Comparative Mining of B2C Web Sites by Discovering Web Database Schemas

By

Bindu Peravali

A Thesis

Submitted to the Faculty of Graduate Studies through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

2016

Comparative Mining of B2C Web Sites by Discovering Web Database Schemas

By

Bindu Peravali

APPROVED BY:

Dr. Zhiguo Hu, External Reader

Department of Mathematics & Statistics

Dr. Subir Bandyopadhyay, Internal Reader

School of Computer Science

Dr. Christie I. Ezeife, Advisor

School of Computer Science

09/09/2016

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Discovering potentially useful and previously unknown historical knowledge from heterogeneous E-Commerce (B2C) web site contents to answer comparative queries such as “list all laptop prices from Walmart and Staples between 2013 and 2015 including make, type, screen size, CPU power, year of make”, would require the difficult task of finding the schema of web documents from different web pages, extracting target information and performing web content data integration, building their virtual or physical data warehouse and mining from it. Automatic data extractors (wrappers) such as the WebOMiner system use data extraction techniques based on parsing the web page html source code into a document object model (DOM) tree, traversing the DOM for pattern discovery to recognize and extract different web data types (e.g., text, image, links, and lists). Some limitations of the existing systems include using complicated matching techniques such as tree matching, non-deterministic finite state automata (NFA), domain ontology and inability to answer complex comparative historical and derived queries.

This thesis proposes building the WebOMiner_S which uses web structure and content mining approaches on the DOM tree html code to simplify and make more easily extendable the WebOMiner system data extraction process. We propose to replace the use of NFA in the WebOMiner with a frequent structure finder algorithm which uses regular expression matching in Java XPATH parser with its methods to dynamically discover the most frequent structure (which is the most frequently repeated blocks in the html code represented as tags < div class = “ ″ >) in the Dom tree. This approach eliminates the need for any supervised training or updating the wrapper for each new B2C web page making the approach simpler, more easily extendable and automated. Experiments show that the WebOMiner_S achieves a 100% precision and 100% recall in identifying the product records, 95.55% precision and 100% recall in identifying the data columns.

Key Words: Web Content Mining, Automatic Web Data Extraction, Data integration, Wrappers

DEDICATION

This thesis is dedicated to my father Koteswara Rao peravali, mother Jayalakshmi Peravali and my brother Gourinath Peravali. Without their patience, understanding, support, and most of all love, the completion of this work would not have been possible.

ACKNOWLEDGEMENT

I would like to give my sincere appreciation to all of the people who have helped me throughout my education. I express my heartfelt gratitude to my parents and brother for their support throughout my graduate studies.

I am very grateful to my supervisor, Dr. Christie Ezeife for her continuous support throughout my graduate study. She always guided me and encouraged me throughout the process of this research work, taking time to read all my thesis updates and for providing financial support through research assistantship.

I would also like to thank my external reader, Dr. Zhinguo Hu, my internal reader, Dr. Subhir Bandhyopadhay, and my thesis committee chair, Dr. Dan Wu for making time to be in my thesis committee, reading the thesis and providing valuable input. I appreciate all your valuable suggestions and the time, which have helped improve the quality of this thesis.

Finally, I express my appreciations to all my friends and colleagues at University of Windsor, especially vignesh aravindan and sravya vangala for their support and encouragement. Thank you all!

Table of Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The World Wide Web is the biggest and most widely used information source that is easily accessible (Liu, 2007). It is essential to provide tools for effective knowledge discovery. It has numerous amounts of web pages that keep on updating and some gets deleted as well. Data is heterogeneous in nature, several types of data such as text, image, video, and audio are within the web.

Web has emerged as one of the important medium for business. E-commerce has evolved as the most potential business area. Huge volume of products are being sold online, where each vendor has his own web site to show case his products. We have identified the necessity of a comparative agent for such product web pages which not only compares current data but also historical. Such agents already exist in the literature but none of them are totally automatic even if some of them are automatic each has its own downside.

The main intuition of the thesis is to develop a scalable and robust approach for automatic web data extraction and storage for further analysis.

1.1 Web Mining:

Web mining can be stated as the application of data mining techniques on the web (Cooley, Mobasher, & Srivastava, 1997). It is performed to discover unknown knowledge existing on web data. Also to provide some value added services to the users. Containing large volumes of data and its versatility, interest's lots of researchers on performing several mining techniques for knowledge discovery.

Web mining is categorized into 3 areas as given below

1) Web Structure mining

2) Web usage mining

3) Web content mining

1.1.1 Web Structure Mining:

Mining structured hyperlinks within the web is termed as Web Structure Mining (Kosala & Blockeel, 2000). Web structure mining involves Mining the web document structure, analysis of the tree-like structure of page structures to describe HTML or XML tag usage.

This area of research also involves study of World Wide Web structure as a whole like social networks and citation analysis etc. Considering web as a directed graph, the analysis of in-links and out-links to a page is the key to search engines. Traditional data mining techniques does not perform the link analysis as they are stored as relational databases no link structure exists. The structure of a web site can be obtained by analyzing the link structure between pages (Kosala & Blockeel, 2000). It also involves web page categorization and discovering communities on the web.

1.1.2 Web Usage Mining:

The web usage mining focuses on techniques that could present the user behavior while the user interacts with the web (Srivastava, Cooley, Deshpande, & Tan, 2000). Such data is stored on the webservers and log files this information can be mined for identifying the user interests and further build recommendations accordingly. But the problem arises in identifying individual user and their sessions.

There are 3 main tasks for performing the web usage mining, Preprocessing, Pattern discovery, and Pattern analysis (Cooley, Mobasher, & Srivastava, 1997). Data can be embedded into a web page by anybody there is no any restrictions, this flexibility gives way to incomplete and in appropriate data which makes the task of pattern discovery difficult. Hence preprocessing the web data before performing any task is must. Pattern discovery is performed on web usage data by using several machine learning techniques and statistical methods, pattern recognition along with data mining. Some of the major approaches involves association rule mining, clustering, classification, sequential patterns and dependency modeling (Srivastava, Cooley, Deshpande, & Tan, 2000). Such discovered patterns can be analyzed for interesting patterns using some already existing forms such as sql and olap depending up on the data.

1.1.3 Web content mining:

Mining the actual web page contents for knowledge is termed as web content mining (Kosala & Blockeel, 2000). This area of web mining involves several data mining techniques. Data in traditional data mining is typically available in relational data tables but data in web is not homogeneous also it does not exist in a single repository, it is

distributed in large scale over numerable servers all over the world. Task of performing web content mining involves gathering relevant information from the World Wide Web.

General Challenges in web content mining:

1) Web data is heterogeneous in nature, several kinds of data such as text, image, audio and video exist on web.

2) Web data is dynamic, whenever an expensive mining task is performed on web data now or then the data on web gets updated and as a result the same task has to be performed again. This scenario gets worse when the data gets updated frequently, unfortunately this is a common scenario in web. Introduction:

1.2 Types of Web Pages

Data on the World Wide Web has been classified by many researchers into 3 types. The unstructured data, the semi structured and the Structured (Kosala & Blockeel, 2000). As the web is not strict anybody can represent their content in any way they wanted. However based on the domain the website / web page belongs to, it follows a similar structure with the other web pages belonging to the same domain. For example all the web pages that belong the news columns will project their content to the readers in a similar format. The type which the web page belong to matters, because wrappers generated to extract the content on the web page highly depends on the type of web page and its structure. Here, we discuss the different type of web pages.

1.2.1 The Unstructured Web Page:

The unstructured type of web page addresses the pages that do not follow any structure also called free text documents. These type of web pages publish their content such as text, images, multimedia in a free manner. Example of this documents are the news blogs and personal blogs etc. Blog writers

place their content according to their desire and no structure is followed necessarily.

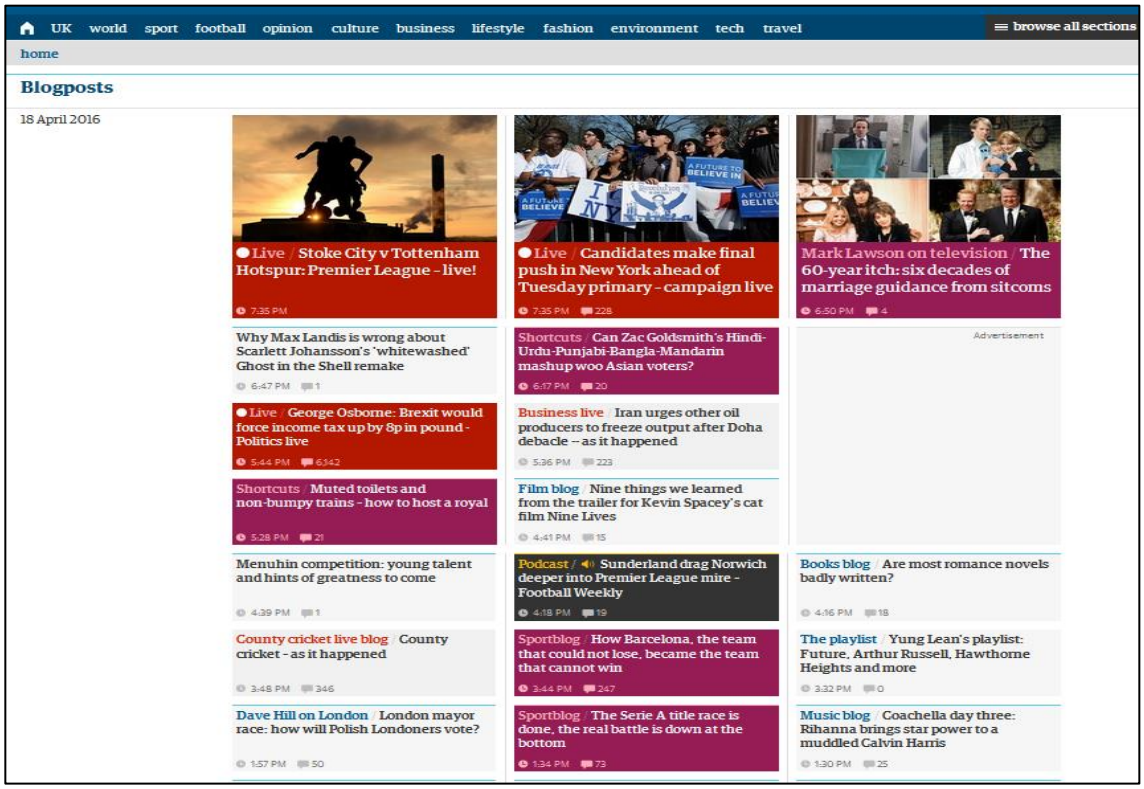


Figure 1: Unstructured web page

On such web pages no template based web data extraction can be applied and these pages are usually represented as a bag of words or phrases which can be mined using several data mining techniques.

