The Pennsylvania State University
The Graduate School

EFFECTIVE METHODS FOR WEB CRAWLING AND WEB INFORMATION EXTRACTION

A Dissertation in
Computer Science and Engineering
by
Shuyi Zheng

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2011

The Dissertation of Shuyi Zheng was reviewed and approved* by the following:

C. Lee Giles
David Reese Professor of Information Sciences and Technology
Dissertation Advisor, Chair of Committee

Jesse Barlow
Professor of Computer Science and Engineering

Daniel Kifer
Assistant Professor of Computer Science and Engineering

Murali Haran
Associate Professor of Statistics

Raj Acharya
Department Head and Professor of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Crawling and information extraction are two fundamental components for almost all web-scale search engines. They are usually the first two steps of a search engine's system pipeline. First, a web crawling system (a.k.a "crawler", "robot", or "spider") traverses the web in certain manner and provides the raw content (crawled webpages) for search engines. Then, an extraction system is used to understand those crawled pages correctly before they can be indexed and presented to the end user.

This research includes several works in the context of real systems which attempt to address the some fundamental issues that are often encountered in web crawling and web information extraction. (1) How to design and manage a large scale web crawler (2) How to select seeds for a web-scale crawler; (3) How to order URLs to effectively obtain relevant documents (4) How to extract information from news articles and homepages that do not conform to any template; (5) How to combine template detection and wrapper generation for template dependent information extraction; (6) How to reduce the labeling cost for wrapper induction.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Professor C. Lee Giles, for his great guidance and support. In these years, he have not only provided me guidance, but also inspired me with his passionate attitude on research. It truly was my honor to be one of his students. Therefore, I'd like to take this chance to sincerely thank Dr. Giles for advising and supporting me.

At the same time, my committee members, Professor Jesse Barlow, Professor Daniel Kifer, and Professor Murali Haran, have spent much of their time helping me improve my thesis. In addition, Ruihua Song and Ji-Rong Wen, and Pavel Dmitriev, who were my internship mentors from Microsoft Research Asia and Yahoo! Labs, introduced me to the competitive industry work environment and guided me to solve many challenging problems that I encountered. I learned much from each of them and appreciate all the things they have done for me.

Again, I am very grateful for the support and opportunities that my advisor, committee members, mentors, and coworkers provided. Its them who made my Ph.D education meaningful and interesting.

Chapter 1

Introduction

1.1 Importance of Web Crawling and Web Information Extraction

Crawling and information extraction are two fundamental components for almost all web-scale search engines. They are usually the first two steps of a search engine's system pipeline. First, a web crawling system (a.k.a "crawler", "robot", or "spider") traverses the web in certain manner and provides the raw content (crawled webpages) for search engines. Then, an extraction system is used to understand those crawled pages correctly before they can be indexed and presented to the end user.

This proposal includes several ongoing works which attempt to address some fundamental issues that are often encountered in web crawling and web information extraction.

1.2 Crawler Seed Selection

Crawling is not only one of the most important tasks of a search engine, but also an indispensable part of many other web applications. The breadth, depth, and freshness of the search results depend crucially on the quality of crawling. As the number of pages and sites on the web increases rapidly, deploying an effective and efficient crawling strategy becomes critical for a search engine.

A typical crawler [1] works as follows. It maintains a list of unvisited URLs called *frontier*, which is initialized with seed URLs. In each crawling loop, the crawler picks a URL from the frontier, fetches the corresponding page, parses the retrieved page to extract URLs, and adds unvisited URLs to the frontier. Typically, the crawler needs to keep revisiting some or all the URLs to check if they are updated since the last crawl. Due to the infinite nature of the web and the competition between getting new URLs and refreshing the old ones, even web-scale crawlers can never crawl "all" the URLs from the web. Therefore, an effective crawling strategy is required by the crawler to avoid getting lost in the infinite web.

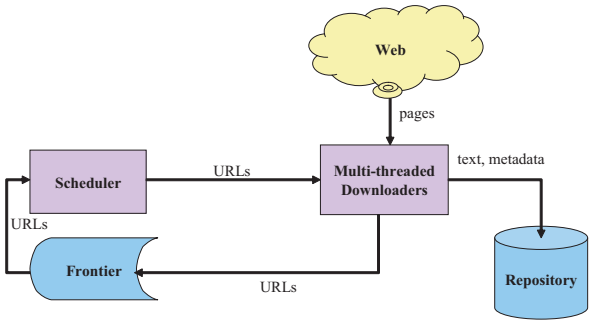


Figure 1.1: Crawler

Crawling strategies mainly differ in the ways of ordering URLs in the frontier, which results in exploring the web differently. However, all the strategies start from the seed pages and proceed by exploring the neighborhoods of the seed pages. Thus, to a large extent, selecting good quality seed determines the quality of the crawl.

One may think that simply starting from root pages of several well known sites and crawling very deep will allow the crawler to reach all useful pages on the web. Unfortunately, this is not so. As Broder et. al [2] showed, even 9 years ago, close to half of all web pages could not be reached from the "central" strongly connected portion of the web. Moreover, the situation is likely to be even worse nowadays. Recently, many websites that contain millions of pages have emerged. They are either dynamically generated from a backend database (e.g., www.amazon.com), or contributed by a user community (e.g., www.wikipedia.org). Due to lack of connections between some content, those websites are not always strongly connected. We found that even www.cnn.com, such a well known web

site, contains components which are disconnected from the rest of the web site, according to our crawler's point of view[1]. Another reason why the web is not well connected from a crawler's point of view is the difficulty in extracting links which are hidden in sophisticated javascript functions.

The above points are illustrated on Figures 1.2 and 1.3. Figure 1.2 shows the distribution of the relative sizes of the largest strongly connected components (SCC) and largest connected components (CC). These distributions are obtained from 100 large web sites selected from the Open Directory, based on an experimental web-scale crawl conducted in summer 2008. Here, "relative size" means the percentage of the largest component in the whole corresponding website. The results indicate that, for almost half of the sites, the largest strongly connected component contains less than 10% of the total number of pages (Figures 1.2), and around 50% of the web sites are not even connected (Figures 1.3). Because of these reasons, using root pages of several web sites alone may only allow the crawler reach a small portion of the whole web. Many seeds in different regions of the web, even multiple seeds from a single website, may be required to reach all useful pages on the web.

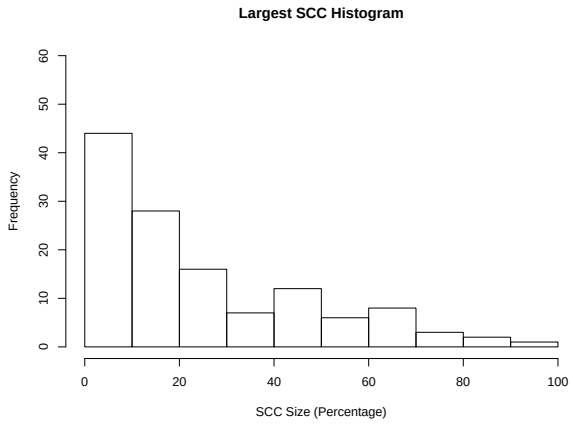


Figure 1.2: Distribution of Largest SCC's Relative Size

In this work, we study the problem of crawler seed selection and propose a seed selection framework based on the analysis of the link structure of the web.

[1]These disconnected components are special news coverage for important past events, such as presidential elections.

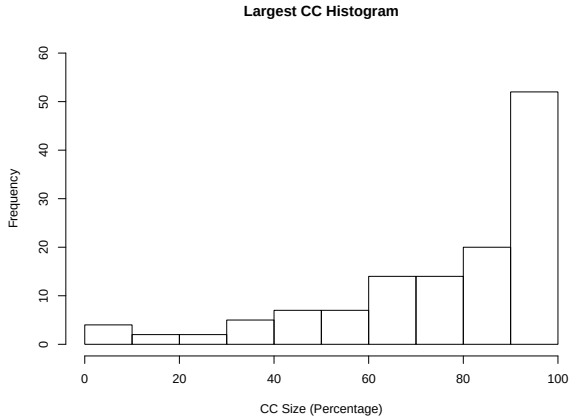


Figure 1.3: Distribution of Largest CC's Relative Size

We assume the incremental crawler model, where the crawler crawls continuously downloading new pages and refreshing the old ones, and seeds are updated periodically to keep up with the changes in the web. For many websites, an incremental crawler could not reach all the pages. From the crawler's point of view, such websites have infinite number of pages. Given limited resources, we usually need to restrict the number of seeds and number of hops from each seed. Otherwise, crawling new pages will consume all the resources and already crawled pages could never be revisited.

The major contributions of our work are as follows.

1. We identify the seed selection problem in large scale incremental crawlers.
2. We propose a graph based seed selection and evaluation framework and designe several seed selection algorithms within that framework.
3. We test the effectiveness of the algorithms on real world data, and show that our algorithms significantly outperform heuristic seed selection approaches.

1.3 Template-Independent Information Extraction

Some webpages are generated from a specific template to present a uniform layout or appearance. Some webpages are manually written and do not conform to any

template. The problem of template-independent information extraction is to train an "universal" extraction module which is able to handle pages from any template or even manually written pages. Such module is usually trained to extracted domain specific information. Our work in this area includes news extraction and author metadata extraction.

1.3.1 Template-Independent News Extraction

News articles are usually embedded in semi-structured format, e.g., HTML, as Figure 1.4 shows. To extract structured data from HTML pages, some tools and algorithms were reported in the literature on Web information extraction [3, 4]. Among these approaches, the most traditional method is to generate a software or program, called wrapper, to extract data based on consistency of HTML DOM trees.

DOM tree-based approaches are template-dependent. The generated wrapper can only work properly for pages that share a specific template. Any change of a template may lead to the invalidation of a wrapper. Therefore, it is costly to maintain up-to-date wrappers for hundreds of websites.

When information sources expand to domain level, it is difficult (if not impossible) to induct a traditional DOM tree-based wrapper because pages from different websites lack structural consistency. First, similar content might be represented by different tags in pages from different Websites. To prove this, we did a statistical study on 295 Web news pages and found more than 10 types of tags used to wrap news headlines. The tags include `<B>`, `<H1>`, `<FONT>`, and `<SPAN>`. Second, DOM trees generated by different templates can have entirely different topological structures. For instance, news content might appear at different levels of DOM trees. We have also performed a statistical experiment on the same 295 Web page set. The results indicate that the depth of news content distributes in a wide range (from two through 29). To demonstrate these points, we compare two news pages (Figure 1.4) whose DOM trees are shown in Figure 6.1a and Figure 6.1b. As the figures illustrate, the DOM trees differ in terms of tag and topological structures.

Despite the structural dissimilarities among pages of different templates, people can still easily identify a news article from a news page at a glance. For example,

(a) from ABCNews.com (in English) (b) from News.BBC.co.uk (in Arabic)

Figure 1.4: Two sample news pages

when people browse the two news pages in Figure 1.4, they can quickly locate the news articles without any difficulty, even in the case in which the news is written in a language they may not understand (e.g., Arabic news page).

In this work, we discuss why human beings can identify news articles regardless of templates. We propose a novel news extraction approach that simulates human beings. The approach is template-independent, because it is based on a more stable visual consistency among news pages across websites. First, we represent a DOM node with a block and basic visual features to eliminate the diversity of tags. Second, extended visual features are calculated as relative visual features to the parent block. Such features can handle the dissimilarity of topological structure. Third, we combine two classifiers into a vision-based wrapper for extraction. Experimental results indicate the proposed approach can achieve high extraction accuracy as about 95% in terms of F1-value which substantially outperforms exisiting techniques.

1.3.2 Template-Independent Author Metadata Extraction

Authors are kernel objects for any online digital library. Most available information of an author is extracted from his/her online documents, e.g., a research paper [5]. This approach has two limitations. First, only a few fields are available in an author's online document. Second, such information tends to be out of date.

In this work, we attempt to extract author meta-data from their homepages.

We choose homepage domain as the extraction source because it is more reliable, comprehensive and up-to-date than any other sources.

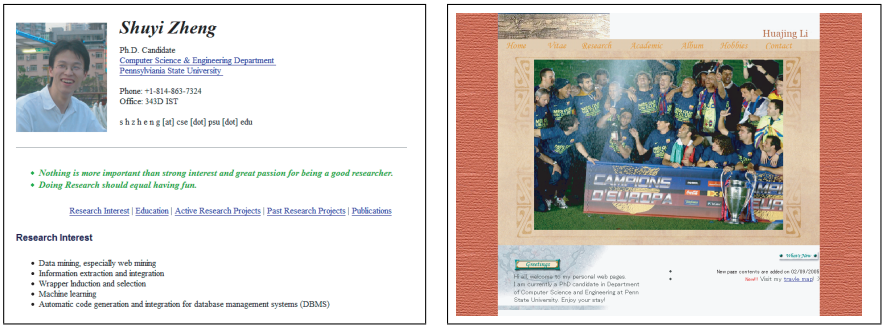
If we could extract authors' relevant information from their homepages, not only this information can serve as meta-data for a digital library, but also many valuable services and applications can be developed based on the enriched author information. Here are some typical examples. 1) Author's meta-data can be used in ranking authors and their publication documents for a digital library. 2) Enriched author meta-data would help a lot to distinguish multiple authors with same name. 3) Based on extracted author meta-data, we can even build a vertical search engine [6] to search scholars over the world.

However, extracting information from homepages is not an easy task because most homepages are manually designed and encoded. People design their homepages in all kinds of ways, which leads to the diversity of their web appearance. Traditional machine learning methods encounter difficulty when dealing with homepage domain because features found on some pages might not applicable on others.

In this work, we proposed to take advantage of visual features to help improve the extraction accuracy. In some sense, visual features are more stable than content-based features for certain types of information which an author puts on his/her homepage. This is understandable because when people design their homepages, they usually will follow some hidden conventions of the way they put certain types of information on the page.

Due to the noisy nature of homepage data, solely depending on individual features is still hard to achieve high extraction accuracy. In this work, we propose to consider the relation of different fields on a same page. Actually, when people design their homepage, they follow some conventions not only for design some individual fields, but also on how to arrange multiple fields on a same page. This observation gives us a hint that we could learn a probability model to simulate the relation between different fields. That is what we call inter-field probability model in this work.

However, with all above ideas taken into account, it is still a hard problem to find a uniform method to handle all kinds of different cases. In order to apply our method, we first need to limit the range where our approach is applicable. By observation, most homepages can be grouped into two types. In the first type,



(a) Type-1 (b) Type-2

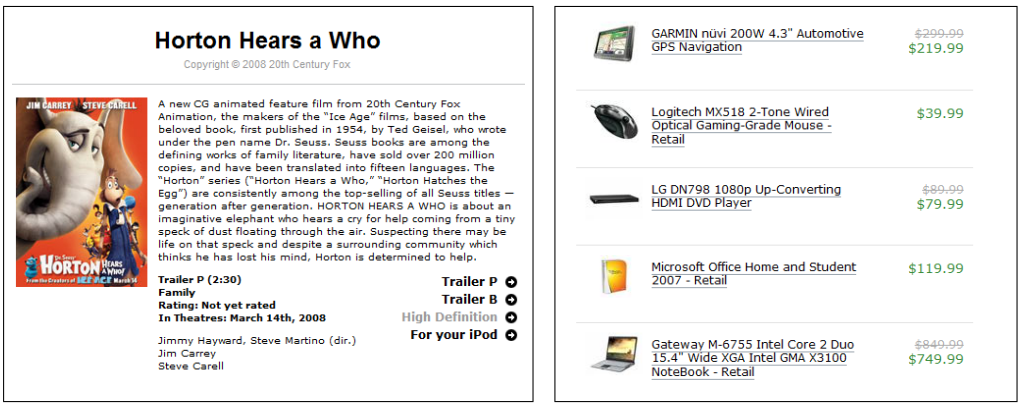
Figure 1.5: Two Types of Author Homepage

most (if not all) relevant information of the author has been presented on the first page (e.g., index.htm); while, in the second type, author information is evenly broken into several categories. Each category has its own page. Normally, the first page of this type only conveys a little self-introduction or greetings. In terms of proportion, type-1 homepages is the majority of all homepages. And our work is focused on dealing with the type-1 homepage and leave type-2 as future work.

1.4 Wrapper Induction: A Template-Dependent Information Extraction Method

Many websites like Amazon.com are data-intensive, and information on them comes from structured sources and is often presented in the form of a *Web record* which exists in both detail and list pages. For example, in Figure 1.6a, a movie record is presented in a detail page from `www.apple.com`; in Figure 1.6b, five product records are presented in a list page from `www.newegg.com`.

Although Web records are normally rendered from a structured source with semantic definitions, e.g., a back-end relational database, they are often encoded into semi-structured HTML pages that employ templates for rendering. Because of that, such web record information is difficult for computers to understand without proper pre-processing. Given a host webpage and related information needs, how to automatically identify relevant records as well as their internal semantic structures is critical to many online information systems. To achieve high accuracy, the task



(a) Detail Page (b) List page

Figure 1.6: Example of Web Records in a Detail Page and a List Page

of extracting structured information from web pages is usually implemented by programs called wrappers. Generally, a wrapper can be defined as a software or program which is used to extract desired information and transform the extracted data to a structured format. The process of leaning a wrapper from a group of similar pages is called *wrapper induction*.

Wrapper induction is template dependent. A wrapper is usually generated just for a specific template and can only handle pages belonging to the same template. Even with this limitation, wrapper induction is still one of the most popular methods to extract web information and is extensively used by many commercial information systems including major search engines. One of the primary reasons is its high extraction accuracy.

Manually writing wrappers for Web sources [7] is a tedious, time-consuming, and error-prone job, thus the study of automatic wrapper induction using machine learning techniques has been motivated in recent years [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18].

Our work in this area also focuses on wrapping Web sources in an automatic manner. In particular, we attempts to address three challenges in wrapper induction problem.

1. How to combine template detection in wrapper induction?

2. How to extract web records with complex structure?

3. How to reduce the human effort in manually labeling?

The following sub-sections describes each challenge in detail.

1.4.1 Joint Optimization of Wrapper Generation and Template Detection

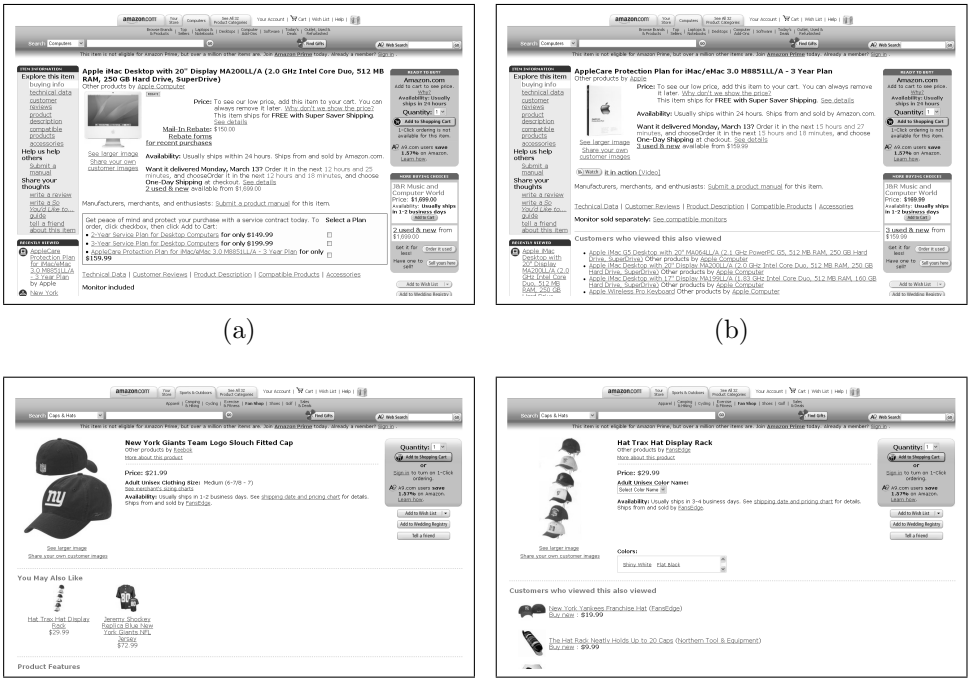
Although different wrapper induction systems employ various techniques and strategies to generate wrappers, they all separate template detection from learning wrappers. The detection groups training pages into several clusters or classes, based on some cues like URLs. For example, Crescenzi et al [19] assume that pages belonging to the same template are located at the same sub-directory of a website. Thus, pages are considered to share a common template if their URLs fit a common schema. Grouped pages are fed into an induction module then. The module generally assumes that a group of pages conform to a common template, and generate a wrapper per class.

The separated template detection strategy has two limitations:

1. With the popularity of dynamic URLs, it is no longer as effective to detect templates by URLs as before, especially for some large-scale websites. Figure 1.7 lists four sample pages collected from Amazon.com. From their appearances, it is easy to tell that Figure 1.7a and Figure 1.7b share a common template, and Figure 1.7c and Figure 1.7d share another template. Comparing their URLs, we find there is no cues to group the pages correctly.

2. Even if URL can group pages that share a template, it is far from optimal to generate only one wrapper for a complex template sometimes. For example, by looking closely at page (c) and (d) in Figure 1.7, we observe that page (d) is different from page (c) in some aspects. Page (d) has an additional attribute named Colors, and it lists some also-viewed items in a column whereas page (c) lists some you-may-also-like items in a row. Building two wrappers for such a complex template may achieve better extraction accuracy.

To solve the problems above, we propose to detect template solely based on the similarity among page representations that are also used in wrapper generation.



(a) (b)

(c) (d)

(a)=http://www.amazon.com/gp/product/B000BNLGJA/
(b)=http://www.amazon.com/gp/product/B00007J8SC/
(c)=http://www.amazon.com/gp/product/B0000DD95R/
(d)=http://www.amazon.com/gp/product/B0000DD95R/

Figure 1.7: Sample pages from www.amazon.com

In our system, tree structures are used as representations for pages and wrappers. Based on a distance metric between a page and a wrapper, a clustering algorithm is employed to cluster similar enough pages into a class and induce a central wrapper for the class at the same time. Such joint approach makes it possible to optimize the final performance of extraction by involving template detection in the training process.

Compared with prior works, our approach has two advantages:

1. Our approach is more stable because it does not rely on URLs or any other external features to detect templates. Instead, we attempt to detect templates based on inner structure similarity of pages, which is consistent with the principle of wrapper induction.

2. Given a set of pages, the number of wrappers is determined by how simi-

lar they are. This number can be optimized under the criterion of overall extraction accuracy of generated wrapper set.

1.4.2 Record-Level Wrapper Induction

Most traditional wrapper techniques have issues dealing with web records since they are designed to extract information from a page, not a record. Specifically, they usually have difficulty in handling the case of *cross records*. For example, Figure 1.8a is a part of a webpage taken from `www.amazon.com` and Figure 1.8b is the corresponding HTML source. In this example, the pictures (`<IMG>`) of three products are presented together in the first row (`<TR>`), while their names (`<A>`) are shown in the second row (`<TR>`). Those three records' HTML sources are crossed with each other. Even though record boundaries are visually clear when such page is rendered for user browsing, as shown in Figure 1.8a, there is no clear boundary for partitioning different records from the HTML source. As a result, the lack of record boundaries makes extracting records difficult for most existing wrapper induction methods, especially for those which learn wrappers by inferring repeated patterns from the HTML source. This problem occurs frequently in real-life websites. Almost all image search engines (e.g., Google Image Search) design their search result pages with such layouts. Many shopping websites (e.g., `www.amazon.com`, `www.bestbuy.com`, `www.diamond.com`, `www.costco.com`, etc.) also list some of their products in a similar way as in Figure 1.8.

Another issue of many page-level wrapper induction methods is the expensive cost of performing tree-mapping for both wrapper induction and data extraction [20]. In these methods, two complete tag-trees need to be aligned with each other, even though most parts of the DOM-tree do not contain user-interested data. Thus, mapping irrelevant information not only wastes lots of runtime, but also interferes with the accuracy of data extraction.

To address the above issues, in this paper, we investigate the wrapper induction technique in record level. We implement a record-level wrapper system and use a novel "broom" structure to represent both records and generated wrappers. With such representations, our system is able to effectively extract records and identify their internal semantics at the same time. In particular, our record-level wrapper

(a) Page Layout

```
<TABLE>
  <TR>
    <TD>
      <IMG SRC="Plates.jpg"/>
    </TD>
    <TD>
      <IMG SRC="Tableware.jpg"/>
    </TD>
    <TD>
      <IMG SRC="Serveware.jpg"/>
    </TD>
  </TR>
  <TR>
    <TD>
      <A>Plates</A>
    </TD>
    <TD>
      <A>Seasonal Tableware</A>
    </TD>
    <TD>
      <A>Serveware</A>
    </TD>
  </TR>
</TABLE>
```

(b) HTML Source

Figure 1.8: Example of Cross Records

technique makes the following contributions:

1. We propose a novel "broom" structure to represent a record, thus our system provides a uniform approach for extracting records from both detail pages and list pages. The record-level wrapper induction approach performs better than the page-level approach and can effectively handle the case of cross records.

2. The record-level approach achieves better efficiency than the page-level approach. The cost of tree-alignment is reduced dramatically by restricting the alignment in the relevant-region of DOM-trees and constructing a wrapper library to avoid duplicated matching.

3. We propose using context words to disambiguate different attributes that are embedded in similar HTML tag trees. Detecting ambiguous attributes and learning context words are fully automatic.

1.4.3 Reduce Labeling Cost

Automatic wrapper induction methods can be divided into two groups based on the type of manual labeling process involved. Some approaches [21, 22, 23, 17] generate wrappers without pre-labeled training samples. In practice, however,

these methods are difficult to apply to commercial systems which demand high extraction performance. In contrast, some other systems require manually labeled examples as training data. A typical workflow of methods in this category is as follows. First, a user defines an extraction schema and labels a set of training pages with the schema. Then, all training pages are fed into a wrapper-induction system that generates one or more wrappers. Finally, when a new page is acquired, a wrapper is selected to extract data and fit the extracted data into the pre-defined schema. This category of method can achieve excellent extraction performance, but can be problematic as labeling is costly and error-prone.

We observed that in some practical applications, we find that using labels in wrapper induction can be beneficial. We argue that if a system is well-designed, we can minimize labeling cost to an acceptable level while maintaining the advantages of labeling.

In the work, we propose an interactive wrapper induction system, called *Pictor*. The system involves users in defining a schema and labeling a few example pages. Initially, a wrapper is generated upon the initial one or two labeled pages. Next, the wrapper is applied to assist further labeling. A page that the wrapper cannot handle well is shown to users as the next page for labeling. In contrast to the initial labeling, users no longer need to work on a clean page but a page with some automatic extractions. Users now only label missing attributes or correct wrong labels. Then, the wrapper is incrementally improved by the newly labeled data. Labeling continues until the predicted performance satisfies a user's requirements.

Our goal is to minimize labeling cost as well as obtaining high extraction accuracy. Three novel techniques are record-level wrapper induction and extraction, wrapper-assisted labeling, and extraction performance estimation. In our Pictor system, labeling effort is reduced primarily in three ways. 1) Our system is able to reduce the labeling cost within one page. Not all records need to be labeled manually. 2) When a new unlabeled page is available for labeling, it is likely that most of the labeling can be done automatically. 3) Our technique has the ability to intelligently select the pages that are really worth labeling.

Experiments are conducted upon three kinds of websites, such as online shops, restaurant reviews and a digital library. The results indicate that our proposed approach can achieve as high as 99% F1-value while saving 90% labeling efforts.

It also shows that the function of predicted performance is highly close to the true performance when a few pages are labeled.

Chapter 2

Related Work

2.1 Web Crawling

In this section, we briefly review related work on crawling and crawler seed selection. Although crawling is a well studied area, there is surprisingly little work directly addressing the problem of seed selection. There are, however, more works related to specific aspects of our work. We discuss these works below. For a comprehensive survey on general crawling please see [1].

