Automatically Extract Information from Web Documents

Dipesh Sharma
Western Kentucky University

Follow this and additional works at: http://digitalcommons.wku.edu/theses
Part of the Computer Sciences Commons

Recommended Citation
http://digitalcommons.wku.edu/theses/376

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Masters Theses & Specialist Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact topscholar@wku.edu.
AUTOMATICALLY EXTRACT INFORMATION FROM WEB DOCUMENTS

A Thesis
Presented to
The Faculty of the Department of Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Computer Science

By
Dipesh Sharma
December 2007
Table of Contents

Abstract .................................................................................................................. v

1. **Introduction** .................................................................................................. 3
   1.1 Research Context ....................................................................................... 3
   1.2 Literature Survey ....................................................................................... 5
   1.3 Contribution ............................................................................................... 7
   1.4 Organization of Chapters ........................................................................... 8

2. **Targeted Information and Extraction Techniques** .................................... 10
   2.1 Basic Concepts .......................................................................................... 12
      2.1.1 Hand-coded wrappers ........................................................................ 12
      2.1.2 Automatic wrapper construction ....................................................... 12

3. **The Tag Similarity** ....................................................................................... 14
   3.1 Similar items contain similar surrounding tags ........................................... 14
      3.1.1 Example of the occurrence of price ................................................... 15
      3.1.2 Real World Example ......................................................................... 16

4. **Frequency of occurrence of data** ................................................................ 18
   4.1 Observation of an HTML page layout ....................................................... 18
      4.1.1 Important information is repeated several times ............................... 18
      4.1.2 Identifying tag pattern is repeated several times .............................. 20

5. **High Frequency Pattern Extraction** .............................................................. 22
   5.1 Selection of high frequency tags ............................................................... 22
   5.2 Creation of candidate prefix tag patterns from selected high frequency tags ... 23
      5.2.1 Selection of tag patterns with data .................................................... 27
      5.2.2 High frequency candidate tag patterns ............................................. 27
   5.3 Selection of patterns with highest frequency ............................................. 28
   5.4 Problems faced during the implementation of the algorithm .................. 28
   5.5 Real World Example ............................................................................... 29

6. **Experimentation and Results** ...................................................................... 31
   6.1 System Architecture for Smart Frequency Data Extractor ...................... 31
   6.2 Experimental Results ............................................................................... 37
   6.3 Advantages over other strategies ............................................................ 36

7. **Conclusions and Future Work** ................................................................. 38
   7.1 Refinement of the existing technique ....................................................... 38
   7.2 Enhancement to extract particular data items .......................................... 38
   7.3 Integration with other techniques ........................................................... 39

Reference ............................................................................................................... 40
Table of Figures

Figure 2.1. An example of target information in a Web page ........................................... 11
Figure 3.1. Prefix Matching .............................................................................................. 15
Figure 3.2. Prefix tag patterns ......................................................................................... 16
Figure 4.1. Typical Web page showing repetition of relevant data ................................ 20
Figure 5.1: HTML source code containing targeted items .................................................... 24
Figure 5.2: High Frequency Tags List ................................................................................ 26
Figure 5.3. Different Attributes in Tags ............................................................................. 30
Figure 6.1: System Architecture of Smart Frequency Data Extractor (SFDE) ................. 32
Figure 6.2: Experimental Results ..................................................................................... 35
Abstract

The Internet could be considered to be a reservoir of useful information in textual form -- product catalogs, airline schedules, stock market quotations, weather forecast etc. There has been much interest in building systems that gather such information on a user's behalf. But because these information resources are formatted differently, mechanically extracting their content is difficult. Systems using such resources typically use hand-coded wrappers, customized procedures for information extraction.

Structured data objects are a very important type of information on the Web. Such data objects are often records from underlying databases and displayed in Web pages with some fixed templates. Mining data records in Web pages is useful because they typically present their host pages' essential information, such as lists of products and services. Extracting these structured data objects enables one to integrate data/information from multiple Web pages to provide value-added services, e.g., comparative shopping, meta-querying and search.

Web content mining has thus become an area of interest for many researchers because of the phenomenal growth of the Web contents and the economic benefits associated with it. However, due to the heterogeneity of Web pages, automated discovery of targeted information is still posing as a challenging problem.
Chapter 1

Introduction

1.1 Research Context

The Internet provides access to information, which was previously very difficult to obtain. Thus, information like product catalogs, airline schedules, stock market quotations, weather forecast, etc. are now readily available on the Internet.

