quotes may be used. Quiotix

[1]An element is defined by a start tag `<...>`, an end tag `</...>` and all the content in between

parser does not manage to parse unquoted or single quoted attributes with some tags. It seems that the parser was designed for former versions of HTML.

Sun Java HTML Parser

This is the parser that comes standard in the JDK (package javax.swing.text.html.parser). It supports document structure and is really high-quality. It works by parsing first a DTD, a grammar that defines the HTML version, so that it seems capable to work with every version of HTML. The issue is that the JDK only comes with HTML3.2 DTD, which is unsuitable. The HTML 4.01 DTD can be found on W3C, but the DTD used by JDK needs to be in the format of a "bdtd" file, which is a binary format used only by Sun Microsystems in this parser implementation. There are many requests for a 4.01 bdtd file in forums or newsgroups on the Web, but they all remain unanswered. Building it from scratch is not so easy, so we need to wait for Sun Microsystems to write it.

Jericho HTML Parser

Jericho [25] is the best HTML parser we found. It has been written by a professional who was bored with facing the bugs of all the HTML parsers he had ever tested. Jericho HTML Parser is a simple but powerful Java library allowing both analysis and manipulation of an HTML document. All classes and methods have been comprehensively documented.

It supports document structure since it works by finding all the elements in a given page. It can reproduce verbatim any unrecognized HTML in a bid of suiting future versions. The library distinguishes itself from other HTML parsers because no parse tree of the entire document is ever generated. In this sense Jericho is strictly speaking not a true parser because it does not infer a grammar as usual parsers do. *"The document source text is searched only for the markup relevant to the current operation"* [25]. This allows the library to analyse and modify documents containing incorrect or badly formatted HTML. Most other parsers can only handle contents that they are explicitly programmed to accept.

Jericho can also recognize special tags, apart from HTML 4.01 specification: ASP, JSP, PSP, PHP and Mason server tags are explicitly recognized. The library then allows any of these segments to be ignored when parsing the rest of the document so that they do not interfere with the HTML syntax.

11.3 Lightening the HTML source

Let us analyse the definition of HTML on Webopedia:

> "Short for HyperText Markup Language, the authoring language used to create documents on the World Wide Web [...]. HTML defines the structure and layout of a Web document by using a variety of tags and attributes [...]. There are hundreds of tags used to format and lay out the information in a Web page."

First, we should remember that the objective is to cluster similar pages, i.e. pages that have a common general layout. If we look at two Web pages, they will be considered as similar if their global structure is common (how the blocks are divided; how the menus, the main text, the images,... are positioned). Not all the tags are responsible of this general arrangement: some tags are designed to build the "block-level" layout of the Web page, while others are used to format specific elements such as text ("inline-level"). Moreover, a few tags are not designed to lay out Web pages but to give information that is not displayed. For example, `<div>`, `<table>`, `<tr>` are block-level tags, `<a>`, `<b>`, `<i>` are inline-level tags, and `<meta>`, `<address>` tags give information that is not displayed.

As we target at extracting the contents of a news article, independently from its formatting, we do not need to keep tags that format only text. More generally, all the tags that do not define the general structure of the page can carefully be considered as useless.

The HTML code sets up the page layout from general to specific features. If we look at the DOM tree of an HTML page, nodes close to the root are the tags that define the general arrangement while nodes close to the leaves concern specific format features.

Then, if we decide to remove those inline-level tags, we considerably reduce the amount of nodes in deep levels of the tree without changing block-level nodes close to the root. As the top-down algorithm used to cluster similar pages works by comparing vertices between two trees from the top to the bottom, the purging of vertices close to the leaves hardly affects the results. These DOM trees which contain only the tags responsible of the general layout are called "layout trees".

11.3.1 Main idea

As said above, the purpose of reducing the amount of nodes in the generated tree is twofold :

- To improve the efficiency of the mapping algorithms.
- To gather parts of text.

"To reduce the amount of nodes" actually means "To ignore or remove some tags during the parsing". We need to carefully choose the tags to be ignored or removed. The main idea can be stated as follows:

- The tags that define the arrangement of the blocks (block-level), i.e. how the page is displayed, must be kept. We need them to cluster similar pages.
- The tags that define the formatting of text (inline-level), i.e. how the text is displayed, can be ignored, which implies that the text inside the element is kept.
- The tags that do not enclose text, that have no (or slightly) effects on the layout (which do not define the arrangement of the blocks), can be removed with their contents.

It is quite important to understand that "ignoring tags" means removing both start and end tags without removing the text in between. The goal is to get rid of the text formatting although keeping the text itself.

The objective is a bit paradoxical: on the one hand we use the "block-level" tags to cluster similar pages and on the other hand we ignore the "inline-level" tags to get rid of the text formatting.

11.3.2 Carefully choose the tags to select

We analyzed each of the 90 tags presented in the HTML 4.01 specification made by the World Wide Web Consortium (www.w3.org) [26] in order to classify the HTML tags into one of the three classes presented above. As this choice is arbitrary, it must be possible for a user to select himself the tags he wants to keep or ignore. The selection that we shall present is a default selection that the program loads when it is launched.

However, since we could not display a frame allowing to (un)select 90 tags, we gathered some tags into generic names (see figure 11.1).

Notice that tags `<HTML>`, `<HEAD>` and `<BODY>` are automatically selected. (D) means that the tag is deprecated.

11.3.3 Tables

Tables are one of the most used structures to tailor the layout on a page. We chose to keep all the tags linked to tables.

TABLE	Keep	Delineates a table
THEAD	Keep	Delineates a row group in the head part in a table
TBODY	Keep	Delineates a row group in the body part in a table
TFOOT	Keep	Delineates a row group in the foot part in a table
TR	Keep	Delineates a row in a table
TD	Keep	Delineates a cell in a row
TH	Keep	Delineates a cell containing a header in a row
CAPTION	Keep	Contains the caption of the table
COL	Keep	Delineates a column in a table
COLGROUP	Keep	Delineates a column group in a table

In order to show a lightened tag selection frame, we grouped the following tags like this:

- (de)select `TABLE` : keep or remove tables in the layout tree.
- (de)select row/col groups : keep or ignore `THEAD`, `TBODY`, `TFOOT`, `COL`, `COLGROUP`.
- (de)select `TR` : keep or ignore tables rows.
- (de)select `TD`, `TH` : keep or ignore table cells.
- (de)select `CAPTION` : keep or remove table caption.

11.3.4 Lists

Lists are also a usual way to structure information. They are often utilized to structure menus in the left part of a news Web site. They can be block-levels and a fortiori inline-levels.

A more seldom kind of list are the definition lists. *"Definition lists vary only slightly from other types of lists in that list items consist of two parts: a term and a description. The term is given by the DT element and is restricted to inline content. The description is given with a DD element that contains block-level content."* [26] Tags `<DIR>` and `<MENU>` are deprecated and often replaced by `<UL>` but the parser should handle them anyway.

OL	Keep	Delineates an ordered list
UL	Keep	Delineates an unordered list
LI	Keep	Delineates a list element
DL	Keep	Delineates a definition list
DT	Keep	Delineates the definition term
DD	Keep	Delineates the definition description
DIR (D)	Keep	Designed to be used for creating directory lists
MENU (D)	Keep	Designed to be used for single column menu lists

In the tag selection frame, it gives:

- (de)select `UL`, `OL` : keep or remove lists.
- (de)select `LI` : keep or ignore lists elements.
- (de)select `DD` : keep or remove definition lists.
- (de)select `DT`, `DD` : keep or ignore definition lists elements.
- (de)select `DIR`, `MENU` : keep or remove directory and menu lists.

11.3.5 Objects

Objects do not contain text but they play a large role in the page layout.

OBJECT	Keep	Includes an object. Object is a generic name for every media (image, sound, video, applet,etc.)
IMG	Keep	Includes an image
APPLET(D)	Keep	Includes an applett
IFRAME	Keep	(for Inline Frame) Includes a frame in an "inline-level"
PARAM	Remove	Contains a "non-displayed" parameter used for the `<Object>` tag

MAP	Remove	Specifies a "non-displayed" map that allows an object to be split in several areas so that each area interacts differently with the user
AREA	Remove	Specifies a "non-displayed" image map area

"The IFRAME element allows authors to insert a frame within a block of text. It allows you to insert an HTML document in the middle of another." [26] See also section 11.3.14 In the tag selection frame, the first four items are displayed and `PARAM, MAP and AREA` are put together in "Objects parameters". Deselecting an object means removing it.

11.3.6 HEAD tags

Some tags are enclosed only in the HEAD tag and contain information on the page.

TITLE	Keep	Contains the title to appear in the window. News Web sites sometimes show the title of the article at this place
META	Remove	Contains "not-displayed" information about meta-data
STYLE	Remove	Contains "not-displayed" information about the style sheets
LINK	Remove	Contains "not-displayed" information about the pages linked by this site
BASE	Remove	Allows to specify a document's base path for the URL's
ADDRESS	Remove	Contains "not-displayed" information about the author of the page

The same structure is kept in the tag selection frame. If title is deselected, the whole `title` element is removed.

11.3.7 Text management

P	Keep	Represents the inline-level paragraph
Q	Ignore	Used for short quotations. Puts quotation marks around the enclosed text

PRE	Ignore	For Pre-formatted Text. Means that the browser should render the text verbatim
BR	Remove	Forces line break
INS	Ignore	Indicates the new text
DEL	Remove	Indicates the old text

"INS and DEL are used to markup sections of the document that have been inserted or deleted with respect to a different version of a document (`A Sheriff can employ <DEL>3</DEL><INS>5</INS> deputies`*)."* [26]

If `P` is kept, every paragraph of the news article will be stored in a separate leaf. If it is ignored, all the paragraphs of the article will be put together in the same leaf (considering that tags inside the paragraph element are also ignored). We shall see later how the extraction algorithm takes into account the nodes `<P>`, `<H1>...<H6>` and `<TITLE>` in order to ensure a basic layout in the extracted text.

As we do not interpret text, the text inside a `<PRE>` element, will keep its original format with or without the `<PRE>` tags.

11.3.8 Miscellaneous

Some tags were too specific and we had to create a "Miscellaneous" class.

A	Ignore	Indicated the enclosed inline-element is an hypertext link
DIV	Keep	Defines its content to be block-level but imposes no other presentational idioms on the content
SPAN	Ignore	Defines its content to be inline-level but imposes no other presentational idioms on the content
Blockquote	Keep	Used for long quotations (block-level content). Browsers generally indent the text inside `BLOCKQUOTE`
BDO	Ignore	Forces to ignore the bi-directional algorithm that allows arabic text to be read from right to left
HR	Keep	Draws an horizontal rule

`DIV` and `SPAN` offer a generic mechanism for adding structure to documents. They are a good example to show how to handle a block-level or an inline-level tag.

Links in text are ignored since their formatting is irrelevant for a plain text extraction. However, in some cases such as "See also this" where this is a link to another article for example, it is obvious that the extracted text will be useless.

11.3.9 Headings

H1...H6	Keep	Indicates the text is a level (1-6) heading

There are six levels of headings available in HTML from `<H1>` to `<H6>`. As they are inline-levels, we normally should ignore them. However, it is better to keep them in a separate node, so that they are not melt with the paragraphs (if they are actually headings belonging to the news article).

11.3.10 Phrase elements

Phrase elements are inline-level tags that add structural information to text fragments.

EM	Ignore	Indicates emphasis
STRONG	Ignore	Indicates stronger emphasis
CITE	Ignore	Contains a citation or a reference to other sources
DFN	Ignore	Indicates that this is the defining instance of the enclosed term
CODE	Ignore	Designates a fragment of computer code
SAMP	Ignore	Designates sample output from programs, scripts, etc.
KBD	Ignore	Indicates text to be entered by the user
VAR	Ignore	Indicates an instance of a variable or program argument
ABBR	Ignore	Indicates an abbreviated form (e.g. WWW, HTTP, URI, Mass., etc.)
ACRONYM	Ignore	Indicates an acronym (e.g. WAC, radar, etc.)

Phrase elements are designed to give information on the text. Browsers or other Web agents can interpret them freely. In our case, we do not want to interpret any form of text and we simply ignore them.

11.3.11 Font styles

This class could also be considered as "text management" but we preferred keeping the structure of the W3C's HTML 4.01 specification as much as possible.

FONT (D)	Ignore	Selects the font for the enclosed content
BASEFONT	Ignore	Selects a default font for the document
SUP	Ignore	Puts the text in superscript
SUB	Ignore	Puts the text in subscript (ex: `H<sub>2</sub>O`)
TT	Ignore	Renders as teletype or monospaced text
I	Ignore	Renders as italic text style
B	Ignore	Renders as bold text style
BIG	Ignore	Renders text in a large font
SMALL	Ignore	Renders text in a small font
STRIKE (D)	Ignore	Render strike-through style text
S (D)	Ignore	idem STRIKE
U (D)	Ignore	Renders underlined text

In order to lighten the tag selection frame, each of the *Headings, Phrase elements* and *Font styles* sections became a generic item.

11.3.12 Forms

"An HTML form is a section of a document containing special elements called controls (checkboxes, radio buttons, menus, etc.), and labels on those controls. Users generally "complete" a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing." [26]

There are many tags involved with forms: `<INPUT>`, `<BUTTON>`, `<SELECT>`, `<OPTGROUP>`, `<OPTION>`, `<TEXTAREA>`, `<ISINDEX>`, `<LABEL>`, `<FIELDSET>` and `<LEGEND>`. We did not decide to list all of them but just to create a check box "select forms". If the check box is deselected, the form and its content will be removed. As a news article is not likely to be inside a form, the entire form can be removed but this can have an impact on the page look. There is often a little form with a text field and a button designed to search for articles through the Web site. We chose to select forms by default in the tag selection frame because we wanted to keep all the block-level elements for an accurate comparison between general layouts.