In [24], the authors use Bayesian classifiers to estimate the link distance between a crawled page and the relevant pages. The classifiers are trained from context graphs generated from training pages. Although the goal of this work is to solve the problem of finding more relevant pages on a specific topic, the idea of using graph structure to find relevant pages is similar to our approach to seed selection.

Another work which utilizes graph structure of the web is [25]. In this work, given several example pages from a web site, a crawler is generated to crawl similar pages from this site, where two pages are defined to be "similar" if they share the same navigation pattern from the entry page to the target page.

One of the most popular problems in crawling research is the problem of scheduling urls from the frontier for crawling. One of the earliest works in this area is [26], which proposes a crawling strategy such that pages with higher PageRank get crawled first. In [27] the authors evaluate several scheduling strategies and use PageRank as a metric to evaluate the quality of downloaded pages. This work shows that breadth-first crawling strategy can yield high-quality pages. This sup-

ports the assumption we made in this paper of crawling exactly h hops from a seed page.

Works more relevant to the seed selection problem belong to the focused crawling area [28, 29, 30, 31, 32]. In focused crawling the goal is to obtain a collection of pages on a given topic, wasting as little crawler resources as possible on crawling off-topic pages. Here pages relevant to the topic are used as crawler seeds. These seeds can be provided by the user or derived from a search engine by submitting queries relevant to the topic [32, 33]. The assumption behind this approach to seed selection is that relevant pages are located close to each other in the web graph [34, 35, 36]. This assumption is similar to the one that motivates the *PageRank* approach to seed selection discussed in this paper.

Another related work to our seed selection problem is the K-Coverage problem. This is a well-known problem in the area of sensor network. In recently years, some approximation algorithms have been proposed ([37, 38, 39]). For example, [38] designed a greedy approximation algorithm that delivers a connected 1-cover within a $O(lgn)$ factor of the optimal solution. [37] used a localized heuristic for the problem with some assumption on the communication behavior between sensors. [39] generalized the work in [38] to the connected K-coverage problem and designed a greedy algorithm that returns a solution within $O(lgn)$ factor of the optimal.

The work most relevant to our work is [40]. In this work, the authors consider the problem of discovering newly appearing pages on the web by recrawling as few old pages as possible. They consider a bipartite graph consisting of old pages, recent new pages, and a link from an old page to a new page if the new page can be discovered from the old page by crawling only new pages. The authors then propose several graph-based algorithms for optimally selecting old pages to cover the maximum number of new pages.

There are two important differences between our work and [40]. First, [40] makes a very strong assumption of having the complete crawl of the web at a certain point in time. In contrast, our work assumes that the crawl is incomplete and selects the seeds taking into account information about pages known to the crawler but not yet crawled. Second, our framework optimizes the value covered, while [40] optimizes overhead - number of old pages crawled to discover the new pages. Optimizing value allows us to select the seeds not only for discovery of new

pages, but also for crawling known uncrawled pages and refreshing crawled good pages. In this sense, the framework we proposed can be viewed as a generalization of [40]. The problem of discovering new pages can be modeled in our framework by assigning non-zero values only to new pages.

2.2 Web Information Extraction

Our work belongs to the area of *Web Information Extraction* (IE). It has received a lot of attention in recent years. We group all related works into three categories and briefly describe each class as follows. We refer the reader to a good survey [3] and two tutorials [41][4] for more works related with IE.

2.2.1 Wrapper Induction Techniques

The most popular technique for this category is wrapper induction. Several automatic or semi-automatic wrapper learning methods have been proposed. For example, WIEN [8] is the earliest method as we know on automatic wrapper induction. Other representatives are SoftMeley [9], Stalker [10], RoadRunner [13], EXALG [23], TTAG [16], works in [15] and ViNTs [17].

In previous work, page clustering for wrapper induction is considered a trivial task in most previous wrapper induction systems. Among them only RoadRunner [13, 19] and works in [15] proposed automatic approaches to implement page clustering for wrapper generation. Other methods all manually collect training pages template by template.

In [19], page clustering for RoadRunner system is discussed. They use four types of page features to calculate the similarity between two pages: (1) distance from the home page; (2) url similarity; (3) tag probability; (4) tag periodicity. Based on page similarity, they adopt a popular non-supervised clustering algorithm *MiniMax* to conduct the page clustering process. This process is isolated from the wrapper generation process, which is the primary difference compared with our WPC algorithm. Wrapper selection problem is also discussed in [19].

In [15], tree edit distance is used to measure the distance between two pages.

They use traditional hierarchical clustering techniques [42] in which the distance measured is the output of a restricted top-down tree mapping algorithm (RTDM). The RTDM algorithm does not distinguish node tag and it is designed only for finding the main contents in news pages. This restricts their method from being applied to general information extraction problems. Similar to our system, works in [15] can also derive a similarity between a wrapper (called extraction patterns in their work) and a page when selecting proper wrapper for extracting data from a new page. However, template detection is still isolated from the wrapper generation process.

We need to mention TTAG because wrappers in TTAG are also presented as a tree structure with wildcards. The authors also employ a top-down layer-by-layer alignment and use dynamic programming method in each level's alignment. But the alignment in any layer is isolated from that in other layers. As a result, child nodes can still be aligned even when their parent nodes do not match. That is a different strategy from ours. Moreover, like most other previous systems, template detection is not discussed in TTAG.

ViNTs [17] uses some visual features in generating wrappers for search engine results. In the sense of using visual cues, it is advantageous when contrasted with previous work. However, the visual features used in ViNTs are only limited to the content shape-related features. They use them to help identify the regularities between search result records. Therefore, ViNTs still depends on structural similarities and must generate a wrapper for each search engine.

2.2.2 Single Page Record Mining Techniques

Some researchers also attempt to extract or segment data records embedded in a Web page without a pre-learned wrapper. The most representative methods are IEPAD [12] and MDR [22]. MDR also has an improved version DEPTA (also called MDR-2) [43].

DEPTA consists of two steps. First, it identifies individual data records in a page. In this step, visual information is used to construct a correct DOM tree thus improve their extraction accuracy. Then, a partial alignment technique is used to align and extract data items from the identified data records.

The precondition for both MDR and DEPTA is that the given page must have more than one data record. Therefore, they can not deal with detail pages (one page for one record).

2.2.3 Vision Assisted Techniques

The problem of judging the importance or function of different parts in a web page attracts a lot of attentions. Some proposed approaches [44, 45, 46, 47] take some advantages of visual information more or less. Kovacevic et al. [44] used visual information to build up an M-Tree, and further defined heuristics to recognize common page areas such as header, left and right menu, footer and center of a page. Some other works try to decide block importance levels or functions in a single web page, by learning algorithm [45] or random walk models [46, 47]. In these works, visual features, such as width, height, position, etc., are reported useful.

Chapter 3

CiteSeer^x Crawler: A Large Scale Incremental Focused Crawler

This chapter describes CiteSeer^x's document crawling system which aims at harvesting academic documents.

Digital libraries are normally built based upon large amount of downloaded documents. One measure for the quality of a digital library is the completeness of its document collection. However, due to the rapid growth of Web, collecting a rather complete document repository is becoming more and more difficult. It is not only time consuming, but also requires a suitable design of the crawling system as well as the deployed crawling policies.

The crawling system plays an important role in the whole CiteSeer^x architecture. It feeds huge amount of raw documents to the rest of the system.

Figure 3.1 illustrates the main flow of CiteSeer^x Crawler.

3.1 Seed Submission and Scheduling

A seed URL is a HTML page from which one or more academic documents are linked within a small number of hops. In our system, the Seed Submitter is used in two places. One is used to accept user submissions from http://citeseerx.ist.psu.edu/submit and pass them to the crawler in real-time. The other one is used by seed scheduler.

To be able to get new documents from URLs we visited before without going through the trouble of discovering these URLs again, every time we get a valid

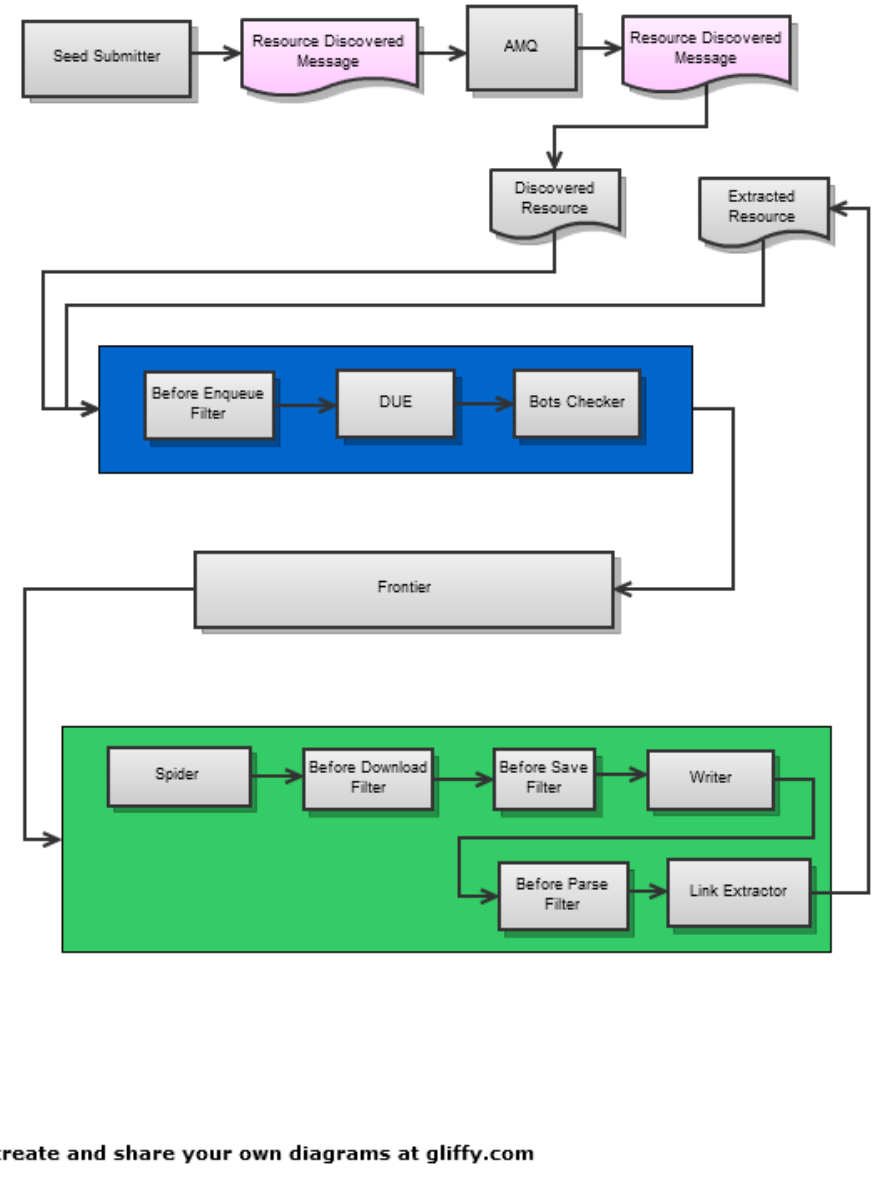


Figure 3.1: CiteSeerX Crawling System

document from a HTML page, we save the URL of this HTML page (called "parent URL") and mark the timestamp of last visit. As we have more than one million documents, the list of parent URLs are also very long. With limited hardware resource, we can only choose a small subset to revisit. To increase the likelihood of getting new documents, these seed URLs need to be chosen wisely. In our current system, the seed scheduler rank all the seeds mainly based on two factors of each

URL (1) Time between now and last visit. The older the URL, the more likely it will be scheduled; (2) Total number of documents we crawled from it. The older the URL is or the more documents we crawled from the URL, the more likely it will be scheduled. In such a way, we can even out URLs over time and also give preference to important URLs.

3.2 Submission Queue and Frontier Queue

We use Apache ActiveMQ as a stand-alone queuing service in our system. We use one queue for passing submitted seeds and one for serving as URL frontier. There are a few advantage of using a stand-alone message queue. First, the content on the queue is persistent to system failures. If the crawler itself is down, we won't lost any submitted seeds or ongoing URLs in the frontier. Second, the message queue is disk-bound instead of memory-bound. Therefore, it can support large scale crawl tasks without going out of memory.

3.3 URL Filtering

We do filtering in multiple stage during the crawling cycle. The rules used to control the filtering can be configured separately for each stage.

1. Before Enqueue: Some URLs can be discarded at this very early stage. E.g, URLs from a black list or non-HTTP URLs.

2. Before Download: At this stage, we mainly limit content type and content length. This help us avoid getting super large video files and other irrelevant content.

3. Before Save: At this stage, we decide whether the fetched content should be saved to disk. As we only need PDF/PS documents, we usually don't save HTML pages. But we do save the metadata of the HTML page which will be used for scheduling.

4. Before Parse: This is mainly just for HTML URLs. This step controls whether we want to further extract and follow links from a HTML. In our

system, we only follow two hops from a seed to avoid getting into crawler traps.

3.4 Metadata Management and Report

All the document metadata are saved in MySQL and managed through Django Framework.

1. URL
2. MD5: hashcode of the URL
3. Host (Website) of this document
4. Discover time of this document
5. Last content update time of this document
6. Parent URL: which URL this document is crawled from
7. State: Indicate whether this document is ingested to CiteSeer^x's indexer

We have a few daily cron job which analysis the metadata and produce various interesting reports. For example, we can rank the websites (institutions) based on the volume of documents we crawled from them. We can also show the Geo distribution of the documents. For a particular website or country, we can also show the historical trend on the number of documents. All these reports are viewable from an internal web site.

Chapter 4

Graph-based Seed Selection for Web-scale Crawlers

4.1 The Problem of Seed Selection

Since there exists a wide variety of crawling strategies, in this paper we make several assumptions about the crawling strategy. First, we assume that the number k of seeds is given. Second, we assume that a crawler crawls pages only within h hops from the seed and it always crawls all pages within h hops. We note that the values of h and k are related. Given a fixed repository size, the larger the number of seeds k is, the less hops h the crawler can crawl on average before it fills the repository. Third, we assume that the cost of downloading and processing a web page (measured in terms of time, CPU, and storage required) is constant.

The above assumptions are not crucial for the algorithms we will describe later, but they will significantly simplify the explanation. The cost of downloading and processing a page can be estimated based on the past statistics and factored into the algorithm. The values of h and k should be defined based on the application. For example, a crawler supplying pages for a large scale comprehensive search engine may need to follow more hops and have more seeds than a crawler for a news or blog search engine.

As we mentioned earlier, in an incremental crawler, seed selection happens periodically in order to keep up with the changes of the web. It is always based

on the information from a portion of the web which is already crawled. Thus, an intuitive definition of the problem of seed selection is, given a currently known portion of the web and the desired number of seeds k, to select seeds so that as many as possible new good pages will get crawled, and as many as possible currently crawled good pages will be retained.

Several heuristic approaches can be used for seed selection. In this paper we consider the following three approaches.

- *Random*. Select k pages at random.
- *PageRank*. Select k pages with the highest PageRank.
- *OutDegree*. Select k pages with the highest number of outlinks.

The *Random* approach is the simplest one which is used as a baseline in our experiments. The intuition behind the *PageRank* approach is that one can expect to find high quality pages around other high quality pages. The *OutDegree* approach is based on the assumption that pages with the high number of outgoing links will lead to a large number of other pages.

A heuristic that can improve the above approaches is to first split the known portion of the web into smaller units, such as web sites, and then select seeds independently for each web site. Such strategy allows distributing seeds "evenly" across the web, allocating to each web site a fraction of the total number of seeds (potentially zero), which is proportional to the site's importance or popularity. It also allows performing seed computation in parallel and on a smaller data set. We use this heuristic in the experiments described later in the paper.

Although the heuristic approaches presented above make sense intuitively, they do not directly optimize the criteria we are interested in. In the next section we formally define the problem of seed selection and present our graph-based framework for this task.

4.2 Graph-based Seed Selection

To formally define the problem of seed selection, we assume that every crawled web page has an associated value. Higher value indicates higher quality or higher

potential to discover new pages. The precise way to set the value of a page depends on the application. For example, for a large search engine, the value of a page can be higher if it was frequently clicked on the search result page. For a news search engine, the value may be higher for a page which represents a recent news article. The value can also be higher if there are uncrawled URLs the page links to. The value can be negative if the page is undesirable, such as a spam page. Given a web graph corresponding to a part of the web which is currently known to the crawler, we can define the seed selection problem as maximizing the total value of the portion of the web graph which is "covered" by the seeds.

Definition 1 (Seed Selection Problem)**.** *Given a directed graph $G = (V, E)$, a function $w : V \to \mathbb{R}$, assigning a weight $w(v)$ to every $v \in V$, and edges unweighted, given the number of seeds k, and the number of hops a seed covers h, select the seeds so that $w(\cup_{i=1}^{k} A_i)$ is maximized, where $A_i = \{v | v \in V, v$ within h hops of seed $i\}$, $w(A) = \sum_{v \in A} w(v)$.*

If $h = 1$, (crawler can travel only 1 hop from the seed) the seed selection problem becomes a variant of the *Dominating Set Problem*: a vertex v_i dominates vertex v_j if there is an edge $(v_i, v_j) \in E$ or $v_i = v_j$. The dominating set problem is to find a set of k vertices dominating the maximum-weight set of vertices in G. The dominating set problem is known to be NP-complete [48].

If $h > 1$, the seed selection problem becomes a version of a more general *Maximum K-Coverage Problem*. In this problem, we are given a set of elements U, each element $u \in U$ has an associated weight $w(u)$, a class of subsets of U, $\{A_i\}$, and a positive integer k. We then need to select k subsets $A_1, \ldots, A_k$ of U so that the total weight in $\cup_{i=1}^{k} A_i$ is maximized. In our scenario, the set U contains an element for each vertex v_i, and there is a subset A_i for each element, consisting of elements u corresponding to all vertices v reachable from v_i in h hops. K-Coverage problem is known to be NP-complete as well [49].

In [49], a greedy iterative algorithm for the maximum k-coverage problem is proposed, which achieves $1 - \frac{1}{e}$ approximation (Algorithm 1). The algorithm proceeds in k steps, selecting on each step i the subset $A_i \in \mathbb{R}$ such that $w(A_i)$ is maximal. If maximal $w(A_i)$ cannot be computed precisely, but is guaranteed to be within a factor of β of the maximum, the algorithm guarantees $1 - \frac{1}{e}\beta$ approximation.

K-Coverage problem is also a well-known problem in the area of sensor network. Recently, there has been a lot of research done to address the problem with approximation algorithms ([37, 38, 39]). Although those works are for different areas and solving different problems, the idea of using approximation algorithms is similar to ours.

Algorithm 1 Seed Selection Algorithm

Input: Weighted Graph G of (a portion of) the web, Maximal Seed Number k, Number of hops allowed h
Output: Selected Seeds $\mathcal{S}$

Algorithm:

1: **FOR** $i = 1$ to k
2: Find vertex v^* which covers maximal value within h hops
3: Make h hops from v^* on graph G
4: Set the values of all covered vertices to zero
5: Add v^* to $\mathcal{S}$
6: **IF** values of all vertices are zero **THEN**
7: BREAK
8: **ENDIF**
9: **ENDFOR**

Unfortunately, performing even a single iteration of the algorithm is computationally expensive for large h, due to the exponential in h complexity of step 2. For example, if the number of outlinks of a page on the website is l, finding the optimal seed v^* will take $O(n \cdot l^h)$. For a rather typical scenario of $l = 20$, $h = 5$, $n = 1,000,000$, we need $O(10^{12})$ operations to find v^*.

Because of the high complexity of step 2, we resort to an approximation again. We propose and evaluate four approximation algorithms for finding the vertex in the graph that covers maximal value within h hops. These algorithms are described below.

4.2.1 Maximal Out-degree First

The first algorithm, called *MaxOut* (Algorithm 2), always selects as a seed the page that has the largest out-degree. While this algorithm does not look at the values of the pages, it is based on the intuition that covering more pages in the

first hop will lead to covering more value in h hops.

Algorithm 2 Maximal Out-degree First

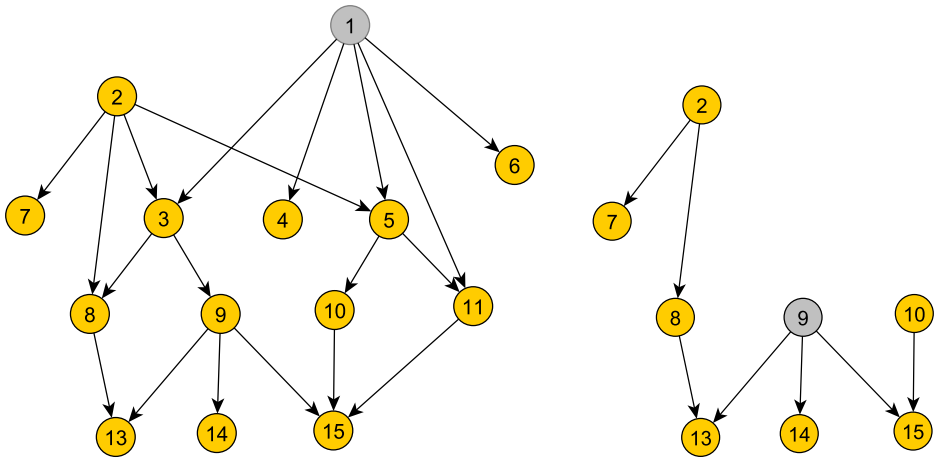
Input: Weighted Graph G of a website, Maximal Seed Number k, Number of hops allowed h
Output: Selected Seeds $\mathcal{S}$ of this website

Algorithm:

```
1: FOR i = 1 to k
2:     Get seed s which has the largest out-degree
3:     Make h hops on graph G, mark all covered vertices
4:     Remove s and all the marked vertices from G
5:     Add s to S
6:     IF G is empty THEN
7:         BREAK
8:     ENDIF
9: ENDFOR
```

This algorithm is different from the heuristic *OutDegree* algorithm, because after each step it updates the graph by removing the covered vertices. As a result, the out-degree of remaining vertices will decrease if they have out-links pointing to the removed vertices. Therefore, the algorithm will select seeds which cover substantially disjoint portions of the web graph.

Here we use the following example to illustrate the process of this algorithm and show the difference with the heuristic *OutDigree* algorithm. For this example, we assume the number of seeds, k, is 2 and the number of hops, h, is 1. In Figure 4.1a, the top two nodes (pages) with highest out-degree is node 1 (out-degree=5) and node 2 (out-degree=4). These two nodes will be selected as seeds if we use the heuristic *OutDegree* algorithm. On the contrary, if we use *MaxOut* algorithm, node 1, 3, 4, 5, 6, and 11 will all be removed after the first step because they are all covered within 1 hop from the first selected node 1. In step 2 (Figure 4.1b), node 2 is no longer the one with the highest out-degree because some nodes it links to have already been covered by previous selected seeds. Instead, node 9 is the best one for this step.



(a) Step 1 (b) Step 2

Figure 4.1: An Example of MaxOut Algorithm

4.2.2 Maximal Depth-d Weight First

This algorithm, called *MaxWeight* (Algorithm 3), selects as a seed the page that has the maximal depth-d weight, where depth-d weight is defined as follows.

Definition 2 (Depth-d weight)**.** *Given vertex v,*

$$\textit{depth-}d\textit{ weight of } v = w(v) + \sum_{u \in C} w(u)$$

where C is the set of vertices within d hops from v.

The *MaxWeight* algorithm tries to approximate the optimal seed, which has a maximal depth-h weight, with a seed that has a maximal depth-d weight, $d < h$. The complete algorithms is shown below.

In our experiments we used values 2 and 3 for d. We found that using values larger than 3 results in prohibitively long running times.

4.2.3 Maximal Weighted SCC First

In this algorithm, called *MaxSCC* (Algorithm 4), we first calculate the *Strongly Connected Components (SCC)* of the link graph. A strongly connected component is a subgraph consisting of all vertices such that there is a path from each vertex

Algorithm 3 Maximal Depth-d Weight First

Input: Weighted Graph G of a website, Maximal Seed Number k, Number of hops allowed h, Number of hops to examine d
Output: Selected Seeds $\mathcal{S}$ of this website

Algorithm:

```
1: Calculate depth-d weight for all vertices of G
2: FOR i = 1 to k
3:     Get seed s which has the largest depth-d weight
4:     Make h hops on graph G, mark all covered vertices
5:     Set the values of all marked vertices to zero
6:     Recalculate depth-d weight for all vertices of G
7:     Add s to S
8:     IF the weight of all vertices in G is zero THEN
9:         BREAK
10:    ENDIF
11: ENDFOR
```

in the subgraph to every other vertex [50]. The algorithm then selects as the seed a page that has the maximal depth-d weight in the maximal weight SCC.

This algorithm is based on the intuition that selecting the seed within the highest-weight SCC will lead to crawling all the pages within that SCC, if the diameter of the SCC is less or equal than h. If the diameter is greater than h, the algorithm tries to select a seed that covers the most high-weight portion of the SCC, based on the *MaxWeight* algorithm.

4.2.4 Root SCC First

Given a link graph G, we can form a higher level graph G' as follows. We collapse each strongly connected component in G into a vertex. Then, we add a directed edge $e(c_1, c_2)$, connecting the vertex corresponding to SCC c_1 to the vertex corresponding to SCC c_2 if and only if there exists at least one hyperlink from a page in c_1 to a page in c_2. It is not difficult to see that G' is a directed acyclic graph [1]. Given two components c_1, c_2 in G', we say that c_1 is a parent of c_2, and c_2 is

[1]This can be proved by contradiction. Suppose there exists a cycle $\mathcal{C}$ in G', which means that from any component $c \in \mathcal{C}$, there exist a directed path to every other component in $\mathcal{C}$. Since all pages in each components are also strongly connected, according to the definition of strongly connected components, all components in $\mathcal{C}$ should become one single strongly connected

Algorithm 4 Maximal Weighted SCC First

Input: Weighted Graph G of a website, Maximal Seed Number k, Number of hops allowed h, Number of hops to examine d
Output: Selected Seeds $\mathcal{S}$ of this website

Algorithm:

```
1: Calculate strongly connected components of G
2: FOR each SCC c
3:     Calculate total weight of all vertices in c
4: ENDFOR
5: FOR i = 1 to k
6:     Pick SCC c which has the largest total weight
7:     Calculate depth-d weight for all vertices of c
8:     Get seed s from c which has the largest depth-d weight
9:     Make h hops on graph G, mark all covered vertices
10:    Set the values of all marked vertices to zero
11:    Recalculate total weight for each SCC
12:    Add s to S
13:    IF the weight of all vertices in G is zero THEN
14:        BREAK
15:    ENDIF
16: ENDFOR
```

a child of c_1, if there is an edge $e(c_1, c_2) \in G'$. Components without parents are called root components.

Suppose there is no limit on the number of hops. Then any seed selected in a parent component will cover all the pages in its child components. This motivates the *RootSCC* algorithm (Algorithm 5), which selects as a seed a page that has the maximal depth-d weight in the maximal weight root SCC of the maximal weight SCC in G.

While the above algorithms can be extended by incorporating other properties of the graph as well as application-specific information, we believe that the four algorithms presented above are enough to illustrate the benefits of the graph-based seed selection framework. In the following section we present an experimental evaluation of the above algorithms, comparing their performance to the performance

component in the original graph G. Apparently, this is not true. Therefore, G' is a directed acyclic graph.

Algorithm 5 Root SCC First

Input: Weighted Graph G of a website, Maximal Seed Number k, Number of hops allowed h, Number of hops to examine d
Output: Selected Seeds $\mathcal{S}$ of this website

Algorithm:

```
1: Calculate strongly connected components of G
2: Build a directed acyclic graph G' by collapsing each SCC into a vertex
3: FOR each SCC c
4:     Calculate total weight of all vertices in c
5: ENDFOR
6: FOR i = 1 to k
7:     Pick SCC c_1 which has the largest total weight
8:     Get SCC c_2 which is the maximal weight root SCC for c_1 in G'
9:     Calculate depth-d weight for all vertices of c_2
10:    Get seed s from c_2 which has the largest depth-d weight
11:    Make h hops on graph G, mark all covered vertices
12:    Set the values of all marked vertices to zero
13:    Recalculate total weight for each SCC
14:    Add s to S
15:    IF the weight of all vertices in G is zero THEN
16:        BREAK
17:    ENDIF
18: ENDFOR
```

of heuristic approaches to seed selection.