The focus of our research is mainly on extraction of data from Web pages, such as product name, price, etc. Extracting such data allows one to provide many useful services like comparative shopping, searching etc. This kind of information mining technique from Web is known as Web content mining or Web data extraction.

Web data extraction is the problem of identifying and extracting target items from Web pages. It is important in practice because it allows one to integrate information or data from Web pages to provide the value-added services.

There are two main types of algorithms used for mining data records from Web pages, namely wrapper induction and automatic extraction. In wrapper induction, a set of extraction rules are learnt from a set of manually labeled pages or data records. These rules are then used to extract data items from similar pages. This method still requires substantial manual efforts.

On the other hand, in automatic methods, one finds patterns or grammars from multiple pages containing similar data records. Requiring an initial set of pages containing similar data records is, however, a major limitation of this type of approach.
because such pages have to be found manually or by another system. In this research, we have considered some techniques used in Web content mining for extracting structured data objects.

Web content mining is related but different from data mining and text mining. It is related to data mining because many data mining techniques can be applied in Web content mining. It is related to text mining because much of the Web contents are texts. However, it is also quite different from data mining because Web data are mainly semi-structured and/or unstructured, while data mining deals primarily with structured data. Web content mining is also different from text mining because of the semi-structure nature of the Web, while text mining focuses on unstructured texts. Web content mining thus requires creative applications of data mining and/or text mining techniques and also its own unique approaches. In the past few years, there was a rapid expansion of activities in the Web content mining area. This is not surprising because of the phenomenal growth of the Web contents and significant economic benefit of such mining. However, due to the heterogeneity and the lack of structure of Web data, automated discovery of targeted or unexpected knowledge information still present many challenging research problems.

In this tutorial, we have examined the following important Web content mining problems and discuss existing techniques for solving these problems. Particularly, we have considered the following two problems.

• **Data/information extraction:** Our focus has been on extraction of structured data from Web pages, such as products and search results. Extracting such data allows one to provide services. Two main types of techniques, machine learning and automatic extraction are covered.
• Web information integration and schema matching: Although the Web contains a huge amount of data, each Web site (or even page) represents similar information differently. How to identify or match semantically similar data is a very important problem with many practical applications. Some existing techniques and problems are examined. All these tasks present major research challenges and their solutions also have immediate real-life applications.

1.2 Literature Survey

The approaches for mining data records from Web pages can be broadly categorized into two categories, namely manual and automatic. In the manual approach a programmer finds some patterns from the Web page and then writes a program to identify and extract all the data items. This approach is not scalable to a large number of pages and is quite laborious. Other approaches all have some degree of automation. As discussed earlier, there are two main types of algorithms, wrapper induction and automatic extraction. In wrapper induction, a set of extraction rules are learnt from a set of manually labeled pages or data records. Popular examples of the wrapper induction systems are [1, 2, 3, 4, 5, 12, 13, 14]. These rules are then used to extract data items from similar pages. This method still requires substantial manual efforts. In automatic methods, [6] find patterns or grammars from multiple pages containing similar data records. Requiring an initial set of pages containing similar data records is, however, a major limitation of this type of approaches because such pages have to be found manually or by another system. Fetch [7] proposes a method that tries to explore the detailed information page behind the current page to extract data records. This need for detailed information pages is also a
serious limitation because many data records do not have such pages. Furthermore, the method assumes that the detail pages are given, which is not realistic in practice. Due to a large number of links in a typical Web page, automatically identifying links that point to detailed information pages is a non-trivial task. Chang and Lui [8] proposes a string matching method. However, its results are weak as shown in MDR [9]. Another assumption that most current systems make is that the relevant information of a data record is contained in a contiguous segment of the HTML code. However, in some Web pages, the description of one object may intertwine with the descriptions of some other objects. For example, the descriptions of two objects in the HTML source may follow this sequence, part1 of object1, part1 of object2, part2 of object1, part2 of object2. Thus, the descriptions of both object1 and object2 are not contiguous. However, when they are displayed on a browser, they appear contiguous to human viewers.

“Record-boundary discovery in web documents” and “Learning the common structure of data”, [15, 16] propose some techniques for finding data objects or data records. However, they do not perform data extraction from the records.

Roadrunner [10] proposes another automatic extraction method. However, this method is less accurate than the systems that ask the user to label training pages.

We propose an automatic extraction method that does not require manual labeling, yet it is accurate. It works on the observation that relevant information is repeated several times and hence it finds and extracts the information that is repeated the most number of times.
1.3 Contribution

This thesis presents a novel concept for the extraction of relevant information from Web pages. The basic idea for this research is inspired by one of the strategies implied in Text Mining, popularly known as “Term Frequency”.