11.3.13 Scripts

"A client-side script is a program that may accompany an HTML document or be embedded directly in it. The program executes on the client's machine when the document loads, or at some other time such as when a link is activated. HTML's support for scripts is independent of the scripting language."

Scripts, such as the famous javascript, aim at giving pages a better look or at modifying the contents of the document dynamically, allowing more interaction with the user. As they are written in various languages, it is impossible to handle them. We actually compare the layouts of pages without having run the scripts. However, if two pages are similar considering their look in a Web browser that handles scripts, they should be as similar if the scripts are not executed on both pages.

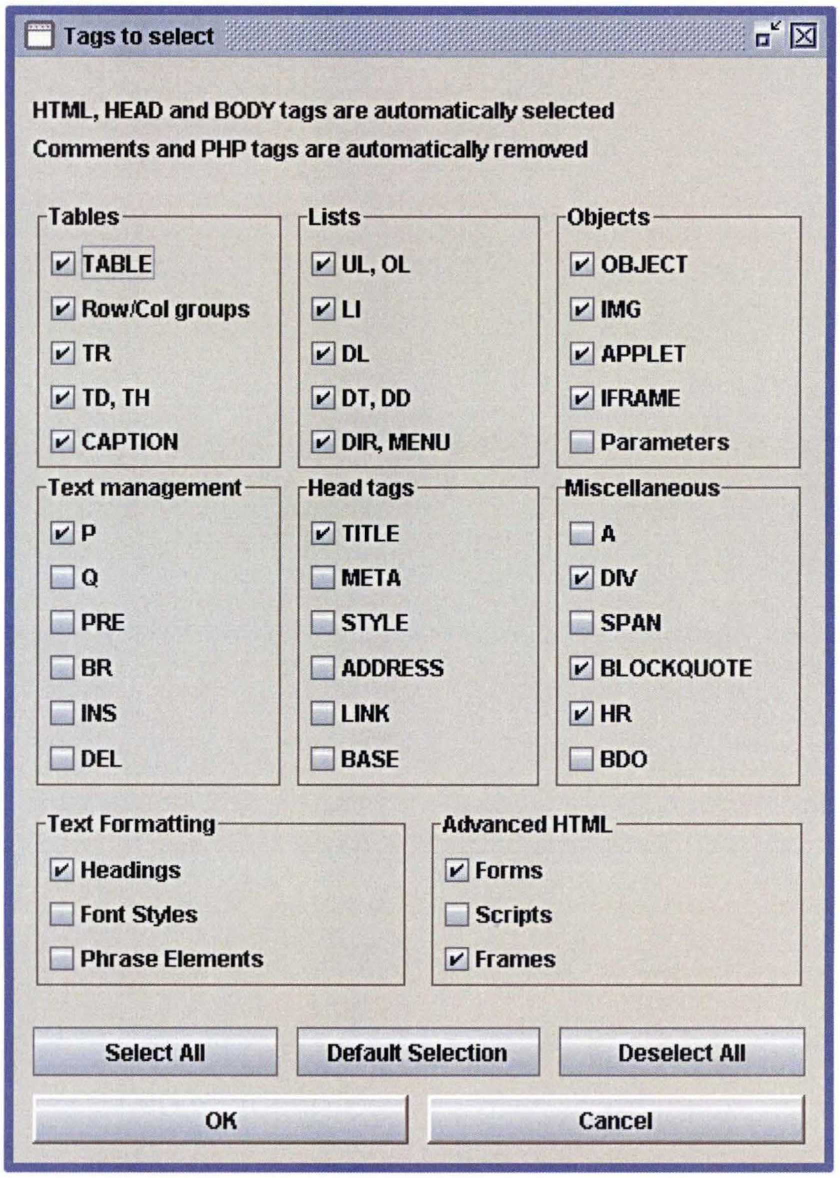
Scripts often represent hundreds of lines in an HTML source. Since we are not able to handle them, we decided to remove tags `<SCRIPT>` and `<NOSCRIPT>` with their contents by default. See also section 11.6.

11.3.14 Frames

"HTML frames allow authors to present documents in multiple views, which may be independent windows or subwindows (...) An HTML document that describes frame layout (called a frameset document) has a different makeup than an HTML document without frames. A standard document has one `HEAD` *section and one* `BODY`*. A frameset document has a* `HEAD`*, and a* `FRAMESET` *in place of the* `BODY`*."* [26]

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
  "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
  <HEAD>
    <TITLE>A frameset document</TITLE>
  </HEAD>
  <FRAMESET cols="33%,66%,33%">
    <FRAME src="contents_of_frame1.html">
    <FRAME src="contents_of_frame2.html">
  </FRAMESET>
</HTML>
```

If a page is structured using frames, the parser does not go further through the source referred by the attribute `src="..."`. The layout tree is then limited to nodes representing the `FRAMESET` and `FRAME` tags, so that the news extraction fails. The tag selection frame has a check box the user can select if he wants the "frameset tree" to be built anyway. See also section 11.6. Finally, the tag selection frame is presented in figure 11.1



HTML, HEAD and BODY tags are automatically selected
Comments and PHP tags are automatically removed

Tables: [✓ TABLE] [✓ Row/Col groups] [✓ TR] [✓ TD, TH] [✓ CAPTION]

Lists: [✓ UL, OL] [✓ LI] [✓ DL] [✓ DT, DD] [✓ DIR, MENU]

Objects: [✓ OBJECT] [✓ IMG] [✓ APPLET] [✓ IFRAME] Parameters

Text management: [✓ P] Q PRE BR INS DEL

Head tags: [✓ TITLE] META STYLE ADDRESS LINK BASE

Miscellaneous: A [✓ DIV] SPAN [✓ BLOCKQUOTE] [✓ HR] BDO

Text Formatting: [✓ Headings] Font Styles Phrase Elements

Advanced HTML: [✓ Forms] Scripts [✓ Frames]

Figure 11.1: *The tag selection frame with the default selection*

11.4 From the source to the layout tree

Once a page has been cleaned up, the construction of the layout tree is rather easy. Thanks to a library provided by the Jericho parser: `public java.util.List findAllElements()` that returns a list of all the elements enclosed between the begin and the end of the source.

The class *Element* represents an HTML *element* (as defined by the W3C, HTML 4.01 specifications, 3.2.1), which encompasses a *StartTag*, an optional *EndTag* and all the contents in between.

> *If the start tag has no corresponding end tag:*
>
> - *If the end tag is optional, the end of the element occurs at the start of the next tag that implicitly terminates this type of element.*
> - *If the end tag is forbidden, the element spans only the start tag.*
> - *If the end tag is required, the source HTML is invalid and the element spans only the start tag. No attempt is made by this library to determine how user agents might interpret invalid HTML. [25]*

The method `findAllElements()` works in a recursive way. It fetches the first element (logically between `<html>` and `</html>`).Then it fetches the first element enclosed in the element `html` (logically between `<head>` and `</head>`). It goes on further by searching the first element enclosed in the element `head` and so on. When an element does not have any enclosed element, it goes one level up and searches for a second enclosed element and so on.

This mean of searching all elements corresponds to a preorder traversal in the HTML source. We can thus write a recursive algorithm that builds the layout tree as the Jericho parser finds the elements.

11.4.1 Pseudo-code of the parser

Main function

```
Create an iterator for all the $elements$ found by jericho.findAllElements();
Get the first $element$ (normally <html> ... </html>);
Create an empty node called $currentNode$;
Call the recursive function $parse\,(element)$;
Return the root node.
```

Recursive function

```
$parse\,(element)$
    Create a new node with the start tag of $element$;
    Add this node as a child of the $currentNode$;
    $currentNode$ := new node;
    WHILE ($element$ encloses the next $element$ in the iterator) DO
        Get this $nextElement$;
        Extract text stored between the start tag of $element$...
        ...and the start tag of $nextElement$;
        $parse(nextElement)$;
    END WHILE
    IF (there was an enclosed element)
    THEN Extract text stored between the end of the enclosed element...
        ...and the end of $element$;
    ELSE Extract text stored between both start and end tags of $element$;
    $currentNode := currentNode.getParent()$ (the $currentNode$ is set on its parent).
```

11.5 From the layout tree to HTML

We shall see later that running the top-down algorithm on two trees T_1 and T_2 outputs a third tree $TopDownTree$ that represents the common nodes between T_1 and T_2.

The more T_1 and T_2 are similar, the bigger $TopDownTree$ will be. As $TopDownTree$ symbolizes the common general layout of T_1 and T_2, it can be interesting to do a "reverse parsing", i.e, parsing the $TopDownTree$ to transform it into HTML code, so that it can be opened by an Internet browser. Notice that this function is of no use for the clustering of Web pages but can intuitively help understand the concept of top-down mapping and similarity between page layouts.

11.5.1 Pseudo-code of the algorithm

Main function

```
Open a String Buffer to write text;
Call $parseTree$ with the root node of $TopDownTree$;
Write the String Buffer in a file.
```

Recursive function

```
$parseTree\ (currentNode)$
        Get the contents of $currentNode$;
                (tag name + attributes if the node is an HTML tag)
                (text if the node represents plain text)
        Append the contents to the String Buffer;
        FOR all the children of the $currentNode$ DO
                $parseTree(child)$;
        END FOR
        IF $currentNode$ is a Start Tag
        THEN    IF this Start Tag requires an End Tag
                THEN    Create an End Tag corresponding to the Start Tag;
                        Append this End tag to the String Buffer;
                ENDIF
        ENDIF
```

11.6 Limits of the parser

Our parser is not perfect. We shall present here three major limits and their possible issues.

Comparing tag names only

We chose to cluster similar pages by comparing tag names in order to build a layout tree that represents the structure of a page. We could also analyse the attributes of the tags which can allow a more accurate comparison between tags. If attributes are replaced by style sheets (.css), the latter should be parsed as well.

Creating a layout tree is not the only solution to detect similar pages. For example, analysing style sheets makes it possible as well.

Scripts

It is well-nigh impossible to cope with the code inside the `scripts` tags in order to understand their effects on the page layout.

Moreover, if a script is responsible of writing the news article, the latter does not appear in the layout tree. Let us consider the following example:

```
<HTML>
       <HEAD>
         <TITLE>News in script</TITLE>
       </HEAD>
       </BODY>
         <SCRIPT type="text/javascript">
           document.write("<p><i>American air force bombed Baghdad<i>")
         </SCRIPT>
       </BODY>
</HTML>
```

As the article is written by the javascript function *write()*, it will not appear in the layout tree as the scripts are deselected by default. Even if scripts were selected, a node *document.write(*`"<p><i>American air force bombed Baghdad<i>"`*)* would be created.

Frames

We saw in section 11.3.14 that a frameset document will be limited to `FRAMESET` and `FRAME`. The algorithm could go further: it could fetch the name of the HTML file inside the attribute `"src"`, download this page and build its layout tree. The layout tree of the framed page could be added as a child of the related node `<FRAME>` in the original layout tree.

11.7 Conclusion

We saw that parsing HTML is not an easy task. Despite all the efforts, the parser is far from perfect. We saw that some limits can be improved, such as parsing the attributes, style sheets or going further through the frames. We analysed each of the 90 tags specified by the transitional.dtd of HTML 4.01 in order to define a default tag selection which aims at improving results of the further algorithms. The main idea of the default tag selection is to keep the block-level tags, to ignore inline-level tags and to remove tags that are not involved in the layout of pages.

The parser can be run in two directions: from an HTML source to its related layout tree or from a layout tree to the HTML source it represents.

We defined in this chapter the structure type that we shall use to cluster similar pages and extract news articles. All the algorithms covered in both next chapters rely on the trees we introduced here.

Chapter 12

News pages clustering

Contents

12.1 Introduction

The approach we have developed for finding and extracting data of interest from Web pages is based on the analysis of the structure of target Web pages. More precisely, by evaluating the structural similarities between pages in a target site we are able to perform tasks such as grouping together pages with similar structure to form page clusters and finding a generic representation of the structure of the pages within a cluster.

Various mappings have been described in chapter 9. To identify similar pages, we use a top-down mapping that finds the common nodes between two layout trees from the root node to the leaves. We shall see that we actually use two top-down mappings:

- A fast mapping: The algorithm is very simplistic and aims at efficiency but do not manage to avoid mistakes.
- A fine mapping: The algorithm is more complicated and much slower but gives better results.

Afterwards, we shall show how clusters are created thanks to the mapping results and how the tag selection affects those results. We shall conclude with an example of clustering with original news Web pages.

12.2 The fast top-down mapping

12.2.1 Pseudo-code of the fast top-down algorithm

Input: T_1 and T_2, both trees to map.
Output: $TopDownTree$, the fast top-down mapping between T_1 and T_2.