4.3 Experimental Results

4.3.1 Experiment Setup

Our experimental methodology is as follows.

1. Select several web sites from a large-scale crawl of the web
2. Obtain all crawled pages from these web sites, and assign a value to every page
3. Run each of the seed selection algorithms on every web site, for different numbers of k and h, and measure the coverage of pages and the coverage of

value based on the seed set generated by each of the algorithms

4. Compare the algorithms in terms of the average improvement over the *Random* approach

Note that the seed computation is performed offline. Therefore the cost of running a seed selection algorithm does not impact the performance of the crawler. This cost mainly depends on the depth and complexity of the neighborhood region around a page which needs to be analyzed in order to perform step 2 of the algorithm 1, and varies from seconds for small sites to minutes for large sites.

The dataset used for the experiments is a sample of 2000 web sites from Malaysian web, each website containing at least 100 pages and having at least 1 external link into the site. We used an experimental web-scale crawl from the summer 2008 to obtain all pages crawled from these web sites. We also obtained other page attributes, such as whether a page was determined to be spam, whether the page was clicked on in search engine results, and how many uncrawled pages the page links to. We assigned a value $w(p)$ to each page p according to the following rules:

- set $w(p) = 1$
- If p is spam, set $w(p) = -10$
- If p is a crawl error, set $w(p) = 0$
- If p was clicked, set $w(p) = 10$
- Let n be the number of uncrawled pages the page links to. Set $w(p) = w(p)+n*P_{uniq}$, where P_{uniq} is the probability of uniqueness of a newly crawled page on the site. This probability is defined as the number successfully crawled unique documents from the site divided by the total number of crawl attempts, and is calculated based on past crawl statistics for the site.

We implemented the three heuristic approaches and the five graph-based algorithms described above. We used Tarjan's algorithm [51] to calculate the strongly connected components of a website graph. We use *Random* algorithm as a baseline, and report the average performance of the algorithms over the 2000 sites in

terms of the percentage of improvement over *Random*. We vary the number of seeds generated for each site from 1 to 10, keeping the number of hops fixed at 5, and vary the number of hops from 1 to 5 keeping the number of seeds fixed at 5.

4.3.2 Experiment Results

Figure 4.2 shows the results for the page coverage and Figure 4.3 shows the results for the value coverage for the fixed number of hops and varying number of seeds (error bars indicate standard errors). The following conclusions can be drawn from these graphs. First, *Random* performs rather poorly. All other strategies outperform it significantly. As one may expect, the improvement over *Random* is larger when the number of seeds is small, and it decreases as the number of seeds grows. One can also see that *PageRank*, while outperforming *Random*, performs much worse than the other heuristic approach, *OutDegree*, in terms of both the page coverage and the value coverage.

Second, all four of the graph based algorithms outperform the heuristic seed selection approaches. The *MaxOut* algorithm, which only uses outdegree of a page to select the best seed, performs best in terms of the coverage of pages. However, as expected it performs much worse than most of the graph-based strategies in terms the coverage of value.

Third, the algorithms using simple depth-d approximation perform better than the algorithms that try to utilize the knowledge of strongly connected components and their connectivity. In fact, the *RootSCC* algorithm performs very poorly in terms of the value coverage, worse than the heuristic *OutDegree* algorithm. The reason for this might be that, as we showed earlier, most of web sites tend to consist of many small-size SCCs, so selecting the seeds based on the direct estimation of the depth-d value is better than selecting seeds based on the the value of the SCC the seed belongs to.

Finally, the two *MaxWeight* algorithms using $d = 2$ and $d = 3$ perform similar, suggesting that using 2 hop approximation is enough to identify a good seed.

Figures 4.4 and 4.5 show the results for the page coverage and the value coverage for the fixed number of seeds and varying number of hops. As once can see, improvement of the algorithms over *Random* increases slightly with the number of

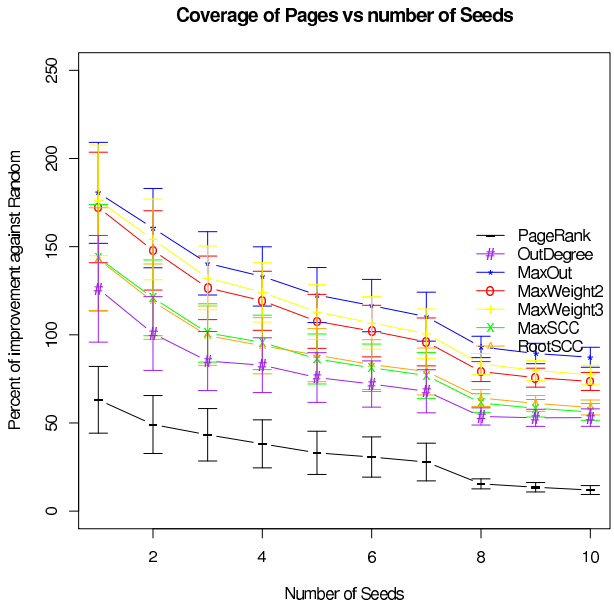


Figure 4.2: Coverage of Pages

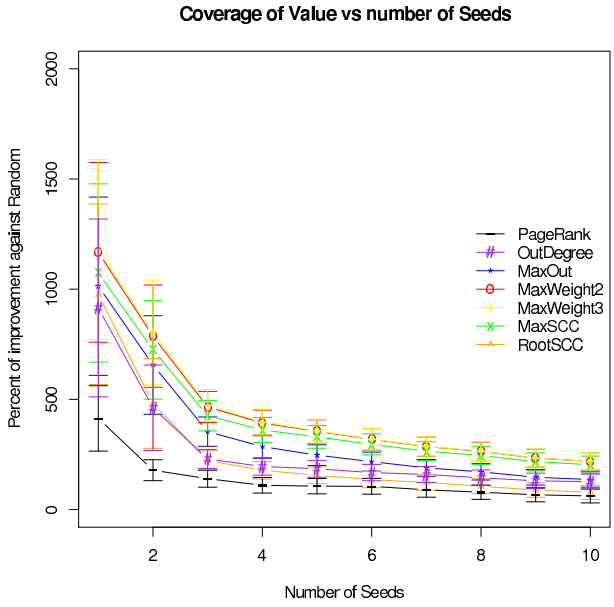


Figure 4.3: Coverage of Value

hops increasing. The increase is higher for the page coverage, and smaller for the value coverage. This is expected because due to the nature of our algorithms most high-value pages should be covered within a few hops. The relative order of the algorithms is the similar to the one on figures 4.2 and 4.3.

Overall, the results show that the graph-based strategies significantly outper-

form the heuristic approaches, demonstrating the advantage of the graph-based seed selection framework. While we only evaluated the algorithms for one specific setting of page values applicable in a web-scale crawler setting, we believe that the results will generalize to other settings, such as the ones arising in a news or blog search engine.

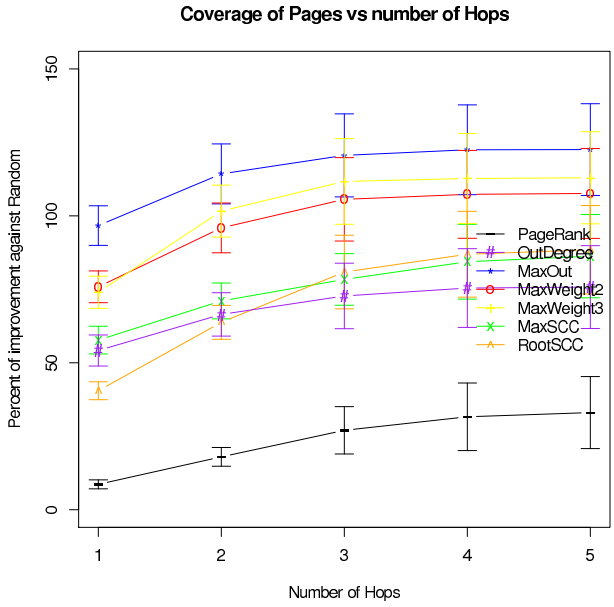


Figure 4.4: Coverage of Pages

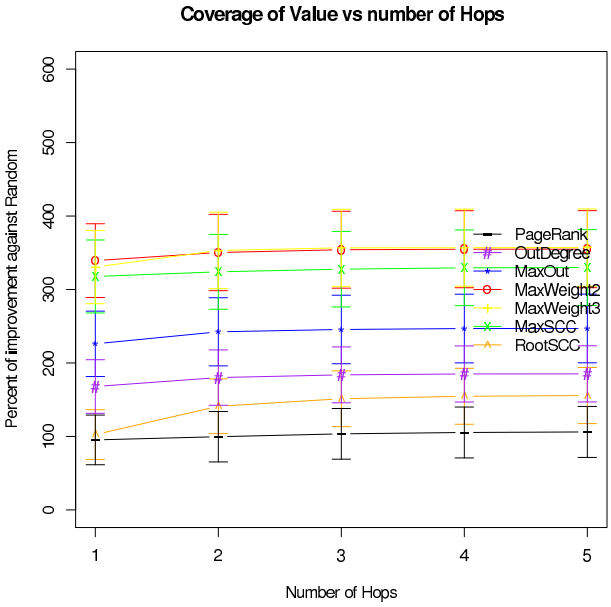


Figure 4.5: Coverage of Value

Chapter 5

Focused Crawling Guided by Positive Signals

In this Chapter, we propose a URL ordering policy for focused crawling in the context of CiteSeerX crawling system.

5.1 Background

The purpose of focused crawling is to harvest relevant documents. Two problems need to be solved for such a task.

1. How to determine whether a document is relevant or not?
2. How to find an effective policy to prioritize URLs in the crawling queue so more relevant documents can be discovered in a given period of time.

Problem (1) can be solved by either using heuristic rules or training a document classifier with state-of-the-art machine learning methods. Problem (2) is a URL ordering problem. In this work, we focus on problem 2.

URL ordering policy is critical for focused crawling tasks. The computing resource affordable to a crawling task is usually limited; therefore, the crawler cannot traverse all pages of a website to discover all relevant documents. By ordering URLs properly, the crawler can obtain important pages first and avoid wasting time on visiting irrelevant pages.

5.1.1 Problem Definition

The goal of URL ordering policy is to maximize the coverage of relevant documents achieved over time t, given a fixed crawl rate [52]. Here, $t << \infty$. When t is infinite, URL ordering no longer matters since all relevant document will eventually be discovered. When time t is limited, different ordering policies will yield different subset of relevant documents.

In CiteSeerX system, we consider academic papers in PDF or PS format as relevant documents. Such documents are usually obtained from academic websites.

5.2 A Supervised Method Using Positive Signals

In this work, we propose a supervised method to prioritize URLs using positive signals. Here, signals can be various features related to each URL. We say a signal is positive, if it can lead to a relevant document with non-zero probability.

The general idea of our method is to train a set of positive signals and use them to guide the crawler. Obviously, this method is domain specific. Different applications should train different sets of signals.

5.2.1 Signal of a URL

To order the URLs, additional information should be added to each URL to represent its signals. In our implementation, we use anchor text, the text used to represent a link. There are two advantages of using anchor text for such purpose.

1. Unlike feature like PageRank, which requires complex computation, anchor text is very easy to extract. This ensures scalability of our method.

2. Anchor text reflects meaningful connections among web pages. For a particular domain, the vocabulary used for the anchor texts are relatively stable.

To reduce noise and improve precision, the raw anchor text of a given URL need to be preprocessed. We do not use its entire anchor text as one signal. Instead, we first remove stop words and meaningless symbols from it, then we use a stemming tool [53] to normalize the rest words. The output words are treated as multiple signals of the input URL.

Although we choose anchor text as signal in our implementation, the method proposed in this work is not limited to this choice. It can be easily extended by replacing anchor text with any other feature related to a URL.

From this point, when we say "signals" of a URL, we actually mean the normalized words extracted from its anchor text.

5.2.2 Computing URL Priority

This work is based on the observation that a relevant document is usually surrounded by some positive signals in the web graph. For a given URL, we need a function to map its surrounding signals to a value which can indicate its priority. We take into account the following factors when designing this function.

- Different signals should be treated differently. For example, the word "research" is usually more likely to lead to an academic paper than "news".
- A same signal in different neighborhood may contribute different likelihood of leading to a relevant document. Compare the signal "download" in two scenarios. One is connected by a page with signal "publication"; the other is connected by a page with signal "software". We would believe that it is more likely to get an academic paper by following the first "download" link. Therefore, when calculating the priority of a URL, we consider not only the signals of the URL itself, but also the signals of its neighbors in the web graph.

Before we can calculate the priority values for URLs, we need to generate a positive signal set S^+ with an offline training process (See section 5.3 for more details). Each signal in S^+ has been assigned with a weight.

Given URL u, its priority value can be calculated by the following function:

$$\mathcal{P}(u) = \sum_{i=1}^{n} w(s_i) + \alpha \cdot \sum_{j=1}^{m} w(t_j) \tag{5.1}$$

where, $\{s_1, s_2, \ldots, s_n\}$ are the signals of input URL u, $\{t_1, t_2, \ldots, t_m\}$ are the signals of u's in-link neighbors within k hops, k is a pre-defined parameter. The weighting function $w(\cdot)$ returns the weight of a signal if it is in S^+; otherwise, it returns 0.

The smoothing factor α is another pre-defined parameter that controls the ratio of the contribution from u's neighbors.

Ideally, we should include all depth-k in-link neighbors in above calculation. However, since the crawling paths only form a tree structure on the web graph, the crawler usually does not "see" the full picture of a URL's in-link structure. To be more precise, it only knows at most k in-link neighbors within k hops. They are actually the ancestor nodes on the crawling tree. This can be illustrated by a simplified example crawl shown in Figure 5.1.

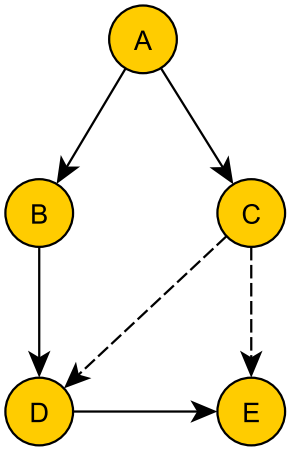


Figure 5.1: Simplified Example Crawl

The crawler starts at URL A, and follows the paths in solid lines. Let's consider the depth-2 in-link neighbors of URL D. It should be A, B, C if we see the full graph. But the crawler doesn't follow the path from C to D; hence, only A, B are considered as depth-2 in-link neighbors of D.

By applying function (5.1), URLs discovered by the crawler will be ordered by their priorities and high priority URLs will be fetched first. In our implementation, a heap based data structure is used as a priority queue to support URL ordering.

5.3 Training Positive Signal Set

The purpose of the training is to find S^+, a set of weighted positive signals.

We need a set of training sites from the same domain. For our system, we choose websites of well-known university and research institutions. For each selected site,

three steps should be done for the training.

1. Build a web graph
2. Find relevant documents on the graph
3. Credit signals surrounding relevant documents.

These three steps are detailed in the following subsections.

5.3.1 Building Web Graph

To build a web graph for a site, we conduct a deep crawl by making up to 10 hops [1] from its root page (homepage of a site). We log all the crawling paths followed by the crawler. We also save metadata (anchor text, out-links, content type, etc.) of all the pages visited by the crawler. Using logged crawling history information, a directed graph can be built by taking URLs as vertices and links as directed edges. We also setup back-links (reversed links) so that each URL can easily locate its in-link neighbors on the graph.

5.3.2 Document Classification

The classification is to label relevant nodes on the graph for each site. In our system, we only consider academic papers in PDF or PS format as relevant documents. Therefore, if a document's content type is HTML, we always label them as irrelevant. For PDF documents, we use a rule based classifier to further separate real papers from others. Here is a list of rules we used in the classifier. They are applied in order to determine whether a document is an academic paper.

1. Consider negative if the URL contains a negative keyword, e.g, CV, resume, homework, quiz, etc. This filters out majority of negative samples in our data.
2. Consider negative if the document has no reference section. We check this rule by matching the "references" keyword and its variations in the second half of the document.

[1] We believe most of relevant content of a site should be reachable in 10 hops from its root.

3. Consider positive if it contains both "abstract" in the first 1/3 of the document and "conclusion" in the last 1/3 of the document. Otherwise, continue to check other rules.

4. Consider negative if the the document has more than 20 pages or less than 2 pages. Otherwise, label as positive.

5.3.3 Depth-d Neighborhood Signal Crediting

Give graphs of all training sites and their positive node sets, we can Algorithm 6 to generate S^+.

The main idea of Algorithm 6 is to credit signals that lead to a relevant document in d hops. d is the only parameter we need to tune for this algorithm. If d is too large, we are basically crediting everything. If d is too small, we will miss some signals with predictive power even they do not immediately connect to a relevant document. In our system, we found that depth-four positive signal set can achieve best performance.

5.4 Experiments

5.4.1 Dataset and Setup

To evaluate the performance of our URL ordering method, we selected 18 academic websites (See Table 5.2. We conduct deep crawl on all the sites and fetched 2,410,701 documents including HTML pages and PDF documents. Among total 64,994 PDF documents, 43,918 PDF documents are labeled as relevant and 21,076 are labeled as irrelevant.

In our experiments, even though we need to try different crawling policies on a website, we do not actually crawl the same site again and again. Instead, we build graph for the site using the logged data of the first crawl. Then, the latter crawls can be virtually conducted on the graph. This saves us a lot of time and avoids heavy load on the target websites. The crawler is implemented in Python and all the experiments were run on a Linux server with 32 GB memory.

Algorithm 6 Depth-d Neighborhood Signal Crediting

Input: Graph set G of all training sites, Neighborhood depth d
Output: Positive Signal Set S^+
Algorithm:

```
Initialize S+ as an empty set
FOR g in G
    CreditOneSite(g)
ENDFOR
```

Function CreditOneSite(g)

```
N+ = relevant node list of g
FOR u in N+
    CreditOneNode(u, 0)
ENDFOR
```

Function CreditOneNode(u, $hops$)

```
IF hops ≤ d THEN
    S(u) = signals of u
    w = 1/|S(u)|
    FOR s in S(u)
        IF s in S+ THEN
            increment its weight by w
        ELSE
            add s to S+ and set its weight to w
        ENDIF
    ENDFOR
    I = in-link neightbors of u in G
    FOR v in I
        CreditOneNode(v, hops + 1)
    ENDFOR
ENDIF
```

5.4.2 Evaluation Metrics

The goal of our crawling policies is to get more relevant documents with less time. In our experiments, we evaluate different crawling policies by comparing the time used to harvest 80% relevant documents of a given website. Since the crawl rate can be considered constant, we can actually compare the number of total URLs a crawler need to visit to get 80

5.4.3 Experimental Results

Top 20 Positive Signals

Here are the top 20 positive signals we extracted from our training set. Unsurprisingly, the word "publication" ranked first. Most of the top words do indicate some sort of relation with a academic paper.

Table 5.1: Top 20 Positive Signals

(13854, 'publication')
(4772, 'lecture')
(2618, 'schedule')
(2568, 'papers')
(1824, 'note')
(1642, 'research')
(1545, 'list')
(1545, 'cmsc')
(1472, 'computer')
(1367, 'system')
(1288, 'algorithm')
(1195, 'cs')
(1181, 'content')
(1173, 'page')
(1092, 'learning')
(1005, 'index')
(982, 'bibtex')
(970, 'abstract')
(959, 'author')
(942, 'theory')

Comparison with Breadth-First Policy

To show the effectiveness of our signal-guided crawling policy, we did a comparison with the widely used breadth-first policy. We compare the percentage of URLs the crawler need to visit in order to get 80% relevant document. The results are listed in Table 5.2. Our method outperforms breadth-first policy by 23.9% on average.

Table 5.2: Breadth-First vs. Signal-Guided: Percentage of visited URLs to get 80% relevant document

Website	**Total URL**	**Breadth-First**	**Signal-Guided**
mizar.org	3906	73.9	64.1
research.microsoft.com	39506	82.3	54.2
www.ai.mit.edu	719	69.7	70.5
www.cc.gatech.edu	5731	88.2	34.9
www.cis.upenn.edu	7269	87.6	51.2
www.cs.berkeley.edu	25434	46.7	38.6
www.cs.cmu.edu	94175	92.1	46.1
www.cs.columbia.edu	125241	72.6	52.5
www.cs.cornell.edu	12855	68.2	21.6
www.cs.huji.ac.il	7357	82.6	87.6
www.cs.umd.edu	53964	87.2	43.6
www.cs.utexas.edu	1334254	71.7	41.2
www.cs.washington.edu	71599	75	28.5
www.cs.wisc.edu	23429	86.2	69.1
www.hpl.hp.com	6661	79.6	36.6
www.ics.uci.edu	554182	58.9	49.5
www.research.att.com	7722	80.4	89.7
www.ri.cmu.edu	36639	69.5	62.7
Average		76.24	52.34

Chapter 6

Template-Independent News Extraction Based on Visual Consistency

Since the birth of the Internet, Web information has continued to proliferate at an exponential pace due to ease of publishing and access. News is among the most popular interests for Web surfers. Some Web services, such as Google News and Yahoo! News, extract news from hundreds of sources. Their aim is to provide a better way for searching the latest news stories.

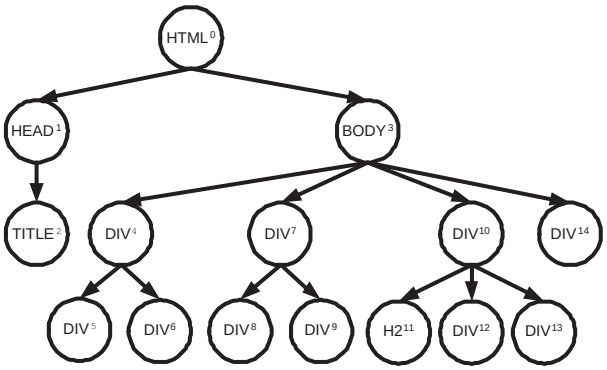
In this work, we propose an template-independent approach to extract news from webpages.

6.1 Our Approach

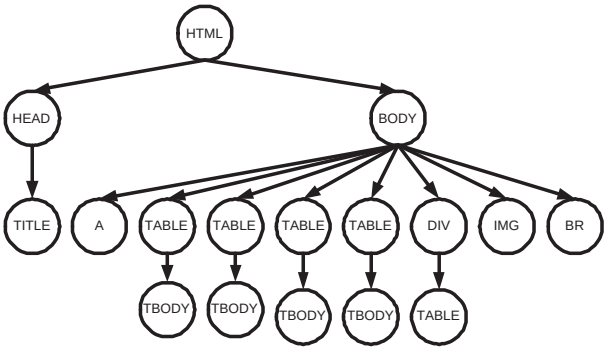
We target deriving domain level visual consistency by using visual features. We argue that visual consistency can lead to high extraction accuracy even though template consistency may be missing.

6.1.1 Visual Consistency

When people browse a Web page, they are subconsciously guided by the experience they have accumulated in browsing other similar pages in similar domains. In the



(a) for page in Figure 1.4a



(b) for page in Figure 1.4b

Figure 6.1: DOM trees for pages in Figure 1.4

case of news, people commonly seek a block with the following visual features: (1) Its area is relatively larger than other page objects around it. (2) At the top of the block, there is usually a bold-faced line which is the news headline. (3) It is mostly occupied by contiguous text paragraphs, sometimes mixed with one or two illustration pictures. (4) Its center is close to that of the whole page. ... They would never go for a slim and long block because they assume that page object in that shape is unlikely to contain the core information.

This is understandable because when a Website developer designs Web pages in a specific domain, usually some hidden conventions of a domain that have already been accepted by most people are followed. For instance, designers generally do not put main content at the corner of a page, occupying a small portion of the page area. We notice that such design conventions are irrelevant to the content text or HTML tags. It is a kind of visual level convention that leads to the visual consistency among all pages in the same domain.

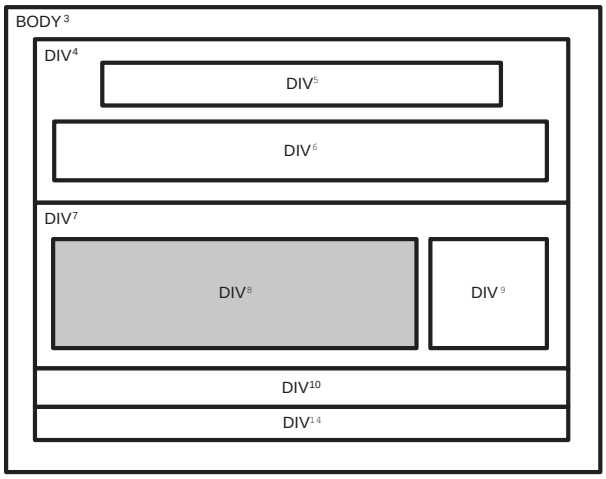


Figure 6.2: Visual blocks of page in Figure 1.4a

6.1.2 Data Representation

In our solution, all DOM nodes are treated as rectangular blocks. HTML tags are not used to distinguish different blocks. Instead, we use a group of visual features to represent them and determine the similarity among them.

Definition 3. *(Visual Block) A visual block is a visible rectangular region on Web page with fixed position and nonzero size. It is rendered from a pair of HTML tags, e.g.,* `<DIV>` *and* `</DIV>`*, or text between them.*

For both coding and description convenience, we assign a unique identity for each block within a Web page, called *Block ID*.

Every Web page can be converted to a visual tree. Again, we take news page in Figure 1.4a for instance. Its block structure is displayed in Figure 6.2 (Blocks at deeper levels are omitted). The superscript numbers (Block ID) of both figures indicates the relations between HTML tags and blocks rendered by them. As we can see, the whole page can be considered as the biggest block (called page level block) rendered by `<BODY>` and `</BODY>`. Note that although each visual block is rendered by a pair of HTML tags, such tag information is not used as a feature for leaning our wrapper.

Given an HTML source, an HTML rendering algorithm or an HTML parsing tool can be used to obtain such a visual tree. In our implementation, we choose the

Microsoft MSHTML library [1] to render and parse HTML source. The MSHTML library provides interfaces to access visual information for any DOM nodes. Based upon these visual cues, we derive needed visual features for our news domain application. The basic visual features are listed below in 4 categories.

1. **Position features**: Left, Top (the coordinate of left-top corner of a block), NestedDepth;
2. **Size features**: Width, Height;
3. **Rich format features**: FontSize, IsBoldFont, IsItalicFont;
4. **Statistical features**: ImageNumber, HyperlinkNumber, TextLength, ParagraphNumber, ItalicParagraphNumber, BoldPargraphNumber, TableNumber.

Consequently, the tag diversity of DOM trees does not affect our algorithm, even at the domain level. For example, the title in Figure 1.4a is wrapped by `<SPAN>` and that in Figure 1.4b is wrapped by `<FONT>`. For most T-Wrapper induction methods, the corresponding DOM nodes of the two titles cannot match each other. In our approach, they are considered visually similar because they are both bold-faced with a high ratio of width and height.

6.1.3 Extended Visual Features

Since the topological structures of DOM trees are quite different at domain level, they are not used for matching purposes. Instead, we use "parent-child" visual relations between two blocks.

Definition 4. *(Parent Block and Child Block) Given two visual blocks b_1, b_2, we say b_1 is b_2's parent block (or b_2 is b_1's child block), iff b_1 covers b_2 and these exist no block b_3 in the same page, where b_1 covers b_3 and b_3 covers b_2.*

Based on this definition, we can define a set of extended visual features to represent relations between a child block and its parent block. E.g.,

[1] More details about MSHTML library can be found at: http://msdn.microsoft.com/workshop/browser/editing/mshtmleditor.asp

$$RelativeWidthOfParent = Width/ParentWidth$$

$$RelativeLeftOfParent = Left - ParentLeft$$

The primary difference between "parent-child" visual relation and topological structure of DOM tree is that it has no constraints on path depth. More specifically, for each DOM node, its position in DOM tree is determined by the tag path from root node to itself. This tag path plays an important role in most traditional DOM tree-based methods. In our approach, the "parent-child" visual relations can be used to match any two blocks as to paternity in spite of their nested depth. In short, topological diversity of DOM trees can also be handled effectively in our approach.