1.2.2 Semi-structured/Structured Web Pages

The semi- structured/Structured web pages are the type of pages that follows a defined structure which are attained due to the structured data source like a database or table. While data on the webpage is being published it will have a format and is aligned. A product rich page from an e-commerce website like bestbuy.ca is the example of structured web page since all the information such as product details are essentially stored in a database which are published on to a webpage following a strict template. Such data can be extracted using wrappers by generating the extraction rules manually or automatically based on the target data. Again in structured web page there are

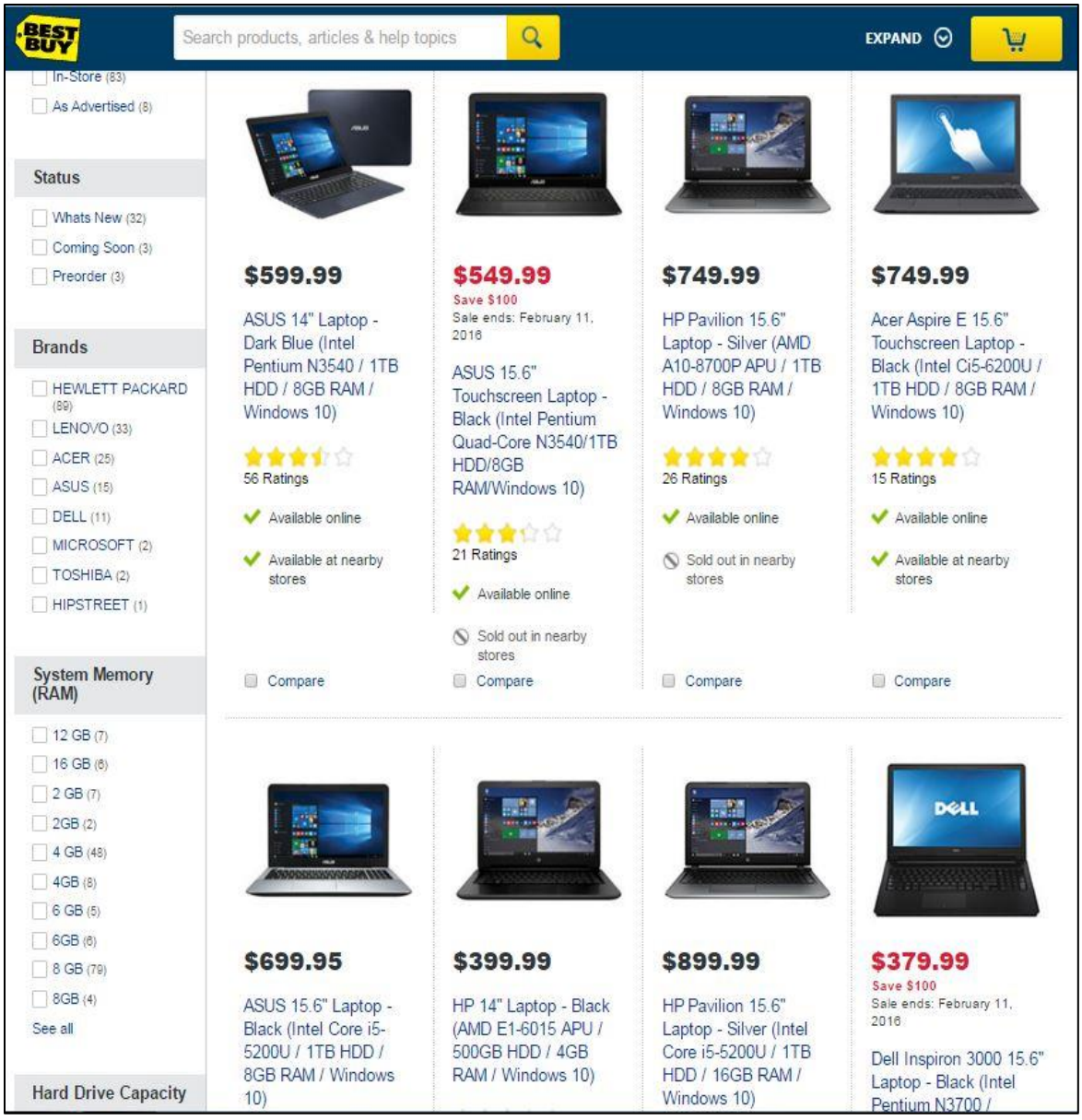


Figure 2: Structured web page

two classifications, the List Page and the detail page. Which are formed due to the type of template used to generate them.

1.2.2.1 The List Web Page

This type of web page consists of several structured data records. For example, e-commerce websites display all the products belonging the same category in list or a grid format in their web page. Considering the object oriented data model treating a single product as an object, a list web page will have a set of objects that belong to the same type with a set of attributes that all the objects

may or may not be associated with. Figure highlights a web page from bestbuy.ca, it is a product web page which is a list type of page having 8 objects displayed with several attributes such as price, brand, size, color, processor, hard disk, ram and operating system. Also object 2, 8 have an attribute discount which none of the other objects in the page has. It is meta data of the product which can be stored into a data warehouse from historical querying and comparative analysis .This page also has other data such as header, navigational data, advertisements (Currently not visible) and footer (currently not visible), all this information has to be ignored and only the target objects has to be extracted by the wrapper. This Figure is an example of our input web page for our system WebOMiner_S performing the data extraction.

Figure 3 : Detail Page

1.2.2.2 The Detail web page

Detail web page displays only one object with all of its attributes. The Figure 4 showcases a detail page from an e-commerce website chaptersindigo.ca. The object is a book and the detail page displays all the attributes of this object such as name of the book, price, format, dimensions, published on ,Published by , language etc. All this can be extracted to store into a data warehouse for value added services.

1.3 Why List Page

We have chosen to mine a list page instead of a detail page because of the following reasons:

1. List pages provide with the most important attributes of the object in the customers perspective whereas the detail pages contains all the attributes of the object. As our main objective is to develop a system that can perform a comparative mining and historical querying as value added service for the customers.
2. Our observation is that the detail page contains more noise than the list page, on Considering the recommendations and advertisements. As we work towards eliminating noisy data in our data ware house we prefer to mine less noisy web pages.
3. It is very resource consuming to extract data of an object and its attributes from the web page then move to another page to extract another object and its attributes when all the objects belong to the same type and while we can extract multiple objects from single page in the list type.

1.4 Understanding the html source code behind the webpage

Before constructing any wrapper to extract the data on a web page understanding the html source code helps in building the better ones. HTML stands for Hyper Text Markup Language, it is a language that is a set of markup tags which helps in creating a web page. The HTML documents are described by using finite set of html tags where each tag has a meaning and usage purpose, typically to display document contents based on the user's requirement. It is the web browsers responsibility to process the HTML document and display the content. A typical html file starts by declaring its <!DOCTYPE> which helps the browser to display the web page correctly. By declaring the doctype to HTML i.e., <!DOCTYPE HTML> it understands that the file is a html file and processes the tags to display contents appropriately. The start of the html code is initiated by the tag<HTML> and the end by </HTML>, thus the html tags are paired but there are few singleton tags as well. Some of the html tags include<BODY>, <HEAD> for the browser to understand what comprises of the header section and what's in the body. Other tags helps to understand how the content should be displayed <

FONT>, <COLOR>, < H1>, <H2>, <IMG>, <A>, < BR> etc., these are called inline tags. Tags like <DIV>, <TABLE>, <OL>, <FORM>, <LI>, <TR>, <TD> helps in grouping, organizing and sectioning the web page also called block level tags according to w3consortium. Figure shows a sample html source code and how it will be rendered by the browser as a web page.

```
<!Doctype html>
<html>
 <head>
    <title> sample</title>
 </head>
 <body>
    <p>this is a </p>
     <h1> Sample page </h1>
  </body>
</html>
```

Figure 4: Sample Html Source Code

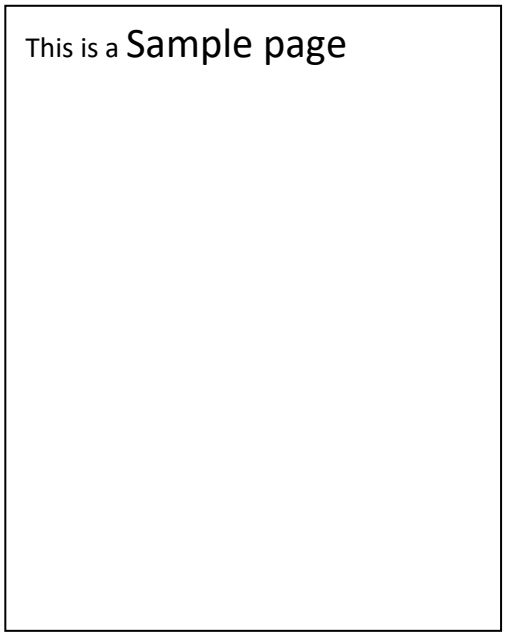


Figure 5: Web page content for the sample html code

1.5 Building DOM Tree:

Dom stands for document object model it has become a necessary step for many data extraction algorithms. Dom tree is constructed using the html source code of the web page. The tags and their hierarchies form the parent child relationship explaining how the content is organized in the page. This insight allows the researchers to build wrappers for data extraction. DOM trees can be built by using 2 methods.