```
function $fastMapping\,(T_1, T_2)$
    Get the root node of $T_1$, $rootT_1$;
    Get the root node of $T_2$, $rootT_2$;
    IF $(rootT_1 == rootT_2)$
    THEN Create the root node of $topDownTree$ with $rootT_1$;
            $mapChildren\,(rootT_1\, rootT_2)$;
    ELSE  STOP.


function $mapChildren\,(node_1,\, node_2)$
    Get the array of children of $node_1$, $children_1[\,]$;
    Get the array of children of $node_2$, $children_2[\,]$;
    FOR $(int\ i = 0;\ i < children_1.length;\ i++)$ DO
        Get the child at $children_1[i]$, $child_1$;
        IF ($i < children_2.length$)
        THEN Get the child at $children_2[i]$, $child_2$;
                IF ($child_1 == child_2$)
                THEN Add $child_1$ to $topDownTree$;
                        $mapChildren\,(child_1,\, child_2)$;
                ELSE  Break;
        ENDIF
    ENDFOR
```

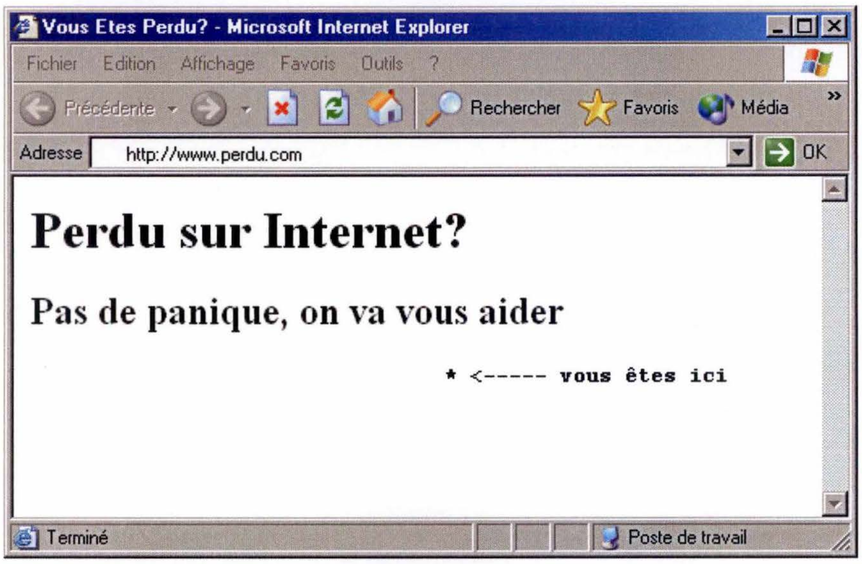
As we can see, the fast top-down mapping algorithm is rather simple. It works by comparing the children of current nodes from left to right recursively and by adding common nodes to $TopDownTree$.

The top-down mapping algorithm gives as output the "top-down tree". This tree is actually not the biggest common part between the trees T_1 and T_2. As the matching between common children stops as soon as different children are found, children that could have been identical after this mismatch are ignored (see section 12.2.3).

12.2.2 Example

So as to understand properly the top-down mapping algorithm here follows a straightforward example. We have downloaded the original Web site *www.perdu.com* and considered its HTML source code (see figures 12.1 & 12.2). In order to display correctly the layout trees, *perdu.com* has been taken as key example on purpose for its simplicity. It is indeed impossible to show the layout tree of a news Web page counting thousands of nodes like a news page from the CNN Web site (www.cnn.com).

As you can see in figure 12.1, this Web site is very elementary. The template of the Web page is basic and the HTML source code very limited. The second layout tree we shall take for our example is the same Web page but slightly changed: the element `<h2>Pas de panique, on va vous aider</h2>` in the HTML source code has been removed. This leads to a new Web page of www.perdu.com without the sentence "Pas de panique, on


Vous Etes Perdu? - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Favoris
Média
Adresse
http://www.perdu.com
OK
Terminé
Poste de travail

Perdu sur Internet?

Pas de panique, on va vous aider

```
* <----- vous êtes ici
```

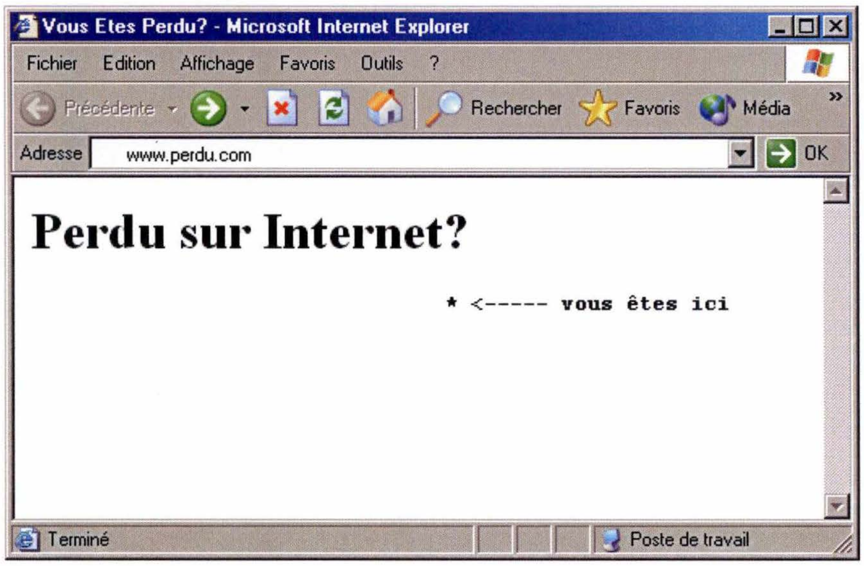
Figure 12.1: *the Web page www.perdu.com*

va vous aider" in the middle of the page; and to a new corresponding HTML source code (see figures 12.3 & 12.4).

```
<html>
  <head>
    <title>
      Vous Etes Perdu?
    </title>
  </head>
  <body>
    <h1>Perdu sur Internet?</h1>
    <h2>Pas de panique, on va vous aider</h2>
    <pre> * <----- vous &ecirc;tes ici</pre>
  </body>
</html>
```

Figure 12.2: *HTML source of www.perdu.com*

As we have the Web pages we want to map and their corresponding HTML source code, we are now able to transform them into layout trees thanks to our program through the HTML parsing. Figure 12.5 shows how the trees are actually displayed in our program. Tree $perdu_1$ corresponds to the original Web page of www.perdu.com and tree $perdu_2$ to the slightly changed Web page of the same Web site (figure 12.5).


Vous Etes Perdu? - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Favoris
Média
Adresse www.perdu.com
OK
Terminé
Poste de travail

Perdu sur Internet?

```
* <----- vous êtes ici
```

Figure 12.3: *Modified version of www.perdu.com*

```
<html>
  <head>
    <title>
      Vous Etes Perdu?
    </title>
  </head>
  <body>
    <h1>Perdu sur Internet?</h1>
    <pre> * <----- vous &ecirc;tes ici</pre>
  </body>
</html>
```

Figure 12.4: *Modified source code of www.perdu.com*

The next operation is the top-down mapping properly speaking. Both trees $perdu_1$ and $perdu_2$ are taken as input of the top-down mapping algorithm. This one identifies the common part of both trees and gives as output the *top-down tree* (see figure 12.6).

12.2.3 Limits of the fast top-down mapping

First the fast top-down algorithm is too restrictive because it only tries to match common nodes from left to right and stops when a mismatch occurs. In figure 12.6, both `<pre>` elements on the right are ignored although they are identical. If we consider the Web

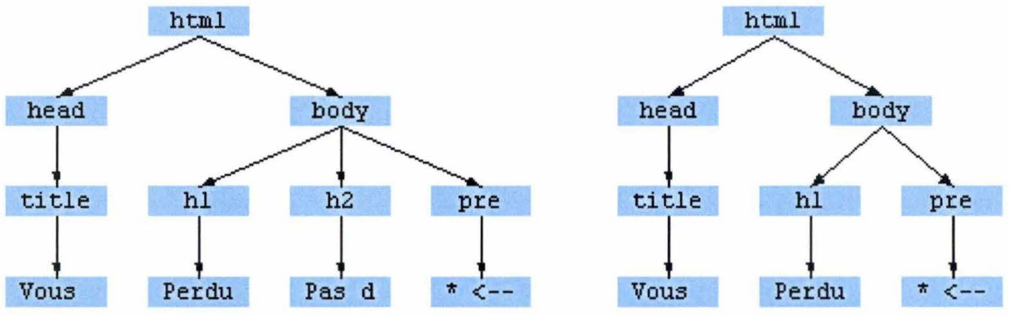


Figure 12.5: *perdu*$_1$ *and predu*$_2$ *trees*

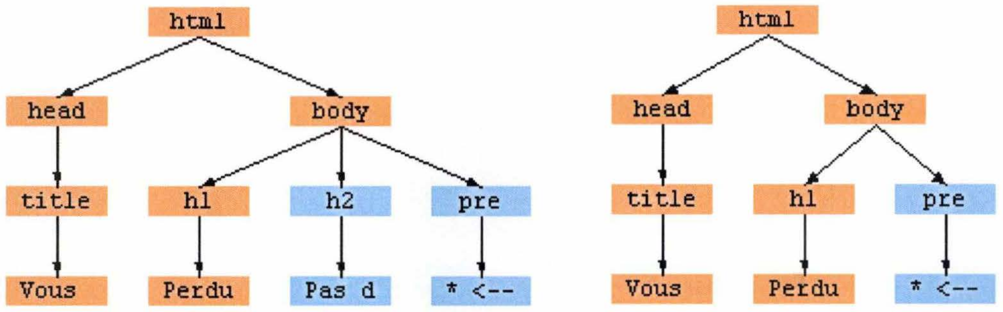


Figure 12.6: *The common part of perdu*$_1$ *and perdu*$_2$ *(orange)*

pages as they are displayed (see figures 12.1 & 12.3), we can consider that the bottoms of both pages are similar. The presence of the extra title "Pas de panique, on va vous aider" does not affect the layout below it. So that we should try to match all the common children when we are analysing two nodes instead of matching them from left to right.

Second, the fast top-down algorithm can make some mistakes. To understand an example of mistake, let us consider figure 12.7. On the right tree, an extra `<table>` fools the algorithm which does not match the corresponding `table` and drops a large set of common nodes.

Finally, the output of $fastMapping\,(T_1, T_2)$ can be different from the output of $fastMapping\,(T_2, T_1)$.

We need to improve the fast top-down algorithm by comparing all the children whatever their position and by avoiding matching "noncorresponding tags".

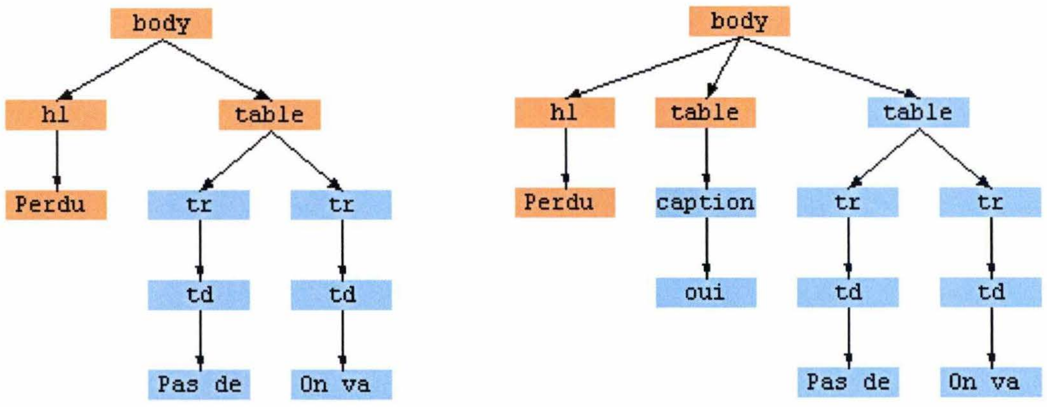


Figure 12.7: *Limits of the fast top-down algorithm*

12.3 The fine top-down mapping

The main idea is to consider nodes according to the size of the children subtrees. Instead of comparing only the children names, the fine top-down mapping takes the size of the subtrees into account in a bid to avoid mistakes. The children are not treated from left to right; they are sorted in descending order of the size of their subtree. Let us explain with an example: consider figure 12.5 and let us assume we are treating the `body` node. Instead of matching children `H1`, `TABLE`, `TABLE` from left to right, we first sort them considering their subtrees. In the left tree, subtrees of `H1` and `TABLE` count respectively 2 and 7 nodes. In the right tree, subtrees of `H1`, the first `TABLE` and the second `TABLE` count respectively 2, 3 and 7 nodes. The subtrees are sorted in descending order, so that the children can be represented like this:

- Left tree, node body, children = `(TABLE-7, TABLE-3, H1-2)`
- Right tree, node body, children = `(TABLE-7, H1-2)`

The algorithm first tries to match `TABLE` and `TABLE`. As it works, it tries to match `TABLE` (the next one) and `H1`, which fails. Since `TABLE` has a bigger subtree than `H1`, the algorithm searches for an instance of TABLE in the second tree, which fails. Finally, it tries to match H1 with H1, which succeeds. Notice that this example was intended for explaining how to handle a set of children. The algorithm actually works recursively when a matching is found, so that normally, after having compared both first TABLE, it goes recursively to both TR nodes, then TD nodes and so on.

Thanks to this technique, the probability of matching noncorresponding nodes is reduced, especially when the tree is large and when the nodes are closer to the root. When there are many children, each having a small subtree, the mapping can still be fooled. Anyway, the objective is to guarantee a matching as best as possible on the high-level nodes. A mistake made close to the leaves, where the subtrees are smaller, matters less than a mistake made close to the root since the second drops a bigger subtree.

As the order of the children does not matter, every child that has its corresponding node in the other tree is matched, whatever the extra "single" children. Notice that calling the function $(fineMapping\,(T_1,\,T_2)$ or $fastMapping\,(T_2,\,T_1))$ gives the one output. Let us consider figure 12.6: if the fine top-down mapping was applied to these trees, the results would be completed by matching both nodes PRE and both nodes $*$<--.

The fine top-down algorithm is more accurate than the fast one but should be utilized only if the fast top-down algorithm does not give suitable results. Most of the time, the results of both algorithm will be the same but the fast top-down algorithm, as it is called, is quite faster.

12.4 Pseudo-code of the fine top-down algorithm

Input: T_1 and T_2, both trees to map.

Output: $TopDownTree$, the largest common tree between T_1 and T_2.

The function that calls $mapChildren$ is the same as the one presented in section 12.2.1