6.1.4 Learning a Vision Based Wrapper

Based on the visual consistency, we target the creations of a robust vision-based wrapper for an entire domain. For statement convenience, we denote our vision-based generated wrappers as *V-Wrapper*, whereas traditional DOM tree-based wrappers as *T-Wrapper*.
Actually, when people browse a Web page, they subconsciously combine various visual features with different priorities and weights in seeking target information. If we can simulate this human behavior, we can likely derive a composite visual feature which is stable enough to form a domain-level visual consistency. In our system, we use Adaboost [54] to learn the composite visual feature. We chose Adaboost because it is an effective algorithm in weighting different features.

Like traditional wrapper induction methods, our approach also requires a set of manually labeled pages for the purpose of training. In our work for news domain, we define two label sets for inner block and leaf block.

Definition 5. *(Inner Block and Leaf Block) A block is an inner block iff it has at least one child block; otherwise, a leaf block.*

For a leaf block, annotators give labels of news *Title* or *Content* to relevant blocks while other leaf block is inferred as the label of *Others*.

$$\mathcal{L} = \{Title, Content, Others\}$$

For an inner block, labels are derived.

Theorem 1. *A inner block is treated as a positive block iff at lease one of its child block is labeled rather than Others.*

If we consider positive inner blocks (PI), negative inner blocks (NI) as two derived labels.

$$\mathcal{L}^{'} = \{PI, NI\}$$

Actually, a label is not only used in training pages, but also in testing pages to present the extraction result. We name the former pre-label and the latter post-label for differentiation purpose. In our experiments, by comparing the post-label with corresponding pre-label, we can easily evaluate extraction accuracy in terms of *precision*, *recall* and *F1-Value*.

To learn a V-Wrapper, we need to simulate human behavior involved in browsing news pages. Actually, we can think of this behavior as a two-step process. The first step is to roughly locate the main block based on all kinds of visual features presented in the page and excluding all irrelevant content like advertisements. The second step is to look into the main block more carefully and identify which is the headline, the news body, and so on. These two steps normally require different visual features.

Similiarly, we developped a two-step learning process for V-Wrappers.

In step 1, we select all inner blocks B_1 with their corresponding derived labels, and train a classifier as to whether an inner block contains some pieces of news.

In step 2, we choose all leaf blocks whose parent blocks are predicted as PI by the classifier in step 1. These leaf blocks compose B_2. Then, similarly, we train a classifier to predict which label a leaf block matches.

6.1.5 News Extraction using a V-Wrapper

The extraction algorithm using generated V-Wrapper for news domain also have two steps:

1. The first step seeks to extract leaves blocks whose parents are positive inner blocks as candidates for target information.
   The algorithm (See Figure 7.1) is a recursive Top-Down process. It starts from the biggest block (page block) and stops at leaf blocks or negative inner blocks. We do not continue to deal with child blocks of negative inner block b because all blocks of the subtree rooted at b are considered as negative blocks, due to Theorem 1.

2. The second step is to label different types of information from candidates blocks obtained in the first step.
   In this step, the leaf block classifier is used to match each candidate block with labels. The output of this step is a post label $\mathcal{L}(p)$ for test page p. It indicates the target information which must be extracted from p.

```
Algorithm: Identify_Candidate_Blocks(p)
1.  begin
2.     Parse p to visual block set B;
3.     pageBlock := page level block in B;
4.     plBlocks := empty block set;
5.     Extract(pageBlock, plBlocks) ;
6.     return plBlocks;
7.  end

Algorithm: Extract(b, plBlocks)
1.  begin
2.     if b is a inner block then
3.        if b is labeled as PI then
4.           childBlock := the first child block of b;
5.           while childBlock is not NULL
6.              Extract(childBlock, plBlocks);
7.              childBlock := childBlock's next sibling;
8.           endwhile;
9.        endif;
10.    else
12.       Add b to plBlocks;
14.    endif;
15. end
```

Figure 6.3: Algorithm for identifying candidate blocks

6.2 Experiments

The experiments in this paper are designed to demonstrate the performance of our V-Wrapper induction system, as well as to show the advantages of V-Wrapper in contrast with traditional T-Wrapper.

To compare it with traditional methods, we have also built a T-Wrapper generation system using state-of-the-art techniques (Tree Edit Distance, Regular Expression Inference and Sequence Alignment). Our T-Wrapper generation system adopts the main ideas from Reis et al.'s work [15] and Chuang et al.'s work [16]. Their methods are close to ours and proved to be effective on template level T-Wrapper induction.

6.2.1 Experiment Setup

We collected 295 pages from 16 online news sites to use in our experiments. For comparison's sake, we only collected pages sharing one common template for each site. i.e., in our experiments, site level equals template level. Pages of each site are randomly divided into three groups to perform cross validation for all experiments. More specifically, we conducted each experiment three times by taking different page groups as test sets for each site. Then we averaged three results as the final result.

All pages are manually labeled for extraction result evaluation. Suppose the extraction result of a test page p is rendered in the format of post label $\mathcal{L}_{post}(p)$ and its corresponding manual label is $\mathcal{L}_{pre}(p)$, we can compute three measures to evaluate the extraction accuracy of certain extraction type τ: *precision*, *recall* and *F1-Value*

$$precision = \frac{|\mathcal{L}_{pre}(p) \cap \mathcal{L}_{post}(p)|}{|\mathcal{L}_{post}(p)|} \tag{6.1}$$

$$recall = \frac{|\mathcal{L}_{pre}(p) \cap \mathcal{L}_{post}(p)|}{|\mathcal{L}_{pre}(p)|} \tag{6.2}$$

$$F1\text{-}Value = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{6.3}$$

where $|\mathcal{L}|$ is the number of items labeled as τ in $\mathcal{L}$.

We implemented two groups of experiments on a PC with 3 GHz Pentium 4

processor and 1015MB RAM.

Experiment One: Domain Level Compatibility

The objective of the first experimental group is to demonstrate what we call domain-level compatibility of V-Wrapper. It is carried out as follows:

First, we take one site training set as the initial training set TrS_1, and a V-Wrapper W_1 can be learned from TrS_1. Then we add another site training set to TrS_1 and get a new training set TrS_2 which generates W_2. Repeat this step until all the training sets of the 16 sites are used to generate W_{16}. Thus, we can get 16 different V-Wrappers. We use these 16 V-Wrappers to extract news text from same test set TeS, which is the combination of 16 site test sets.

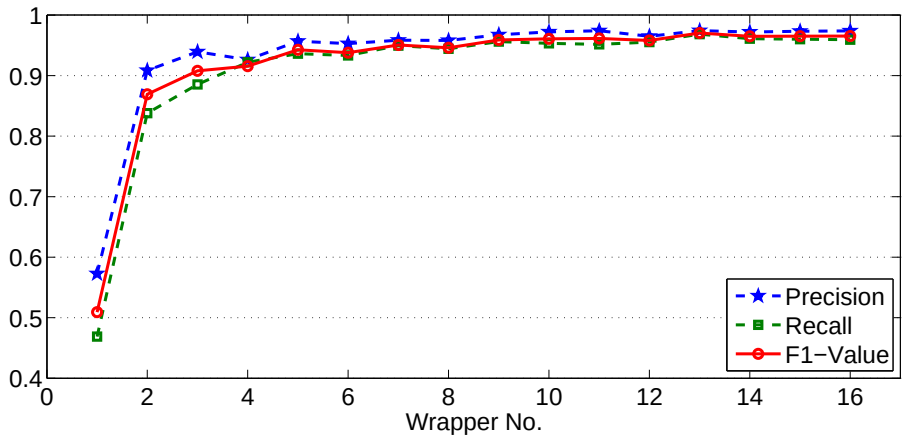


Figure 6.4: Result of experiment one

Figure 10.6 shows changes of extraction accuracy as the training sets increase. Initially, the first V-Wrapper learned from one site's training set can only achieve 50.93% accuracy in terms of F1-Value. After utilizing three sites training sets, the F1-Value stays above 90% and becomes quite stable. This adequately proves the domain level compatibility of V-Wrapper generated by our approach. We can imagine that given another news Website, our V-Wrapper can still achieve optimal extraction accuracy without re-training.

Experiment Two: V-Wrapper vs. T-Wrapper

The second group of experiments are designed to compare V-Wrapper with T-Wrapper site by site. Since V-Wrapper is domain level wrapper, we only generate one V-Wrapper from all site training sets and apply it to all test sets. This wrapper is same as W_{16} used in Experiment One. On the contrary, T-Wrapper is site level wrapper, every site generates a T-Wrapper and use it to test corresponding test set.

Extraction results of V-Wrapper are listed in Table 6.1. The average F1-Value is 94.96%.

Table 6.1: Result of V-Wrapper on 16 news Websites

Site No - Site Name	**Precision**	**Recall**	**F1-Value**
01-NYTimes.com	0.9868	0.9730	0.9793
02-Guardian.co.uk	0.8895	0.9714	0.9229
03-RealCities.com	0.9780	0.9357	0.9544
04-ABCNews.com	0.9805	0.8850	0.9287
05-UPI.com	0.9917	0.9223	0.9547
06-Newsday.com	0.9667	0.9118	0.9381
07-USAToday.com	0.9327	0.8990	0.9125
08-GlobeAndMail.com	0.9777	0.9345	0.9553
09-News24.com	1.0000	0.8173	0.8841
10-Reuters.com	0.9965	0.9765	0.9860
11-TheAge.com.au	0.9155	0.9140	0.9116
12-VOANews.com	0.9629	0.9921	0.9763
13-CNN.com	0.9744	0.9071	0.9378
14-FT.Com	1.0000	0.9306	0.9597
15-News.Yahoo.com	0.9816	0.9578	0.9692
16-BBC.co.uk	0.9739	0.9630	0.9669
Average	0.9693	0.9307	0.9496

Figure 9.9 shows the comparison results of V-Wrapper and T-Wrapper. As displayed in Figure 9.9, although T-Wrapper outperforms V-Wrapper on several sites, even achieve 100% accuracy on NYTimes.com, its average extraction accuracy is not as good as V-Wrapper (F1-Value: 88.32%). This can be explained by two reasons.

First, T-Wrapper's performance highly depends on DOM structure consistency of each site, mainly in DOM tree structure. If pages in a site are very similar

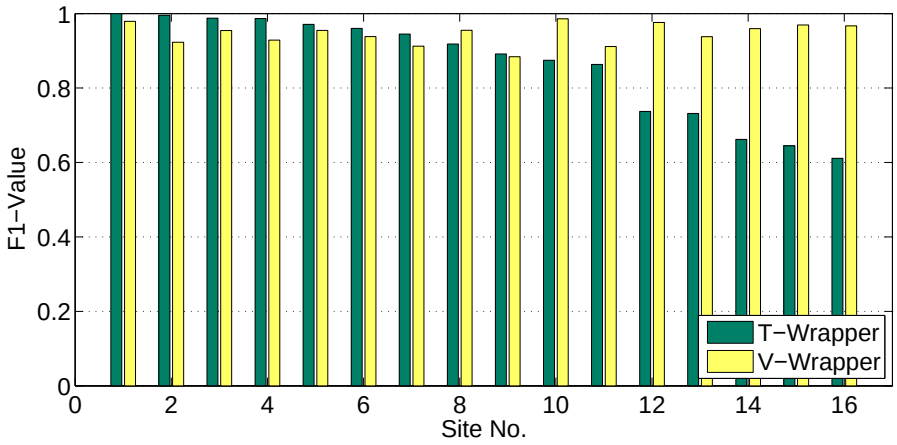


Figure 6.5: T-Wrapper vs.V-Wrapper: Site By Site

(e.g., NYTimes.com), T-Wrapper can achieve almost perfect extraction results. However, if pages in a site are diverse in structure (e.g., News.Yahoo.com), T-Wrapper's extraction accuracy drops quickly.

Second, as stated early, it is difficult to decide the size of training sets for T-Wrapper induction; thus, if the randomly selected training set is not large enough, the T-Wrapper probably cannot cover all pages in the test set. To prove this, we have performed an experiment for case study purpose on FT.com which has the lowest extraction accuracy in all test sets. We use pages in test set to refine the generated T-Wrapper and implement extracting process again. We find that the extraction accuracy in terms of F1-Value obtains an improvement from 61.11% to 94.25%).

In contrary to T-Wrapper, V-Wrapper's performance is pretty stable in the whole domain. Most errors of V-Wrapper are caused by noise information (e.g., copyright information), which is visually similar to news text.

Chapter 7

Extracting Author Metadata from Web using Visual Features

7.1 Data Representation and Problem Definition

7.1.1 Visual Block and Visual Tree

Similar as Chapter 6, we also parse a webpage into a visual tree which consists of many visual blocks. Given a visual block b, we use $Width(b)$, $Height(b)$, $Left(b)$ and $Top(b)$ to indicate its size and position respectively where $(Left(b), Top(b))$ is the coordinate of left-top corner of b.

For evaluation convenience, we assign a unique identity for each block withn a Web page, call *Block ID*.

7.1.2 Problem Definition

The input of our problem is a HTML page p and a predefined author ontology O. p can be parsed into a visual tree consisting of a set of visual blocks, namely B. Note that each block b is a node on the visual tree. Ontology O defines a set of fields, in this work,

$$O = \left\{ \begin{array}{c} \textit{name, title, affiliation, address} \\ \textit{picture, email, telephone, fax} \end{array} \right\}$$

Given B and O, the problem of extracting author meta-data is actually a multi-classification problem. We need to design a function $Predict : B \rightarrow O$, such that for each leaf block $b \in B$, $Predict(b) = l$, where $l \in \{None\} \cup O$.

If $l = None$, it means the input block belongs to no class.

Since it is a classification problem, some state-of-the-art methods can be used to train the classifier. In our implementation, we use adaboost because it is an effective algorithm in automatic feature selection.

7.2 Extraction Method

In this work, we take a tradition machine learning approach to address the problem. Our method consists of two phases: training and extraction. First, Adaboost [54] is used to train a binary classifier for each field. Then, all these binary classifiers are combined to predict the label of a given block.

7.2.1 Visual features

One characteristic of our method is that we take advantage of visual information for extraction. By parsing HTML source, we can derive various visual features for a webpage. In our implementation, we choose the Microsoft MSHTML library to render and parse HTML source.

1. **Position fetures**: *Left*, *Top*, *CenterX*, *CenterY*, *RatioOfCenterXCenterY*[1], *RelativeToPageLeft*[2], *RelativeToPageTop*, *RelativeToPageCenterX*, *RelativeToPageCenterY*, *RatioOfCenterXPageWidth*[3], *RatioOfCenterYPageHeight*, *Nested Depth*

2. **Size features**: *Width*, *Height*, *Area*, *RelativeToPageArea*, *RelativeToPageWidth*, *RelativeToPageHeight*,

3. **Shape features**: RatioOfWidthHeight

[1] $CenterX/CenterY$
[2] $this.Left - page.Left$
[3] $this.CenterX/page.Width$

4. **Rich format features**: *FontSize*, *RelativeToPageFontSize*, *FontStyle*[4], *FontWeight*, *RelativeToPageFontWeight*, *TextAlign*, *Visibility*, *IsHyperLink*

There are three things worth noting in above feature definition. First, some features are calculated based on other basic features, e.g., *area*. Second, some features are relative features whose values are calculated based on the different between a block iteself and the root block (page). Third, some features will be converted to a numerical number before used for training, e.g., normal *FontStyle* is converted to 0 and italic *FontStyle* is converted to 1.

7.2.2 Content based features

Besides visual information, 70 context-based features are used to capture those subtle characteristics of different fields. Defining features based on context text is not new and has already been widely used in all kinds of content-based classification problems.

7.2.3 Algorithm of the prediction function

Here we present the algorithm used to predict a label l of a input block b.

Algorithm: Predict(b)
1. **begin**
2. Load 8 binary classifiers $C = \{c_i\}, (i \in O)$
3. $l := None$
4. $maxScore := 0.0$
5. **foreach** c_i in C
6. $score := \mathcal{S}(c_i, b)$
7. **if** $score > maxScore$ **then**
8. $maxScore := score$
9. $l := i$
10. **endif**
11. **endforeach**
12. **return** l
13. **end**

Figure 7.1: Algorithm for predicting label of a input block

[4]normal, italic, oblique

Note that, in line 6, function $\mathcal{S}$ returns a float value indicating the confidence of predicting block b as label i. This value is obtained by normalizing the original prediction score of binary classifier c_i to range (-1, 1). The final label is associated with the binary classifier whose score is a maximal positive number. If all prediction scores are negtive, then the input block will be labeled as *None*.

7.3 Apply Inter-Fields Visual Correlation

Due to the miscellaneity of homepage domain, method proposed in previous section has several issues. 1) It is hard to distinguish fields whose feature vectors are very close. For example, address and affiliation sometimes "look" similar. 2) It tends to produce fault-positive label for blocks which are not describing the author itself, e.g., an collaborator's email address.

Another contribution of our work is that we propose to address above issues by considering the visual correlation between different fields. The idea is natural. By observation, we find that most people do not put those fields randomly on their homepage. They tend to arrange them in a way which is widely used by other professionals. For instance, a large number of people will put Name above Title and place them together beside a picture, as illstrated in Figure 7.2.

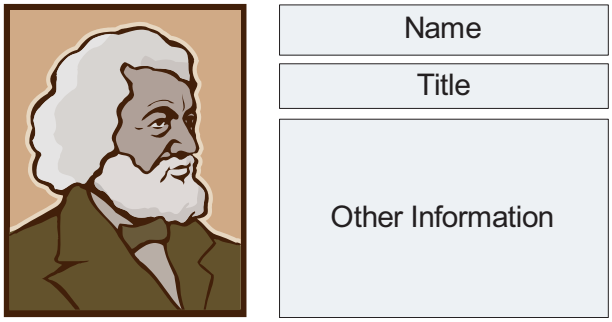


Figure 7.2: A typical homepage layout

Based on such observation, we can define a probability model to let the inter-fields visual relation play a role in determining the label of a block. Here, the basic idea is that when a block b_1 is labeled as a certain field l_i, another block b_2 who has a certain visual relation with b_1 have a conditional probability of being another field l_j.

In this section, we first formally define 12 types of visual relation. Then we describe our probability model and discuss how those parameters can be learned.

7.3.1 Twelve Types of Visual Relations

Except a few extreme examples (less than 1%), an inner block is cut into several child blocks either horizontally or vertically. We define all visual relations based on this assumption. Those exceptional pages are excluded before feed to our extraction process. For description convenience, we use *h-block* to denote an inner block which is horizontally segmented and use *v-block* to denote an inner block which is vertically segmented.

Based on blocks' placement relation and nested depth on visual tree, twelve types of visual relations (indicated as set R) are defined. They are: *Left, Right*, *Above, Below*, *Up-Left, Right-Down*, *Up-Right, Left-Down*, *Up-Above, Below-Down*, *Up-Below, Above-Down*.

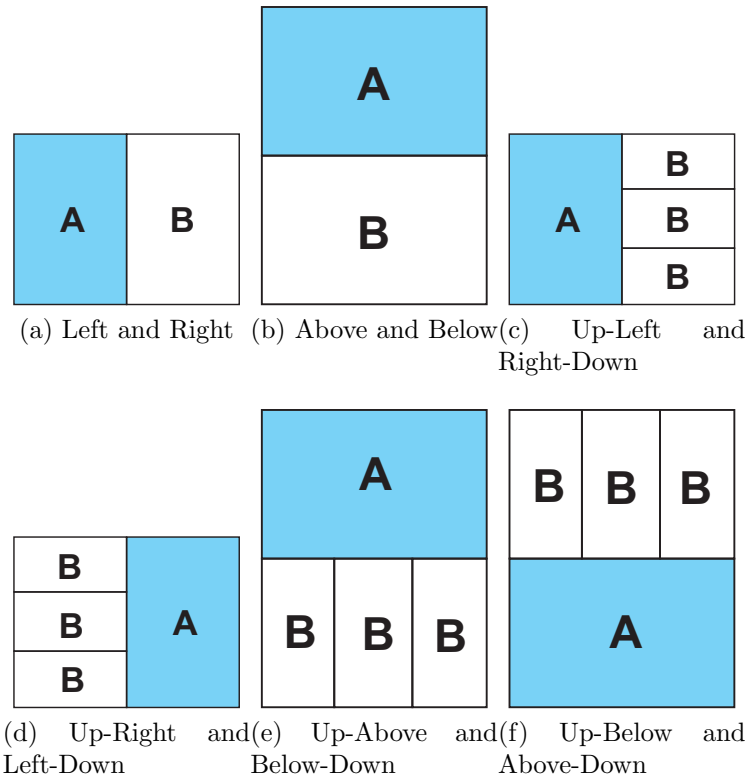
Note that we use "up" and "down" to denote the relation in terms of nested depth and use "above" and "below" to denote the placement relation at same depth.

Given any two blocks b_1 and b_2 on a same page, we use function $\mathcal{R}(b_1,\ b_2)$ to return their visual relation. It can also return *None* if no above relation exist between b_1 and b_2. All above relations are displayed in Figure 7.3. Each sub-figure shows $\mathcal{R}$(A, B) and $\mathcal{R}$(B, A).

Now, the question is how to implement function $\mathcal{R}$ for any two input blocks. Actually, visual relation can be easily inferred from the visual tree structure of given input blocks. For example, Figure 7.4 is the corresponding visual tree structure of Figure 7.3c. In this figure, v-block and h-block are distinguished by a symbol inside their corresponding nodes.

Given a block b and a visual relation r, from the visual tree we can also find a block set $\mathcal{N}(b, r)$ where all blocks have relation r with b.

$$\mathcal{N}(b, r) = \{b' | \mathcal{R}(b, b') = r\}$$



(a) Left and Right (b) Above and Below (c) Up-Left and Right-Down

(d) Up-Right and Left-Down (e) Up-Above and Below-Down (f) Up-Below and Above-Down

Figure 7.3: Twelve Types of Visual Relations

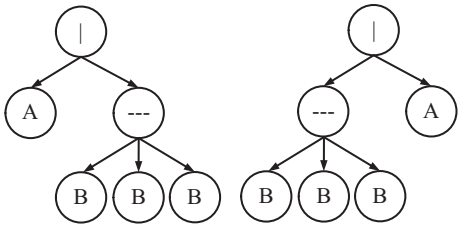


Figure 7.4: Visual Tree Structure of Figure 7.3c and Figure 7.3d

7.3.2 Inter-Fields Probability Model

Here we define the conditional probability model to express how inter-fields relation contributes in predicting the label of a given block. Suppose we are given that a block b_1 is labeled as l_i, and $\mathcal{R}(b_1, b_2) = r$, then the conditional probability of b_2 being l_j is

$$P(b_2 \in l_j | b_1 \in l_i, \mathcal{R}(b_1, b_2) = r)$$

In this model, we do not distinguish feature difference among different blocks,

which means above probability is independent with b_1 and b_2 themselves. Thus we can simply use $P(r, i, j)$ to represent above probability.

Now we are in a place to define a function $\Phi(b, l)$ to calculate the confidence of labeling a block b as field l solely based on the inter-fields model where defined here.

$$\Phi(b, l_j) = \sum_{r \in R} \sum_{b' \in \mathcal{N}(b,r)} \sum_{i \in O, \mathcal{S}(b', c_i) > 0} \mathcal{S}(b', c_i) \cdot P(r, i, j)$$

Here $\mathcal{S}(b', c_i)$ serve as a weight and is directly calculated by the binary classifier c_i.

Now we can get two confidence value for any block being labeled as a certain field. One is returned by the corresponding binary classifier, the other is returned by the inter-fields model.

Natually, the best solution is to combine these two values with a smoothing factor $\alpha \in (0, 1)$. Thus, we have a final confidence functon $\mathcal{S}'$ to replace $\mathcal{S}$ (line 6, Figure 7.1) in function *Predict*.

$$\mathcal{S}'(b, c_i) = \alpha \cdot \mathcal{S}(b, c_i) + (1 - \alpha) \cdot \Phi(b, l_i)$$

We call the new function with $\mathcal{S}'$ as *Predict*$'$.

7.3.3 Learning Parameters

One parameter in our model is $P(r, i, j)$ for all possible combinations of relations and fields. They form a $12 \times 8 \times 8$ joint probability matrix. This matrix can be learned from manually labeled traning samples simply by the definition of conditional probability.

$$P(r, i, j) = \frac{\left\| \{(b_1, b_2) | \mathcal{R}(b_1, b_2) = r, b_1 \in l_i, b_2 \in l_j\} \right\|}{\left\| \{(b_1, b_2) | \mathcal{R}(b_1, b_2) = r, b_1 \in l_i\} \right\|}$$

$(r \in R, i, j \in O)$

In our implementation, another parameter, the smoothing factor α, is set to a fixed value 0.4 according to experience.

7.3.4 When No Pattern Is Found

Although our work is based on the assumption that people follow certain conventions when they design their homepages, it is still not a surprise to see weird designs since they are written manually. For an unconventional homepage, it is imaginable that inter-fileds model might not help, which means $\Phi(b, l)$ is close to zero.

7.4 Experimental Results

7.4.1 Data Collection

We collected our homepage data by meta-search. First, we preapred a list of author name obtained from online digital library *CiteSeer*. Then, we throw each name as a query to Google and get a list of urls. Using several heuristic rules, we choose one url which is most likely to be the homepage of the input query.

7.4.2 Evaluation Method

500 pages are randomly selected from our homepage repository. All pages are manually labeled for extraction result evaluation. Evaluation for each page is performed by comparing manually assigned labels with automatically assigned labels in our extraction process. We use precision, recall, and F1 as measures. A three-folder cross-validation is implemented for the sake of fairness.

We implemented two groups of experiments on a PC with 3 GHz Pentium 4 processor and 1015MB RAM.

Experiment I: The Power of Visual Features

The objective of the first experimental group is to show that utilizing visual features greatly improves the extraction accuracy. We implemented two sets of classifiers using different features. For the first set, all features are used to train the classifiers. For the second set, only content based features are used. Both classifier set are used to perform extraction on the same data set. In both experiments, we use

algorithm *Predict* described in section 7.2 without applying the inter-fields model. For the sake of convenience, we use set-1 and set-2 to denote these two sets.

Figure 10.6 shows the comparison results of using visual features in training and not using visual features in training. As displayed in Figure 10.6, although set-2 outperforms set-1 for extracting telephone and fax, for most fields, using visual features greatly improves the extraction accuracy. Especially, using visual features improve the F1-Value by 49.9% for name field and 52.2% for picture field. This can be explained as follows. For picture field, the only non-visual feature can play a role is the tag-name (`IMG`) since we do not analysis pixelss. Therefore, without considering their position and size, there is no way to tell which different `IMG` is an author's picture and which `IMG` is an unrelated. For name field, normally, there will be more than one name mention on a homepage. Solely based on context text, it is hard to distinguish those names. With the help of visual features, name of the author itself can be easily identified.