1) Using tags alone 2) Using both tags and visual cues

1.5.1 Using Tags Alone:

Most of the html tags occur in pairs with an open tag and a closed tag. Within such pairs there can be other pair of tags, giving rise to nested structure. A DOM tree for an html page is built by considering each pair of tag as a node.

Html is not a strict language, missing or incomplete tags may be encountered so there a necessity to properly embed the tags in order to derive a correct DOM tree. This gives rise

to the preprocessing task called data cleaning. Followed by building a DOM tree from the clean html file.

1.5.2 Using Both Tags and visual cues:

Relying on the visual information (location of tags on the screen) to infer the structural relationship among tags for constructing a DOM tree is another method. Nested rectangles are used to interpret the tag structure. This is performed by finding the 4 boundaries of a rectangle of each html element by calling the rendering engine of a browser. Containment check is performed to see if one rectangle is contained in another, this interprets the nested structure of html tags.

1.6 Cascading style sheets:

The cascading style sheets helps the web developers to layout the information on the web page. It consists of the information on how the html elements should be displayed on the web page. A cascading style sheet can be created to define styles for the webpages, including the design, layout and variations in display. All the B2C websites uses the external css to get the look for their webpages. An external css contains several styles each identified by a unique class name. The class attribute is referred in the html tag to which the preferred style has to be allocated. Figure given an example, an external css and its usage in the html source code. Figure shows the sample css file with two styles indicated by two classes' prodlook and prodprice. The first style prodlook describes that the content of any block that uses this class should set the font weight to be bold, the content should maintain a left margin of 20 points. While the second style prodprice describes that the content in any block that refers this class should be assigned of font sans_serif, with a color red and a background of green. Second figure shows their usage in the html code using the class attribute.

```
div.prodlook{
   font-weight:bold;
   margin-left:30px}
div.prodprice{
   font: sans-serif;
   color:red;
   background:green}
```

```
<div class="prodlook">…….</div>
<div class="prodprice">…..</div>
```

Figure 6: Refering css stylr using Class attribute

Figure 7: Sample css

1.7 Xml and Xpath Parser:

Similar to HTML, XML is also a markup language and is a w3c recommendation. While HTML has a predefined set of tags xml allows users to define their own self descriptive because of which it is not only machine readable but also human readable. XML is designed to store and transport data and exclusively used in IT systems. The major difference between the HTML and XML is that the former one is designed to display data while the later one is to carry data. XPath (the XML Path language) is a language for finding information in an XML document. XPath is used to navigate through elements and attributes in an XML document. It uses path expressions to navigate in XML documents. Also, it contains a library of standard functions. XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system. Some of the sample expressions are /-> selects from the root node//-> Selects nodes in the document from the current node that match the selection no matter where they are. .-> Selects the current node , ..-> Selects the parent of the current node, @->Selects attributes. It also has wild cards, XPath wildcards can be used to select unknown XML nodes.* ->Matches any element node, @* -> Matches any attribute node and node()-Matches any node of any kind. It provides with some methods to retrieve information from the xml document such as, xpath evaluate() method to return specific data from xml file that matches the xpath expression. It can take as input an xml document and returns its nodes as a nodelist. getlength() method to get the count of no of matching nodes to the path expression in the xml.

We have made use of all these functional features in our system to make the data extraction much easier.

1.8 Thesis Contributions:

1) We propose to replace the use of the NFA in step 4 of the WebOMiner system with a frequent structure finder (FSfinder) algorithm using both web content and structure mining. The FSfinder uses the Java xpath parser to summarize the frequency of each tag occurrence in the web html code, retaining only tags that meet a certain minimum occurrence (e.g. 3), then for each tag, it uses the web page DOM tree to find the tag's block and retrieve the most frequently used block. This most frequently used block is then marked as the data block and its data region/table name is retrieved with its schema.

2) Schema can be dynamically discovered for any given web page using our technique while with the WebOMiner, to add and learn about a new or re-structured B2C web page, the NFAs need to be refreshed so that it can recognize new features not previously in its structure. Our proposed system uses regular expression matching in the Java xpath parser through its methods such as compile(), evaluate() which will discover the most frequent structure block level tag (e.g.,< divclass = " ″ >) in the Dom tree.

3) Our algorithm does not follow any hard matching techniques like tree alignment (such as recognizing zones and blocks of the DOM tree with series 1 and 2 observations) or NFA as done in . Instead we summarize and observed the occurrence of tags that create similar block structure.

4) WebOMiner_Simple is extendable and has the potential to be scalable to large websites because of its simple and effective technique.

5) It is highly automated and no manual efforts (such as description of sample training B2C pages) are required in the extraction process. We only have to give the system a product page web page address and it will discover the database schema.

1.9 Outline Of Thesis:

The remainder of the thesis is organized as follows:

Chapter 2: Related literature in the area is presented. We have identified problems which are related to the problem studied in this thesis. We categorized these problems in three sections and each section explains the related work done in these problems and surveys the various solutions proposed. Different works are compared in this section and we tried to identify the advantages and disadvantages of the approaches.

Chapter 3: Detailed discussion of the problem addressed and new algorithms are proposed

Chapter 4: Explain performance analysis and the experiments conducted in detail.

Chapter 5: Concludes this thesis by explaining the work done. The contribution of this thesis is explained in this section. An outline of future work is provided in this chapter.

CHAPTER 2: RELATED WORK

In this section we present the related work in the literature on web data extraction. There have been several works conducted in this domain here we present most significant work performed in the area of data extraction. Researchers had used several data mining techniques on the web data to extract information, over the years web data extraction had improved from manual to automatic using several data mining techniques. We have categorized the existing work based on the scientific technique used and thus we have the comparison based, vision based and the grammar based approaches in web data extraction. The first category of systems build their wrappers based on the comparisons between the new web page and the template. For comparing the web pages or the data blocks researchers had used string matching, tree matching and multiple sequence alignment methods. In the second group of systems unlike the other researchers the authors proposed to use the visual information on the web page instead of underlying html code. Information like coordinates of the data blocks on the web page and vision based Dom trees were used in the web data extraction. In the third category of systems used grammar based approach for building their wrappers. They observed patterns and used regular expressions to identify and further extracted the data.

2.1 Grammar Based Approaches:

The data extraction systems that performed data extraction by inferring a grammar from the html code of the webpage. The common tag structure shared by the web page is uncovered by using regular expressions, string alignments which are further used to build rules for extraction.

2.1.1 IEPAD: Information Extraction Based On Pattern Discovery

(Chang & Lui, IEPAD: information extraction based on pattern discovery, 2001)

IEPAD is the semi-supervised learning system which extracts data from web without using some manual methods such as labeling the Sample pages. It highlights the usage of repetitive patterns occurred in web pages in order to present the target data as a structure called PAT trees. And, multiple sequence alignment is used to comprehend all record instances.

The Major contributions of the paper includes 1) Identifies the target data record without any human intervention by examining the fact that data records occur as repetitive patterns

in a web document and data is pushed into web pages from underlying databases using a common template. 2) Uses a data structure called PAT trees a Binary suffix tree, to discover repetitive patterns as they only records exact match for suffixes. 3) Uses multiple sequence alignment by the center star method which begins from each occurrence of a repeat and ends before the beginning of next occurrence.

The Architecture of the system has 3 modules first is an Extraction rule generator, the pattern viewer and the Extractor module. Where the first module analyses the html tag structure and generates rules, the second module displays the rules to the uses to select one rule among them to perform extraction and the third module extracts all the data records that match with the rule selected. Their algorithm is explained with a running example as follows

Example 1: For Instance consider this 2 lines of html code as input for the system in their first step.

<B>Congo</B>< I>242</I>

<B>Egypt</B><I>342</I>
$

The HTML page is given as an input to this component it then translates into a string of abstract representations called tokens which is identified by a binary code. Here tokens are divided into two types, tag tokens and text tokens where tag tokens are the html tags and text tokens are the original text in between the tags denoted by Text(). Discovering patterns based on tag tokens majorly block level tags excluding text tags will result in more abstract patterns that represent the whole web page.