```
function $mapChildren\,(node_1,\ node_2)$
    Get the array of children of $node_1$, $children_1[\,]$;
    Get the array of children of $node_2$, $children_2[\,]$;
    Sort $children_1[\,]$ in descending order of the sizes of the children subtrees;
    Sort $children_2[\,]$ in descending order of the sizes of the children subtrees;
    WHILE $children_1[\,]$ and $children_2[\,]$ have non treated children DO
        Get the next child in $children_1[\,]$ $child_1$;
        Get the next child in $children_2[\,]$ $child_2$;
        IF $(child_1 == child_2)$
        THEN Add $child_1$ to $topDownTree$;
             $mapChildren\,(child_1,\ child_2)$;
        ELSE
             IF (size of the subtree $(child_1)$ $\geq$ size of the subtree $(child_2)$)
             THEN Search in $children_2[\,]$ for a node that matches $child_1$;
             ELSE Search in $children_1[\,]$ for a node that matches $child_2$;
    END WHILE
```

12.5 News pages clustering

Now that the structural similarities between two Web pages can be identified thanks to our top-down mapping algorithms, it is possible to put various Web pages having structural similarities together to form clusters. At this point, we faced two problems: the top-down mapping algorithm allows us to map only two layout trees at once while the clustering task needs to map many layout trees and the percentage of similarity is not yet known. For these two reasons, we created a clustering algorithm that can apply one of the top-down mapping algorithms to map an unlimited set of trees and we defined a similarity rate.

12.5.1 Similarity rate and similarity threshold

In order to cluster Web pages, we need to specify a similarity rate. This rate is the number of identical nodes between two trees compared with the number of nodes of the largest tree. It is mathematically defined as follows:

$$similarity\ rate = \frac{size\,(\,TopDownTree\,)}{max\,(\,size(T_1)\,,\,size(T_2)\,)}$$

where $TopDownTree$ is the result of the top-down mapping between T_1 and T_2, both trees to map.

The similarity rate is based on the number of nodes of the trees T_1, T_2 and $TopDownTree$. As we are working on trees having hundreds of nodes, we deem that it is rather a correct value. The reason why we chose the maximum size between both trees as denominator is straightforward: if two trees have a high-level of similarity, the top-down tree resulting of their mapping will be large and the choice of the maximum size between T_1 and T_2 does not really matter. But if they have a low-level of similarity, the size of the top-down tree will be small. So if we take the size of the largest tree as denominator, the ratio will be very small as well, so that both trees are not likely to be clustered.

While the similarity rate has been defined, we must now estimate a similarity threshold, that is a value from which similar Web pages are accepted within the same cluster. This threshold has to be efficient, that is it has to be restrictive enough in order to put together in the same cluster pages which are very similar to each other. Yet, it must not be too restrictive to allow some structural differences.

12.5.2 Impact of the tag selection

To illustrate how the tag selection (see chapter 11) affects the results, let us take figure 12.8 as key example. In this example, all the tags have been transformed into nodes but we saw in chapter 11 that we can actually ignore some tags during the parsing, which leads to reducing the amount of nodes.

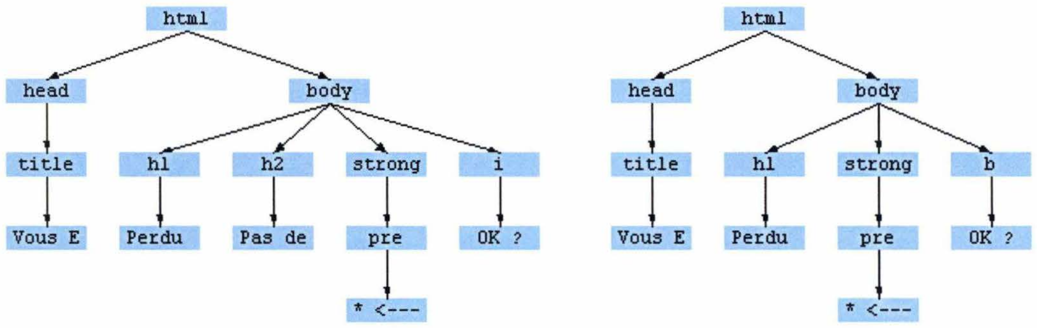


Figure 12.8: *Key example to explain the tag selection impact*

Henceforth we shall analyse two scenarios: the first one assumes that all the tags were selected during the parsing and the second one assumes that the default tags selection has been applied. As we have already shown the results of a fast top-down mapping, this time the scenarios are covered using the fine top-down mapping.

All tags selected

In this case, the left tree counts 13 nodes and the right tree counts 11 nodes. The fine top-down tree (colored in orange) counts 8 nodes. The similarity rate is $\frac{8}{max\,(13,11)} = 0,615$.

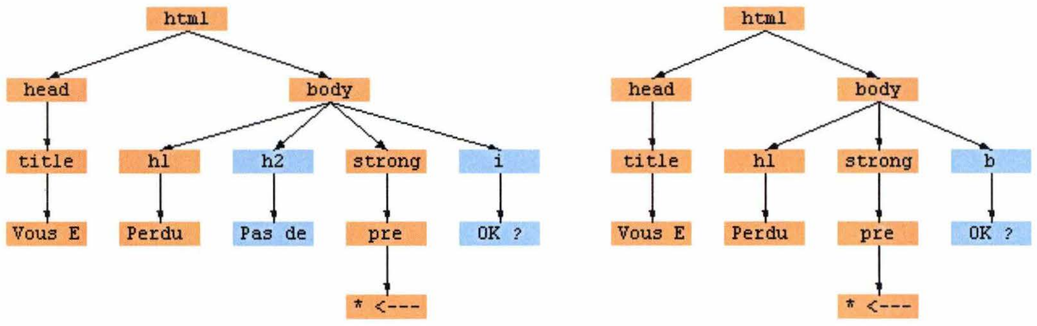


Figure 12.9: *Results of a fine top-down mapping if all nodes are selected*

Default tag selection

With the default tag selection, tags <PRE>, <B>, and <I> have been ignored, so that their contents have been put together (see the source code frame). In this case, the left layout tree counts 9 nodes and the right layout tree counts 7 nodes. The fine top-down mapping (colored in orange) has totally mapped the right layout tree, it counts 7 nodes as well. The similarity rate is $\frac{7}{max\,(7,9)} = 0,778$.

This demonstrates the benefit of the default tag selection. By removing nodes that are not involved in the block-level structure of the page, we can cluster similar pages in a better way.

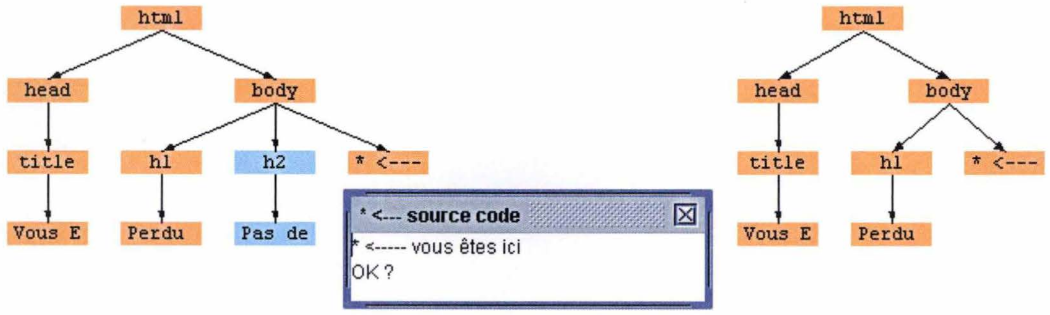


Figure 12.10: *Results of a fine top-down mapping with default selection*

12.5.3 The clustering algorithm

As previously explained, the clustering is the operation taking all the Web pages we need to analyse in order to group them together into clusters depending on their similarity rates. All the Web pages having structural similarities higher than the *similarity threshold* value will be put together into the same cluster.

To perform this task, we created the clustering algorithm. This one can use the fast or the fine top-down mapping algorithms. The user first chooses a tag selection model in the tag selection frame (or he can use the default selection model). Then he selects the pages he wants to cluster, a cluster threshold and a fast or fine mapping. Each page is then converted into its related layout tree and the latter are stored on a vector.

Pseudo-code of the clustering algorithm

Input: $treeVector$ (a vector that contains all the layout trees) and $simThreshold$ (the similarity threshold above which pages are clustered).