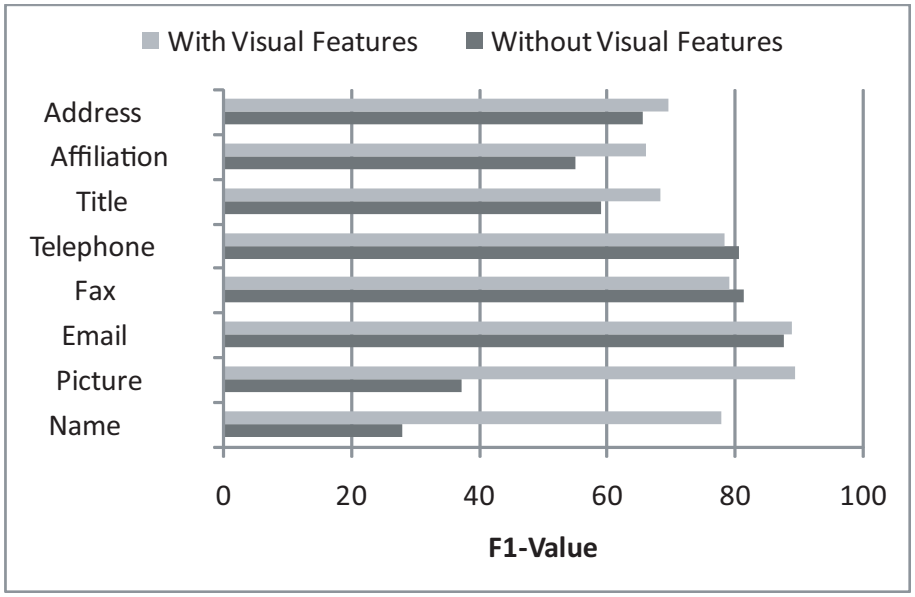


Figure 7.5: Result of experiment one

Experiment II: Effectiveness of Inter-Fields Model

The aim of this group of experiments is to demonstrate the effectiveness of our inter-fields model. Two extraction algorithm *Predict* and *Predict*′ are tested on

the same data set. The comparison results are shown in Figure 9.9. As we can see, applying inter-fields model leads to improvement for most fields. For address and affiliation, the extraction accuracy is improved by around 10% in terms of F1-Value. The reason why inter-fields model works more effective on these two fields than other fields is that these two fields are very similar and are hard to tell from each other without considering the inter-fields relation.

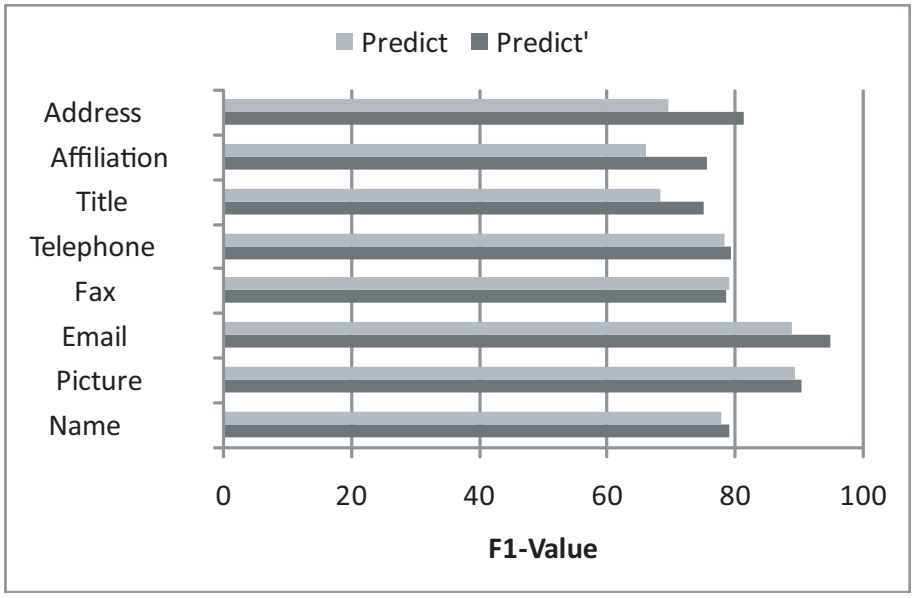


Figure 7.6: Result of experiment two

We also list detailed results of *Predict'* in Table 7.1. The average accuracy of eight fiels in terms of F1-Value is 81.92%.

Field	**Precision**	**Recall**	**F1-Value**
Name	75.28	83.72	79.28
Picture	92.01	89.14	90.55
Email	94.08	95.89	94.98
Fax	82.67	74.88	78.58
Telephone	80.62	78.33	79.46
Title	74.24	76.19	75.2
Affiliation	80.46	71.52	75.73
Address	77.82	85.71	81.57

Table 7.1: Result of meta-data extraction

Chapter 8

Joint Optimization of Wrapper Generation and Template Detection

8.1 Ground-Truth Templates and Similarity-Based Templates

Before going to the details of our approach, in this section we will first describe the main idea of this paper.

What's a ground-truth template? In previous related works, the concept of template has already been presented by various descriptive definitions. Most of them associate a template with a script that encodes the data of a category into a group of HTML pages, called a page class. For example, we can guess that both page (a) and page (b) in Figure 1.7 are generated dynamically by a script for the category of computer while page (c) and page (d) generated by another script for the category of cap. This kind of templates does exist and it is indispensable for wrapper induction systems to generate effective wrappers reversely. We call this kind of templates *Ground-Truth Templates* for they denote the original relations among pages of a website. The corresponding page classes are called *Ground-truth Page Classes*.

However, the ultimate purpose of template detection is not to guess which pages are encoded by a ground-truth template, but to generate more effective wrappers that can extract data correctly. What we propose is to cluster pages into several

groups based on how similar they are. In general, a page is less different from the pages in the same group than those pages in other groups. Each group is corresponded to a template that is called *Similarity-based Template* and the group itself is a *Similarity-based Page Class*.

For a particular page set P, since the ground-truth templates are invisible to us, it is hard and not necessary to ensure that detected similarity-based templates are exactly same to ground-truth templates. For example, for pages like page (c) and page (d) in Figure 1.7, a ground-truth page class may be divided into two similarity-based page classes because the attribute of color is optional so that some pages like page (d) have such attribute while others like page (c) have not. It is very likely that extracted results by using two similarity-based templates are good and even better than those by inducing one ground-truth template because the complexity of these templates gets lower than that of one ground-truth template.

In addition, we found that the definition of similarity is highly related to alignment in the stage of wrapper induction. We propose to use consistent representations in both template detection and wrapper generation and optimize these two stages together to achieve better extraction performance. In our system, tree-structures are used as representations for pages and wrappers, although the representation is not restricted to tree-structure.

8.2 Data Representations

We describe the representations of a Web page and a wrapper in this section.

8.2.1 DOM (Document Object Model) Tree

DOM tree is the representation of a HTML page in our system. Each DOM node of a DOM tree represents an HTML tag pair (e.g., `<TABLE>` and `</TABLE>`). The nested structure of HTML tags corresponds to the parent-child relationship among DOM nodes. Thus a DOM tree is formed naturally. More information about DOM specification can be found at [55].

In our experiments, DOM trees used for wrapper generation are manually labeled so that the generated wrappers can extract values and assign labels in one

step. Labels are only assigned to leaf nodes of a DOM tree. DOM nodes with different labels are considered different no matter whether they have the same tag or not. In the rest of this paper, for a given DOM node σ, we use $\mathcal{T}(\sigma)$ and $\mathcal{L}(\sigma)$ to denote its tag and label.

8.2.2 Wrapper

In our system, a wrapper is also presented in tree structure that can be regarded as extended DOM trees with *Sign* for each nodes.

Definition 6. *(Node Sign) Given a wrapper node σ, its sign $\mathcal{S}(\sigma)$ indicates its matching rule in the alignment between its owner wrapper and a DOM tree. $\mathcal{S}(\sigma)$ can be 1 or an integer $N(N \geq 2)$ or one of the following wildcards: $?, +, *$.*

Rule 1. *Given a wrapper node σ,*

- $\mathcal{S}(\sigma) = 1$ *means σ can only match one DOM node.*
- $\mathcal{S}(\sigma) = N(N \geq 2)$ *means σ can match consecutive N DOM nodes.*
- $\mathcal{S}(\sigma) =?$ *means σ can match one DOM node or no DOM node at all.*
- $\mathcal{S}(\sigma) = +$ *means σ can match consecutive N DOM nodes $(N \geq 1)$.*
- $\mathcal{S}(\sigma) = *$ *means σ can match consecutive N DOM nodes $(N \geq 1)$ or no DOM node at all.*

Such wrapper node signs are similar to the wildcards used in other works like [13, 15].

Another difference between a wrapper and a DOM trees is that a wrapper may have a kind of special nodes that act like pairs of parentheses, called *Parentheses Nodes*. These nodes have no corresponding tags and must be inner nodes with at least one child. For the sake of convenience, we call other DOM nodes or wrapper nodes as *Tag Nodes*.

8.3 Problem Definition and System Overview

In this section we formally define the extraction problem and briefly overview our solution.

8.3.1 Problem Definition

We define the problem as follows:

Given a set of labeled DOM trees D parsed from pages of a particular website, a group of wrappers (w_1, w_1, ..., w_n) should be learned from D. And the target is to maximize the overall extraction accuracy $\mathcal{P}$ when generated wrappers are tested on another DOM-tree set D' that comes from the same website.

In this paper, we use manually labeled training data to explain and verify our ideas. Although the idea of joint optimization of wrapper induction and template detection is not constrained to labeled data, we do so for several reasons. First, our main focus is not on the algorithm of wrapper induction but on how to detect similarity-based templates and how the detection influences extraction performance. Labeled data can simplify the evaluation of extraction results. Second, using labeled data to generate wrappers is commonly used in some scenarios, such as comparison shopping. As the accuracy of price is required close to 100 percent, automatic attributes labeling methods cannot meet the requirement. Furthermore, inducing wrappers based on labeling data is selectively used for only a few of head sites. For each site, only tens of pages are enough to train a robust wrapper set. Thus, the cost of labeling is acceptable.

8.3.2 System Overview

The flowchart of our system is shown in Figure 8.1.

To begin with, training pages are parsed into DOM trees before they are processed by our system. We will not discuss the HTML parsing technique since it is beyond the scope of this paper.

Second, the DOM trees will be fed to the wrapper-oriented page clustering module that combines template detection and wrapper generation into one step and outputs a set of wrappers. A byproduct in the step is that the training DOM trees are also clustered into similarity-based page classes.

When a new Web page comes, it will be parsed into a DOM tree first. Then, our system can automatically select a wrapper from the generated wrapper set,

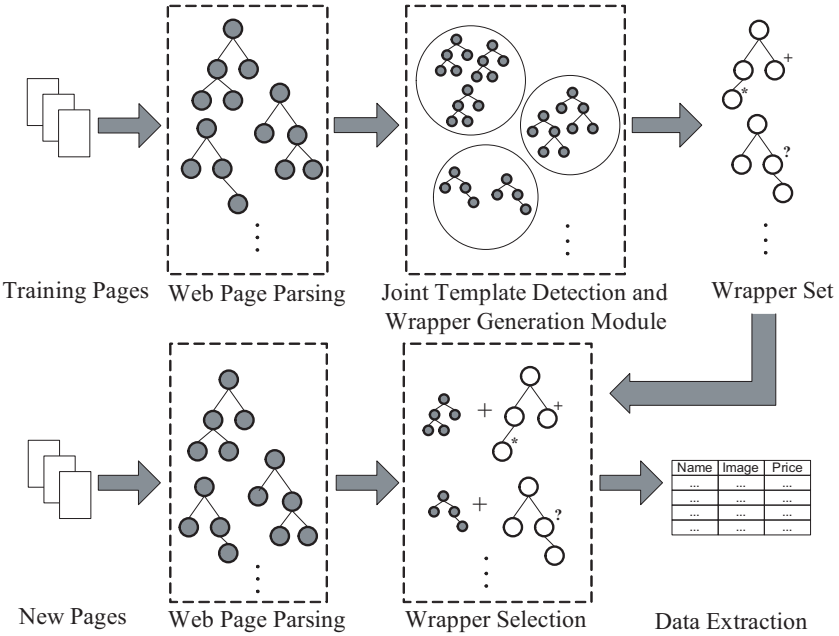


Figure 8.1: System overview

which makes a best match with the DOM tree. At last, data is extracted and saved in a structured format like a relational database.

8.4 Joint Optimization of Wrapper Generation and Template Detection

In this section, we present the idea of joint optimization of wrapper generation and template detection in detail. We first introduce wrapper generation algorithm that is implemented in our system. Then we describe how template detection is combined with wrapper generation by a proposed algorithm called wrapper-oriented page clustering.

8.4.1 Wrapper Generation

In Section 8.4.1.1, we describe how to convert a DOM tree to a wrapper tree. This is the first step for a page before it is evolved in wrapper generation in our system. In Section 8.4.1.2 and Section 8.4.1.3, we implement a cost-driven algorithm to perform wrapper induction. This algorithm synthesizes several state-of-the-art

techniques [13, 16, 15], e.g., regular expression inference.

8.4.1.1 Convert a DOM tree to a Wrapper

Given a source DOM tree T_d, supposing that the converted wrapper is T_w, we use $T_d \rightarrow T_w$ to indicate this conversion.

In $T_d \rightarrow T_w$, we need to perform a repeat pattern combination algorithm to make T_w more compact than T_d. This combination algorithm is similar to the work in [22]. If $N(N \geq 2)$ identical consecutive sub-trees are detected in T_d, they will be merged as one sub-tree rooted at a tag node σ in T_w, where $\mathcal{S}(\sigma) = N$. If $N(N \geq 2)$ identical consecutive sub-forests are detected in T_d, they will be merged as one sub-forest rooted under a parentheses node p in T_w, where $\mathcal{S}(p) = N$. Node labels are considered in the algorithm. Figure 8.2 illustrates this procedure, where letters indicate nodes' tags and subscripts indicate nodes' labels.

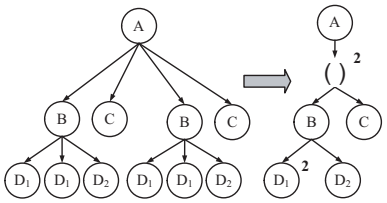


Figure 8.2: Repeat pattern combination

8.4.1.2 Cost-Driven Tree Alignment

Tree alignment is a frequently-used algorithm in our system. There are two types of tree-alignment algorithms: one is for aligning two different wrappers, called WW-alignment, and the other is for aligning a wrapper and a DOM tree, called WD-alignment. We employ cost-driven dynamic programming for both algorithms.

In the tree-alignment algorithm, DOM nodes and wrapper nodes are the basic units for matching. Mismatched nodes will cause cost in the alignment. In order to calculate the cost, we need to assign weight to each node before aligning.

Definition 7. *(DOM-Node Weight) Given a DOM node σ, its weight $\mathcal{W}(\sigma)$ equals the number of nodes in the sub-tree rooted at σ, including itself.*

Definition 8. *(Wrapper-Node Weight) Given a wrapper node σ, its weight $\mathcal{W}(\sigma)$ can be calculated as follows:*

- *If σ is a leaf tag node and $\mathcal{S}(\sigma) = 1$, then $\mathcal{W}(\sigma) = 1$.*
- *If σ is a inner tag node, and $\mathcal{S}(\sigma) = 1$, then $\mathcal{W}(\sigma) = 1+$ sum of its child nodes' weight.*
- *If σ is a parentheses node and $\mathcal{S}(\sigma) = 1$, then $\mathcal{W}(\sigma) =$ sum of its child nodes' weight.*
- *If $\mathcal{S}(\sigma) =?$ or $\mathcal{S}(\sigma) = *$, then $\mathcal{W}(\sigma) = 0$.*
- *If $\mathcal{S}(\sigma) = +$, then $\mathcal{W}(\sigma) = \mathcal{W}(\sigma')$, where σ' is the same to σ except for $\mathcal{S}(\sigma') = 1$.*
- *If $\mathcal{S}(\sigma) = N$, then $\mathcal{W}(\sigma) = N * \mathcal{W}(\sigma')$, where σ' is the same to σ except for $\mathcal{S}(\sigma') = 1$.*

The reason we set a wrapper node σ's weight as 0 if $\mathcal{S}(\sigma) =?$ or $\mathcal{S}(\sigma) = *$ is that this kind of wrapper node is allowed to be mismatched without causing any cost.

In this sub-section, we only describe WW-alignment and leave WD-alignment to Section 8.4.2.2.

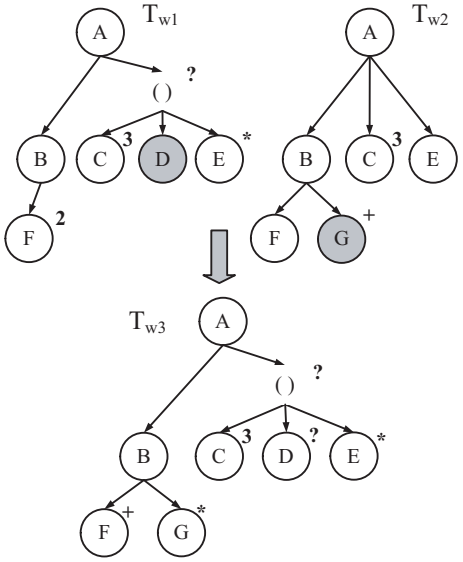
Given two wrappers T_{w_1} and T_{w_2}, the basic procedure of WW-alignment is to align two forests: $\mathcal{A}(F_{w_1}, F_{w_2})$. It is performed in a top-down order layer by layer. Only nodes at the same layer of T_{w_1} and T_{w_2} can be aligned with each other.

Rule 2. *Given two wrapper nodes σ_{w_1} and σ_{w_2}, we say σ_{w_1} matches σ_{w_2}, iff all the following rules are satisfied,*

1. *σ_{w_1} and σ_{w_2} are either both inner nodes or both leaf nodes*
2. *$\mathcal{T}(\sigma_{w_1}) = \mathcal{T}(\sigma_{w_2})$*
3. *If σ_{w_2} and σ_{w_2} are both leaf nodes, $\mathcal{L}(\sigma_{w_1}) = \mathcal{L}(\sigma_{w_2})$*

At each layer, $\mathcal{A}(F_{w_1}, F_{w_2})$ performs a sequence alignment between the array of F_{w_1}'s root nodes and that of F_{w_2}'s. Dynamic programming is adopted here to obtain a minimal cost. All mismatched root nodes in F_{w_1} and F_{w_2} contribute their weight as cost to $\mathcal{A}(F_{w_1}, F_{w_2})$. For a pair of matched nodes σ_{w_1} and σ_{w_2} that are inner nodes, $\mathcal{A}(childF_{w_1}, childF_{w_2})$ will be invoked recursively, where $childF_{w_1}$ and $childF_{w_2}$ are the sub-forests consisting of sub-trees rooted at the child nodes of σ_{w_1} and σ_{w_2}. The cost caused by $\mathcal{A}(childF_{w_1}, childF_{w_2})$ will be counted in the cost calculation of $\mathcal{A}(F_{w_1}, F_{w_2})$. Because our WW-alignment algorithm works in such a top-down recursive way, it attempts to align nodes in two wrappers if and only if their parent nodes are aligned with each other. Such mechanism saves some unnecessary alignment.

Figure 8.3 shows an example of WW-alignment. In this example, label difference is ignored for statement convenience. The alignment algorithm works as follows. First, $\mathcal{A}(\mathtt{A}, \mathtt{A})$ recursively invokes $\mathcal{A}\big(\mathtt{B}(\mathtt{C}^3\mathtt{DE}^*)^?, \mathtt{BC}^3\mathtt{E}\big)$. Then, according to the matching rule of wildcards ? (Rule 1), $\mathcal{A}\big(\mathtt{B}(\mathtt{C}^3\mathtt{DE}^*)^?, \mathtt{BC}^3\mathtt{E}\big)$ seeks a better solution between $\mathcal{A}(\mathtt{BC}^3\mathtt{DE}^*, \mathtt{BC}^3\mathtt{E})$ and $\mathcal{A}(\mathtt{B}, \mathtt{BC}^3\mathtt{E})$. Obviously, the former one costs less by now. Then, $\mathcal{A}(\mathtt{F}^2, \mathtt{FG}^+)$ is invoked recursively by both $\mathcal{A}(\mathtt{BC}^3\mathtt{DE}^*, \mathtt{BC}^3\mathtt{E})$ and $\mathcal{A}(\mathtt{B}, \mathtt{BC}^3\mathtt{E})$ to calculate the cost of these two solutions. In this example, WW-alignment algorithm can find an optimal result as Figure 8.3 shown with the cost of 2.



(Gray nodes are mismatched nodes.)

Figure 8.3: Wrapper induction

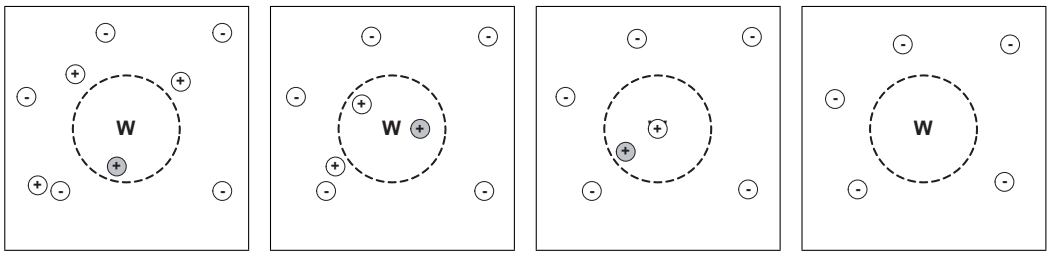
8.4.1.3 Wrapper Induction

After WW-alignment obtains an optimal result between two wrappers, a new wrapper can be constructed according to the sign-inference function $\mathcal{I}$:

$\mathcal{I}(1, \texttt{NULL})$	$=$	$?$	$\mathcal{I}(?, N)$	$=$	$*$
$\mathcal{I}(?, \texttt{NULL})$	$=$	$?$	$\mathcal{I}(?, +)$	$=$	$*$
$\mathcal{I}(n, \texttt{NULL})$	$=$	$*$	$\mathcal{I}(1, *)$	$=$	$*$
$\mathcal{I}(+, \texttt{NULL})$	$=$	$*$	$\mathcal{I}(N, *)$	$=$	$*$
$\mathcal{I}(*, \texttt{NULL})$	$=$	$*$	$\mathcal{I}(?, *)$	$=$	$*$
$\mathcal{I}(1, 1)$	$=$	1	$\mathcal{I}(+, *)$	$=$	$*$
$\mathcal{I}(N, N)$	$=$	N	$\mathcal{I}(1, N)$	$=$	$+$
$\mathcal{I}(+, +)$	$=$	$+$	$\mathcal{I}(N, +)$	$=$	$+$
$\mathcal{I}(?, ?)$	$=$	$?$	$\mathcal{I}(1, +)$	$=$	$+$
$\mathcal{I}(*, *)$	$=$	$*$	$\mathcal{I}(N_1, N_2)$	$=$	$+$
$\mathcal{I}(1, ?)$	$=$	$?$			

where NULL represents a mismatch of a wrapper node. For example, $\mathcal{I}(1, \texttt{NULL})$ is applied for D node because D has the sign 1 in T_{w_1} while it is mismatched in T_{w_2}.

Given two source wrappers T_{w_1} and T_{w_2}, supposing that the generated wrapper



(a) Level-1 wrapper (b) Level-2 wrapper (c) Level-3 wrapper (d) Level-4 wrapper

Figure 8.4: Wrapper-oriented page clustering for one template

is T_{w_3}, we use $T_{w_1} + T_{w_2} \rightarrow T_{w_3}$ to denote this induction procedure. Figure 8.3 also illustrates how to construct a new wrapper based on the alignment result.

8.4.2 Combine Wrapper Generation with Template Detection

In Section 8.4.2.1, we describe the clustering algorithm that combines template detection and wrapper generation to achieve joint optimization. For clustering, a distance metric is defined in Section 8.4.2.2.

8.4.2.1 Wrapper-Oriented Page Clustering Algorithm

Wrapper-oriented page clustering (WPC) is the most novel part of our system. Given a set of DOM trees D, our WPC algorithm clusters DOM trees in D and generates a wrapper for each cluster. Actually, templates are detected one by one in WPC. After a template's clustering process is completed, all clustered DOM trees of this template will be removed and then clustering for another template will start. The cycles will not stop until no DOM tree is left.

Here, we use Figure 8.4 to illustrate the clustering process of one template. In Figure 8.4, "W" represents a wrapper for this template, and positive points represent DOM trees that belong to the same template as the centered wrapper. Each gray point represents a chosen DOM tree that will be used to refine the centered wrapper.

In WPC algorithm, we use *Wrapper Level* to indicate a wrapper's complexity and generality. It is defined as the number of training DOM trees used to learn

this wrapper.

First, a level-1 wrapper T_w is converted from a randomly chosen DOM tree and taken as the center for this template (Figure 8.4a). DOM trees whose distance to T_w is less than a given threshold ϵ (dashed circle in Figure 8.4) are considered belonging to the same template of T_w and will be used to refine T_w. Refining T_w with a DOM tree T_d includes three steps: converting T_d to a level-1 wrapper $T_w^{'}$; generating a new wrapper from T_w and $T_w^{'}$; and replacing T_w with the new wrapper.

After T_w is refined, it is upgraded by one level and becomes more general. Actually, for any DOM tree T_d, the recalculated distance between T_w and T_d is expected to decrease. For the DOM trees that match the wrapper perfectly (e.g., central DOM tree in Figure 8.4c), we will not use them to refine the centered wrapper because they will not bring any change to it. For those DOM trees whose distance to the centered wrapper is less than threshold ϵ, T_d will be employed to refine T_w (Figure 8.4a and Figure 8.4b).

Finally, WPC algorithm stops for one template when no DOM tree is within the given threshold (Figure 8.4d).

The full algorithm of WPC is listed in Figure 8.5.

Our proposed WPC algorithm has only one parameter, i.e., the distance threshold. Fortunately, there is a wide range of the threshold to assure high performance. Please refer to experimental results shown in Section 10.4.

8.4.2.2 Distance Metric

In our system, instead of measuring the similarity between two DOM trees directly, we derive a *Wrapper-DOM Distance* (WD-Distance) to measure the distance between a wrapper and a DOM tree. This distance is used in both wrapper-oriented page clustering module and also wrapper selection module.

WD-Distance is calculated based on the WD-alignment's cost. WD-alignment algorithm is similar to WW-alignment. Thus, we will not describe it in detail but present the difference between them only. In WW-alignment, nodes are aligned in a one-to-one manner; while in WD-alignment a wrapper node whose sign is +, * or N can be aligned with multiple DOM nodes (Figure 8.6).

For WD-alignment between a wrapper T_w and a DOM tree T_d, we use $\mathcal{C}_w(T_w, T_d)$

Algorithm: WPC(D: DOM tree set, ϵ: threshold)
1. **begin**
2. $\mathbb{R}$:= page cluster set;
3. $\mathbb{W}$:= wrapper set;
4. **while** D is not empty
5. create a new page cluster C;
6. select a DOM tree T_{d_1} from D randomly;
7. $T_{d_1} \rightarrow T_{w_1}$;
8. move T_{d_1} from D to C;
9. **for each** T_d in D
10. **if** $\Psi(T_{w_1}, T_d) = 0$
11. move T_d from D to C;
12. **endif**
13. **endfor**
14. **while** $\exists T_{d_2} \in D : \Psi(T_{w_1}, T_{d_2}) < \epsilon$
15. $T_{d_2} \rightarrow T_{w_2}$;
16. $T_{w_1} + T_{w_2} \rightarrow T_{w_3}$;
17. $T_{w_1} := T_{w_3}$;
18. move T_{d_2} from D to C;
19. **for each** T_d in D
20. **if** $\Psi(T_{w_1}, T_d) = 0$
21. move T_d from D to C;
22. **endif**
23. **endfor**
24. **endwhile**
25. add C to $\mathbb{R}$, add T_{w_1} to $\mathbb{W}$;
26. **endwhile**
27. **return** $\mathbb{R}$ and $\mathbb{W}$;
28. **end**

Figure 8.5: Wrapper-oriented page clustering algorithm

to denote the total cost caused by mismatched wrapper nodes and use $\mathcal{C}_d(T_w, T_d)$ to denote the total cost caused by mismatched DOM nodes. When calculating the WD-Distance, it is necessary to normalize the cost $\mathcal{C}_w(T_w, T_d)$ and $\mathcal{C}_d(T_w, T_d)$ by the weight of a whole tree because more nodes the tree have, more nodes are likely to be mismatched. Thus Wrapper-DOM Distance is defined as follows:

Definition 9. *(Wrapper-DOM Distance) Given a wrapper T_w and a DOM tree T_d,*

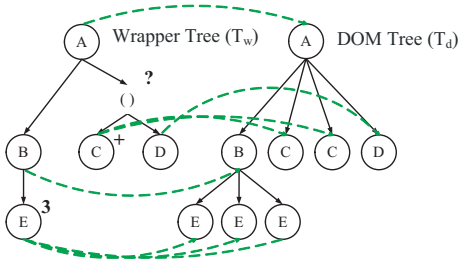


Figure 8.6: Alignment of a wrapper with a DOM tree

the wrapper-DOM distance

$$\Psi(T_w, T_d) = \Big(\frac{\mathcal{C}_w(T_w, T_d)}{\mathcal{W}(T_w)} + \frac{\mathcal{C}_d(T_w, T_d)}{\mathcal{W}(T_d)}\Big)\Big/2$$

Here, weight of a DOM tree and weight of a wrapper are defined as below:

Definition 10. *(DOM-Tree Weight) Given a DOM tree T_d whose root node is τ, then $\mathcal{W}(T_d) = \mathcal{W}(\tau)$*

Definition 11. *(Wrapper Weight) Given a wrapper T_w whose root node is σ, then $\mathcal{W}(T_w) = \mathcal{W}(\sigma)$*

According to Definition 9, WD-distance is the arithmetic mean of the normalized cost caused by the wrapper side and that caused by the DOM-tree side. Thus values are normalized in the range between 0 and 1. $\Psi(T_w, T_d) = 0$ means T_w perfectly matches T_d without any cost, and $\Psi(T_w, T_d) = 1$ means none of the nodes in T_w and T_d match in the alignment.