Example 2: Token string generated by the translator for the above example is as follows

Html(<B>)Text(_)Html(</B>) Html(<I>)Text(_)Html(
) Html(<B>)Text(_)Html(</B>)Html(<I>)Text(_)Html(
)$

These as encoded into binary strings as shown below

Html (<B>) 000, Html (</B>) 001, Html (<I>) 011 then the pattern would be 000001011$

The binary pattern which is the output of the previous step is given as input to this step. A Patrica tree is a specification of binary suffix tree where 0 goes to left and 1 goes to right. Here, each

internal node is an indication of which bit is to be used for branching. Every sequence of bits starting from each of the encoded token extending it to the end of the token string. Each sistring is represented by a bit position. An important observation in pat tree is that, all suffix strings with the same prefix will be under the same subtree. Also, each edge is associated with a virtual edge label which is the sistring between those two nodes. We can uniquely identify a sistring by its prefix. The edges labelled with the $ sign indicates the end of the string as shown in figure 8

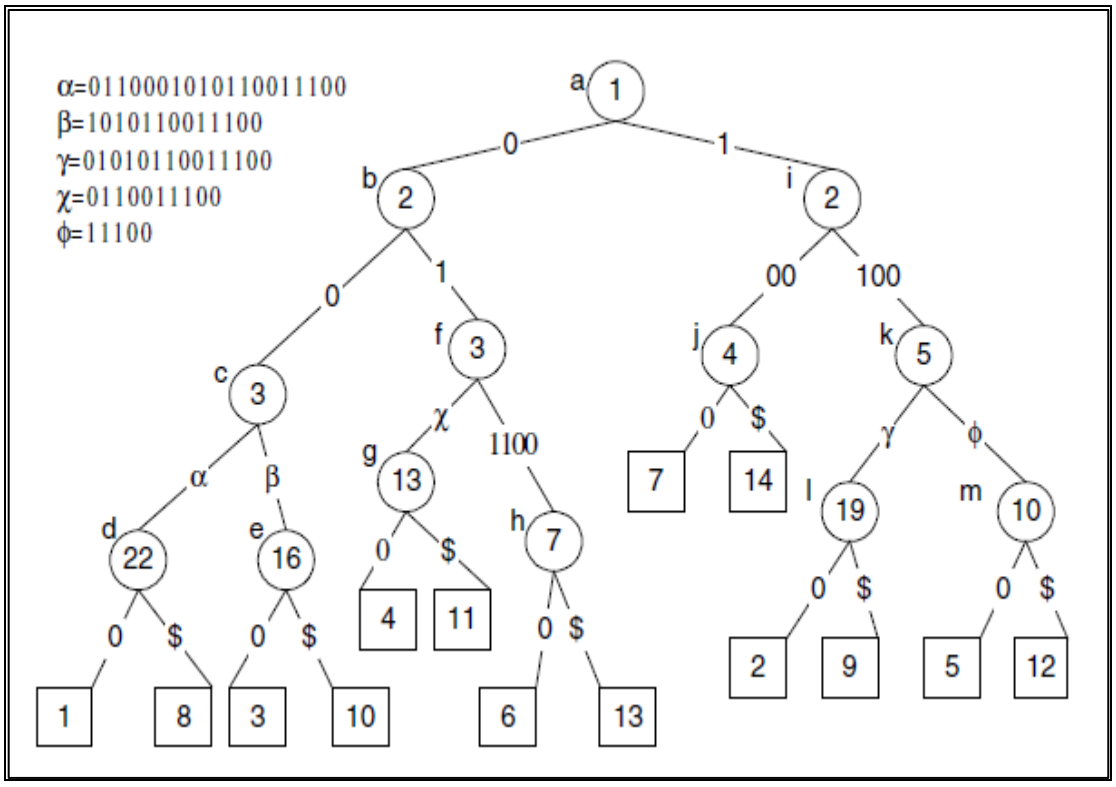


Figure 8: A Pat Tree

Every Path label of the internal node represents a repeated sequence of inputs. Hence to discover the repeated patterns we now only have to observe the path labels. Thus the problem of identifying data records is narrowed to fetching repeated patterns in IEPAD. For a path label in the internal node v if its sub tree is having different left characters then it is called Left_Diverse. Based on this a lemma is proposed that maximum repeat should be a left diverse. The number of times any pattern has occurred can be easily known by the occurrence counts and the reference positions. Hence the user has to define a threshold value or it can even be a default value. Regularity, compactness and coverage are defined as the standard deviation of the interval between two adjacent occurrences is calculated as regularity. Whereas compactness is a measure of the density of a maximal repeat, this is

used to eliminate maximal repeats that are scattered apart beyond a given bond. Along with these, Coverage measures the volume of content in the maximal repeats. IEPAD uses multiple string alignment for pattern matching, it does not allow any partial or inappropriate matching only strings exactly matched are considered. For Example, Suppose "kjl" is the pattern discovered for token string, if we have multiple alignments kjlmn, kjlm, kjlmo the extraction pattern can be generalized as "kjl[m\x]o[p\-]" where " –" represents missing characters. Thus the problem is transformed to find the multiple alignment of the k strings. The approximation algorithm is used to find the center string S_c in K strings. Once the center string is found, each string is iteratively aligned to the center string to construct multiple alignment, which is in turn used to construct the extraction pattern.

The extraction process starts when the user selects one or more patterns from the pattern viewer. The extractor searches the PAT tree for maximum occurrences to find the target data but if the web page cannot be expressed as PAT tree then a normal pattern matching algorithm can be used for extraction. Their results shows 97% accuracy in successfully identifying the target data.

Figure 9: Sample String Alignment

However the major Short comings of the paper includes the following

1) The assumption that multiple data records will be homogenous fails as data records in web pages can have different attributes because of the missing values.

2) Requires a trained user that can infer rules to extract actual data records.

3) Their assumption that target data will be under the same parent node fails for large number of web pages as there is no necessity in HTML5 after using the block tags to section web pages.

4) Does not propose any strategy to overcome noise or unnecessary data.

5) Their use of binary suffix trees only produces exact matches is a huge drawback because of the missing attribute values in data records.

6) Did not discuss about the heterogeneity of the web data such as the Images and their extraction process.

2.1.2 STALKER: A Hierarchical Approach to Wrapper Induction

(Muslea, Minton, & Knoblock, A hierarchical approach to wrapper induction, 1999)

This is a supervised approach for data extraction, where the user labels some sample pages from which the system generates the extraction rules. There by data in web pages having similar structure can be extracted by matching those rules. It performs a hierarchical data extraction by using embedded catalog formalism. The Major Contributions of the paper includes 1) Introducing the concept of hierarchical data extraction using embedded catalogue tree in which leaves are the attributes and internal node are the tuples.2) web content can be extracted by some consecutive tokens called landmarks that enable the wrapper to locate data x with in a data region p. 3) Multiple scans are performed to handle missing values and multiple permutations. The Stalker proposed to represent a web page as typical embedded catalogue tree where each leaf node contains the target data and the internal nodes form the list of K tuples. Again the items in k tuples can be a leaf node l or another list L. Each edge of the tree is associated with an extraction rule in order to extract x from p.

For example consider the Restaurant description as given below

1:<p> Name: Yala <p> Cuisine: Thai <p>

2:4000 Colfax, Phoenix, AZ 85258 (602) 508-1570

3:
 523 Vernon, Las Vegas, NV 89104 (702) 578-2293

4:
 403 Pica, LA, CA 90007 (213) 798-0008

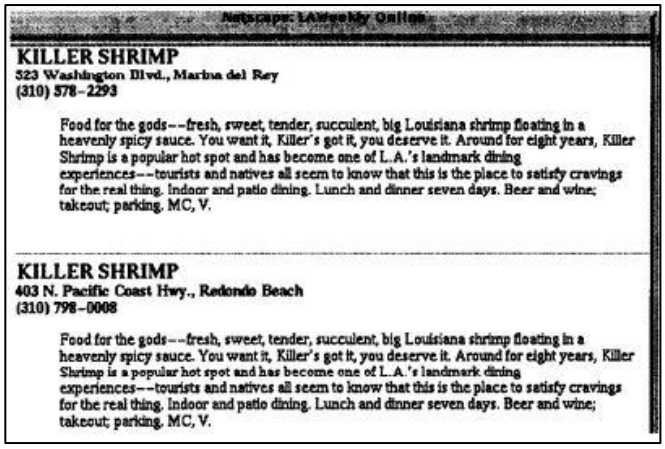


Figure 10:EC tree for LA weekly

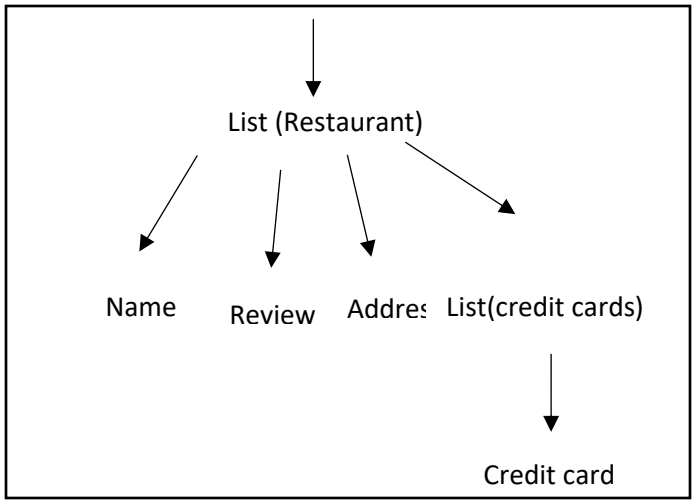


Figure 11: Description of LA weekly

Data is extracted by using 2 rules the start rule and the End rule.For the above given example in order to identify the name of restaurant the start rule is SkipTo(and the end rule would be SkipTo(). Thus for each data item an extraction rule is associated and the wrapper has to generate it automatically. The SkipUntil() and NextLand mark () rules can also be used.

The Extraction rules can also be expressed as landmark automata where each transaction is labelled by a transaction. Disjunct rules can be used in extraction process.