- Term Frequency approach in Text Mining

Text Mining technique is basically applied to extract the relevant information from a long piece of text. In other words, it extracts the meaning (or summary) of the text. The term Frequency stands for count. Thus in the context of Text Mining it means the number of occurrences of a particular term in a document. Clearly, the most relevant information in a text is most likely to be repeated several times throughout the text. For example, an article on a cardiovascular disease called “Hypertrophic cardiomyopathy” or commonly known as HCM is likely to have the words “Hypertrophic cardiomyopathy” or HCM repeated throughout the text. It is also very likely that the term “cardiovascular disease” might as well be repeated throughout the article. In either case, if either one of them is repeated several times, one could easily infer (by counting the frequency of their occurrence) that the article focuses on a cardiovascular disease.

This approach favors the long documents even more, since such documents are likely to have those words repeated even more, hence causing an increase in the frequency of relevant words.
• **Term Frequency approach in Web Mining**

Web mining is related to text mining because much of the Web contents are texts. Hence the Term Frequency approach discussed above can be used to some extent to extract relevant data from Web pages as well.

However Web mining is not completely similar to text mining. It is different from text mining because of the semi-structure nature of the Web, while text mining focuses on unstructured texts. Web content mining thus requires creative applications of its own as well.

### 1.4 Organization of Chapters

The content of this thesis is organized as follows:

Chapter 2 explains what we are trying to do, i.e. it covers the concept of targeted or relevant data by taking a real world example. In this chapter, we have also thrown some light on the basic and existent concepts that are used for information extraction from Web pages. Finally we have discussed two main techniques namely “Hand-coded wrappers” and “Automatic wrapper construction” in here.

Chapter 3 explains the extraction techniques at a much deeper level. It discusses the phenomenon that forms the base of data extraction technique by taking a real world example.

Chapter 4 explains our unique observations of the frequency of occurrence of various data in a Web page and also discusses our strategy to employ these facts to our advantage by taking an example.
Chapter 5 discusses in detail our approach and also gives an outline of our algorithm. It explains the various filtering phases in our algorithm that lead to the selection and consequent extraction of correct data. Later in this chapter, we discuss a few problems that we faced during the implementation of our algorithm. The problem in comparison of tag has been addressed in particular.

Chapter 6 outlines implementation and the results of our approach applied on actual Web pages. We show and explain the general architecture of our system firstly. Also, in this chapter we discuss three popular approaches and compare our approach with them on the basis of the results obtained. We also list various advantages of our approach over other existent approaches.

Chapter 7 presents the resulting conclusions and provides an insight into the future work. It gives some valuable tips for future work, and also guides about completely new directions for new and related work.
Chapter 2

Targeted Information and Extraction Techniques

Other than the information of interest, a Web page also consists of many other and relatively less useful information like advertisements, links to other pages, links to other facilities, acknowledgements etc. By targeted information we mean, the information of interest, e.g. if it is a site that shows deals, we are interested in deals that are listed on that site; if it is a site for shopping, we are interested in product information; if it is a site that gives news, then the targeted information is news.

For example, Fig.2.1 below shows the contents of a Web page taken from a site called www.deals.com. This Web page contains a listing of deals placed by various vendors. Each of the deals is accompanied with a detailed description of the product. One may want to extract just the deal information while ignoring the rest of the contents, which can be assumed to be relatively less important or useless.

Please note that each of the items is marked with a lined box in the figure. We call each required piece of information a target item (or simply item).
Extracting such information helps in providing value added services like comparative shopping and searching. The basic idea here is to create computer programs to interact with the information resources and gather information on a user's behalf.

The Internet was designed with the intention of direct use by people. Hence making systems that can automatically extract information from Web pages is a challenging and complex task. In this paper, we focus on regularly structured data which are produced by computer programs following some fixed templates.

**Figure 2.1:** An example of target information in a Web page
2.1 Basic Concepts

Data extraction from HTML pages is performed by computer programs called wrappers. A wrapper is a procedure for extracting particular resources content. This technique of extracting data using wrappers can be broadly categorized into two categories as discussed in the sections below.

2.1.1 Hand-coded wrappers

In this technique, the wrapper was coded manually. The underlying concept is that the targeted information (which is basically a text), is always surrounded by certain specific tags. Hence the surrounding tags can be used to extract the targeted item. In hand-coded wrappers, such surrounding tags were supplied manually.

The major problem with this approach is that manually coding the wrapper is a laborious and difficult task. Another difficulty with this approach is that web sites often change their format. Each time such modifications are made, the wrapper has to be rewritten, making the hand-coded wrapper technique even more tedious and error prone.