```
Create a vector to store the similarity rate after each top-down mapping: $tdrVector$;
(Each element of $tdrVector$ represents a triplet $[tree_i, tree_j, similarityRate_{ij}]$)
FOR each tree $tree_i$ in $treeVector$ $(0 \leq i \leq treeVector.length - 1)$ DO
        FOR each tree $tree_j$ in $treeVector$ $(i + 1 \leq j \leq treeVector.length)$ DO
                Run a top-down mapping (fast or fine) between $tree_i$ and $tree_j$;
1               Calculate the similarity rate between $tree_i$ and $tree_j$ : $similarityRate_{ij}$;
                Add the triplet $[tree_i, tree_j, similarityRate_{ij}]$ to $tdrVector$;
        END FOR
END FOR
Sort $tdrVector$ in descending order of $similarityRate_{ij}$;
Get the first triplet $[tree_i, tree_j, similarityRate_{ij}]$ in $tdrVector$);
IF $(similarityRate_{ij} < simThreshold)$ THEN STOP.
Create a new cluster containing $tree_i$ and $tree_j$;
FOR each triplet $[tree_i, tree_j, similarityRate_{ij}]$ in $tdrVector$ DO
        IF $(similarityRate_{ij} \geq simThreshold)$
        THEN IF ($tree_i$ is not yet clustered)
                THEN Create a new cluster with $tree_i$;
                ELSE IF ($tree_j$ is not yet clustered)
2                       THEN Create a new cluster with $tree_j$;
                FOR each tree $tree_k$ of $treeVector$ that is not yet clustered DO
                        Get the $similarityRate_{ik}$ corresponding to $tree_i, tree_k$ in $tdrVector$;
                        IF $(similarityRate_{ik} \geq simThreshold)$
                        THEN Put $tree_k$ into the current cluster;
                END FOR
        ELSE STOP.
END FOR
```

Output: A set of clusters, each containing pages at least $simThreshold$ % similar.

The first loop (1) aims at calculating the similarity rates between all the trees combinations. Since the mapping algorithms can be run only with two trees at once, we need to generate all the different "mapping combinations". The algorithm takes the first tree of $treeVector$, applies the top-down mapping algorithm to all the other following trees of $treeVector$ and for each triple $(Tree_1, Tree_2, TopDownTree)$, calculates the similarity rates. It also puts the similarity results and its associated trees into the $tdrVector$. Then it takes the second tree of $treeVector$, applies the top-down mapping algorithm to all the other following trees and so on.

Before entering the second loop , the clustering algorithm sorts the $tdrVector$ in descending order of similarity rate. Then, if the first element of the top-down mapping results (so the best result) has a similarity lower than the similarity threshold, all the trees are a fortiori dissimilar and the algorithm stops. On the contrary, if the similarity rate of the best top-down mapping is higher than the threshold, both trees involved in this result are put together in a first cluster.

The second loop (2) goes on for each couple of trees in $tdrVector$ associated to a similarity result higher than the similarity threshold. It creates a new cluster or puts trees into an already existing cluster depending on their similarity results.

The output is a set of clusters, each containing pages at least $simThreshold$ % similar.

12.5.4 Application

To illustrate the clustering algorithm, here follows a complete example. We have chosen a set of Web pages coming from the World Wide Web. Some are news Web pages coming from well-known news Web sites such as Le Soir, CNN, etc, some are coming from other Web sites such as perdu.com, Webkot.be, etc. Some have nothing to do with each other, some present at the first glance high-level structural similarities (see figure 12.11).

We actually selected these pages in order to cover some important scenarios:

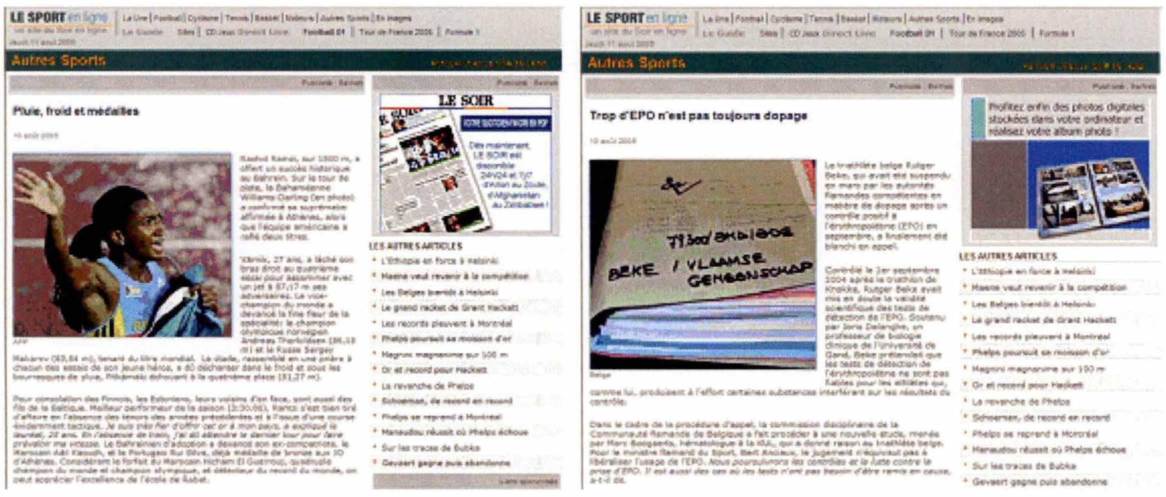
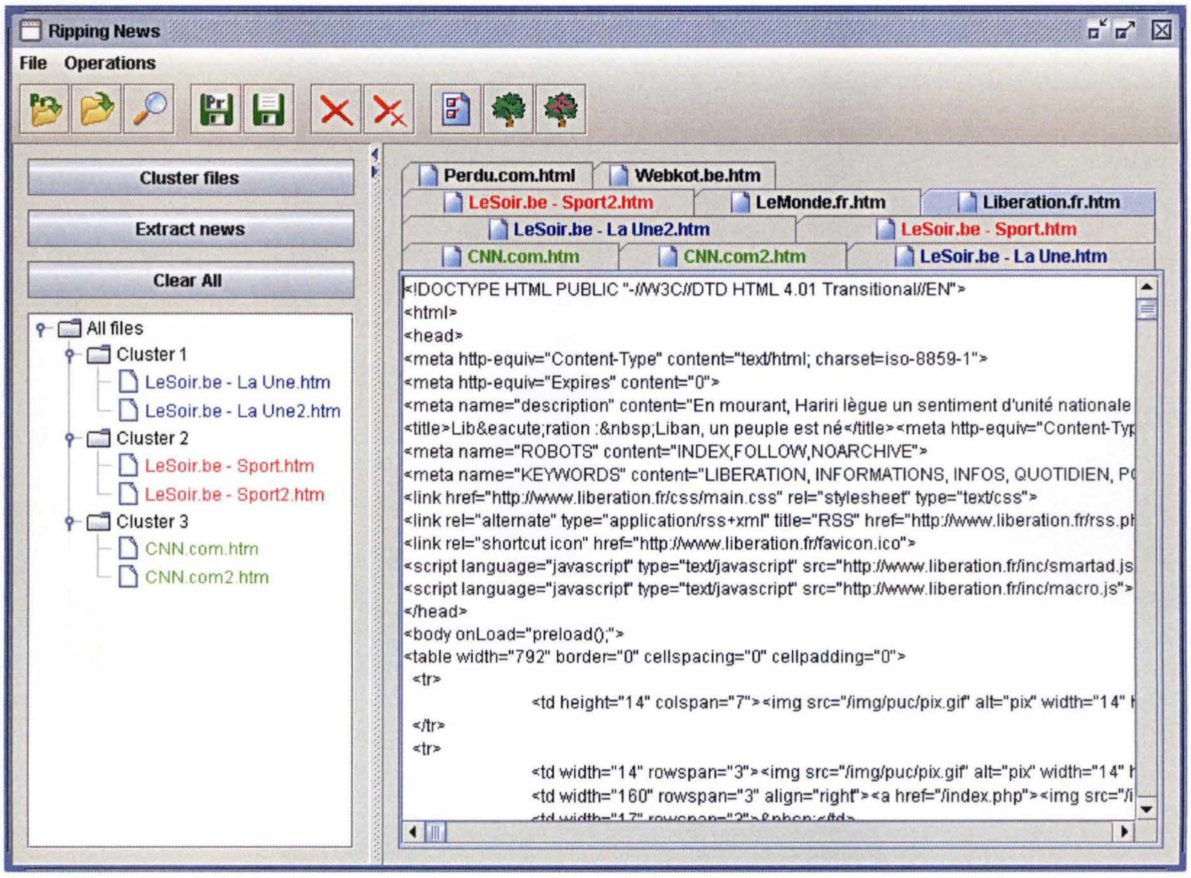


Figure 12.11: *two sport news Web pages from www.LeSoir.be showing a high degree of similarity*

- For the first scenario, we fetched four news pages coming from LeSoir.be. Two from these four pages present sports news and the two others are on the front page. Normally, the first two pages should be grouped together into one cluster and the last two others into another one.
- In the second scenario, we downloaded two news pages from CNN.com. They should be clustered together.
- The third scenario takes into account four pages having nothing to do with each other. These four Web pages come from LeMonde.fr, Liberation.fr, Perdu.com and Webkot.be. They should not be clustered.
- The last scenario is the consequence of the three previous ones. Each of the three scenarios must not interfere with each other.

The first step is to choose the Web pages we want to cluster, to choose a fast or fine top-down mapping and a similarity threshold (see User Guide for details). For our experiment, we selected all the opened files, a fast top-down mapping and a similarity threshold of 60%.

The results are as expected (see figure 12.12). The first scenario is covered since both sports news Web pages are in the same cluster (Cluster2) as well as both news being on the headlines (Cluster1). The second case is correct: both CNN news Web pages are in the same cluster (Cluster3). The third scenario has been respected: the four distinct pages were not clustered. Finally, the non-interference condition of the fourth scenario has been covered as well.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Expires" content="0">
<meta name="description" content="En mourant, Hariri lègue un sentiment d'unité nationale
<title>Lib&eacute;ration :&nbsp;Liban, un peuple est né</title><meta http-equiv="Content-Typ
<meta name="ROBOTS" content="INDEX,FOLLOW,NOARCHIVE">
<meta name="KEYWORDS" content="LIBERATION, INFORMATIONS, INFOS, QUOTIDIEN, P
<link href="http://www.liberation.fr/css/main.css" rel="stylesheet" type="text/css">
<link rel="alternate" type="application/rss+xml" title="RSS" href="http://www.liberation.fr/rss.ph
<link rel="shortcut icon" href="http://www.liberation.fr/favicon.ico">
<script language="javascript" type="text/javascript" src="http://www.liberation.fr/inc/smartad.js
<script language="javascript" type="text/javascript" src="http://www.liberation.fr/inc/macro.js">
</head>
<body onLoad="preload();">
<table width="792" border="0" cellspacing="0" cellpadding="0">
  <tr>
                <td height="14" colspan="7"><img src="/img/puc/pix.gif" alt="pix" width="14" h
  </tr>
  <tr>
                <td width="14" rowspan="3"><img src="/img/puc/pix.gif" alt="pix" width="14" h
                <td width="160" rowspan="3" align="right"><a href="/index.php"><img src="/i
```

Figure 12.12: *The four scenarios have been covered*

12.6 Conclusion

Two top-down mapping algorithms have been implemented to identify similarities between layout trees, from the root to the leaves. The fast top-down mapping is an efficient algorithm, giving satisfying results most of the time but which can be easily fooled. To compensate for this weakness, we developed the fine top-down mapping algorithm, which is more accurate but slower. It actually finds the largest common subtree between two trees.

We saw how the tag selection has an impact on the mapping algorithms. Thanks to this tag selection and the clustering, the extraction of the news articles related to the layout trees that were clustered together is now made easier (see next chapter).

Chapter 13

Extracting the news articles

Contents

13.1 Introduction

This is the last step of our extraction process. We assume here that:

- the Web pages have been transformed into layout trees with the same tag selection options;
- the tag selection was complete enough to build layout trees that reflect the layout of the Web pages;
- the similarity threshold was significant enough to detect (dis)similarities between pages.

This last step is the easiest. Once the similar layout trees have been clustered, the extraction of the news article can be summarized as follows:

- get all the layout trees from a chosen cluster (the cluster must have at least two elements);
- for each layout tree, fetch all the leaves that contain text (section 13.2);
- remove the "text leaves" that are common between layout trees (section 13.3);
- for each layout tree, concatenate all the remaining leaves to a text file (section 13.4).

Each of this four operations will be explained in a separate section. We shall finally introduce how we could extract the news article with its formatting thanks to slight changes in the parser.

13.2 Fetching the leaves that contain text

For each layout tree in the same cluster, a preorder traversal[1] fetches all the leaves in the right order. When a leaf is found, we must check whether it is a node representing a tag or a node containing text. We only keep nodes that contain text. Notice that, at this time, we extract the "text leaves" themselves and not their contents. At the end of this process, the output is a vector containing all the "text leaves".

This approach can be compared with the RoadRunner [18] (see chapter 7) algorithm that fetches all the #PCDATA elements (i.e. everything that is not an HTML tag). A text leaf in our case would be a #PCDATA in RoadRunner.

13.3 Removing common text leaves

This step is a bit harder. We need to compare in twos by two all the text leaves in all the text leaves vectors. The leaves that are common are eliminated from their vector, so that at the end of the process all the leaves are different, notwithstanding the vector they belong to.

[1]A postorder (or depth-first) traversal is also suitable but not a breadth-first traversal.

Pseudo-code of the algorithm

Here is the pseudo-code of the algorithm, $textLeavesVectors[\,]$ is an array of text leaves vectors.

```
FOR $(int\ i = 0;\ i < textLeavesVector[\,].length;\ i++)$ DO
    Get the $textLeavesVector$ stored at index $i$ (called $vectorOnI$);
    FOR $(int\ j = i + 1;\ j < textLeavesVector[\,].length;\ j++)$ DO
        Get the $textLeavesVector$ stored at index $j$ (called $vectorOnJ$);
        FOR every text leaf $tfI$ in $vectorOnI$ DO
            FOR every text leaf $tfJ$ in $vectorOnJ$ DO
                IF $(tfI == tfJ)$
                THEN   Remove $tfJ$ from $vectorOnJ$;
                       Set $tfI$ to be removed at the end of this loop;
            ENDFOR
            IF $tfI$ is set to be removed
            THEN Remove $tfI$ from $vectorOnI$;
        ENDFOR
    ENDFOR
ENDFOR
```

13.3.1 Complexity

Both first loops are designed to compare in twos all the text leaves vectors with each other. If there are four text leaves vectors A, B, C, D, it will compare $A - B$, $A - C$, $A - D$, $B - C$, $B - D$ and $C - D$.

Both inside loops are designed to compare in twos all the text leaves from both current vectors. They remove the text leaves that are identical.

The theoretical complexity of this algorithm is very high, $O(n^2 \cdot m^2)$, where n is the number of text leaves vectors and m is the number of text leaves. The practical complexity is lower: both first loops are repeated respectively $(n - 1)$ and $(n/2)$ times. So that in the example with four vectors we have 6 loops instead of 16. Both inside loops are executed faster as the algorithm progressively goes on because the amount of text leaves decreases.

However, this algorithm is too trivial and should be improved for a better efficiency.

13.3.2 Assumptions

If we remove common text leaves, we assume that all the news articles in a cluster do not have common text. This assumption becomes restricting in two cases:

- As the amount of pages in the cluster grows. Moreover, as the complexity of the algorithm is high, it is better to prevent clusters to be too big. This can easily be done by increasing the similarity threshold before the clustering (see chapter 12).
- As the amount of words in the text leaves decreases. This can be prevented by ignoring the tags in charge of the text formatting during the parsing (see section 11.3). Let us explain with an example:

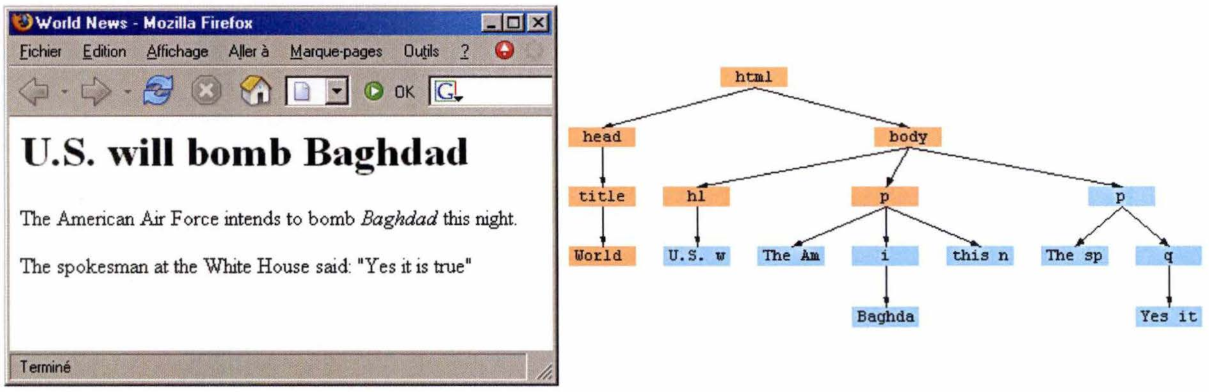
13.3.3 Example

Let us consider two trivial news articles (figure 13.1 and figure 13.2)

Both trees have been built by selecting all tags. Let us assume that both trees have been clustered (the top-down tree is in orange).

If the tag `<i>` that emphasizes the word "Baghdad" is kept, a leaf containing "Baghdad" is created in both trees. Since the leaves are identical, they will be removed during the text extraction, which fails.

On the contrary, if tags `<i>` and `</i>` are ignored during the parsing (thanks to the default tag selection), leaves will be created with the whole sentence below the parent `<p>` (figure 13.3). By the way, notice that the fast top-down mapping gave better results while tags that format text are ignored.