8.5 Experiments

We test the performance of our approach through experiments.

8.5.1 Experiment Setup

We use a data set of 1,700 product pages from Amazon.com and a data set of mixed 1,000 pages from 10 shopping websites. We call the former data set Amazon and the latter M10 hereinafter.

In each page, product records and their three attributes, namely product name, product image, and product price, are manually labeled. For each website, 2-fold cross validation is conducted. We use precision, recall, and F1 as measures in evaluation of data extraction results. All experiments were run on a PC, with a 3.06 GHz Pentium 4 processor and 3.87 GB RAM.

8.5.2 Experimental Results

Experiment I: Effectiveness Test

On the Amazon data, our joint optimization approach achieves as high F1 as 94.88% by setting the threshold as 0.3 (Figure 8.7) with 44 wrappers generated. For comparison, we implement the separated template detection strategy based on URLs. For specific, training pages are divided into several templates by their URLs. Then, each template generates a wrapper using the same wrapper induction technique as that used in our WPC algorithm. The experimental result shows that these wrappers can only achieve 78% accuracy in terms of F1. Therefore, our approach outperforms the traditional method by about 17 points.

To further evaluate the performance of the WPC algorithm, we run the experiment on M10 data. Table 1 shows the evaluation results for each site. As we see, the average F1 is as high as 97.2%. For seven sites out of ten, the proposed WPC algorithm achieves F1 higher than 98%. The lowest F1 is got on pages from ftd.com. By case study, we find that the number of templates are unbalanced between the training set and the test set. When some templates are unseen in the stage of training, the generated wrappers reject those pages and extract nothing in testing. But for those seen templates, the wrappers can handle them well. That is why we get high precision and low recall.

We notice that pages in site Costco.com are clustered into 23 similarity-based templates. It is surprising because in the viewpoint of a human, training pages of this site share only one template. Then, we use all training pages of this site to generate one wrapper. The wrapper can only achieve 87% accuracy in terms of F1 that is lower than the 23 wrappers generated by our approach.

Table 8.1: Results on 10 shopping websites

Website	**Wra. #**	**Pre.**	**Rec.**	**F1**
ashford.com	1	1.0000	1.0000	1.0000
circuitcity.com	2	1.0000	1.0000	1.0000
costco.com	23	0.9667	0.9153	0.9403
diamond.com	1	0.9875	0.9975	0.9925
ebags.com	2	0.9976	1.0000	0.9988
ftd.com	2	0.9833	0.7528	0.8527
officedepot.com	1	0.9850	1.0000	0.9924
overstock.com	6	0.9224	0.9979	0.9587
pricegrabber.com	1	0.9970	0.9773	0.9870
sears.com	3	0.9960	1.0000	0.9980
Average	4.2	0.9835	0.9640	0.9720

Experiment II: WPC with Different Thresholds

We evaluate the performance of WPC algorithm as the threshold changing on Amazon data. We run WPC 18 times under different thresholds: from 0 to 0.85 with a 0.05 interval. Figure 8.7 shows three curves on performance when the threshold increases from 0 to 0.85. There is no result with greater thresholds because pages chosen are too diverse to learn a wrapper.

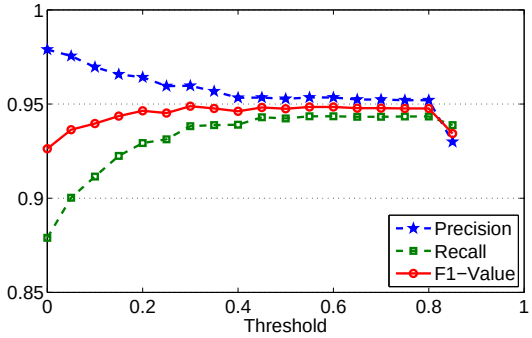


Figure 8.7: WPC performance under different thresholds

When the threshold is set as 0, only exactly matched pages can be absorbed by the wrapper. Thus we got the highest precision while the recall is the lowest. It means that a generated wrapper is specific for the small number of pages although the wrapper is precise in its scope.

As we increase the threshold, the precision drops down and the recall goes up.

In terms of F1, the peak value of 94.88% is achieved by setting the threshold to 0.3. After that, F1 stays above 94% and becomes stable until the threshold is set to 0.85. The stable range, from 0.3 to 0.8 indicates that it is not hard to set an appropriate fixed threshold in the approach.

We also list comparison of the number of wrappers generated with different thresholds in Figure 8.8. The number of wrappers or similarity-based templates decreases quickly from 832 to 44 as the threshold increases from 0 to 0.3. Then the wrapper number decreases slowly. There is an obvious drop if the threshold is set to 0.85. All training pages are clustered into four only. Such wrappers can cover more pages by sacrificing the effectiveness of extraction.

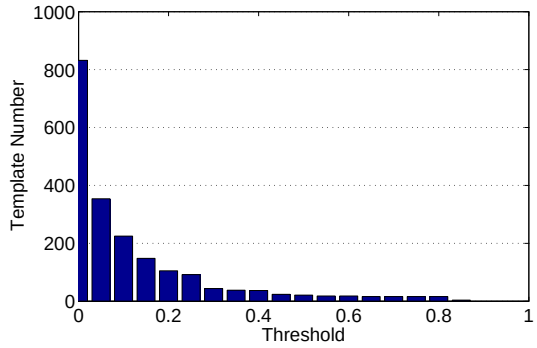


Figure 8.8: Template detection under different thresholds

The impact of different thresholds is also presented by the runtime of our algorithm in the wrapper generation process. When the threshold is set to 0, it takes 13,197 seconds to generate 832 wrappers. Then it drops to 1,424 seconds when the threshold increase to 0.15 and keeps stable around 2,000 seconds until the threshold reaches 0.8. After that, the runtime increases dramatically to 24,666 seconds when the threshold is set to 0.85. The runtime gets too much to tolerate when the threshold is greater. So we treat the situations as it fails to learn a wrapper.

Experiment III: Stability Test

Since our algorithm chooses the initial DOM tree for clustering in a random way, we evaluate how the initial choice impacts performance of our approach in the experiment. We conducted WPC algorithm five times with the threshold set to 0.2

on Amazon dataset. Table 10.1 lists the number of template, extraction precision and recall of each run.

Table 8.2: Stability test result

	Template #	**Precision**	**Recall**	**F1**
1	99	0.9683	0.9249	0.9461
2	116	0.9460	0.9254	0.9356
3	111	0.9606	0.9238	0.9418
4	113	0.9510	0.9138	0.9320
5	111	0.9664	0.9236	0.9445

As Table 10.1 shows, in terms of F1, the mean is 94% while the standard variance is smaller than 4E-5. It indicates that our proposed approach is quite stable with the random strategy to select an initial DOM tree.

Experiment IV: Labeling Cost

As stated earlier, our approach requires manually labeling for wrapper generation. This experiment was designed to show how many training pages are required for learning wrappers to achieve an accuracy higher than 95% in terms of F1. Table 8.3 shows the results for all sites in M10. Actually, most websites only need a handful of labeled pages to meet the demand of accuracy. That proves that the cost of manually labeling in our approach is acceptable.

Table 8.3: Labeling test result

Website	**Page #**	**Website**	**Page #**
ashford.com	12	circuitcity.com	19
costco.com	31	diamond.com	12
ebags.com	19	ftd.com	N/A
officedepot.com	19	overstock.com	16
pricegrabber.com	7	sears.com	27

Chapter 9

Efficient Record-Level Wrapper Induction

9.1 Fundamentals

In this section, we first describe the new data structure used to present records and wrappers in our system. Then we present an overview of our record-level wrapper system.

9.1.1 Data Representation

In our system, a record consists of multiple attributes. For example, a product record can have attributes like "title", "price", "picture", etc. When a page has more than one record, we assign unique IDs ("record id") to them.

Like many other Web information extraction methods, we use the Document Object Model (DOM) [55] to represent an HTML page. In particular, an HTML page is first parsed into a DOM-tree before it can be processed by our system. Nevertheless, our system does not use a full DOM-tree for wrapper induction and data extraction. Instead, we propose a novel data structure, *broom*, to represent a record on a DOM-tree and use such representations for all operations in our system.

As the name implies, a broom has two parts: the "head" and the "stick". The broom head is a record region consisting of sub-trees of a DOM-tree; the broom

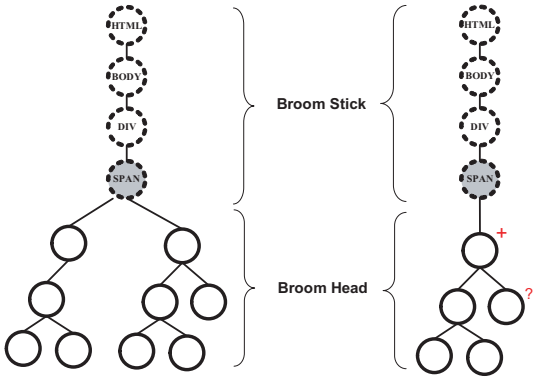


Figure 9.1: Broom Representation

stick is a tag-path starting from the root tag `HTML` to the top of the record region.

In our system, generated wrappers are also represented in such broom structures. The difference is that special wildcards are introduced in their broom-heads in order to give them more powerful matching ability. (Figure 9.1)

The reason to include a tag-path in the representation of records and wrappers is two-fold:

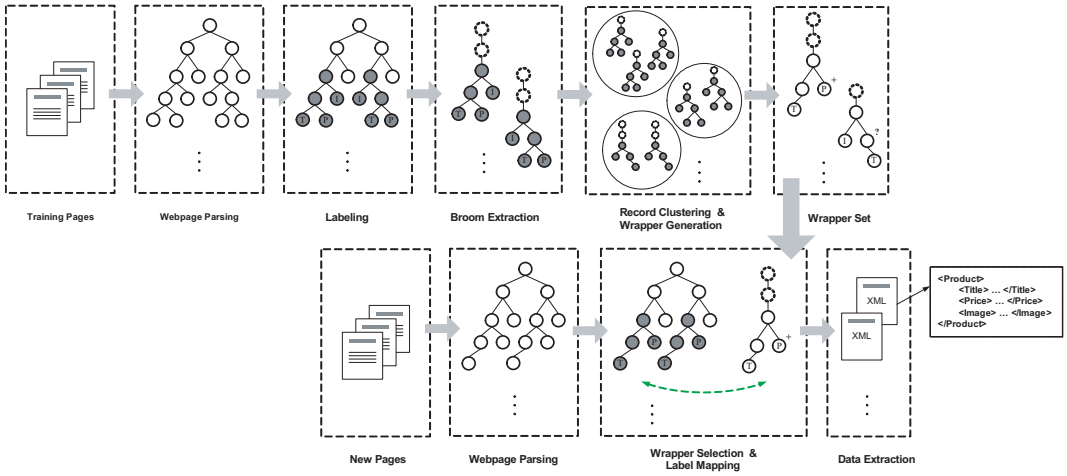
1. For a specific website, different types of records may have the same sub-tree structure. However, they rarely have the same tag-path at the same time. Normally, a wrapper which is generated for one type of records should not be used to extract other types. Including records' tag-paths makes it much easier to distinguish different types of records by comparing their tag-paths.

2. Records in a website can be grouped by their tag-paths. Then we pose two restrictions to our wrapper system. 1) A wrapper should be learnt only from records which share a same tag-path. 2) A wrapper should be used to only extract records which have the same tag-paths as itself. These two restrictions greatly reduce the computational complexity of our system.

9.1.2 System Overview

Figure 9.2 shows the flowchart of our system.

The upper part of this flowchart is the offline training process. First, a set of training pages are converted to DOM-trees by an HTML parser. Then, semantic



Note: 1. Dashed DOM-nodes are tag-path nodes;
2. Letters inside circles are attribute labels.

Figure 9.2: System Overview

labels of a specific extraction schema[1] are manually assigned to certain DOM nodes to indicate their semantic functions. Based on these labels, a broom-extraction algorithm can be applied on each DOM-tree to extract records represented by broom structures. Then extracted records are fed to a module to jointly optimize record clustering and wrapper generation [20]. The main output of this process is a set of wrappers.

The lower part of this flowchart is the online extraction process. When a new page enters our system, it is first converted to a DOM-tree. Then, from the wrapper set generated in the training process, one or more wrappers will be automatically selected to align with the DOM-tree. Labels on selected wrappers will be accordingly assigned to the nodes on the DOM-tree. At last, data contained in those mapped nodes will be extracted and saved in an XML file.

Although most of the previous work [23, 21, 22, 17] attempts to conduct fully automatic wrapper induction without labels, labels do matter when wrapper induction plays an important role in a practical system and high extraction accuracy is required. We choose to include an affordable manually labeling process for three reasons: 1) Labels provide more information to distinguish nodes with the same tags. 2) A wrapper with labels explicitly organizes the extracted data into a cer-

[1] *Extraction Schema* refers to the semantic structure of extracted records. It specifies the type of extracted records (e..g, Product) and the attributes in each record.

tain schema. 3) Labeled training data could improve training efficiency by allowing wrapper induction algorithms to focus on the labeled records on a page and ignore the irrelevant parts.

In our system, we use a user-friendly and easy-to-use labeling tool [56] which greatly reduces the effort made in the labeling process. As illustrated in Figure 10.2, labeling a record can be easily done by several mouse clicks. Those candidate labels shown in the context-menu are loaded from user specified extraction schema. To inform the system that a group of attributes belongs to a same record, users also need to specify a record ID for each attribute they would label. As shown in Figure 10.2, the record ID selected by the user will appear as prefix in each context-menu item. This helps the user to clearly know which record they are assigning labels to. Once a DOM-tree is labeled, attribute labels and record IDs are all attached to the corresponding DOM-nodes.

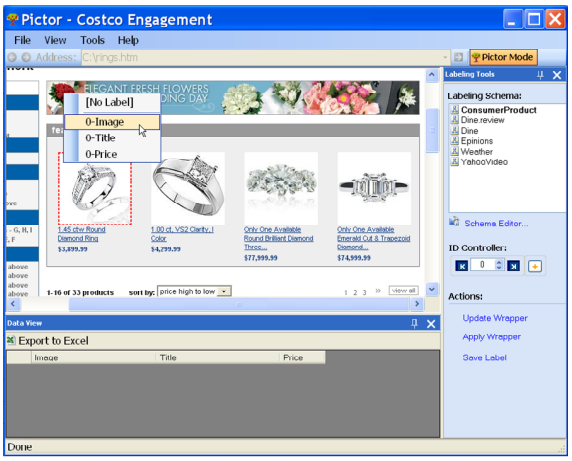


Figure 9.3: Labeling Tool Interface

9.2 Record Wrapper Induction

In this section, we describe how to conduct record-level wrapper induction based on broom structures. First, we formally define two important concepts frequently used in our system. Then, we discuss two scenarios encountered in broom extraction process. At last, we walk through the algorithm with an example.

9.2.1 Minimal Covering Forest & Record Region

Definition 12. *(Boundary Node) Given a labeled DOM-tree and a record ID i, then the boundary node of record i is the root node of a minimal sub-tree which can fully cover all nodes of record i.*

Definition 13. *(Record Region) Given a labeled DOM-tree and a record ID i, then the record region of record i is the smallest set of sub-trees (a forest) which satisfies the following conditions: (1) They can fully cover all nodes of record i (2) They are consecutive siblings rooted at the boundary node of record i.*

In Figure 9.4, 9.5, and 9.6, the gray nodes are boundary nodes and the dotted boxes are record regions for the corresponding labeled records.

Based on the above definitions, the main task of the broom-extraction process is to find the record region (broom head) of each record ID. Once the region is fixed, the corresponding tag-path (broom stick) is also fixed.

9.2.2 Two Scenarios

Our broom-extraction algorithm is designed to handle two possible scenarios.

Figure 9.4 illustrates the first scenarios, which is the most common and simple scenario we encountered in real-world data. In this scenario, different records are rooted at different inner nodes (boundary nodes) without overlapping. After locating a boundary node, extracting the record simply equals to copying consecutive child-sub-trees of the boundary node which all contain nodes of the corresponding record ID. As shown in Figure 9.4, the boundary node is also the last node of the tag-path in an extracted broom.

A special case is shown in Figure 9.5 where multiple records are rooted at a same boundary node. This case explains why we use a forest instead of a tree to define a record region. In this case, we are unable to find a sub-tree which can fully and only cover one record.

The second scenario is about crossed records. In such scenario, as shown in Figure 9.6, multiple records can overlap with each other on a DOM-tree. Consequently, their record regions also overlap. When extracting brooms for such

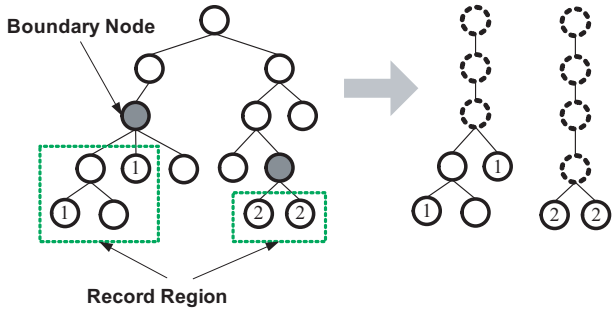


Figure 9.4: Scenario 1 of Broom Extraction

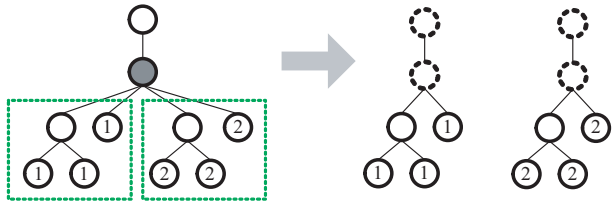
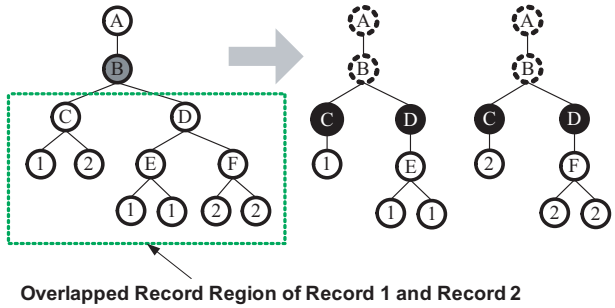


Figure 9.5: A Special Case of Scenario 1

records, some inner nodes will be copied to multiple brooms. We call these inner nodes *Non-Exclusive Nodes* (Black node in Figure 9.6) and the rest DOM-nodes *Exclusive Nodes*. DOM-nodes in the previous scenario are all exclusive nodes.

The property of being exclusive or not will be transferred from extracted brooms to generated wrappers. Exclusive nodes and non-exclusive nodes in a generated wrapper have different matching rules when the wrapper is used in data extraction. See Section 9.3 for more details.



(Black Nodes are non-exclusive nodes)

Figure 9.6: Scenario 2 of Broom Extraction

9.2.3 Algorithms

The detailed broom-extraction algorithm is listed in Algorithm 7.

Algorithm 7 Broom Extraction

Input: Labeled DOM-Tree d, Record ID i
Output: Extracted broom b of ID i

1: Locate boundary node n_B of ID i in d
2: Get Tag-Path $\mathcal{P}$ starting from root node of d to n_B
3: Output $\mathcal{P}$ to b
4: In all sub-trees rooted at n_B, find the first sub-tree t_F and the last sub-tree t_L that contains record ID i
5: Construct a forest $\mathcal{F}$ starting from t_F to t_L
6: **foreach** sub-tree t **in** $\mathcal{F}$
7: SelectivelyCopyOneSubtree(t, i, b)
8: **end foreach**

Method: SelectivelyCopyOneSubtree(t, i, b)
Input: Sub-Tree t, Record ID i, Extracted broom b of ID i

1: **if** t only contains one record ID i or no record ID **then**
2: copy the whole sub-tree t to b
3: set all copied nodes as exclusive node
4: **elseif** t contains record ID i and other record IDs **then**
5: copy root node n_R of t to b
6: set copied n_R as non-exclusive node
7: **foreach** child-sub-tree t_c of t
8: SelectivelyCopyOneSubtree(t_c, i, b)
9: **end foreach**
10: **else**
// t only contains record IDs other than i
// do nothing
11: **end if**

Given a labeled DOM-tree d, the extraction routine in Algorithm 7 should be repeated for all record IDs in d and output one broom per record ID. We use the example in Figure 9.6 to demonstrate how Algorithm 7 works. Just for the ease of explanation, different inner nodes in Figure 9.6 are labeled with different letters. Here we only show how to extract a broom b_1 for record 1. Broom extraction for record 2 can be done similarly. First, we need to find a minimal sub-tree which can fully cover nodes of record 1 and the root node `B` of the minimal sub-

tree is the boundary node. Then, the tag path `A - B` can be fixed and output to the extracted broom. There are two sub-trees rooted at boundary node `B`. We call `SelectivelyCopyOneSubtree` for each of them. For the sub-tree rooted at `D`, since it also contains nodes of record 2, we cannot copy the whole sub-tree to b_1. Instead, we only copy node `D` and recursively call `SelectivelyCopyOneSubtree` for sub-trees rooted at `E` and `F`. The sub-tree rooted at `E` only contains nodes of record 1 and can be copied to b_1 as a whole. The sub-tree rooted at `F` does not contain any node of record 1 and should be ignored. `SelectivelyCopyOneSubtree` for the sub-tree rooted at `C` is similarly processed. Then, we get the broom as shown in Figure 9.6.

After all brooms/records are extracted from the labeled DOM-trees, they will then be fed to a joint optimization process of record clustering and wrapper generating. This process is adapted from our previous work in page level [20]. The work takes mixed DOM-trees for training as input and then combines clustering similar DOM-trees with the same template and generating a wrapper for each cluster in one step. As both template detection and wrapper generation are based on a well-defined pair-wise similarity metrics, that approach can achieve a joint optimization by the criterion of extraction accuracy. To deal with records, we made two main changes:

1. Instead of clustering full DOM-trees, we cluster extracted records represented with broom structures. Consequently, the generated wrappers are also record-level wrappers.

2. In [20], all DOM-trees belonging to a same website will be fed to a single clustering process, whereas, in our system, only records with exact same tag-path will be fed to a common clustering process.

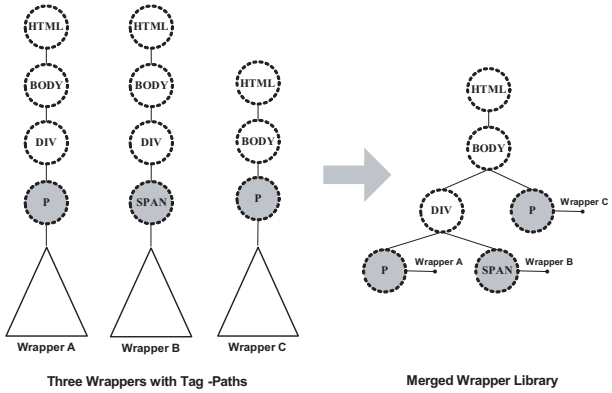
9.3 Record Extraction

In this subsection, we describe how to assemble generated small record wrappers into a wrapper library so that duplicated matching of tag-paths can be collapsed.

9.3.1 Constructing Wrapper Libraries

To improve efficiency, wrappers generated from a certain website should only be applied to new pages from the same site. Within this website, each wrapper only attempts to extract records with the same tag-path. Normally, for a potential record which needs to be extracted, only wrappers generated for the same tag-path from the same website can extract it correctly. Therefore, it is wise to impose the above restriction to make the data extraction process more efficient and effective.

In our system, generated wrappers are hierarchically organized according to their original host websites and tag-paths. In particular, we construct a wrapper library for each website. The main task of this construction process is to merge different tag-paths into a tree structure, called wrapper directory, where individual wrappers are linked to the nodes constructed from original boundary nodes (last node of a tag-path). This is a top-down process of merging same prefixes of multiple tag-paths. A simple example is show in Figure 9.7 to illustrate this process.



(Gray nodes are boundary nodes)

Figure 9.7: A Simple Example of Constructing a Wrapper Library

Two facts should be stressed for the output wrapper library. (1) Sibling nodes will always have different tags. Otherwise, they should be merged to form a longer common prefix for their tag-paths. (2) Given a tag-path of a wrapper, its location on the library tree is determined. This means wrappers with same tag-path will also be linked at the same boundary node on the library tree.

9.3.2 Extracting Records with a Wrapper Library

When a new page of a certain website comes to our system, it is first converted to a DOM-tree by an HTML parser; then the wrapper library generated for this website will be used to perform data extraction from the DOM-tree. If the DOM-tree contains multiple records, they will be assembled as a metadata of the given page. Detailed extraction algorithm is listed in Algorithm 8.

Note that nodes that have already been matched with non-exclusive nodes of a wrapper will not attempt to match another wrapper. This is to make sure the same record will not be repeatedly extracted.

The algorithm consists of two major steps: tag-path mapping and record extraction. By comparing the tag-path of the target DOM tree with that of the wrapper library, in a top-down manner, only wrappers whose tag-paths can be identified on the target DOM tree will be considered as candidates for record extraction. Once a tag-path of the DOM tree is identified, which means a DOM node $root_d$ is mapped with a boundary node $root_d$ on the wrapper library tree (step 4), then the sub-tree rooted at $root_d$ is considered as the extraction region. All wrappers linked at the $root_d$ will try to match this extraction region. The matching process is basically a top-down tree alignment. The wrapper with a best match will be used to perform the extraction.

Given a DOM tree and a wrapper library, suppose there are t different tag-paths on the DOM tree which can be identified by the wrapper library. The average number of wrappers for each tag-path is k. The average runtime of aligning two trees are A. The runtime of tag-path mapping can be ignored compared to A. Then a rough estimation of the extraction runtime for a given DOM tree is $t(k+1)A$. According to our experiments on various datasets, $t = 1$ for most cases and k is mostly less than five. By applying various programming techniques, we can also control the tree alignment cost A to be 10 - 100 milliseconds. Therefore, the average runtime for extracting records from a page is usually much less than one second.