Because of the laborious work and numerous problems associated with it, this technique is relatively less popular in current times. Modern day algorithms make use of another approach, popularly known as automatic wrapper construction. In this technique, we intend to automate the process of wrapper writing.

2.1.2 Automatic wrapper construction

In this technique, we automate the process of wrapper construction. Hence the user is freed of the burden of manually supplying the surrounding tags. Since in case of automatic wrapper construction we don’t have to depend on prior knowledge of the Web
page and its contents, we are also rid of the other problems associated with hand-coded
wrappers such as updating the wrappers each time a Web site changes its contents.

A wrapper is simply a computer program, therefore we are trying to do automatic
programming. Since automatic programming is generally a difficult task, automatic
generation of wrapper is a difficult task indeed. Hence we need a smart algorithm,
something like a soft-robot, which can actually scan the Web page and figure out the
contents of interest and surrounding tags of such contents.
Chapter 3
The Tag Similarity

3.1 Similar items contain similar surrounding tags

The identification of surrounding tags is not the only issue with automatic wrapper construction; the major problem lies in identifying whether an item under consideration in a Web page is actually the targeted item or not.

This paper discusses all such problems and their solutions. But let us start with a more basic problem first. Given a particular instance of targeted item, how do we identify other instances? In order to understand the answer to the above problem, we must first note the fact that since we are using surrounding tags to extract the targeted item, we do not compare the items themselves; instead, we compare the prefix or suffix tag strings.

Our approach is mainly based on the fact that similar data objects, in a Web page, are formatted (hence surrounded) using similar HTML tags. A group of data records which contain descriptions of a set of similar objects are typically presented in a particular region of a page and are formatted using similar HTML tags. Such a region is called a data region.

This phenomenon results into the formation of partially structured representation of the data in a Web page; hence it is useful in pattern formation and consequently is helpful in data extraction.
If we regard the HTML tags of a page as a string, we can use string matching to compare different sub-strings to find the similar ones, which may represent similar data records.

3.1.1 Example of the occurrence of price

This phenomenon can further be explained with the help of an example. Say, if we are searching for price of products in a Web page, and the prefix tags of first occurrence of a price item are found to be say `<table><tr><td><i><b>` as shown in figure below, then all the subsequent price items will have the same prefix of HTML tags as shown in the figure below. Consequently, this prefix (or suffix) tag pattern can be used to identify relevant and targeted data in a Web page. Hence this prefix pattern could be referred to as an identifying tag pattern.

---

**Figure 3.1:** Prefix Matching
3.1.2 Real World Example

This phenomenon could further be clarified by taking an example of a real HTML page. The HTML document shown below is taken from the Website http://www.laweekly.com. The figure displays the particular region of the Web page that contains the target items.