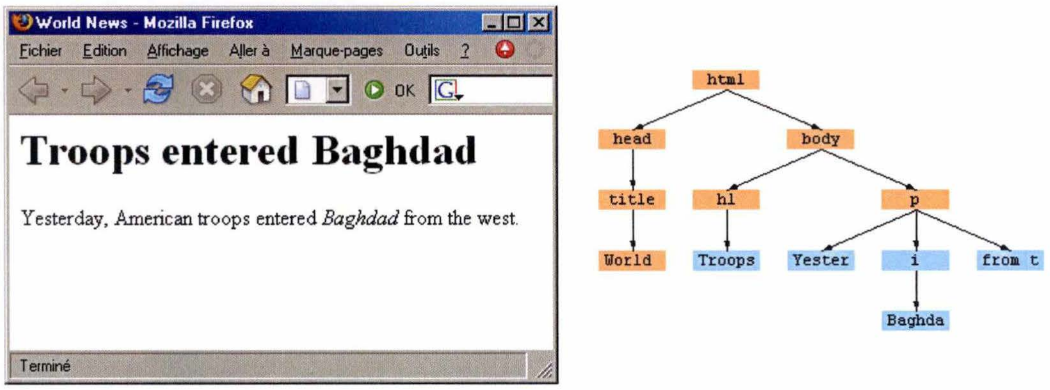

World News - Mozilla Firefox
Fichier Edition Affichage Aller à Marque-pages Outils ?
OK
Terminé
html
head
body
title
h1
p
p
World
U.S. w
The Am
i
this n
The sp
q
Baghda
Yes it

U.S. will bomb Baghdad

The American Air Force intends to bomb *Baghdad* this night.

The spokesman at the White House said: "Yes it is true"

Figure 13.1: *The first article and its related tree (all tags selected)*


World News - Mozilla Firefox
Fichier Edition Affichage Aller à Marque-pages Outils ?
OK
Terminé
html
head
body
title
h1
p
World
Troops
Yester
i
from t
Baghda

Troops entered Baghdad

Yesterday, American troops entered *Baghdad* from the west.

Figure 13.2: *The second article and its related tree (all tags selected)*

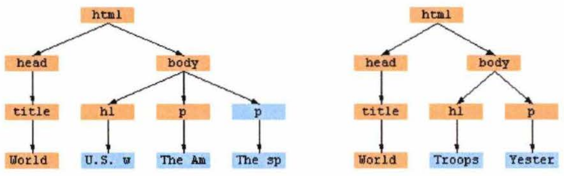


Figure 13.3: *Both layout trees with the default tag selection*

13.4 Concatenation of the remaining leaves

Once the text leaves that were shared with other pages in the cluster have been removed, the remaining text leaves are likely to be the news article.

For each page in the cluster, the remaining text leaves are appended in a text file, so that the extracted article (in plain text) can be stored.

Yet, the text leaves sometimes need a cleaning-up before appending them in order to display the article correctly. We actually want to lay out the article as best as possible despite the plain text format.

In order to achieve this objective, the algorithm:

- removes useless spaces between lines or words;
- transforms the characters coded between & and ; (such as &ecirc; for $\hat{e}$) into their corresponding value in ANSI format;
- must identify paragraphs and headings. The parent of the text leaf must be taken into account:
  - If the parent is a `<P>` node, we found a paragraph and a blank line is inserted;
  - if the parent is a `<TITLE>` or a `<H1>` to `<H6>`, we found a heading and it is written in upper-case.

That explains the reason why we needed to keep the tree node type until the last step of the extraction. If the text leaves were transformed into strings, we would not be able to detect paragraphs and headings, which are essential for the understanding of the article.

The extracted text should be laid out like this:

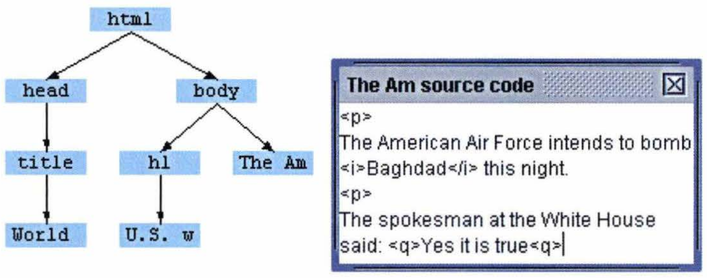
```
U.S. WILL BOMB BAGDAD

The American Air Force intends to bomb Baghdad this night.

The spokesman at the White House said:Yes it is true
```

The leaves below `<title>` are identical so that they have been removed. They actually do not represent the title of the article but the name of the Web site. On the contrary, leaves below `<h1>` have been kept and put in upper-case. A blank space has also been added before each paragraph.The text formatting on "Baghdad" and "Yes it is true" has been dropped.

13.5 Extracting the article with its formatting

The goal of this thesis is to extract a news article without its formatting. However, the news article can be extracted although keeping its layout easily.

During the parsing of an HTML page, we saw that tags responsible of the text formatting are ignored; i.e. both start and end tags are removed but the text between them is kept. These tags could be completely ignored, i.e. they could be considered as text, so that they would be included in the text leaves (see figure 13.4). The extracted text would consequently contain the HTML tags that format it. Yet in this case, the extracted text can be read only by an agent that interprets HTML code.



```
<p>
The American Air Force intends to bomb
<i>Baghdad</i> this night.
<p>
The spokesman at the White House
said: <q>Yes it is true<q>
```

Figure 13.4: *The same tree where tags* `<p>`, `<i>` *and* `<q>` *have been considered as text*

Then, the output would be:

```
U.S. WILL BOMB BAGDAD
<p>
The American Air Force intends to bomb <i>Baghdad</i> this night.
<p>
The spokesman at the White House said:<q>Yes it is true</q>
```

13.6 Conclusion

We saw that we simply extract the news article by fetching the leaves that contain text and by removing text leaves that are common between layout trees in a given cluster. As the amount of layout trees in the cluster grows, the extraction is slower but the results are better, on condition that the text leaves representing the article are significative enough. Actually, text leaves which contain words instead of sentences are likely to be found in more than one layout tree, so that they unfortunately would be removed. This is the reason why it is important to ignore most of the tags that format text when the HTML source is transformed into a layout tree.

After the extraction, the article needs a cleaning-up before storing or displaying it. Even if we aimed at extracting plain text, it is necessary to give the article a minimal layout in order to keep it understandable.

Conclusion

The extraction of news articles from the Internet, as we understand it, is far beyond a simple retrieval of the main text in a given news Web page. We wanted an agent to be able to extract any news article about any topic without any prerequisite processing. To be considered as fully automatic, the process must be covered from A to Z. The extraction process should start with the classification of both Web sites and Web pages of interest. Once the relevant pages have been downloaded, the HTML source code must be transformed into a given data structure. The latter allows to cluster similar pages (i.e. pages having a common layout) before extracting the news articles.

In the literature, tools that make the extraction of data possible, such as wrappers, assume nevertheless that similar pages have been downloaded beforehand. In our opinion, agents should not have to search for relevant pages themselves, or worse, to cluster similar pages manually. We saw that various approaches have been developed in order to classify sites and pages on the Internet. Although the Web sites classification is not a big hit, it could be useful to reduce the search space dramatically, improving the results of the Web pages classification.

News Ripper, our application, does not deal with classification but manages to cluster similar pages. In order to group pages together according to their similarities, we had to build a data structure that mirrors their layouts, the layout tree. This tree is built along the parsing of an HTML source by keeping only tags related to the block-level layout of a page and by ignoring tags involved in the inline-level formatting. We saw that this tag selection conspicuously improves both efficiency and effectiveness of the clustering algorithms.

Two top-down mappings algorithms have been implemented to identify similarities between layout trees, from the root to the leaves. The fast top-down mapping is an efficient algorithm, giving satisfying results most of the time but which can be easily fooled. To compensate for this weakness, we developed the fine top-down mapping algorithm, which is more accurate but slower. It actually finds the largest common subtree between two layout trees.

The extraction of the news articles related to the layout trees that were clustered together is now made easier thanks to the tag selection and the clustering algorithms. A smart tag selection, that ignores text formatting, allows to reduce the amount of leaves that make up the article. The clustering of similar pages, as for it, is necessary to identify both common and uncommon leaves between similar layout trees. Given the assumption that articles from news pages are not likely to have common sentences, we can extract the news articles within a cluster by concatenating the contents of the dissimilar leaves in each layout tree.

The news article, in plain text format, can be eventually conveyed to agents for further use.

Despite our infatuation with News Ripper, each of the three steps of our application can be enhanced. We only parse the tag names of an HTML source although its attributes and style sheets could bring more exhaustive information. If the fast top-down mapping algorithm has been fooled, the user should be warned instead of having to check the output manually. The news extraction algorithm has a high degree of complexity because it compares all the combinations of leaves between layout trees within a cluster.

Anyway, the crucial enhancement would be the coverage of the Web classification, so that an agent would not have to fetch pages of interest by himself. Once this upgrade combines with News Ripper, the whole news extraction process will be deemed as fully automatic.

Appendix A

News Ripper User Guide

A.1 Introduction

News Ripper is the application we developed in JAVA during our internship at the University of Technology in Sydney. This program allows the user to select a set of news Web pages, to cluster them and to extract the news article inside each page. Here follows an exhaustive description of all its functionalities.

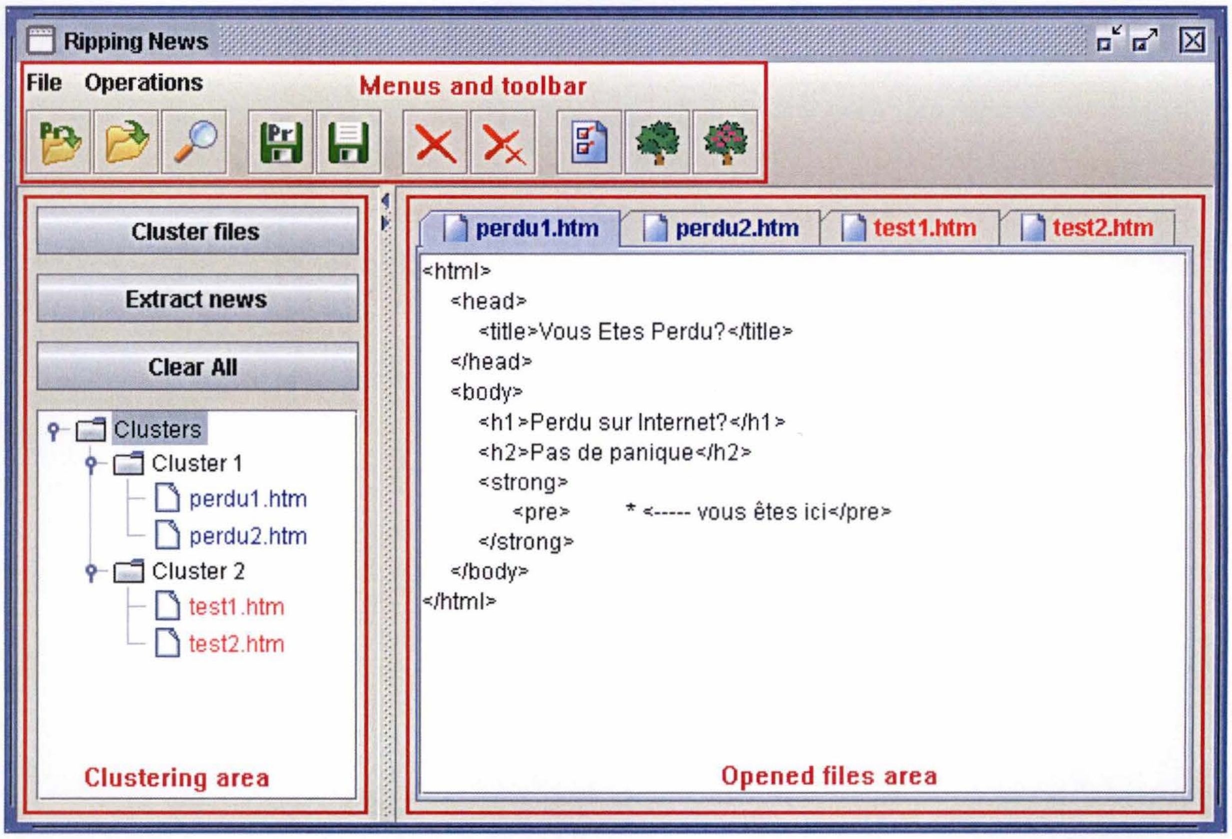
A.2 Launching News Ripper

To run the application, you have to launch the batch file called NewsRipper.bat. The main window will then be displayed in the centre of the screen.

A.3 Main window

The main window (figure A.1) is made up of three main elements:

1. the menus and the tool bar;
2. the Opened files area;
3. the Clustering area;



```
<html>
   <head>
      <title>Vous Etes Perdu?</title>
   </head>
   <body>
      <h1>Perdu sur Internet?</h1>
      <h2>Pas de panique</h2>
      <strong>
         <pre>     * <----- vous êtes ici</pre>
      </strong>
   </body>
</html>
```

Figure A.1: *The main window of News Ripper*

A.4 Menus & toolbar

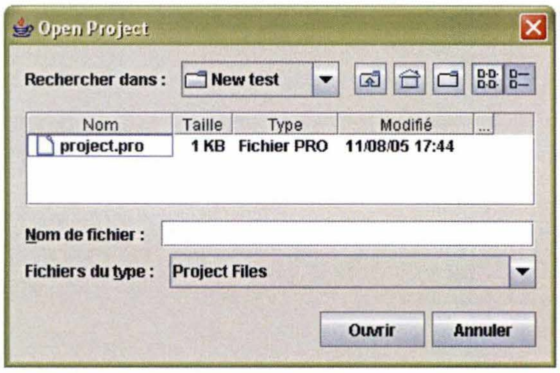
A.4.1 File

Open Project...

Opens an *Open Project* dialog box (figure A.2, left) allowing to open a project file (.pro) beforehand saved. A project is a set of HTML files and a set of clusters. Thanks to the projects, the user can save the results of a clustering for further use.

Open File(s)...