Algorithm 8 Data Extraction with Wrapper Library

Input: DOM-Tree d, Wrapper Library $\mathcal{L}$
Output: Extracted Record Set $\mathcal{R}$

$root_d$: Root node of d
$root_L$: Root node of $\mathcal{L}$
$\mathcal{R}$ = ExtractFromSubDomTree($root_d$, $root_L$)
Return $\mathcal{R}$

Method: ExtractFromSubDomTree($root_d$, $root_L$)

$\mathcal{W}$: Wrapper set linked at $root_L$
$\mathcal{R}$: Extracted record set
$\mathcal{F}$: forest consisting of all sub-trees rooted at $root_d$
if $root_L$ is boundary node **then**
    while (true)
        search wrapper w in $\mathcal{W}$ with a best match with $\mathcal{F}$
        if w is found **then**
            Extract a record r from $\mathcal{F}$ with w
            $\mathcal{N}$: nodes of $\mathcal{F}$ mapped with w's exclusive nodes
            Set all nodes in $\mathcal{N}$ as matched
            Add r to $\mathcal{R}$
        else
            break
        end if
    end while
end if
foreach child wrapper library node $child_L$ of $root_L$
    foreach child DOM-node $child_d$ of $root_d$
        if $child_d$'s tag = $child_L$'s tag **then**
            $\mathcal{R}'$ = ExtractFromSubDomTree($child_d$, $child_L$)
            Add all records in $\mathcal{R}'$ to $\mathcal{R}$
        end if
    end foreach
end foreach

9.4 Record Disambiguation

In this section, we describe how to use content text to disambiguate attributes which are embedded in similar HTML tag trees.

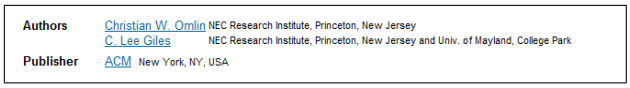
By default, we do not consider content text during wrapper induction and data extraction. In wrapper induction process, two wrappers are aligned with each other

in order to generate a more general wrapper. During such wrapper aligning, only nodes with same tag and same label can be mapped. Similarly, in data extraction, when a wrapper is aligned with a potential record region of a DOM-tree, only nodes with same tag can be mapped.

Ignoring content does not result in accuracy loss for most cases. The tag-tree structures of records and generated wrappers are already very informative to avoid ambiguity. However, there are still cases where tags are not enough to distinguish different nodes in the aligning process we mentioned above. It can lead to mismatch of different attributes when aligning two tag-trees and thus decrease the extraction accuracy of generated wrappers.

To show this problem, we use an example of two records taken from `portal.acm.org` (Figure 9.8). The attributes we want to extract for each record are "Authors", "Sponsor", and "Publisher". In this example, the tag-tree structure of the sponsor row and that of the publisher row are identical. Therefore, if we align these two records to infer a wrapper, the publisher row of the second record will be aligned with the sponsor row of the first record because the sponsor row is absent in the second record. Apparently, such mismatch would bring error to the generated wrapper.

(a)



(b)

Figure 9.8: Examples from portal.acm.org

To resolve the attribute ambiguity caused by similar tag trees, we propose to utilize content text. In above example, by comparing the content text in the first column, the publisher row of the second record will tend to align with the publisher row of the first record since they both have the same text "Publisher" in the beginning.

Obviously, considering content text will increase the runtime complexity for

both wrapper generation and data extraction. We need to compare content text every time we attempt to align two DOM nodes. To limit such complexity increase as well as improve the extraction accuracy, we implement an approach which only considers content text when necessary. First, our approach considers surrounding text in wrapper induction selectively. Only when two consecutive identical tags happen to have different labels, we use text to differentiate them. It is not necessary to waste time on other cases because they are distinguishable by tags. Second, our approach is able to automatically infer template text. When two wrappers are aligned to generate a new wrapper, only overlapped text will be transferred to the new wrapper. In above example, only text in the first column of both records will be transferred to generated wrappers. Third, when a wrapper is aligned with a DOM-tree in data extraction, if there are multiple possible alignments with the same smallest aligning cost, our system will make a decision based on how well the inferred template text is aligned with the text on the DOM-tree. The one with less text mismatch will be chosen as the final solution.

9.5 Experiments

Experimental results are presented in this section to show the performance of our record-level wrapper system. For comparison purpose, we also implement a page-level wrapper induction approach using techniques proposed in [20]. However, we do not compare our system with other wrapper induction systems for the following reasons. First, our system treats DOM nodes as the extraction unit while others treat strings as one. Second, their labeling methods are quite different from ours as well. Therefore, directly utilizing their datasets is impractical.

All experiments were run on a PC, with a 3.06 GHz Pentium 4 processor and 2.0 GB RAM.

9.5.1 Experiment Settings

9.5.1.1 Dataset

We collected our experimental data from 16 real-life large-scale websites belonging to four different domains (i.e., online shops, user reviews, digital libraries, search

results). Four different extraction schemas are defined for these domains.

Table 9.1: Extraction Schemas

Record	**Attributes**
Product	title, price, picture, description
Review	author, title, rating, date, comment
Article	title, author, affiliation, year, pages, ...
Search Result	title, snippet, url

There are seven detail page datasets (Table 9.2) and 11 list page datasets (Table 9.3). Two websites, i.e. amazon.com and circuitcity.com, provide both list page datasets and detail page datasets. That is why 18 datasets come from 16 websites. Each dataset contains more than 1000 pages. All these pages are manually labeled for the purpose of evaluation.

9.5.1.2 Evaluation Metrics

In all experiments, records are equally weighted despite of whether or not they are extracted from a same page. Given a page p for evaluation, we have the extracted metadata $\mathcal{M}_e$ and the manually labeled ground-truth metadata $\mathcal{M}_g$. Both $\mathcal{M}_e$ and $\mathcal{M}_g$ consist of records which belong to page p. The goal of the evaluation is to calculate the *precision*, *recall*, and *F1-value* for $\mathcal{M}_e$ by comparing it with $\mathcal{M}_g$.

First, records in both $\mathcal{M}_e$ and $\mathcal{M}_g$ are sorted according to the order they appear on the DOM-tree of p. Then, we need to align records in M_e with those in M_g because a ground-truth record might be falsely extracted as two records. A dynamic-programming based algorithm is executed to find an optimal alignment which maximizes the overall F1-Value of p.

For each candidate alignment, suppose record r_e in $\mathcal{M}_e$ is aligned with record r_g in $\mathcal{M}_g$, the attribute-level precision (P_{attr}) and recall (R_{attr}) for record r_e can be calculated with the following equations:

$$P_{attr} = \frac{|\text{correctly extracted attributes}|}{|\text{attributes in } r_e|} \tag{9.1}$$

$$R_{attr} = \frac{|\text{correctly extracted attributes}|}{|\text{attributes in } r_g|} \tag{9.2}$$

Then, we can calculate the average attribute-level precision ($\overline{P_{attr}}$) and recall ($\overline{R_{attr}}$) for all aligned records in $\mathcal{M}_e$. Besides, precision (P_{rec}) and recall (R_{rec}) can also be calculated to show the accuracy in record level:

$$P_{rec} = \frac{|\text{aligned records}|}{|\text{records in } \mathcal{M}_e|} \tag{9.3}$$

$$R_{rec} = \frac{|\text{aligned recordss}|}{|\text{records in } \mathcal{M}_g|} \tag{9.4}$$

Finally, overall accuracy of $\mathcal{M}_e$ is computed based on averaged attribute-level PR value and record-level PR value:

$$precision = \overline{P_{attr}} \cdot P_{rec} \tag{9.5}$$

$$recall = \overline{R_{attr}} \cdot R_{rec} \tag{9.6}$$

$$F1\text{-}Vallue = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{9.7}$$

9.5.2 Effectiveness Test

We compare our proposed record-level wrappers with the page-level wrappers proposed in [20] upon all 18 datasets. Results are shown in Table 9.2 and 9.3.

The extremely high accuracy indicates that our approach is effective to extract structured data. For list pages, the record-level wrappers improve the page-level wrappers by 7%. For example, Dataset L1 (`www.amazon.com`) contains 300 pages with crossed records and the results proves that record-level wrappers can deal with such scenarios much better than page-level wrappers. For detail pages, the page-level wrappers and the record-level wrappers perform equally well in terms of F1-Value, but the record-level wrappers use less time for all these websites.

9.5.3 Effect of Content Text

Results in Table 9.2 and 9.3 are obtained without using the strategy proposed in Section 9.4. If we also consider the context text, the F1-Value of dataset D11 (`portal.acm.org`) can be increased to 1. As a byproduct, this strategy also results in about 20% increase in extraction time. This is the only website in our dataset which requires content text for disambiguation purpose.

Table 9.2: Results on List Pages (F1-Value)

Website	**ID**	**P-Wrapper**	**R-Wrapper**
www.amazon.com	L1	0.8105	0.9572
www.circuitcity.com	L2	1	1
www.diamond.com	L3	0.9262	0.9926
www.ebags.com	L4	0.9361	0.9962
www.epinions.com	L5	0.9639	0.9701
www.google.com	L6	0.9232	0.9916
scholar.google.com	L7	0.87	1
Average		0.918557143	0.986814286

Table 9.3: Results on Detail Pages (F1-Value)

Website	**ID**	**F1-Value**
www.amazon.com	D1	0.9601
www.buy.com	D2	0.9952
www.circuitcity.com	D3	1
www.compusa.com	D4	1
www.costco.com	D5	1
www.jr.com	D6	1
www.newegg.com	D7	1
www.overstock.com	D8	1
www.target.com	D9	0.9856
www.walmart.com	D10	1
portal.acm.org	D11	0.9292
Average		0.988190909

9.5.4 Efficiency Test

This experiment is designed to compare the average runtime used for extracting a record by page-level wrappers and record-level wrappers. The time for training is limited because it runs upon a small number of pages, while the extraction time is critical for wrapper systems because a huge number of pages will be processed. The 11 detail-page datasets are used for this experiment. Figure 9.9 shows the results.

Obviously, the runtime difference is significant. On average, record-level wrappers are four times faster than page-level wrappers. It is explainable because aligning two broom structures in record-level wrapper system is much easier than

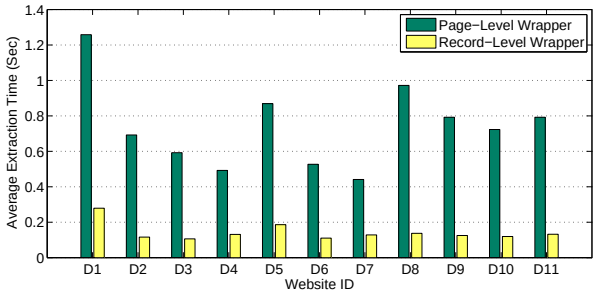


Figure 9.9: Comparison on Efficiency

aligning two full DOM-trees in the page-level wrapper system. Also note that, for page-level wrappers, the extraction time notably varies across different websites because the amount of irrelevant information differs in the websites. On the contrary, the extraction time for record-level wrappers has less variation for most websites since the record size is relatively stable.

In addition, we compare our system with page-level wrapper method regarding labeling effort. We do not directly account the labeling time because it varies a lot with regard to different labelers. Instead, we implement the comparison by counting how many records are used in different methods to train a wrapper set which can achieve a specified accuracy. This somehow shows how much labeling effort can be saved in our system. According to the results, the average record number required by training page-level wrapper is about 129, whereas the number required by training record-level wrapper is about 16. Therefore, our system roughly saves 87% labeling cost compared to the page-level wrapper induction method in [20].

Chapter 10

Pictor: Generating Semantics in Hypertext with Minimal Labeling Cost

With the rapid growth of information technologies, especially the emergence of Web 2.0 environments, data growth on the web continues. Therefore, high quality knowledge-providing services such as Vertical Search Engines [57] and RSS [58] are very useful resources for data acquisition. In such services, the original information has to be understood and reorganized before being used. However, most information on the Web are presented in the HTML format and does not intrinsically have machine-understandable semantics. We argue that adding semantics to hypertext would significantly help the process of knowledge acquisition.

Most Web information can be considered as different types of Web records (Web objects). Typical examples would be products on a shopping site such as Amazon.com, or individual profiles on Social Utility website such as facebook.com. In fact, these web records are normally rendered from a backend relational database, which means their original format is most likely to be a database record. Consequently, given any webpage, if we are able to effectively identify Web records as well as their internal attributes (or fields), we can generate some semantic understanding of the information on that page. The identified Web records can be extracted and re-organized into a structured format (e.g., XML).

One of the most effective methods to extract Web information is wrapper in-

duction. In this work, we describe our wrapper system, *Pictor*, which can identify records from webpages with minimal labeling cost.

10.1 Label Matters

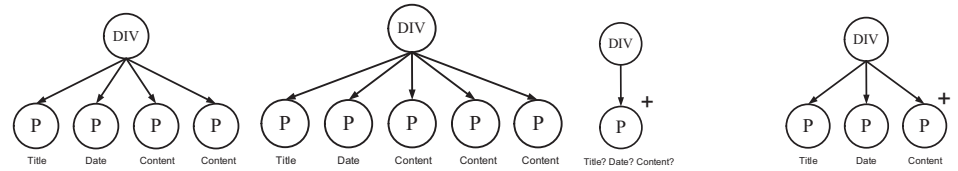
In this section, we explain how manually labeling benefits wrapper generation when we aim to bring semantics to hypertext. Although, fully automatic extraction methods without any human involved is considered as the mainstream in academics for years [23, 21, 22, 17], we argue that manually labeling would lead to better performance of generated wrappers in terms of both effectiveness and efficiency in practice, especially in the era of Web 2.0.

10.1.1 Eliminate Ambiguity

In the methods of extracting structured data from hypertext, tags are widely used as signs to distinguish different parts, whether the hypertext is represented as a string or a tree. However, just using tags can lead to ambiguity. In this work, we will explain this problem by an example based upon DOM trees [55], although similar ambiguity also occurs in the methods using strings. In our example (Figure 10.1), DOM Tree 1 (Figure 10.1a) and DOM Tree 2 (Figure 10.1b) are parsed from two news pages. (Both DOM trees are simplified to make the explanation more clear.) The text under each node is a manually assigned label indicating what type of information is contained by the node. Suppose we want to generate a wrapper which can extract three fields, namely *Title*, *Date*, and *Content*, if we do not consider nodes' labels, a wrapper generated from these two DOM trees most likely looks like the one in Figure 10.1c, where the wildcard "+" denotes the followed one or more consecutive nodes with tag `P` contain content of the same field. Thus, it is unable to distinguish three fields when we apply this wrapper to a new page.

Labels can eliminate some ambiguity. In the above example, if we consider nodes' labels, we can generate a wrapper like the one in Figure 10.1d. The difference is that a tag and its label together represents a node. As a result, the `P` nodes are not merged in this wrapper because of different labels. Now we could extract

the three fields from a new page successfully.



(a) DOM Tree 1 (D1) (b) DOM Tree 2 (D2) (c) Wrapper (W1) (d) Wrapper 2 (W2)

Figure 10.1: Label helps eliminate ambiguity

10.1.2 What We Extract Is What We Need

In general, we need a mapping from attribute values to attribute names in applications based on data extraction. Without using labels in the training phase, a manual/automatic post-labeling is necessary after attributes values are extracted and aligned. Such a mechamism has two drawbacks.

1. Attribute alignment is prerequisite for post-labeling. Without labels, we do not know which parts users do care and which parts not. A blind alignment is costly and difficult. That results in relatively low extraction accurracy.

2. Automatic attribute names mining is too difficult to be used. Normally, it requires prior knowledge about the data. The trial to mine attribute names from surrounding text is interesting but the achieved accuraccy cannot meet requirements from real applications [59].

By manually assigning labels to corresponding DOM nodes in training pages, those labels can be inherited by generated wrappers, as illustrated in Figure 10.1d. Consequently, given a new page, records extracted by such labeled wrapper could be directly fit into the predesigned schema and form a structured format without any post-processing. In a word, labels impose semantics to extracted records.

10.1.3 Improve Efficiency

For most wrapper induction systems, wrappers are generated based on pairwise comparison among training pages. If pages are represented as DOM trees, then

in each step of the training phase, two DOM trees need to be aligned to infer a wrapper from the alignment result[1]. In the testing phase, the DOM tree of a given page is also required to align with a wrapper to extract data. Dynamic programming is frequently used for such aligning purpose. Due to the nature of dynamic programming, these kinds of aligning and induction operations become the main contributor for the overall runtime complexity of a wrapper induction system.

Fortunately, with the presence of manually assigned labels in each train page, we can bound one or more interest regions which contains all the data we need to extract. On the corresponding DOM tree, these interest regions are normally sub-trees of the original DOM tree. By setting interest regions, irrelevant parts of the original DOM trees can be pruned. This will greatly reduce the time used in tree alignment, because now, we only need to align two small sub-trees rather than align two complete trees.

10.2 Interface and system

In this section, we present our wrapper induction system *Pictor* and use an example to explain how it works.

10.2.1 Interface of Pictor

A user-friendly interface plays an important role in achieving the goal of minimizing labeling cost. As shown in Figure 10.2, the main part of the Pictor interface is an embedded web browser. This browser can be used in two different modes: browsing mode and labeling mode (a.k.a. Pictor mode). In browsing mode, online webpages or offline local pages can be opened for labeling purpose. When an opened page is completely loaded, we can switch to labeling mode by clicking the switch button as shown in Figure 10.2.

[1]If pages are represented as plain text, alignment is among strings instead of DOM trees

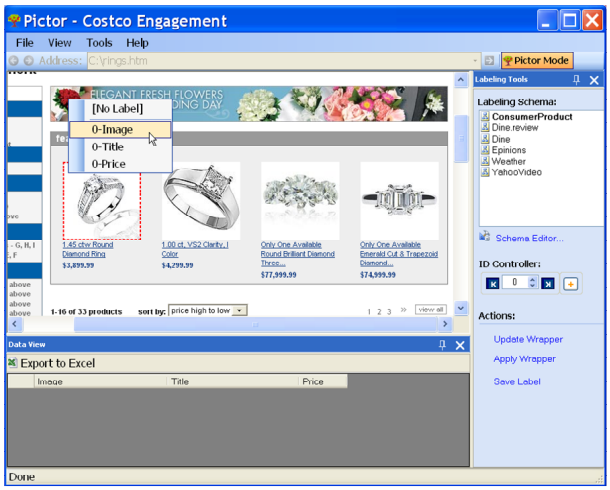


Figure 10.2: Interface of Pictor

10.2.2 Extraction Schema

Before a labeling task can be performed, an extraction schema has to be defined by the user. This schema specifies the structure of extracted data and provides candidate labels for labeling purpose. In Pictor system, an extraction schema can be expressed in a plain text file or a structured XML file, both of which can be loaded before labeling.

Although the semantics of the generated wrappers come from the pre-defined schema, they do not depend on the schema. Changes made to the original schema will not affect already generated wrappers.

The use of schema makes it possible to share similar semantics structures for different tasks. Our system allows user to modify an existing schema by adding more attributes. Thus, they do not always need to start from scratch.

10.2.3 System Workflow

Given a set of sample pages, a typical workflow of page labeling and wrapper generation is shown in Figure 10.3.

During the entire labeling process, each individual sample page is labeled separately. Here, we use an example from www.costco.com to explain several main steps. In this example, the extraction schema is designed as follows: each record contains three fields, namely *Image*, *Title*, and *Price*.

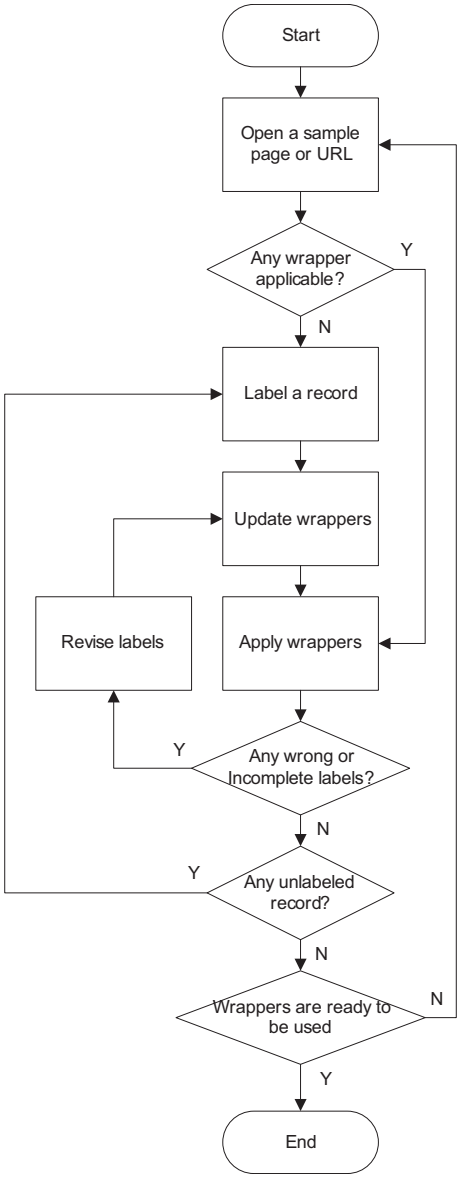


Figure 10.3: Work Flow

10.2.3.1 Detect applicable wrappers for a new page

Given a new page, the labeling process does not always need to start from scratch. If one or more wrappers generated from other similar pages, they could be applied to current page to predict some labels. More detail about predicting labels using existing wrapeprs will be explained in Section 10.2.3.4. Note that the detection of existing applicable wrappers is automatically done by the Pictor system when it

enters the labeling mode.

10.2.3.2 Label a record

Suppose there are no existing wrappers applicable to current page, which means we have to start from scratch, the first thing we need to do is to assign labels to a complete record. In the labeling mode of our system, assigning labels to a record can be easily done with several mouse clicks as illustrated in Figure 10.2. To let the system know that a group of attributes belong to a same record, users need to specify a record id for each attribute they would label. As shown in Figure 10.2, the record id selected by the user will appear as prefix in each menu item. This helps the user to clearly know which record they are assigning labels to.

10.2.3.3 Generate or Update wrappers

After a single record is labeled, it can be used to generate or update underlying wrappers. This process is automatically done by the system at backend and invisible to users. The algorithm of updating underlying wrappers is presented in Section 10.3.

10.2.3.4 Apply wrappers

The most interesting part of this work is the wrapper-assisted labeling strategy. The basic idea of this labeling strategy is to use previously generated wrappers to predict labels of similar records within a same page. In Figure 10.4, wrapper generated from the first record is used to predict labels for the other three similar records.

In most cases, records in same page are often similar even identical in terms of internal structure. Therefore, labeling a very small number of records may generate wrappers which can automatically label all rest records without more human labor.

10.2.3.5 Revise labels

In the early stage of labeling, the underlying wrappers may not be able to correctly predict all labels. For some records, there might be wrongly assigned labels. For

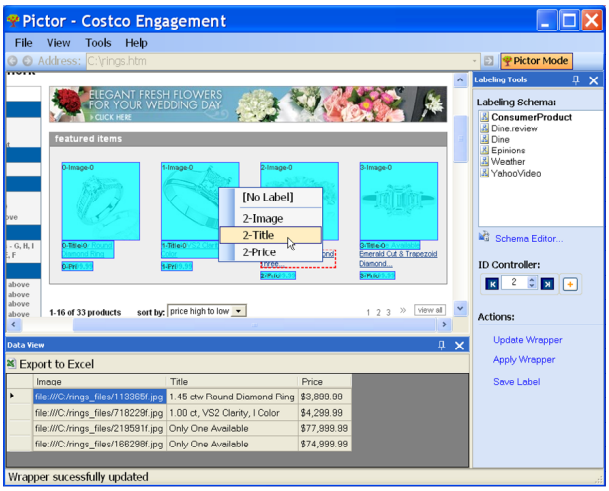


Figure 10.4: Apply wrappers and revise labels

some records, their attributes might not be completely labeled. For example, in Figure 10.4, the third and forth records have an unlabeled attributes.

Our Pictor system allows modifying labels assigned by underlying wrappers. Actually, the process of modifying labels of a record is almost same as assigning new labels to it. Figure 10.4 illustrates how to append a label to a partially labeled record.

As we can see from Figure 10.3, the labeling task for one page can be finished by repeat the process of labeling a new record, updating wrappers, applying wrappers, and revising labels. The final labeling result of previous example is shown in Figure 10.5.

10.2.3.6 Decide if the wrappers are ready

One tricky thing in wrapper training is to decide the number of training pages. More specifically, it is difficult to predict whether generated wrappers can handle all kinds of cases in real dataset because the amount of real dataset is often much larger than the training set. In order to ensure the quality of output wrappers and to avoid redundant labeling, in our system, we propose an effective method to predict the performance of generated wrapper. To do so, users need to supply an unlabeled test set and our system is able to predict the precision/recall values of generated wrappers. Of course, the test set should be randomly selected from

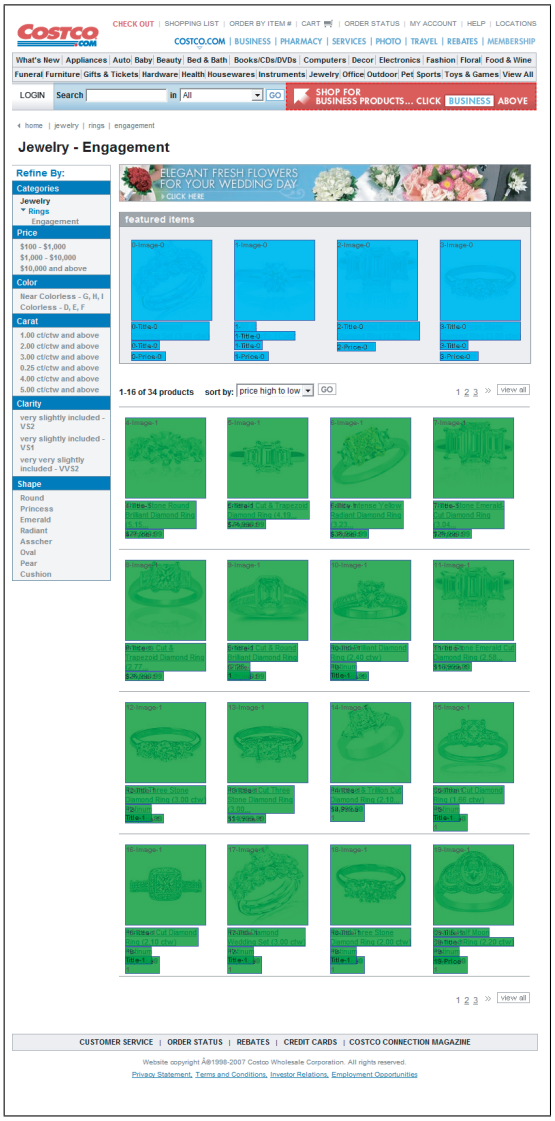


Figure 10.5: Completed labeling results

real dataset to make the prediction as accurate as possible. More details about wrapper performance prediction will be explained in section 10.3.

10.3 Underlying Techniques

In this section, we describe three key techniques used in our Pictor system. It is those techniques that make it possible to minimize the labeling cost as well as achieving high extraction accuracy of generated wrappers.

10.3.1 Record-Level Wrapper Induction and Extraction

The first important technique we implement in our Pictor system is the idea of record level wrapper induction. This idea makes it possible to avoid labeling records in the same page with identical structures. This wrapper induction method is quite general and proved to be effective for most websitese.

10.3.1.1 Record-Level Wrapper Training

Like many wrapper induction systems, we also use DOM tree to present a webpage. And the approach we used to represent and generate wrappers can be considered as a variation of works in [20]. The major difference is that we train wrappers in record level while the system proposed in [20] generates page-level wrappers. More specifically, when a user finishes labeling one record, the system will extract the sub-DOM-tree corresponding to the record and feeds it to the incremental wrapper updating process.

Like in works [20], several basic operations on DOM-trees and wrappers are supported in our system.

1. Distance between any DOM tree and wrapper or between any two wrappers can be measured by aligning two tree structures. We use $\Psi(\cdot)$ to indicate the distance function.

2. Any DOM tree t can be converted to a primitive wrapper w. We use $w \leftarrow t$ to indicate such conversion.

3. Any two wrappers w_1 and w_2 can be merged to generate a new wrapper w'. We use $w' \leftarrow w_1 + w_2$ to indicate this process.

Detailed algorithm of updating exisiting wrapper set using a new record is shown in Algorithm 9.