Since this is a news site, the most important information (and hence also the target item) here is the news headline, and prefix tag patterns of each such headline has been marked using a lined box as shown in figure below.

```html
<dl>
  <dt>Source Family Reunion, 11/15</dt>
    &gt;Source Family Reunion, 11/15&lt;/a&gt;</dd>
  <dt>By Tom Christie</dt>
  <dd>&lt;p class="Intro_Text">L.A.'s legendary Source Family cult/commune and house band Ya Ho Wa</p&gt;</dd>
  <dt>Deadline Hollywood: On a Collision Course</dt>
  <dd>&lt;a href="/news/deadline-hollywood/on-a-collision-course/17683/"&gt;Deadline Hollywood: On a Collision Course&lt;/a&gt;</dd>
  <dt>By NIKKI FINK</dt>
  <dd>&lt;p class="Intro_Text">Why TV showrunners could be the collateral damage of this writers' &lt;/p&gt;</dd>
  <dt>Counter Intelligence: Ribbed for Her Pleasure</dt>
  <dd>&lt;a href="/eat-drink/counter-intelligence/ribbed-for-her-pleasure/17678/"&gt;Counter Intelligence: Ribbed for Her Pleasure&lt;/a&gt;</dd>
  <dt>By Jonathan Gold</dt>
</dl>

Figure 3.2: Prefix tag patterns
As one can clearly see, for each instance of the target item (which is the headline in our case) we have the same prefix tag pattern in the page namely `<div><h4><a>`.

Similarly we also have the same suffix tag patterns for the same. These are the in fact the surrounding tags of interest to us. These tags can thus be used to extract the headlines, and hence the target items from the page. If we can identify these tags for one instance of the target item, we can be sure that all the subsequent occurrences of the target items will have the same surrounding tags. Hence we can extract the other instances as well.
4.1 Observation of an HTML page layout

Having made the reader familiar with the underlying concepts, we will now discuss some major issues with data extraction. As mentioned earlier, the major problem lies in identifying whether an item under consideration in a Web page is actually the targeted item or not. We need to know that in order to be sure that the item’s surrounding tags are the tags we are looking for, because if the item is not the targeted item, we would end up using incorrect surrounding tags, which would further result into extraction of incorrect data or no data at all.

In other words, when our algorithm scans a Web page, how does it know the tag that it has in its hand is the tag we are looking for? How can we make our system smart enough to make such decisions on its own?

This chapter answers all such questions, but before explaining our approach and algorithm, let us first discuss some of our important observations, which led to the formation of our strategy.

4.1.1 Important information is repeated several times

In order to make a smart algorithm, which is capable of making decision, we have to have a strong strategy. What we are looking for is a phenomenon that is true, if not all the time, then at least most of the time.
The title of this section forms the base of our approach. In a Web page, usually the most relevant information is repeated more than any other information, which is true in most of the cases. If we have a Web site that sells some product, then it would repeatedly talk about the product it is trying to sell. For example, a site that sells cars would have the model and make of the cars listed throughout the page, and most of the web sites that sell cars would have that information repeated more than any other information contained on those sites. This point could further be clarified by taking an example of an actual Web site. The page below shows the deals available on the site called www.slickdeals.net.

As one can see in this page, the information that is repeated the most often is the deals (shown in an enclosed box); it is repeated more often than any other data in the Web page. Even if there exists some other data in a Web page (which is not the target item, but occurs more than the target item), the target item would still have a very high frequency of occurrence. In fact, in such Web pages the target item would probably have the second or third highest frequency of occurrence in general.
This phenomenon plays a crucial role in extraction of relevant data from a Web page as explained in the section below.

4.1.2 Identifying tag pattern is repeated several times

As explained in chapter 3, we already know that similar data objects have the same prefix and suffix tag patterns in a Web page. Consequently, this prefix (or suffix) tag pattern can be used to identify all targeted items in a Web page. Therefore this prefix pattern could be referred to as an identifying tag pattern.
In combination with the phenomenon explained in section 4.1.1, this fact results in the formation of another phenomenon namely, “identifying tag patterns of the relevant information are repeated most number of times”.

Thus, for the example shown in section 3.1.2, the identifying tag pattern <div><h4><a>, would therefore be repeated several times throughout the Web page.

The fact that identifying tag patterns of relevant information are repeated several time can therefore be tapped to identify those patterns, and consequently extract the targeted data. This point will further be discussed in detail later.
Chapter 5
High Frequency Pattern Extraction

As mentioned earlier, our focus is on the extraction of relevant data from a Web page. Such data is repeated several times for the reasons explained in chapter 4. Hence, the identifying prefix tags for such data are repeated several times throughout the Web page.

Therefore, our focus should be on the tag patterns that are repeated several times, which implies that each of the individual tags in the pattern themselves have a high frequency of occurrence. Our algorithm filters those tags, and finally the prefix tag patterns they form, using the technique explained below.

5.1 Selection of high frequency tags