Opens an *Open File(s)* dialog box (figure A.2, right) allowing to open one or several HTML files. The HTML source code of the selected files is then displayed in the Opened files area.



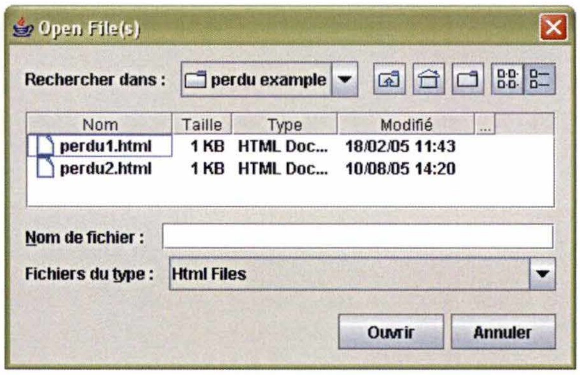


Figure A.2: *The Open Project (left) and Open File(s) (right) dialog boxes*

Open URL...

Opens a *Search URL* dialog box (figure A.3) allowing to enter the URL of a given Web page. The HTML source code of the selected Web page is then downloaded and displayed in the Opened files area. If the *Launch browser* check box is selected, the default Internet browser is launched to display the Web page.

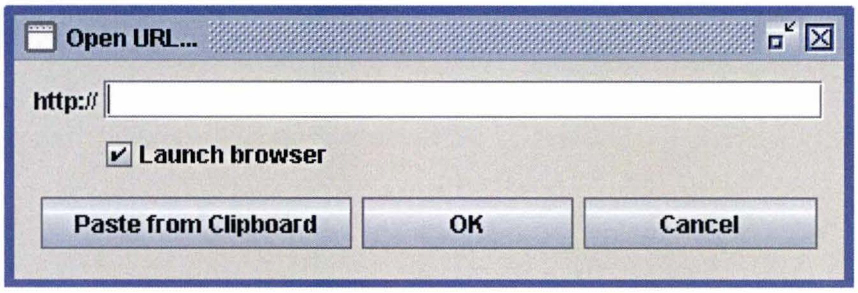


Figure A.3: *The Search URL dialog box*

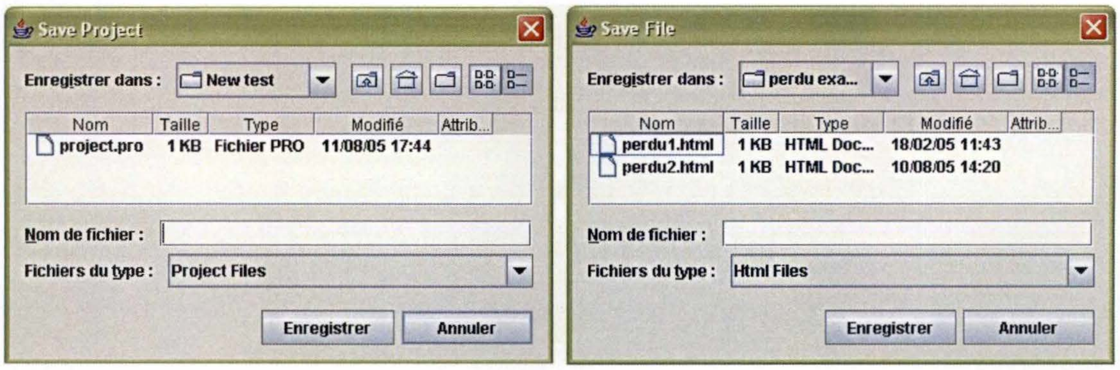
Save Project As...

Opens a *Save Project* dialog box (figure A.4, left) allowing to save a project (.pro) for future work. It actually saves the opened files of the Opened files area and the state of the Clustering area.

Save File As...

Opens a *Save File* dialog box (figure A.4, right part) allowing to save the HTML file selected in the Opened files area. As the contents of the Opened files area are editable, this options makes saving a modified file possible.

If a layout tree is selected in the Opened files area, *Save File* generates the HTML code represented by the layout tree before saving it. This can be useful for showing the similarities between two pages (the top-down tree) in an Internet browser.



Save Project

Enregistrer dans : New test

Nom	Taille	Type	Modifié	Attrib...
project.pro	1 KB	Fichier PRO	11/08/05 17:44	

Nom de fichier :

Fichiers du type : Project Files

Enregistrer Annuler

Save File

Enregistrer dans : perdu exa...

Nom	Taille	Type	Modifié	Attrib...
perdu1.html	1 KB	HTML Doc...	18/02/05 11:43	
perdu2.html	1 KB	HTML Doc...	10/08/05 14:20	

Nom de fichier :

Fichiers du type : Html Files

Enregistrer Annuler

Figure A.4: *The Save Project (left) and Save File (right) dialog boxes*

Close, Close All

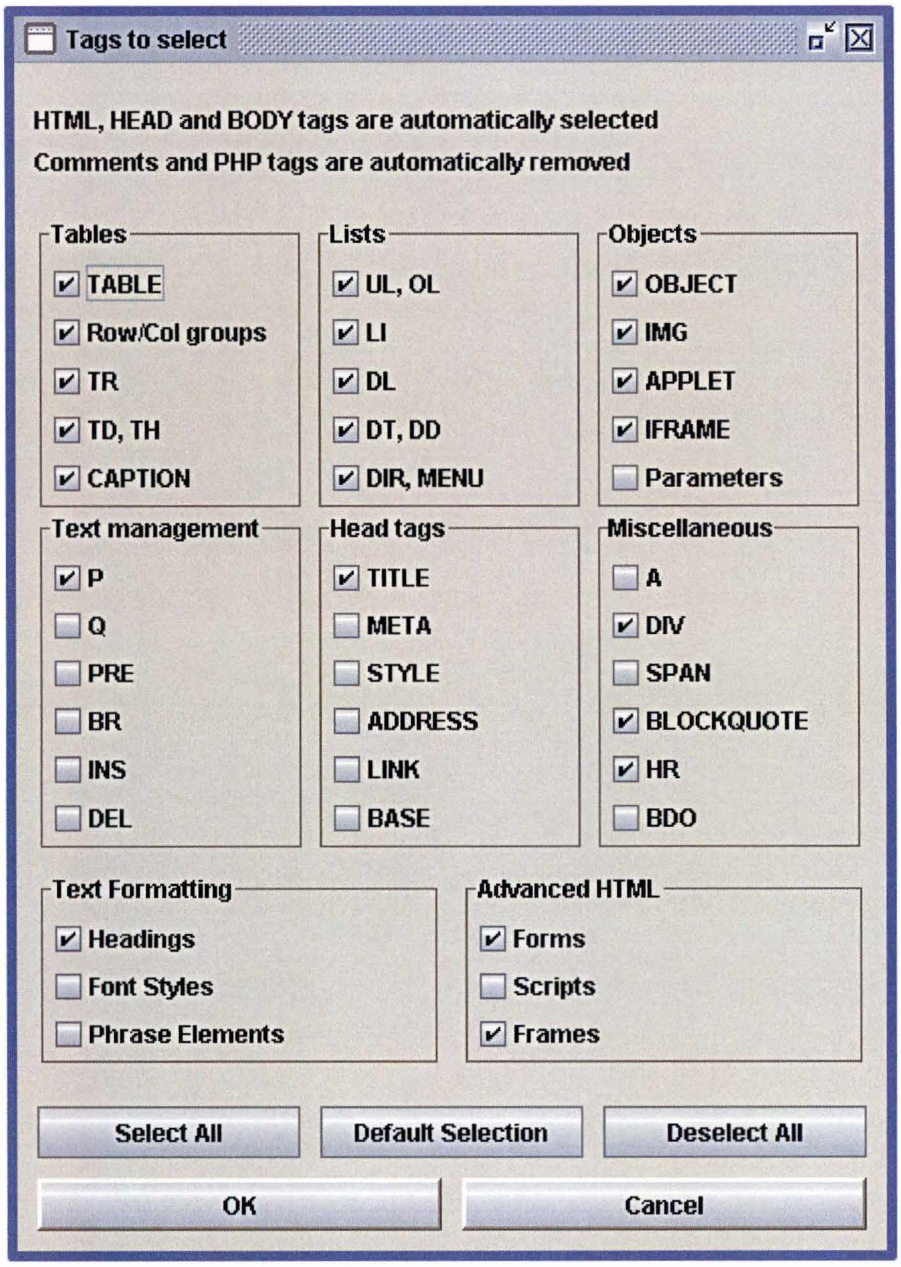
Closes the selected file (or all the files) in the Opened files area.

A.4.2 Operations

This menu offers all the functions managing the various operations allowed on HTML files and/or layout trees.

Tags selection...

Opens the *Tags Selection* dialog box (figure A.5) allowing to choose the list of HTML tags that will be kept for the building of layout trees. The *Default Selection* button selects a list of the most relevant tags for the clustering.



Tags to select

HTML, HEAD and BODY tags are automatically selected

Comments and PHP tags are automatically removed

Tables
- [✓] TABLE
- [✓] Row/Col groups
- [✓] TR
- [✓] TD, TH
- [✓] CAPTION

Lists
- [✓] UL, OL
- [✓] LI
- [✓] DL
- [✓] DT, DD
- [✓] DIR, MENU

Objects
- [✓] OBJECT
- [✓] IMG
- [✓] APPLET
- [✓] IFRAME
- [ ] Parameters

Text management
- [✓] P
- [ ] Q
- [ ] PRE
- [ ] BR
- [ ] INS
- [ ] DEL

Head tags
- [✓] TITLE
- [ ] META
- [ ] STYLE
- [ ] ADDRESS
- [ ] LINK
- [ ] BASE

Miscellaneous
- [ ] A
- [✓] DIV
- [ ] SPAN
- [✓] BLOCKQUOTE
- [✓] HR
- [ ] BDO

Text Formatting
- [✓] Headings
- [ ] Font Styles
- [ ] Phrase Elements

Advanced HTML
- [✓] Forms
- [ ] Scripts
- [✓] Frames

Figure A.5: *The tag selection frame with the default selection*

Build Tree

Builds the layout tree of the selected file in the Opened files area (figure A.6). If the selected file is not an HTML source, an error message is shown.

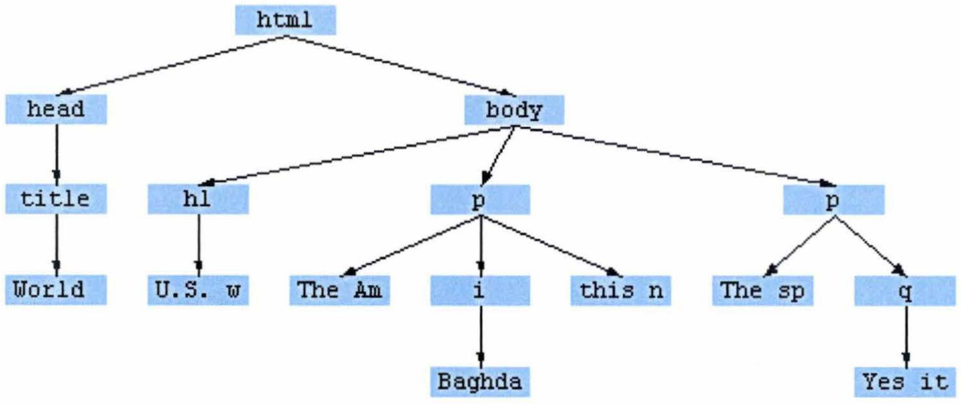


Figure A.6: *An example of layout tree in the Opened files area*

Mapping...

Opens a *Mapping* dialog box (figure A.7) allowing to choose two layout trees to map. Two kinds of mappings are available : *Fast top-down mapping* and *Fine top-down mapping*. The result of the chosen mapping is displayed in the Opened files area. Notice that at least one layout tree must have been built beforehand.

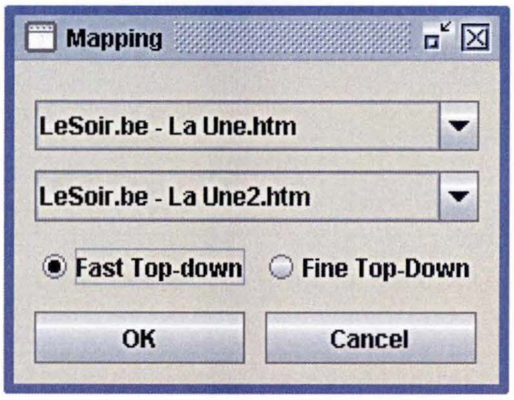
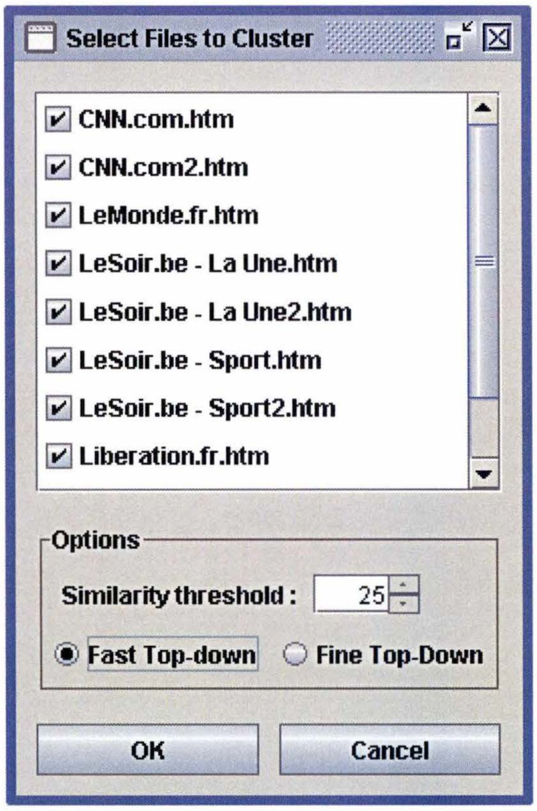


Figure A.7: *The mapping frame*

Cluster files...