Note that there is a wrapper merging sub-procedure in Algorithm 9. The reason we can do so is that a new wrapper produced by merging two old wrappers tends to be more general and have a smaller (sometimes equal) distance to any existing

wrapper than the other two wrappers. The way we define distance in our system ensures that distance between wrappers will shrinks in a reasonable manner. By merging wrappers whose distance is less than a given threshold, we can avoid generating too many wrappers. The advantage of controlling the wrapper number is to reduce the extraction time when generated wrappers are used to extract records from new pages.

10.3.1.2 Record-Level Extraction

When a generated wrapper need to be used in extraction, it is supposed to extract data from a record not a whole page. In our system, a wrapper is able to match sub-DOM-tree corresponding to a record of a test page then extract data from matched sub-tree. When multiple wrappers can match a same sub-tree, our system will automatically select a wrapper with the best match (minimal distance to matched sub-tree) to perform the extraction.

10.3.2 Wrapper-Assisted Labeling

Applying previously generated wrappers to predict labels of similar records is the key point of our wrapper-assisted labeling strategy. This is done by performing a data extraction using existing wrappers. Of course, no data is actually extracted. They are just assigned with labels by underlying wrappers and displayed in the labeling interface. The main advantage of using wrapper-assisted labeling strategy is that it can avoid repeatedly labeling records with completely same internal structure and thus reduce the labeling effort. The amount of labeling effort it can save can be formalized as follows:

Suppose we need to label a group of pages from a same template. The total number of records is N_r. According to records' internal structures, all N_r records can be virtually partitioned into N_s subsets. Records in each subset have exactly same internal structure. Then, with wrapper-assisted labeling strategy, we only need to label one representative record for each subset and use generated wrapper to automatically assign labels for the rest records. Therefore, the maximal number of records we need to label is N_s. By observation, $N_s << N_r$ for most websites.

Wrapper-assisted labeling not only can reduce labeling effort but also can im-

prove the labeling quality by reducing the possibility of error. With more and more records are labeled, the underlying wrappers will keep updating and become more and more robust. In the meanwhile, these wrappers' ability in predicting labels is also being improved. As a matter of fact, after enough records are used to train the underlying wrappers, those wrappers seldom make mistakes in predicting labels for a new record.

10.3.3 Wrapper Performance Prediction

Another contribution of our system is the ability to predict the performance of generated wrappers. With such ability, users are able to know how many training pages are enough to generate a satisfactory wrapper set, thus to know when is a good time to stop labeling. To do such performance prediction, a test set is required. Pages in the test set should be also from the same template with those used to train the wrappers and they should be randomly selected from real dataset to make the prediction as accurate as possible. Good news is those test pages do not need to be labeled.

Given a test set $\mathcal{D}$ and generated wrapper set $\mathcal{W}$, the goal of wrapper performance prediction is to approximate the precision and recall values defined as follows:

$$Pre(\mathcal{D}, \mathcal{W}) = \frac{\sum_{p \in \mathcal{D}} \sum_{r_e \in p} Pre(r_e, \mathcal{W})}{\sum_{p \in \mathcal{D}} N_e(p)} \tag{10.1}$$

$$Rec(\mathcal{D}, \mathcal{W}) = \frac{\sum_{p \in \mathcal{D}} \sum_{r_e \in p} Rec(r_e, \mathcal{W})}{\sum_{p \in \mathcal{D}} N_t(p)} \tag{10.2}$$

where r_e indicates an extracted reocrd, p indicates a page, $Pre(r_e, \mathcal{W})$ and $Rec(r_e, \mathcal{W})$ are the precision and recall of $\mathcal{W}$ applied on r_e, $N_e(p)$ is the number of extracted records in a given page p, $N_t(p)$ is the number of total records in a given page p.

One assumption in our prediction method is that if a wrapper can perfectly match a record in a test page, the extraction accuracy for this record will be close to 100% in terms of precision and recall. Our experimental results show that this assumption works well for most cases. In our system, when a wrapper is applied

on a test page to match records and extract data, an aligning threshold $\theta \in [0, 1]$ is used to control the roughness of wrapper-record alignment. In extreme case, if θ is set to 1, a wrapper can map to any record; if θ is set to 1, only perfect mapping is allowed.

Bases on above assumption, if we set the aligning threshold to zero, $Pre(r_e, \mathcal{W})$ and $Rec(r_e, \mathcal{W})$ can be considered as constant 1. Thus, $Pre(\mathcal{D}, \mathcal{W})$ in Equation (10.1) will always be 100%. As shown in Equation (10.2), we also need to approximate the total number of records in the test set to evaluate $Rec(\mathcal{D}, \mathcal{W})$. This number is unknown since the test set is not labeled. By observation, we found that most pages from a same template normally contain same number of records. Therefore, we simply use the average number of records in labeled training pages, $\bar{N}$, to approximate the average number of records in each test page. Then Equation (10.2) can be written as:

$$Rec(\mathcal{D}, \mathcal{W}) = \frac{\sum_{p \in \mathcal{D}} N_e(p)}{|D| \cdot \bar{N}} \tag{10.3}$$

where $|\mathcal{D}|$ is the number of pages in test set $\mathcal{D}$ and $N_e(p)$ can be calculated by apply $\mathcal{W}$ to given page p.

One remaining question is how to decide the size of test set? Apparently, it should be much larger than the training set to make the prediction more accurate. According to our experience, less than 100 test pages are enough for most templates.

10.4 Experiments

We test the performance of our Pictor system through experiments.

10.4.1 Experiment Setup

Our dataset comes from 13 real-life large-scale websites as shown in Table 10.1. The first eight websites are online shopping websites and the following four are restaurant review websites. The last website is the ACM digital library portal. Among eight shopping websites, pages from the first seven sites are all list pages

and pages from Yahoo.com are all detail pages. Pages from four restaurant review websites and ACM portal are all detail pages.

For online shopping websites, the extraction schema contains three attributes, namely product name, product image, and product price. For restaurant review websites, their extraction schemas are separately defined. For ACM portal, we are interested in extracting docuemnt title, source (where is it published), author and their affiliations.

In our experiments, we use precision, recall, and F1 as measures in evaluation of data extraction results. These measures are calculated based on whether a labeled node is correctly extracted.

All experiments were run on a PC, with a 3.06 GHz Pentium 4 processor and 2.0 GB RAM.

10.4.2 Experimental Results

Experiment I: Labeling Effort

In this experiment, we try to show how much labeling effort can be saved by using our wrapper assisted labeling strategy. Instead of counting the labeling time used by different labeling methods, we implement this comparison in a differently way. In traditional labeling methods, all records in training pages need to be labeled. In our system, we only need to label those representative records and let the system automatically label the rest. Therefore, the labeling effort in traditional methods is proportion to the total number of records in training pages. On the contrary, the labeling effort in our system is proportion to the number of records which are actually manually labeled. By compare these two different record numbers, we can somehow show much labeling effort can be saved in our system.

Comparison results of seven online shopping websites are listed in Figure 10.6.

As shown in Figure 10.6, the average record number of all seven websites are 16.1 while the average manually labeled record number are 1.68. Therefore, our system roughly saves 89.61% labeling cost compared to traditional labeling methods.

Note that the impact of wrapper performance prediction is not taken into account in this experiment, which means our system can actually save more time

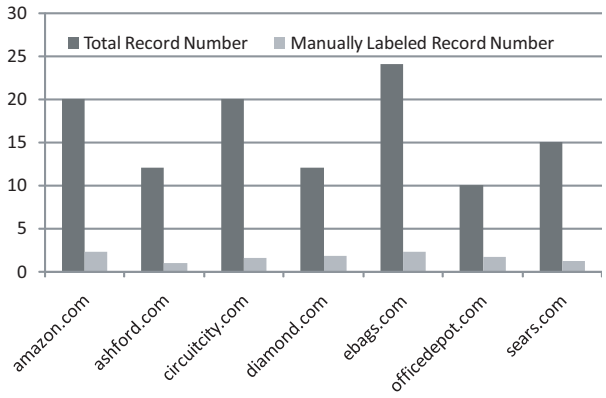


Figure 10.6: Experiment one

than the results listed here because traditional labeling methods do not have a clear idea about how many pages should be labeled and tend to label more pages than necessary.

Experiment II: Extraction Quality

This experiment is designed to show the extraction accuracy of generated wrappers. Table 10.1 shows the evaluation results for each site. As we can see, the average F1 is as high as 99%. This is good enough for most real applications.

Although our wrapper performance prediction model is able to tell us when to stop during the labeling process for each site, in this experiment, we still label all pages for evaluation purpose. The page number listed in Table 10.1 is the number of page labeled for each site when the prediction model considers they are enough for training.

Experiment III: Wrapper Performance Prediction

This experiment is implemented to demonstrate the accuracy of our wrapper performance prediction module. We choose two websites, amazon.com and yahoo.com, to do this experiment. The reason we choose these two websites is that they are more complex than other websites and we want to show our wrapper performance prediction model works well in both list pages (amazon.com) and detail pages (yahoo.com).

Table 10.1: Results on 12 websites

Website	**Page #**	**Pre.**	**Rec.**	**F1-Value**
amazon.com	23	0.9700	0.9300	0.9496
ashford.com	12	1.0000	1.0000	1.0000
circuitcity.com	19	1.0000	1.0000	1.0000
diamond.com	12	0.9875	0.9975	0.9925
ebags.com	19	0.9976	1.0000	0.9988
officedepot.com	19	0.9850	1.0000	0.9924
sears.com	27	0.9960	1.0000	0.9980
yahoo.com	14	1.0000	1.0000	1.0000
dine.com	8	1.0000	1.0000	1.0000
menupages.com	5	1.0000	1.0000	1.0000
restaurant.com	10	1.0000	1.0000	1.0000
wcities.com	5	1.0000	1.0000	1.0000
portal.acm.org	4	1.0000	1.0000	1.0000
Average	13.6154	0.9951	0.9944	0.9947

As discussed early, wrapper performance prediction requires a test set. In this experiment, we prepare 100 test pages for each site. For evaluation and demonstration purpose, all test pages are also labeled. Then, for each site, we label training pages one by one. Every time when one more page is labeled, it is used to update underlying wrappers. Based on current status, a predicted precision value and a recall value are reported using the provided test set. And the corresponding actual precision and recall values are also calculated by using the same test set.

In Figure 10.7 we plot the comparison between predicted values and actual values for these two websites. Note that the predicted precision values are always 1.

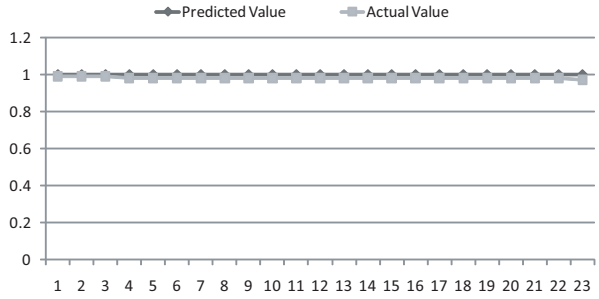
Two conclusions can be drawn from these results. 1) The assumption is true that wrappers can extract perfectly aligned records with accuracy close to 100%. 2) The more training samples are, the more accurate our wrapper performance prediction module is.

Algorithm 9 Incremental Wrapper Update

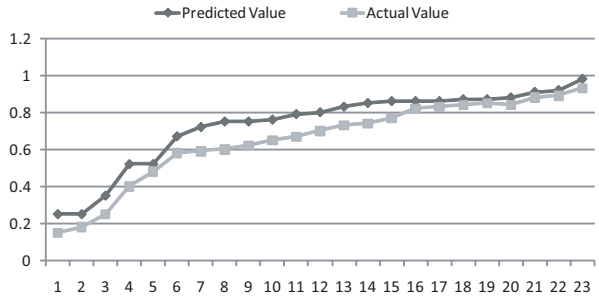
Input:
sub-DOM-tree t of a record
existing wrapper set $\mathcal{W} = \{w_1, w_2, \ldots, w_N\}$
N: number of wrappers in $\mathcal{W}$
θ: distance threshold (set to 0.2 in our system)
Ψ^*: minimal distance
$w^* \in \mathcal{W}$: wrapper with minimal distane with t

begin wrapper updating procedure
$\Psi^* = \inf$
for $i = 1$ **to** N
    calculate the distance between t and w_i: $\Psi(t, w_i)$
    if $\Psi(t, w_i) < \Psi^*$
        $\Psi^* = \Psi(t, w_i)$
        $w^* = w_i$
    end if
end for
if $\Psi^* < \theta$
    update wrapper w^*: $w^* \leftarrow w^* + t$
    begin wrapper merging sub-procedure
    flag = **true**
    while flag = **true do**
        flag = **false**
        for $i = 1$ **to** N
            if $w^* \neq w_i$
                calculate $\Psi(w^*, w_i)$
                if $\Psi(w^*, w_i) < \theta$
                    merge w^* and w_i: $w_i \leftarrow w_i + w^*$
                    $\mathcal{W} = \mathcal{W} - \{w^*\}$
                    $N = N - 1$
                    $w^* = w_i$
                    flag = **true**
                end if
            end if
        end for
    end while
else
    generate a new wrapper w': $w' \leftarrow t$
    $\mathcal{W} = \mathcal{W} \cup \{w'\}$
    $N = N + 1$
end if

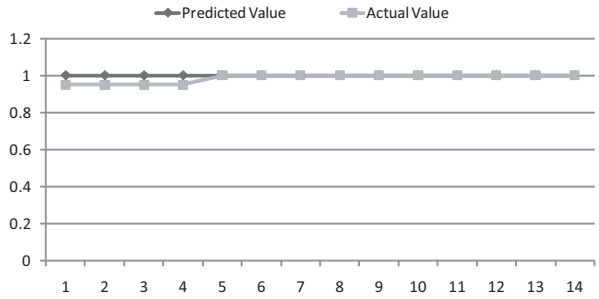
Output: $\mathcal{W}$



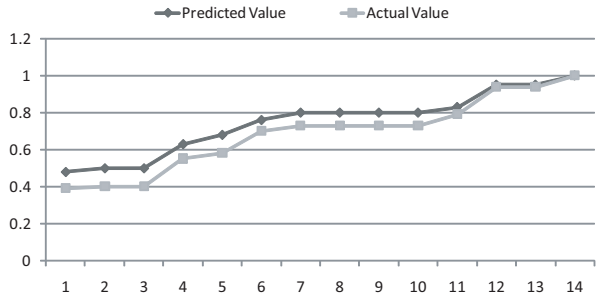
(a) Precision (Amazon)



(b) Recall (Amazon)



(c) Precision (Yahoo)



(d) Recall (Yahoo)

Figure 10.7: Experiment Three

Chapter 11

Conclusion

In this dissertation, we proposed a few methods in the area of web crawling and web information extraction.

We discuss the problem of seed selection for a web-scale crawler. We propose a graph-based framework for seed selection, analyze the complexity of the problem, and propose several approximate seed selection algorithms within this framework. Experimental results on a dataset of 2000 real web sites demonstrate that our algorithms significantly outperform heuristic seed selection approaches. One of the surprising conclusions of our work is that using higher level graph properties such as strongly connected components and their relationships is not proved to be beneficial for seed selection.

The heuristic baseline used in our experiments are relatively simple and naive. To further validate our approach, future work can be done to conduct comparison with more complicated approximation algorithms proposed in related works ([37, 38, 39]).

For URL ordering problem, we propose a supervised method which relies on the anchor text of each URL and guides the crawler based on the likelihood of each anchor text leading to relevant documents. The experimental results on 18 academic websites shows that, given pre-trained domain knowledge, our method is more effective than bread-first approach.

We also proposed a template-independent news extraction approach based on visual consistency. We first represent a page as a visual block tree. Then, by extracting a series of visual features, we can derive a composite visual feature set

that is stable in the news domain. Finally, we use a machine learning approach to generate a vision-based wrapper. Once a V-Wrapper is learned, it is able to deal with all pages of the domain without re-training. Thus, our approach saves a lot of time and money for wrapper maintenance. Experimental results demonstrate that the generated V-Wrapper not only has a beneficial domain compatibility, but also can achieve extraction accuracy of as much as 95% in terms of F1-value. As future work, we plan to broaden our application to other domains.

This work addresses the problem of extracting author's meta-data from their homepages. We sufficiently utilizes visual features and applies an inter-fields probability model to capture the relation among different fields. Experimental results demonstrate that utilizing visual features and applying inter-fields probability model can significantly improve the extraction accuracy.

In the area of template-dependent information extraction, we propose a novel wrapper induction system that expresses a different opinion regarding the relation between template detection and wrapper generation. Our system takes a miscellaneous training set as input and conducts template detection and wrapper generation in a single step. By the criterion of generated wrappers' extraction accuracy, our approach can achieve a joint optimization of template detection and wrapper generation. Experimental results on real-life shopping websites prove the feasibility and effectiveness of our approach. The preliminary comparison demonstrates that our approach significantly outperforms the separated template detection strategy.

An extension of the work above is a record-level wrapper induction system which is able to effectively extract records and identify their internal semantics at the same time. Compared to traditional page-level wrapper methods, the proposed approach not only saves a lot of effort made in manually labeling but also performs data extraction more efficiently. Experimental results on 16 real-life websites from four different domains demonstrate 99% extraction accuracy in terms of F1-Value.

Lastly, we designed a wrapper labeling and training system, called *Pictor*, which is able to minimize the labeling cost as well as obtaining high extraction accuracy. Our system generates record-level wrappers in contrast to traditional page-level wrapper induction methods using a labeling strategy called "*wrapper-assisted labeling*" which utilized initially generated wrappers to help label more records of a partially labeled page or a totally new page. Our wrapper performance prediction

module is used to decide how many training pages are enough to generate a satisfactory wrapper set. Experimental results upon 13 websites indicates that our performance prediction module works well.

Bibliography

[1] Pant, G., P. Srinivasan, and F. Menczer (2004) "Crawling the web," *Web Dynamics*, pp. 153–178.

[2] Broder, A., R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener (2000) "Graph structure in the Web," *Computer Networks*, **33**(1-6), pp. 309–320.

[3] Laender, A. H. F., B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira (2002) "A brief survey of web data extraction tools," *SIGMOD Record*, **31**(2), pp. 84–93.

[4] Liu, B. (2005) "Web Content Mining (Tutorial)," in *Proceedings of the 14th International Conference on World Wide Web*.

[5] Han, H., C. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. Fox (2003) "Automatic document metadata extraction using support vector machines," in *Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*, IEEE Computer Society Washington, DC, USA, pp. 37–48.

[6] Nie, Z., J. Wen, and W. Ma (2007) "Object-level vertical search," in *To appear by the Third Biennial Conference on Innovative Data Systems Research (CIDR)*.

[7] Hammer, J., H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha (1997) "Extracting Semistructured Information from the Web," in *Proceedings of the Workshop on Management fo Semistructured Data*.

[8] Kushmerick, N., D. S. Weld, and R. B. Doorenbos (1997) "Wrapper Induction for Information Extraction," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 729–737.

[9] Hsu, C.-N. and M.-T. Dung (1998) "Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web," *Information Systems, Special Issue on Semistructured Data*, **23**(8), pp. 521–538.

[10] Muslea, I., S. Minton, and C. Knoblock (1999) "A hierarchical approach to wrapper induction," in *Proceedings of the 3rd Annual Conference on Autonomous Agents*, pp. 190 – 197.

[11] Liu, L., C. Pu, and W. Han (2000) "XWRAP: an XML-enabled wrapper construction system for Web information sources," in *Proceedings of the 16th International Conference on Data Engineering*, pp. 611–621.

[12] Chang, C.-H. and S.-C. Lui (2001) "IEPAD: information extraction based on pattern discovery," in *Proceedings of the 10th International Conference on World Wide Web*, pp. 681–688.

[13] Crescenzi, V., G. Mecca, and P. Merialdo (2001) "RoadRunner: Towards Automatic Data Extraction from Large Web Sites," in *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 109 – 118.

[14] Cohen, W. W., M. Hurst, and L. S. Jensen (2002) "A flexible learning system for wrapping tables and lists in HTML documents," in *Proceedings of the 11th International Conference on World Wide Web*, pp. 232 – 241.

[15] Reis, D. C., P. B. Golgher, A. S. Silva, and A. F. Laender (2004) "Automatic web news extraction using tree edit distance," in *Proceedings of the 13th International Conference on World Wide Web*, pp. 502 – 511.

[16] Chuang, S.-L. and J. Y.-j. Hsu (2004) "Tree-Structured Template Generation for Web Pages," in *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 327 – 333.

[17] Zhao, H., W. Meng, Z. Wu, V. Raghavan, and C. Yu (2005) "Fully automatic wrapper generation for search engines," in *Proceedings of the 14th International Conference on World Wide Web*, pp. 66 – 75.

[18] Hogue, A. and D. Karger (2005) "Thresher: automating the unwrapping of semantic content from the World Wide Web," in *Proceedings of 14th International Conference on World Wide Web*, pp. 86 – 95.

[19] Crescenzi, V., G. Mecca, and P. Merialdo (2002) "Wrapping-oriented classification of web pages," in *Proceedings of the 2002 ACM symposium on Applied computing*, pp. 1108 – 1112.

[20] Zheng, S., R. Song, D. Wu, and J.-R. Wen (2007) "Joint Optimization of Wrapper Generation and Template Detection," in *SIGKDD-2007*, pp. 894–902.

[21] Wang, J. and F. H. Lochovsky (2003) "Data extraction and label assignment for web databases," in *Proceedings of the 12th International Conference on World Wide Web*, pp. 187 – 196.

[22] Liu, B., R. Grossman, and Y. Zhai (2003) "Mining Data Records in Web Pages," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 601 – 606.

[23] Arasu, A. and H. Garcia-Molina (2003) "Extracting structured data from Web pages," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 337 – 348.

[24] Diligenti, M., F. Coetzee, S. Lawrence, C. Giles, and M. Gori (2000) "Focused crawling using context graphs," in *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 527–534.

[25] Vidal, M., A. da Silva, E. de Moura, and J. Cavalcanti (2006) "Structure-driven crawler generation by example," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM New York, NY, USA, pp. 292–299.

[26] Cho, J., H. Garcia-Molina, and L. Page (1998) "Efficient crawling through URL ordering," *Computer Networks and ISDN Systems*, **30**(1-7), pp. 161–172.

[27] Najork, M. and J. Wiener (2001) "Breadth-first crawling yields high-quality pages," in *Proceedings of the 10th international conference on World Wide Web*, ACM Press New York, NY, USA, pp. 114–118.

[28] Chakrabarti, S., M. v. d. Berg, and B. Dom (1999) "Focused crawling: a new approach to topic-specific Web resource discovery," in *Proceeding of the eighth international conference on World Wide Web*, Elsevier North-Holland, Inc., Toronto, Canada, pp. 1623–1640.

[29] Ben-Shaul, I., M. Herscovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim, V. Soroka, and S. Ur (1999) "Adding support for dynamic and focused search with Fetuccino," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, **31**(11-16), pp. 1653–1665.

[30] Menczer, F., G. Pant, P. Srinivasan, and M. Ruiz (2001) "Evaluating topic-driven web crawlers," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM New York, NY, USA, pp. 241–249.

[31] Menczer, F. (2003) "Complementing search engines with online web mining agents," *Decision Support Systems*, **35**(2), pp. 195–212.

[32] Pant, G. and F. Menczer (2002) "MySpiders: Evolve Your Own Intelligent Web Crawlers," *Autonomous Agents and Multi-Agent Systems*, **5**(2), pp. 221–229.

[33] Srinivasan, P., J. Mitchell, O. Bodenreider, G. Pant, and F. Menczer (2002) "Web crawling agents for retrieving biomedical information," in *Proc. Int. Workshop on Agents in Bioinformatics (NETTAB-02)*.

[34] Menczer, F. (1997) "ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery," in *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, MORGAN KAUFMANN PUBLISHERS, INC., pp. 227–235.

[35] Davison, B. (2000) "Topical locality in the Web," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, ACM New York, NY, USA, pp. 272–279.

[36] Menczer, F., G. Pant, and P. Srinivasan (2004) "Topical web crawlers: Evaluating adaptive algorithms," *ACM Transactions on Internet Technology (TOIT)*, **4**(4), pp. 378–419.

[37] Wang, X., G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill (2003) "Integrated coverage and connectivity configuration in wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ACM, pp. 28–39.

[38] Gupta, H., S. Das, and Q. Gu (2003) "Connected sensor cover: self-organization of sensor networks for efficient query execution," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, ACM, pp. 189–200.

[39] Zhou, Z., S. Das, and H. Gupta "Connected k-coverage problem in sensor networks," in *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, IEEE, pp. 373–378.

[40] Dasgupta, A., A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins (2007) "The Discoverability of the Web," in *Proceedings of the 16th international conference on World Wide Web*, ACM Press New York, NY, USA, pp. 421–430.

[41] Sarawagi, S. (2002) "Automation in information extraction and data integration (Tutorial)," in *Proceedings of the 28th International Conference on Very Large Data Bases*.

[42] Willett, P. (1988) "Recent trends in hierarchic document clustering: a critical review," *Information Processing and Management*, **24**(5), pp. 577–597.

[43] Zhai, Y. and B. Liu (2005) "Web data extraction based on partial tree alignment," in *Proceedings of 14th International Conference on World Wide Web*, pp. 76 – 85.

[44] Kovacevic, M., M. Diligenti, M. Gori, and V. Milutinovic (2002) "Recognition of Common Areas in a Web Page Using Visual Information: a possible application in a page classification," in *Proceedings of the 2002 IEEE International Conference on Data Mining*, pp. 250 – 257.

[45] Song, R., H. Liu, J.-R. Wen, and W.-Y. Ma (2004) "Learning block importance models for web pages," in *Proceedings of the 13th International Conference on World Wide Web*, pp. 203 – 211.

[46] Yin, X. and W. S. Lee (2004) "Using link analysis to improve layout on mobile devices," in *Proceedings of the 13th International Conference on World Wide Web*, pp. 338–344.

[47] ——— (2005) "Understanding the function of web elements for mobile content delivery using random walk models," in *Special interest tracks and posters of the 14th International Conference on World Wide Web*, pp. 1150–1151.

[48] Garey, M., D. Johnson, et al. (1979) *Computers and Intractability: A Guide to the Theory of NP-completeness*, WH Freeman San Francisco.

[49] Hochbaum, D. and A. Pathria (1998) "Analysis of the Greedy Approach in Problems of Maximum k-Coverage," *Naval Research Logistics*, **45**(6), pp. 615–627.

[50] Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001) *Introduction to Algorithms (2nd ed.)*, MIT Press and McGraw-Hill.

[51] Tarjan, R. (1972) "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, **1**, p. 146.

[52] Olston, C. and M. Najork (2010) "Web Crawling," *Foundations and Trends in Information Retrieval*, **4**(3), pp. 175–246.

[53] "http://www.nltk.org/," .

[54] Freund, Y. and R. E. Schapire (1997) "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, **55**(1), pp. 119–139.

[55] "http://www.w3.org/DOM/," .

[56] Zheng, S., M. R. Scott, R. Song, and J.-R. Wen (2008) "Pictor: An Interactive System for Importing Data from a Website," in *SIGKDD-2008*, pp. 1097–1100.

[57] Nie, Z., J.-R. Wen, and W.-Y. Ma (2007) "Object-level Vertical Search," in *CIDR-2007*, pp. 235–246.

[58] Wikipedia (2007), "RSS," [Online; accessed 20-October-2007]. URL `http://en.wikipedia.org/w/index.php?title=RSS&oldid=164714605`

[59] Crescenzi, V., G. Mecca, and P. Merialdo (2001) "Automatic Web Information Extraction in the RoadRunner System," in *Proceedings of International Workshop on Data Semantics in Web Information Systems (DASWIS-2001) in conjunction with 20th International Conference on Conceptual Modeling (ER 2001)*.

Vita

Shuyi Zheng

Shuyi Zheng was born in Zhoushan, Zhejiang Province, China. He enrolled in the Ph. D. program in Computer Science and Engineering at The Pennsylvania State University in 2006. Prior to that, he received a B.A degree in 2002 and a M.E degree in 2005 from Tsinghua University, China. His research work is mainly focused on web crawling and information extraction.