We start by parsing the code using an open source HTML parser, which can be downloaded from the site http://htmlparser.sourceforge.net/. As the code is parsed, we maintain a frequency list, which contains frequency of occurrence of each tag. So every time a tag is encountered, we first check the list. If that tag already exists in the list, the frequency associated with it is incremented. If not a new tag is added to it. Once the frequency of a tag exceeds a certain limit, which we refer to as threshold frequency (which is 8 in our system), we start monitoring that tag.

We do get some unwanted tags in our list, which are not a part of prefix tag patterns that corresponds to the target items. The most common examples of such tags are the ones that are a part of patterns that correspond to some data other than target items.
like the advertisements, links to other pages, etc. They add up in our list simply because they have a high frequency of occurrence. There are some other high frequency tags that do not contain any data, but are repeated several times in a Web page. We do want to get rid of all such tags. Our algorithm discards these tags by a series of filtering techniques which are explained later in this chapter.

5.2 Creation of candidate prefix tag patterns from selected high frequency tags

As the parsing continues, we start finding patterns amongst the high frequency tags. We look for consecutive occurrences of high frequency tags. If a set of consecutive high frequency tags occur simultaneously, we know that we have a candidate for the prefix tag pattern. This tag pattern is then added to a list of candidate identifying prefix tags.

This can further be explained with the help of an example. Let us assume that we have a part of HTML source code, as given below, that contains the targeted items. We also assume that tags <br> and <img> have been repeated several times in the initial part of the page, where they have been displayed.
Figure 5.1: HTML source code containing targeted items
By the time our algorithm reaches up to the tenth target item we will have a frequency list as shown in figure below. Clearly, <div>, <img>, <br>, <a>, and <span> are the high frequency tags. As the code is parsed, let’s say we encounter tag <div>, so its frequency increases to 10. The tag that immediately follows it is say for example <a>, so its frequency becomes 10 as well. The tag which occurs after <a> is <span>, and hence its frequency, too, goes up to 10.

Since these tags occur consecutively, we know that we have a pattern in our hand that comprises of tag <div>, followed by tag <a>, which is further followed by tag <span>. The order of the occurrence of tags determines their position in the pattern.

Please note that we are assuming that the high frequency of tags <br> and <img> is due to their repetition in the initial part of the page. Since they have crossed our threshold frequency, they are of interest to our algorithm now. Hence, each time any of those tags (<br> or <img>) is encountered, our algorithm keeps track of the next consecutive tag, and since in both the cases it corresponds to a closing tag, our algorithm discards them.

However, when a tag <div> is encountered, our algorithm checks to see which is the next tag, and when it finds that the next one is <a>, which too has a high frequency, it continues checking the tags next in the line, until it discovers a text. At this point it immediately knows that it has found a piece of text that may be of interest to us since it has been repeated many times in the page. Hence it adds the prefix tag pattern <div><a><span> to the list of candidate tag patterns.
<table>
<thead>
<tr>
<th>Tag#</th>
<th>Tag Name</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Html</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>Body</td>
<td>1</td>
</tr>
<tr>
<td>3.</td>
<td>Table</td>
<td>3</td>
</tr>
<tr>
<td>4.</td>
<td>Div</td>
<td>9</td>
</tr>
<tr>
<td>5.</td>
<td>H3</td>
<td>4</td>
</tr>
<tr>
<td>6.</td>
<td>Img</td>
<td>14</td>
</tr>
<tr>
<td>7.</td>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>8.</td>
<td>Br</td>
<td>17</td>
</tr>
<tr>
<td>9.</td>
<td>Span</td>
<td>9</td>
</tr>
<tr>
<td>10.</td>
<td>Tr</td>
<td>3</td>
</tr>
</tbody>
</table>

**Figure 5.2:** High Frequency Tags List

The table shows various tags with their frequency of occurrence. The tags of high frequency have been marked in bold letters.
5.2.1 Selection of tag patterns with data

Typically in a Web page, we still have some unwanted tag patterns that do not contain any data. These tag patterns are filtered out further by considering the fact that, each true candidate prefix tag pattern must be followed by a text field (or data). Let’s assume in our case that there occurs a text field immediately after the tag <span>. Hence we know that we have a potential candidate for identifying tag pattern, since it contains some data.

If we have a pattern, which is not followed by a text field, we can safely assume that it does not contain any data, and hence is not a candidate; since if there is no text contained, we have don’t have a data. We simply ignore such patterns. This also serves as a filter in short listing patterns and therefore getting closer to the correct one.

5.2.2 High frequency candidate tag patterns

As the candidate tag patterns are found, they are added to a list. We also maintain a frequency list, which contains the frequency associated with the candidate tag pattern. Hence whenever the algorithm encounters a candidate tag pattern it checks to see if it is already contained in the candidate tag pattern list. If it already contains such a pattern, the tag pattern frequency associated with that pattern is incremented. If however the pattern does not exist in the list, it is added to the list and its frequency is initialized to 1.

In our example, since the algorithm has already found the candidate tag pattern <div><a><span>, it has been added to the list and its frequency has been set to 1. Now, as the algorithm executes further, it finds new patterns and adds them to the list in a