Opens a *Clustering* dialog box (figure A.8) allowing to choose the various news Web pages to cluster, to select a fast or a fine top-down mapping and a similarity threshold.


Select Files to Cluster

[✓] CNN.com.htm
[✓] CNN.com2.htm
[✓] LeMonde.fr.htm
[✓] LeSoir.be - La Une.htm
[✓] LeSoir.be - La Une2.htm
[✓] LeSoir.be - Sport.htm
[✓] LeSoir.be - Sport2.htm
[✓] Liberation.fr.htm

Options

Similarity threshold : 25

[✓ Fast Top-down] Fine Top-Down

OK Cancel

Figure A.8: *The Clustering frame*

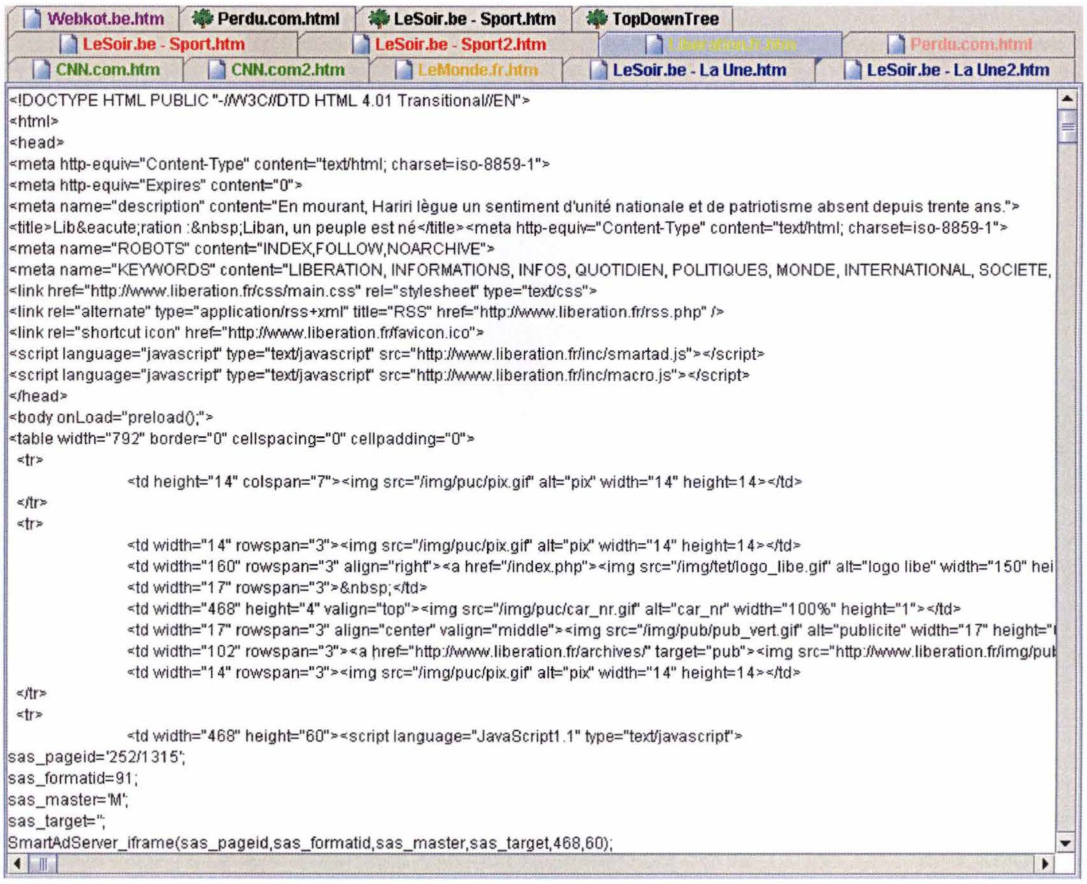
Extract News

Extracts the news articles from the Web pages inside the selected cluster in the Clustering area and displays them in the Opened files area.

A.5 The Opened files area

The Opened files area (figure A.9) is the right part of the main window. The HTML source code of chosen news Web pages, the related layout trees, the results of the mappings and the extracted news articles are displayed in this area. When an HTML source code is displayed, the area becomes a text editor so that the HTML source code can be changed if necessary. Each file of the Opened files area is labelled at the top of the area. The labelling of the different files in the Opened files area is made as follows:

- each file is labelled under its file name,
- each HTML file is labelled with a "page" icon,
- each layout tree is labelled with a "tree" icon,
- each file that is not clustered is labelled in black,
- the news Web pages within a same cluster are labelled in the same colour,
- the selected file has its label background in light blue.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Expires" content="0">
<meta name="description" content="En mourant, Hariri lègue un sentiment d'unité nationale et de patriotisme absent depuis trente ans.">
<title>Lib&eacute;ration :&nbsp;Liban, un peuple est né</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="ROBOTS" content="INDEX,FOLLOW,NOARCHIVE">
<meta name="KEYWORDS" content="LIBERATION, INFORMATIONS, INFOS, QUOTIDIEN, POLITIQUES, MONDE, INTERNATIONAL, SOCIETE,
<link href="http://www.liberation.fr/css/main.css" rel="stylesheet" type="text/css">
<link rel="alternate" type="application/rss+xml" title="RSS" href="http://www.liberation.fr/rss.php" />
<link rel="shortcut icon" href="http://www.liberation.fr/favicon.ico">
<script language="javascript" type="text/javascript" src="http://www.liberation.fr/inc/smartad.js"></script>
<script language="javascript" type="text/javascript" src="http://www.liberation.fr/inc/macro.js"></script>
</head>
<body onLoad="preload();">
<table width="792" border="0" cellspacing="0" cellpadding="0">
 <tr>
		<td height="14" colspan="7"><img src="/img/puc/pix.gif" alt="pix" width="14" height=14></td>
 </tr>
 <tr>
		<td width="14" rowspan="3"><img src="/img/puc/pix.gif" alt="pix" width="14" height=14></td>
		<td width="160" rowspan="3" align="right"><a href="/index.php"><img src="/img/tet/logo_libe.gif" alt="logo libe" width="150" hei
		<td width="17" rowspan="3">&nbsp;</td>
		<td width="468" height="4" valign="top"><img src="/img/puc/car_nr.gif" alt="car_nr" width="100%" height="1"></td>
		<td width="17" rowspan="3" align="center" valign="middle"><img src="/img/pub/pub_vert.gif" alt="publicite" width="17" height="
		<td width="102" rowspan="3"><a href="http://www.liberation.fr/archives/" target="pub"><img src="http://www.liberation.fr/img/pu
		<td width="14" rowspan="3"><img src="/img/puc/pix.gif" alt="pix" width="14" height=14></td>
 </tr>
 <tr>
		<td width="468" height="60"><script language="JavaScript1.1" type="text/javascript">
sas_pageid='252/1315';
sas_formatid=91;
sas_master='M';
sas_target='';
SmartAdServer_iframe(sas_pageid,sas_formatid,sas_master,sas_target,468,60);
```

Figure A.9: *The Opened files area*

A.6 The Clustering area

The Clustering area is composed of three buttons (see figure A.10) and of a frame that displays the clusters (figure A.11).

Both first buttons are actually the main functions of the News Ripper application. They have already been described in section A.4. The third button allows to reset all the clustering results.

Each cluster in the frame contains the news Web pages having at least *similarity threshold %* of structural similarities. The colour of the clusters (of the pages inside a cluster actually) are applied to the corresponding labels in the Opened files area.

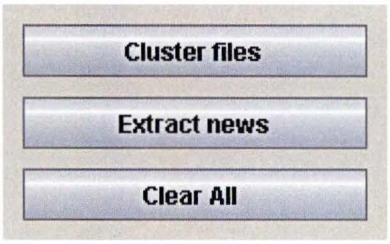
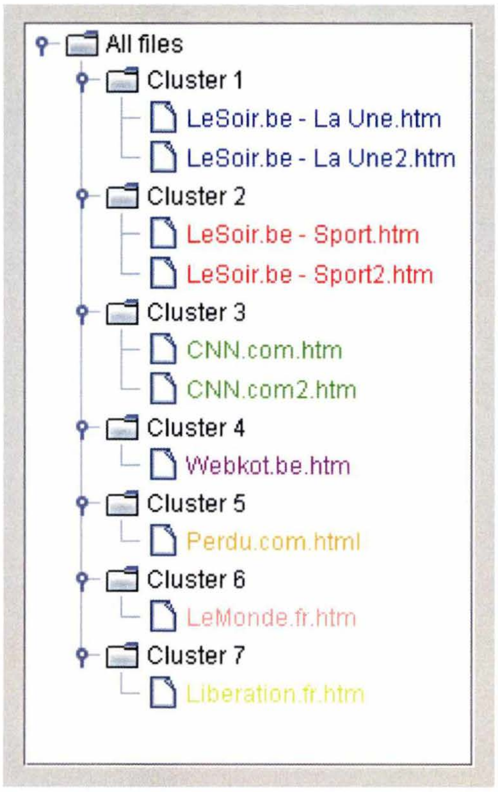


Figure A.10: *Buttons of the Clustering area*



All files
- Cluster 1
  - LeSoir.be - La Une.htm
  - LeSoir.be - La Une2.htm
- Cluster 2
  - LeSoir.be - Sport.htm
  - LeSoir.be - Sport2.htm
- Cluster 3
  - CNN.com.htm
  - CNN.com2.htm
- Cluster 4
  - Webkot.be.htm
- Cluster 5
  - Perdu.com.html
- Cluster 6
  - LeMonde.fr.htm
- Cluster 7
  - Liberation.fr.htm

Figure A.11: *Frame of the Clustering area*

Bibliography

[1] S.J. Simoff D. Zhang. Informing the curious negotiator: Automatic news extraction from the internet. 2004.

[2] M. Champion S. Isaacs A. Le Hors G. Nicol J. Robie P. Sharpe B. Smith J. Sorensen R. Sutor R. Whitmer C. Wilson V. Apparao, S. Byrne. Document object model (dom) level 1 specification version 1.0. Technical report, 1998.

[3] J.M. Pierre. On the automated classification of web sites. In *Computer and Information Science*, volume 6, 2001.

[4] D. Lewis. Text representation for intelligent text retrieval: A classification-oriented view. In *Text-Based Intelligent Systems*, 1992.

[5] M. Schubert M. Ester, H-P. Kriegel. Web site mining: A new way to spot competitors, customers and suppliers in the world wide web. 2002.

[6] D.R. Karger L.K. Shih. Using urls and table layout for web classification tasks. 2004.

[7] Q. Yang H-J. Zeng B. Zhang Y. Lu W-Y. Ma D. Shen, Z. Chen. Web-page classification through summarization. 2004.

[8] A.A. Freitas N. Holden. Web page classification with an ant colony algorithm. 2004.

[9] X. Li L. Yi, B. Liu. Eliminating noisy information in web pages for data mining. 2003.

[10] G. VALIENTE. An efficient bottom-up distance between trees. 2002.

[11] B.A. RIBEIRO-NETO J.S. TEIXEIRA A.H.F. LAENDER, A.S. DA SILVA. A brief survey of web data extraction tools. 2002.

[12] S. MINTON C.A. KNOBLOCK, K. LERMAN and I. MUSLEA. Accurately and reliably extracting data from the web: A machine learning approach. In *IEEE Data Engineering Bulletin*, 2000.

[13] G.O. AROCENA and A.O. MENDELZON. Web-qql: Restructuring documents, databases ans webs. In *Proceeding of the 14th IEEE International Conference on Data Engineering*, pages 24–33, Toronto, Canada, 1998.

[14] W. HAN L. LIU, C. PU. An extensible wrapper construction system for internet information. In *Proceedings of the 16th International Conference on Data Engineering*, 2000.

[15] I. MUSLEA. Extraction patterns for information extraction tasks: A survey. 1999.

[16] B.NODOSE. Nodose, a tool for semi-automatically extarcting structured ans semi-structured data from text documents. In *SIGMOD Recors 27,2*, pages 283–294, 1998.

[17] A.S. DA SILVA A.H.F. LAENDER, B.A. RIBEIRO-NETO. Debye, data extraction by bxample. In *Data & Knowledge Engineering*, 2002.

[18] G. MECCA V. CRESCENZI and P. MERIALDO. ROADRUNNER: Towards automatic data extraction from large web sites. In *Proceedings of the 26th International Conference on Very Large Databases System*, Rome, Italy, 2001.

[19] S. MINTON C.A. KNOBLOCK and I. MUSLEA. Hierarchical wrapper induction for semistructured information sources. 2001.

[20] U. GAST. *Information Extraction*, chapter 3. Information Extraction Techniques. 2003.

[21] B. Theodoulidis H. Karanikas, C. Tjortjis. An approach to text mining using information extraction. 2000.

[22] D. Freitag. Toward general-purpose learning for information extraction. 2002.

[23] A.S. Da Silva D.C. Reis, P.B. Golgher and A.F. Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th International Conference on World Wide Web*, 2004.

[24] Elliotte Rusty Harold. Web client programming with java: Parsing html is hard. 2000.

[25] Anonymous. Jericho parser homepage - http://jerichohtml.sourceforge.net/.

[26] World Wide Web Consortiom W3C. Html 4.01 specification - www.w3.org/tr/html4. 1999.

[27] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. 2001.