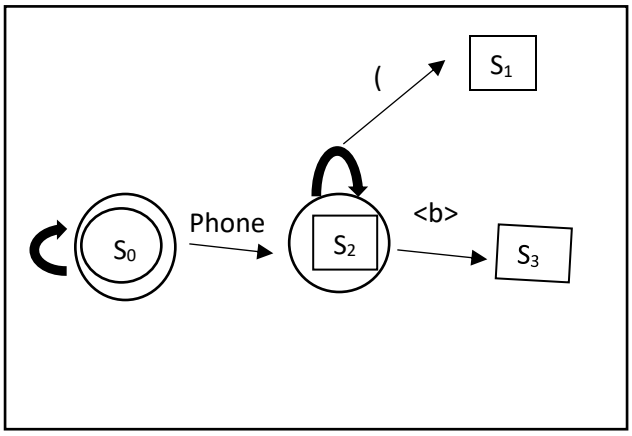


Figure 12: A slg for the start of area

Example: 2

El: 513 Pica, Venice, Phone: I-800/b>-555-1515

E2: 90 Colfax, Palms , Phone: (818) 508-22222

E3: 523 1st St., LA , Phone: l-mk-578-2293

Ed: 403 Vernon, Watts , Phone: (310) 798-0008

The stalker inductive algorithm generates extraction rules to capture each data item by taking as input the set of sequence of prefix tokens. User identifies the start of x from the sample page using the GUI. For instance consider that the user has marked the area codes then the sequential covering algorithm starts by generating linear l, As it then covers all the examples and generates disjuncts. In the fore mentioned example algorithm initially generates SkipTo(() it matches E_2 and E_4 but does not match E_1, E_3. While in the second iteration algorithm considers only uncovered examples and generates SkipTo(phone) and SkipTo(). The fig 1 represents the slg that accepts both the rules. Thus Stalker tries to generate slg that accepts all the positive examples and reject all the negative ones. The algorithm contains a function called learnDisjuncts() generates all the candidates repeatedly selects and refines the best candidate until it finds a perfect disjunct which only accepts the positive pages. In order to find best disjunct stalker searches for a disjunct that accepts the largest number of positive examples. Another function Refine() obtains netter disjuncts() by making its landmarks more specific or by adding new states to the automata. Followed by the GetTokens() which gives out the tokens that have appeared at least once. Their

experimentation proves that their methodology achieves 97% accuracy. But Short comings of the paper are 1) Their algorithm requires Multiple passes over the input data to extract the target data. 2) The performance of the algorithm entirely depends on the sample training pages given, making the system dependent on a trained user. 3) Their Technique is scientifically complex and time consuming that it cannot be used in real time applications. 4) Maintaining wrapper is another huge dis-advantage of this system, every time new presentation structure arrives the wrapper has to be updated.

2.1.3 The WebOMiner : Towards Comparitive Mining Of Web Documents Using NFA

(Ezeife & Mutsuddy, 2012)

Ezefie and Titas proposed a framework for orject oriented web scema extraction and their integration as a two level mining process. They liad paths for the extraction of heterogeneous web data by a unified approach using NFA. Major Contributions of Paper are 1)Proposes an architechture with 4 modules for web Data extraction and storage, this involves eliminating noise to enter the database. 2) Identifies data blocks and data records by defininig the seperator. 3) An NFA for each content type is generated such that target data can be extracted from dom trees. 4)Associates leaf level tags with contents as it consists of imortant information about the associated content. In order to identity data region and data block they have constructed the DOM tree T represents the entire web page then data regions are subtrees of T, Also author argues that data region and block can be with in any tags such as <*div*>,<*table*>,<*tr*>,<*span*> and this list is not complete. They observe that data regions are embedded in between and <table > tags and they used "{,}" to represent a data a data region. They define data block as a tuple when it is decomposed to a flat structure this is accomplished by the schema. To identify tuple fromation and data block they identified 6 types of content blocks in a business 2 customer domain, They are Product data block, list data block , form,text, decorative/singleton,noise. But the target is the product data block which is usually of the form {< *image* >,{< *title* >, < *number* >, . . .,< *brand* >, < *price* >}} , some pages may also be like {< *image* >,< *title* >, < *brand* >, < *price* >}. This is the information that has to be identified and extracted. Inn order to Extract content tuple types they used NFA. NFA is a non-deterministic finite state automata, where each pair of states and a nput there is a next state. In order to build

amNFA that can extract target data from all the porduct list pages author observes a comprehensive list of schemas. Some of them are as follows,

Product (title:string, image:image-file,prodNum:string, brand:string, price:long);

Product (title:string, image:image-file,prodNum:string, price:long);

Thus, the NFA should be able to identify any of the scema and extract the data tuples and send them to product content object class. The case is similar with list, form, text and image contents as well.

Author proposed an architechture with 6 modules called sequentially by the main algorithm and always output of the previous module is input to the next one. Each modul

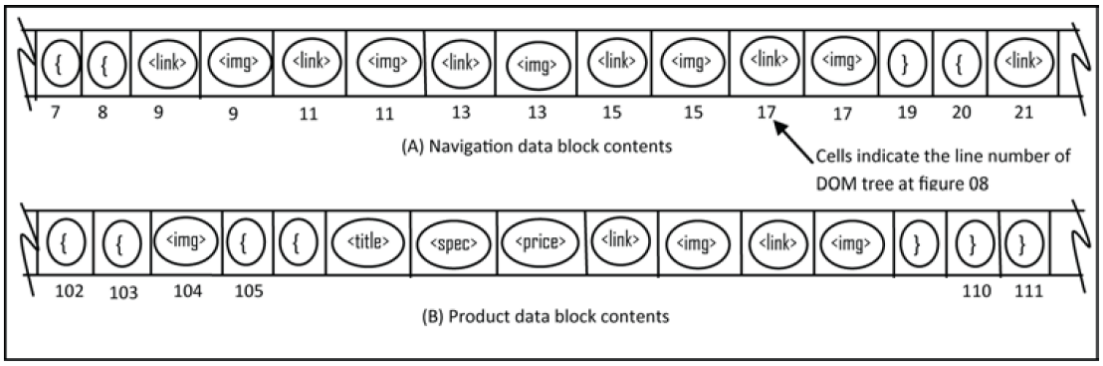


Figure 13: Data block contents

is described as follows. They proposed a mini crawler algorithm that takes input as a url, it crawl the web fetches the page nad stores it in a local directory. It discards the comments from the document. Followed by the cleaner module. Their reason for using a cleaner is that the web data is usaully dirty. Data cleaninig has to be performed to avoid any missing tags or in approporiate ones because DOM tree is built based on these tags, ao it is importan to have quality tags. After the cleaning is done content Etractor Module is called.The task of this module is to take the clean html file to build a DOM tree and assign respective class object class and push objects into content array list.Each data block is seperated by the seperator object. Then the algorithm calls Mine contentObject.Identifytuple() which extracts diffetent tuple types and places them in corresponding containers This algorithm generates Seed NFA pattern for data blocks from positive pages. It extracts objects of all tuples by matching them with the refined NFAs and storing identical tuples(e.g., product

record object tuples, list tuples) into appropriate TupleList. A count check is performed to verify if all the tuples wre extracted. Tuple squeezing is used to generalize tuples of the same data block. so that pattern of any tuple containing varying length pattern can be represented in the same category. In order to Create Data base table **t**he data ware house star schema approach is used to form the fact and dimentional tables from the tuple array list along with the company name. They have achieved an accuracy of 100% and recall of 96%. Major Short Comings Of The paper includes 1) Matching each tuple in the ArrayList with NFA sequentially is a hard matching technique, can not be applicable to large web sites 2)Wrapper Maintenance : has to be updated every time a new structure is encountered

2.1.4 RoundRunner: Towards Automatic Data Extraction From Large Websites

(Crescenzi, Mecca, & MerialdO, 2001)

Automatically generates wrapper for extracting data from html pages by comparing pages based on their similarities and dissimilarities. Given a set of sample html pages of the same class finds the nested type of source data set and extract the source data set from which the pages have been generated. The Major Contributions of the Paper are 1) Wrappers are generated in an automated way no user interaction is needed neither any training examples are necessary. 2) Capable of handling any data records either nested or flat and no prior knowledge of schema is required. 3) Pattern discovery is based on similarity and dissimilarity between two web pages. Here, miss matches are used to identify relevant content. 4)According to the Paper a regular grammar is generated and it is used to parse the page, in this case identifying such grammar only by positive pages does not yield good results. 5) A lattice-Theoretic approach is proposed, this is based on a close correspondence between nested types and union free regular expression. According to the Author, Website generation is a process of encoding original data sources into html tags now extracting information must be a decoding process. As the nested structure of web pages is hierarchical representing them in union free regular expressions and there by finding the minimum UFRE by iteratively computing the least upper bounds of the RE lattice to generate a common wrapper is the solution for finding web page schema. Here the html tags are the input strings $S_{1,}S_{2,\ldots.}S_{k.}$

A Matching algorithm called ACME, Align, Collapse under mismatch, and extract is proposed the input to this algorithm are the list of prepared html tokens and two web pages in which initially considered as a version of wrapper and progressively will be transformed into a wrapper that extracts from similar pages. The wrapper is generalized whenever a mismatch has occurred. According to this Paper, two mismatches can take place String mismatch and the tag mismatch.String mismatches are nothing but the different values to the attributes so they can be commonly generalized by replacing #PCDATA. In the mentioned example strings “john smith” , “Paul jones” there is string mismatch which can be overcome by replacing them with #PCDATA. However the tag mismatch is not as easy