similar manner. But whenever it comes across the pattern `<div><a><span>`, the
algorithm simply increments its frequency.

### 5.3 Selection of patterns with highest frequency

Finally, we start selecting the tag patterns in the descending order of the
frequency and use them to extract the data. We start with the one that has the highest
frequency and use it to extract the data. The extracted data is then presented to the user
for verification.

If the user is not satisfied with the results, the next tag pattern in the list (which
has the second highest frequency of occurrence) is then selected and is used to extract the
data. This process is repeated until the targeted tag pattern is reached, which produces
the desired results. Normally such a pattern is reached in three to five attempts. It actually
depends upon the frequency of occurrence of tag items relative to the frequency of
occurrence of other data in a page.

### 5.4 Problems faced during the implementation of the algorithm

In reality, tag comparison is not that simple. The common problem that arises is
the fact that although the tags are similar, they usually have different attributes for
different data objects. One common example of such an attribute is the tag I.D. Since an
I.D. is a unique value, it has different values for each occurrence of the tag.

This phenomenon created a problem in string extraction, because even if the
algorithm compares similar tags, it never will get a positive result since the tags are
usually not similar in totality. This point is further clarified by taking an example of a real
HTML page in the section below.
5.5 Real World Example

The HTML document shown below is taken from the Website www.slickdeals.net and has been modified to include only the information relevant to our concept, i.e. it shows the portion of the HTML code that contains the target items and has been indented to improve the readability.

As one can see, there is a repetition of a tag pattern in the page namely <div><a><span>. But if you look closely, the <div> tag itself has the different I.D. values viz. “deal header 10340”, “deal header 10339”, “deal header 10334”, and so on.

Similarly, for tag <a>, we have different parameters for the “toggledeal” function, which is called for in the case of an “onclick” event. So if we would compare the <a> tags in totality it would not give a true value. However if we ignore the attributes, we would get the correct result.
Figure 5.3: Different Attributes in Tags

Hence to perform string matching, we must compare just the tag names, and ignore the attributes (which are usually different and play no major role in comparison and hence in our algorithm).
Chapter 6

Experimentation and Results

In this chapter, the implementation of our procedure and the results obtained from it are discussed and compared with the results of other approaches. Our approach has been implemented on various Web sites and the results obtained are presented here.

Before we discuss the results of our experiment, we must first understand a few things like the general architecture of our system which I call Smart Frequency Data Extractor (SFDE).

6.1 System Architecture for Smart Frequency Data Extractor

Our system parses the HTML source code of Web pages first. It then performs various computations on the tokens obtained, like frequency counting, filtering, etc. Finally it selects some combinations of tokens and creates a list of such combinations. These combinations are used as prefix tags to identify and extract target items. The system starts with combination, which has the highest frequency of occurrence. The results then obtained are displayed to the user for verification purposes. If the user is not satisfied, he may simply reject the output, in which case the tag pattern next in the list, which has the second highest frequency, is used. This process is repeated until the user is satisfied or if our system runs out of patterns (the possibility of which is rare). In the latter case, our attempt is declared as failed. Fig 6.1 below is the pictorial representation of the architecture of our system.
6.2 Experimental Results

Based on the algorithm described above, we have developed a prototype of the wrapper generation system (which we call Smart Frequency Data Extractor or SFDE) and used it to run a number of experiments on real HTML sites. The system has been completely written in Java. All experiments have been conducted on a machine equipped
with an Intel Core 2 Duo processor working at 1.8 GHz, with 2 GB of RAM, running Windows Vista and Eclipse SDK version 3.3.1.

Unlike others, our algorithm is not initialized at all. Our system is directly fed with the particular web pages of each web site that actually contains the product information. The algorithm produces more than one candidate wrapper for each such page. It then applies the wrapper with the highest frequency to extract the data. If the user is not satisfied, he may discard the result by clicking on the reject button. In which case, the algorithm selects the next wrapper from the list containing candidate wrappers in the order of descending frequency.

Figure 6.2 consists of two tables that list results relative to several well known data-intensive Web sites. While Table A refers to experiments we have conducted independently, in Table B we compare our results with those of other data extraction systems for which experimental results are available in the literature, namely RoadRunner [10], Wien [2] and Stalker [3].

Wien and Stalker are two wrapper generation systems based on a machine learning approach. But the major limitation of these approaches is their functionality:

1. First, the wrapper generator works by using a set of labeled pages provided by the user; the wrapper is inferred by looking at these examples and trying to generalize them.

2. Second, it is assumed that the wrapper induction system has some a priori knowledge about the page organization, i.e., about the schema of data in the page.