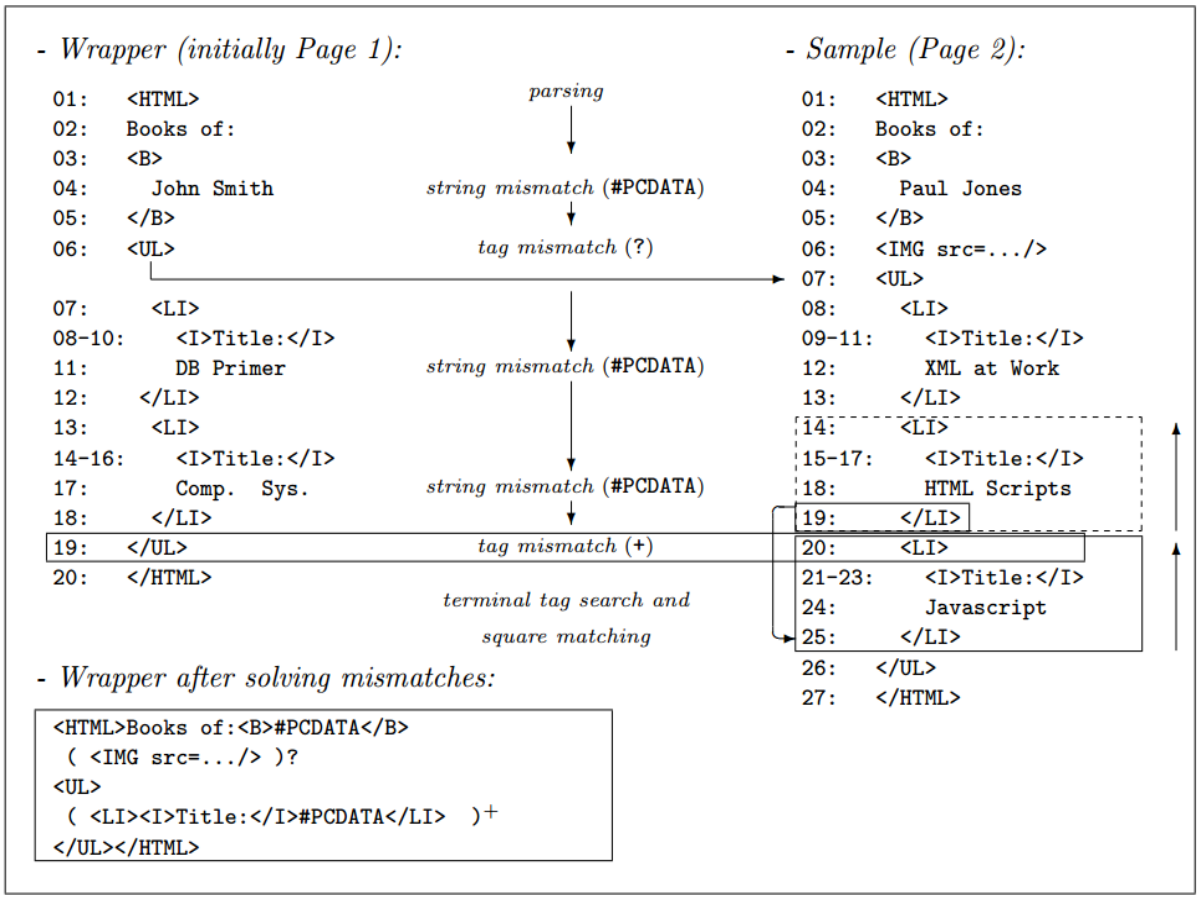


Figure 14: matching two book web pages

to handle as string mismatch. In this case the optional are used. The algorithms tries to find out the repeated pattern once there a mismatch it considers that attribute as optional. For instance in the example a mismatch occurs at token 6 due to the presence of an <IMG > tag in page 2 then the algorithms performs cross matching and considers <IMG> tag as optional

finally finding the repeated pattern with the next <UL> tag.when a mismatch occurs due to the cardinality of the data values it can be solved by identifying the squares and generalizing the wrapper accordingly. Algorithm initially finds where the minimum cardinality exists whether in the sample or in the wrapper, then finds the terminal tag. There by explores the possibility of candidate square of form <UI>……</LI> in the wrapper which is a fail and next in the sample <LI>……..</LI> it succeeds here. Therefore it concludes that the sample contains candidate occurrence of tokens from 20 to 25.Now that the algorithm found square s the wrapper has to be generalized as S^+ by continuously searching the wrapper for repeated occurrences.

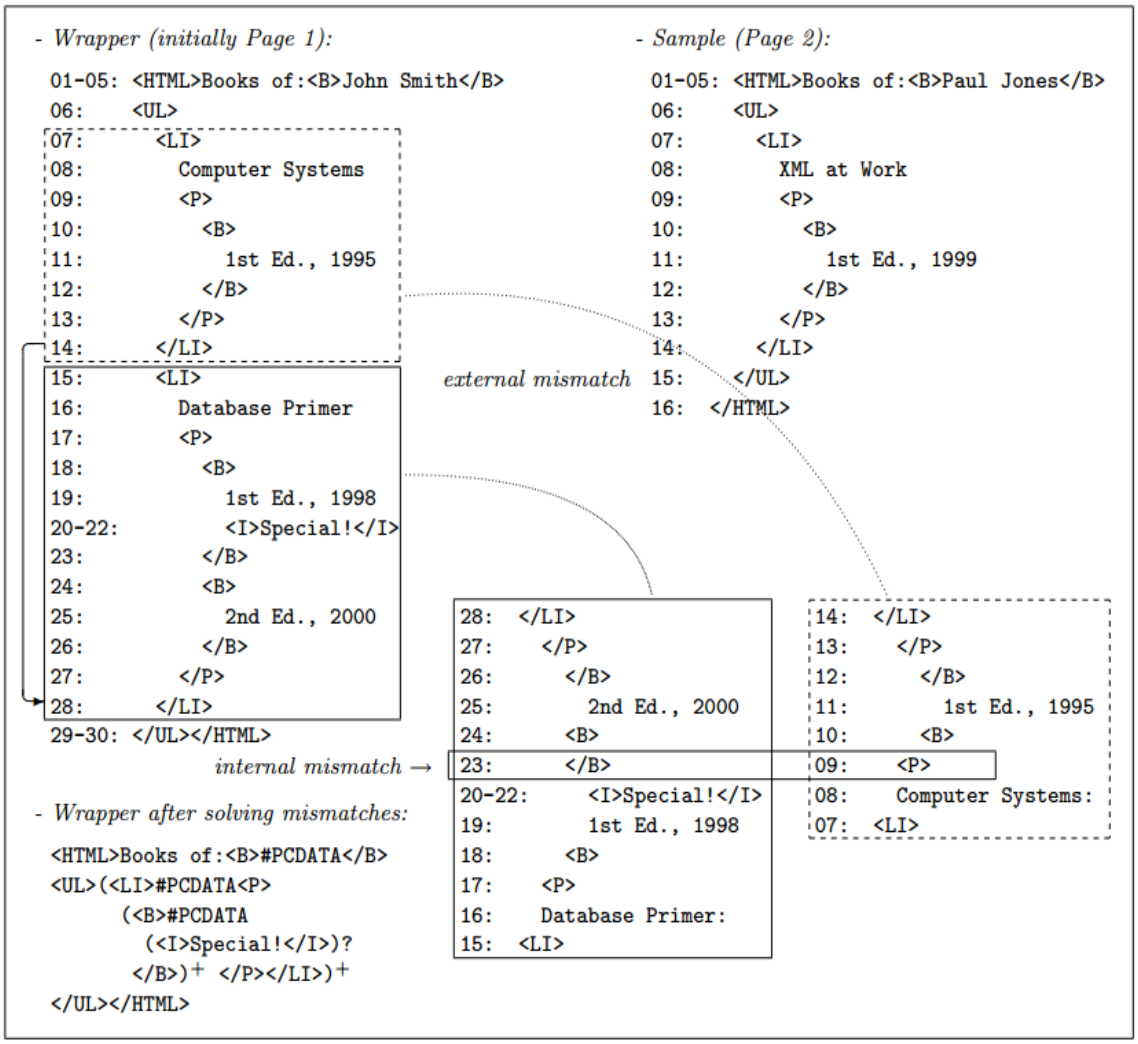


Figure 15: complex mismatch

But the problem arises when we try to resolve a mismatch and it results in more internal mismatches. A variant of pages about author is showcased in example 2, parsing stops at line 15 due to tag mismatch when the algorithm tries to find the candidate tokens by matching upward from the terminal string it results in more and more mismatches. This situation can be handled by considering each internal mismatch as another tag mismatch try to solve it in the same way as external mismatch. In the example internal mismatch involves tokens 23, 9 after solving it leads to the list of editors and the second mismatch is treated as an optional pattern <I>Special!</I>. Hence from all the above cases the algorithm Match can be summarized as follows, Input to the algorithm is a wrapper w and a sample page s it tries to generalize the wrapper by finding the mismatches occurred. Therefore finding a solution to the match (w,s) is a visit to AND-OR tree in fact the solution is finding solution to every mismatch that has occurred in the process of parsing the tree and will involve adding a node or searching for one. Short coming of the Paper are 1) Manual labelling the fields after extraction is necessary as they are anonymously named (like a,b,c,d etc.,) 2) The algorithm has exponential time complexity with respect to input lengths so it cannot be used in real time. 3) It requires more than one training page and is semi-supervised approach.

2.2 Comparison Based Approaches:

These approaches find commonalities to identify data records by making comparisons between the page fragments considering the web page as a tree and by performing tree edit distance, string edit distance. These kind of systems are explained in detail as follows.

2.2.1 DEPTA: Data Extraction Based On Partial Tree Alignment

(Zhai & Liu, 2005)

This paper handles the problem of data extraction by using the tree alignment technique. It compares adjacent substrings with starting tags having the same parent node instead of comparing all suffixes. Their Major contributions include 1) Using a vision based approach for building the Dom tree and identifying the data records. 2) Nested structure of data records is not lost due to the tree alignment method used in extraction process. 3) Irrelevant substrings are not compared to avoid unnecessary computations.

<table>
<tr>
<td>….</td>
<td>…….</td>
<tr>
<td>…..</td>

left	right	top	bottom
100	300	200	400
100	300	200	300
100	200	200	300
200	200	200	400
100	300	300	300
200	300	300	400
100	300	300	400

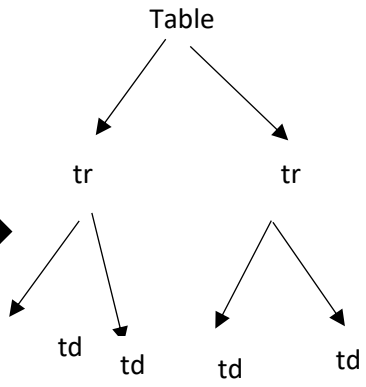


Figure 16: Vision tree

Figure 17: Sample HTML code and Coordinates rendered

Author argues that data records may not be contiguous and in a nested structure it can begin from the middle of any other record as well. Also he states that trees based on tag information does not always guarantee correctness so vision based tag trees are reliable. Based on these observations he proposes a novel tree alignment method on vision based trees.

Building a vision based HTML tag tree involves Each HTML element being represented as a rectangle. A Tag tree is constructed based on the nested rectangles by finding the 4 boundaries of the rectangle and by detecting the containment Relationship between the rectangles. Instead of immediately mining data records author proposes to mine data regions first then followed by data records. A data region is found by performing the string edit distance. A generalized node is used to indicate a subtree. Unlike the other methods author says that data records may not be contiguous but generalized nodes are contiguous. The data records are identified by the fact that the distance between the data records in a data region are less than the distance between any gaps with in a data record. There can be a case in which a data region can have more than one generalized node or even two or more data regions from multiple generalized node such cases can be handled by the algorithm. Data extraction is performed by creating a rooted tag tree for each data record, once all the data records are identifies the subtrees of all the data records are re-arranged into a single subtree. Then the tag trees of all the data records are in each data region are aligned using partial alignment method based on tree matching. In this approach author aligns multiple

tag trees by progressively growing a seed tree T_s, here the seed tree is tree which has a maximum child nodes. For each T_i the algorithm tries to find out for each node in T_i a matching node in T_s. If a match is found then a link is created between those nodes if not, a node is inserted into the seed tree. In that case where to insert a new node or a set of unmatched sibling nodes uniquely is a question. In such case it is added to R set flag to true, whenever a new alignments are found iteratively R is compared for matching. The procedure is explained by using an example as follows

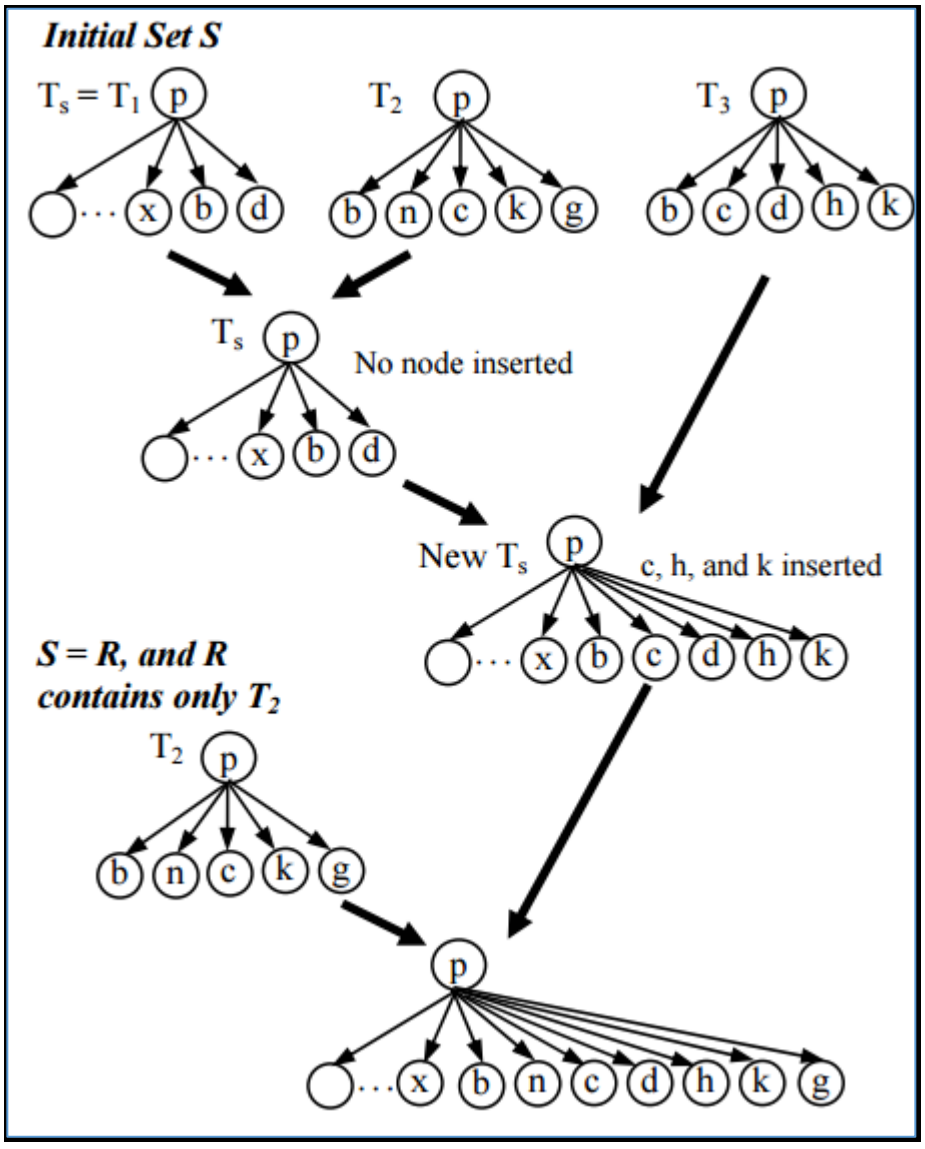


Figure 18: Iterative tree alignment with two iterations

There are 3 trees T_1, T_2, T_3 and consider T_1 as the seed as it has more number of child nodes. Algorithm assigns T_s to the seed tree which is T_1 now. The rest of the two trees are compared to the seed tree for alignments. Ts and T_2 produce one match, node b. Nodes n, c, k and g are not matched to Ts. So, it attempts to insert them into Ts. However none of the nodes n, c, k and g in T_2 can be inserted into Ts because no unique location can be found. Then it inserts T_2 into R, which is a list of trees that may need to be further processed. In when matching T3 with Ts, all unmatched nodes c, h and k can be inserted into Ts. Thus, T_3 will not be inserted into R set "flag = true" to indicate that some new alignments/matches are found or some unmatched nodes are inserted into Ts. When the algorithm encounters S=∅ and flag="true" it understands that there are no more trees in S but there are some new alignments and insertions took place. If there is any data still unprocessed it will be placed into a single column in the database table. Thus all the data records are extracted and stored in the database. The results showed that their approach has yielded 99% of recall and accuracy. Short comings of the Paper includes 1) This approach does not work nor cannot be extended with the pages having single data record such as the detail page 2) Nested data records cannot be extracted using this approach.

2.2.2 DEBYE: Data Extraction By Example

(Laender, Ribeiro-Neto, & da Silva, DEByE–data extraction by example, 2002)

A supervised learning system where user marks example pages there by the structure is learnt from the pages through patterns and from it structures can be extracted by comparing the new data records with previous one. The Major contributions of the paper involves 1) This paper proposes Object oriented content extraction without loss of relationship between the data items. 2) It extracts atomic components first and then it assembles them into objects. 3) Uses pattern matching to identify the similar tokens for extraction. In this paper each piece of data is considered as an object in the mentioned example of 4 four authors and some of their books,

each piece of data is represented by an object. There exists multilevel objects they term these as complex objects