3. Finally, these systems generate a wrapper by examining one HTML page at a time.
RoadRunner on the other hand proposes a novel approach to the wrapper generation process. It automates the wrapper generation process to a larger extent and has some salient features such as:

1. It does not rely on user-specified examples, and does not require any interaction with the user during the wrapper generation process; this means that wrappers are generated and data are extracted in a completely automatic way.

2. The wrapper generator has no prior knowledge about the page contents, i.e., it does not know the schema according to which data are organized in the HTML pages.

3. This system works with two HTML pages at a time. Pattern discovery is based on the study of similarities and dissimilarities between the pages, and mismatches are used to identify relevant structures.

Although, like us RoadRunner proposes a completely automatic extraction method, this method is less accurate than the systems that ask the user to label training pages.

We propose an automatic extraction method, that does not require manual labeling at all, yet it is highly accurate. It works on the observation that relevant information is repeated several times and hence it finds and extracts the information that is repeated the most number of times.
Table A: Results of experiments conducted using SFDE on randomly chosen websites

<table>
<thead>
<tr>
<th>Serial #</th>
<th>Web Site</th>
<th>Experimental Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Alight.com</td>
<td>Yes</td>
</tr>
<tr>
<td>2.</td>
<td>Amazon.com</td>
<td>Yes</td>
</tr>
<tr>
<td>3.</td>
<td>Avenue.com</td>
<td>Yes</td>
</tr>
<tr>
<td>4.</td>
<td>Bargainoutfitters.com</td>
<td>Yes</td>
</tr>
<tr>
<td>5.</td>
<td>Circuitcity.com</td>
<td>Yes</td>
</tr>
<tr>
<td>6.</td>
<td>Computer4sure.com</td>
<td>No</td>
</tr>
<tr>
<td>7.</td>
<td>Pacificgeek.com(formerly known as comput ersurplusoutlet.com)</td>
<td>Yes</td>
</tr>
<tr>
<td>8.</td>
<td>Dell.com</td>
<td>No</td>
</tr>
<tr>
<td>9.</td>
<td>Slickdeals.net</td>
<td>Yes</td>
</tr>
<tr>
<td>10.</td>
<td>Laweekly.com</td>
<td>Yes</td>
</tr>
<tr>
<td>11.</td>
<td>Kmart.com</td>
<td>No</td>
</tr>
<tr>
<td>12.</td>
<td>Kohls.com</td>
<td>Yes</td>
</tr>
<tr>
<td>13.</td>
<td>Deals.com</td>
<td>Yes</td>
</tr>
<tr>
<td>14.</td>
<td>Officemax.com</td>
<td>Yes</td>
</tr>
<tr>
<td>15.</td>
<td>Dealsea.com</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table B: Comparison of results of SFDE with the results obtained from other systems

<table>
<thead>
<tr>
<th>Serial #</th>
<th>Web Site</th>
<th>Comparative Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>URL</td>
<td>SFDE</td>
</tr>
<tr>
<td>1</td>
<td>iaf.net</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>bigbook.com</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>laweekly.com</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 6.2: Experimental Results

6.3 Advantages over other strategies

SFDE has a unique approach that has many advantages over other techniques. Some of them have been discussed in this section.

• No dependence on Knowledge Base

Unlike many other systems, SFDE does not have a knowledge base. It does not rely on, and hence does not require, any previous knowledge. It is smart enough to understand the design of the HTML Web page on its own and takes the required decisions accordingly.
• **Does not require manual labeling of pages**

Since there is no knowledge base in SFDE, it does not require manual labeling of the pages, which is an essential requirement in many other systems. This results into least amount of efforts on the user’s part.

• **Ease of use**

This advantage is in direct consequence from the first advantage listed, independence from knowledge base. Since no manual labeling is required for SFDE, it can be used even by a novice user who does not have any knowledge related to labeling of Web pages.

• **Open to all the Web pages**

The systems that are based on knowledge base, generally work only on a cluster of closely related Web pages that fall under the domain and comprehension of the information contained in the knowledge base. SFDE stands one step ahead of such systems in this regard since it is open for use on any Web page.
Chapter 7

Conclusions and Future Work

This work presents a novel approach for the extraction of relevant information from Web pages. Experimental results have shown that our approach can be implemented for practical purposes. Following are some suggestions which would definitely be helpful for future work.

7.1 Refinement of the existing technique

For future work, the approaches implemented in this work can be further refined to give better results e.g. the threshold frequency for the selection of high frequency tags and high frequency tag patterns can be further calculated thorough experimentation to give better results.

The concept itself can be implemented using different techniques. Some other choice of technique may produce better results than the one presented in this paper. The selection of appropriate technique would determine the outcome and accuracy of the results.

7.2 Enhancement to extract particular data items

Our algorithm extracts relevant information as a whole. It can further be refined to extract specific data from the relevant information. This can be done by following the same technique but applying it to deeper levels. This would probably be a revolutionary work.
7.3 Integration with other techniques

Our concept can be combined with some other (existing or nonexistent) technique to improve the existing technique, or to produce a new hybrid approach.

Also, the combination could be done totally or partially; instead of using the whole concept, only a part of it could be used to bring some improvements to existing works.
Reference


[4] Pinto, D., McCallum, A., Wei, X. and Bruce, W. Table Extraction Using Conditional Random Fields. SIGIR-03.