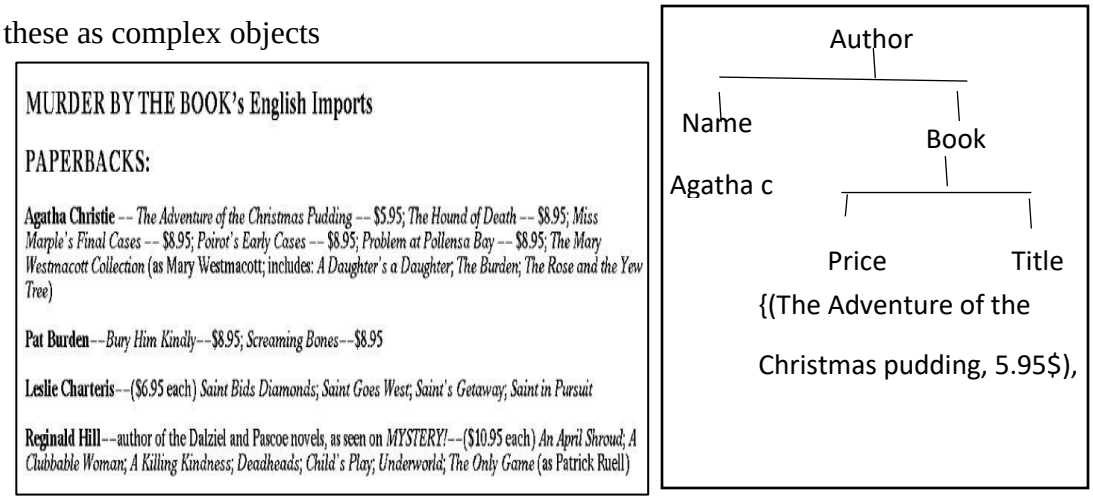


Figure 19: Snap shot of a web page and hierarchical structure for the object

Debye has 2 modules The Graphical interphase and The Extractor. An object is denoted as a pair O= <T, V> such that V ϵ domain (T) where V is a type of O. thus O is an instance of T. domain(T)={a1,a1,a_3….a_n}(n>=1).Instances of A-types are termed as atoms. The pair <T, a_i> is termed as attribute value pair $_{User}$ has to highlight the target data from the web page using the GUI. This target data is treated as objects. Such assembled objects are used to generate oe_patterns in xml. These patterns are used to extract data from the new web pages. The initial and final positions of occurrence of data are indicated by ipos, fpos respectively. Attribute value pair patterns are described as <T,a_i>, a local syntactic context is associated to each avp it is the tokens surrounded by avp. Using this context information avp patterns are built. In the given example in order to identify avp pattern for type price would be as expressed in (a) that the value should be prefixed by S_{pre} = "A deadly lie--$" in particular and suffixed by

S_{suf} = ";dead online".so it strictly extracts only specific values. A more generalized pattern is indicated by (b) it denotes that price value is prefixed by "$" and followed by ";" and "*" indicates any value in that place.

```
<TUPLE type="Author">
   <ATOM type="Name">
      <VALUE ipos="2058" fpos="2073">Agatha Christie</VALUE>
   </ATOM>
   <LIST type="Book">
      <TUPLE type="Book">
         <ATOM type="Title">
            <VALUE ipos="2084" fpos="2122">The Adventure of the Christmas Pudding</VALUE>
         </ATOM>
         <ATOM type="Price">
            <VALUE ipos="2131" fpos="2135">5.95</VALUE>
         </ATOM>
      </TUPLE>
      <TUPLE type="Book">
         <ATOM type="Title">
            <VALUE ipos="2257" fpos="2280">The Hound of Death</VALUE>
         </ATOM>
         <ATOM type="Price">
            <VALUE ipos="2289" fpos="2293">8.95</VALUE>
         </ATOM>
      </TUPLE>
   </LIST>
</TUPLE>
```

Figure 20: Extracted object in xml

Malcolm Hamer--GOLF MYSTERIES!--
A Deadly Lie--$10.95; Dead on Line--$13.95;
Shadows on the Green--$10.95; Sudden Death--$9.95;

Fig. 6. A sample text.

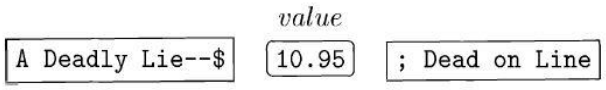
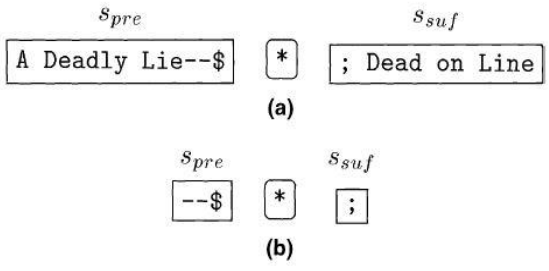


Fig. 7. Example of an AVP context.



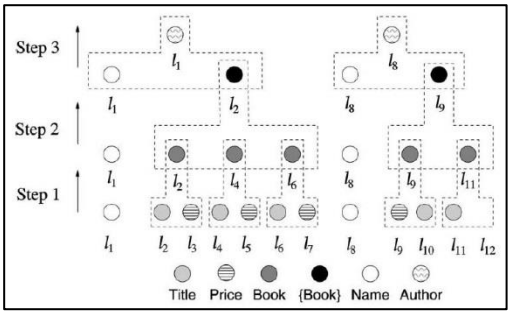


Figure 21: The execution of bottom up

OE patterns represent the structure of the data that the user needs to extract, its internal nodes form another OE tree or type nodes (like price, book) and leaves are avp patterns. There are two extraction strategies specified, the bottom up and the top down strategy. The Top down strategy is a straight forward, the oe patterns are assembled for each example object and using these patterns to discover new objects. This procedure is repeated based on the number of example pages provided. Here the whole object is demolished into components thus called a top down approach. This approach does not work well if the page is variable structure.

In the Bottom up strategy the avp's are priory recognized and extracted. For this approach objects are represented as triple O = <T,V,L> where t is the type of the object, v is the value and l is the index. The last row of circles indicate the avps. Each is avp is associated with a label l_i. Contiguous pairs of title, price are formed as book instances. The values in L_9 are in reverse order and L_{11} does not have the price component, such cases can be handled by the bottom up strategy. The list of objects assembled after the first step are indicated in 2nd row. Further the book instances are grouped up as list of book instances. From the example l_1 is combined with l_2 and l_8 is combined with l_9. Thus the data can be extracted from similar pages. Major Short Comings are 1) Extraction precision falls low in case of missing attributes or multi-ordered attributes. 2) Multiple scan of input page is needed to extend each atomic attribute.

2.2.3 Information Extraction From Web pages:

The paper presents techniques to extract data from web pages in any format, if the page contains multiple records it proposes an ontology based tecnique, if it is a detailed page then comparision with template takes place. Major Contributions of the paper are 1) Presents web data as objects and data extractoin as the process of object attribute values extraction. 2) Ontology based extraction mechanism for list pages and tree matching based approach for detail pages are proposed. 3)Extraction rules are discovered based on the positions of differences in the dom tree. In Their approach author treats web pages as 2 types 1) Master page 2) Detail page A master page is the one having list of taeget data

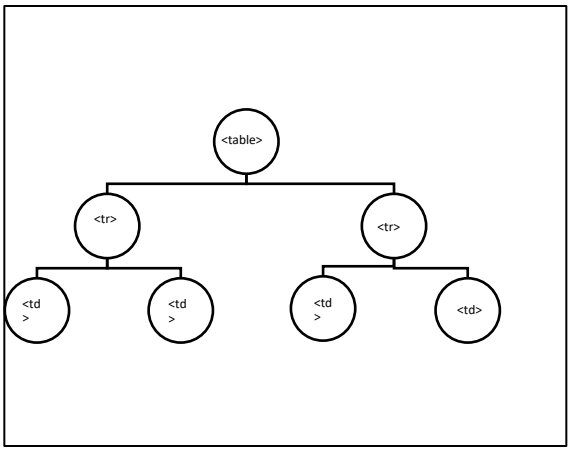


Figure 23: Sample Dom tree

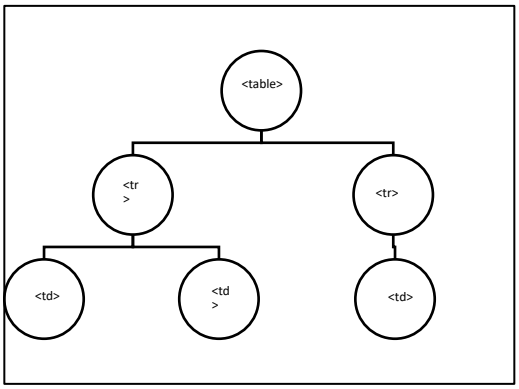


Figure 22: Sample DOM tree with 2 sub trees

for example list of products in a e-commerce environment.When ever user clicks on a particular product all the inforamtion appeers in a detail page. Most of all the techniques proposed so far treat web page either as a list or as adetail page but it not the actual case.A web page is crawled as a DOM tree and the nodes are classified by their weight, performing prune operation on nodes with weight 0. Now that the web page is modelled as a dom tree, the search for the data records is Breadth first search. Finds the repeated similar sub trees by comparing the text similarity of nodes by string edit distance. In the tree text nodes are tokenized by considering the html, special characters and conjunctions as seperators. In order to Identifying attribute values from master page the attribute values are obtained from ontology, They may be atomic or non-atomic sometimes even both. However we have to match the tokens to the attribute values which is obtained with the help of the ontology. The leverage of ontology allows us to detect the structure of data records, the cardinality of extracted attribute values and possibly the relationships between them. The extraction precision can be improved by the onyology extraction which can specify the attribute labels, regular expression or key words. But certain data values can not be extracted from domain, for example data record which has to be extracted a student name has an infinate domain. To Identify attribute values from detail pages data extraction from detail page is based on the observation that, web page is a template generated by embedded values form the underlying databases. Thus they represent web page as a HTML page with variables. These variables are in the process of template evaluation merged with the data model having the attributes of products. Author believes that dissimilarity between web pages

occurs 3 places 1) Difference between the structure of DOM tree and the Structure of Sub trees in the web page. Below given sample dom trees to represent the similar structure but the sub strees are not similar as their number of child trees differ. 2)Difference between the HTML attributes or values. Because, Not that every data record will have the same attributes. 3)Differences in text nodes , representing different values to nodes. By identifying the differences, author proposes extraction as, Initailly retrieving the HTML source code of 2 detail pages is performed, then apply diff algorithm to retrieve all the different spots. Followed by finding the nearest HTML element which contains different spot in the sub tree. If the different spot is in the child node it s clear that the pages have the same structure except variations in values, so declare the entire next node as candidate to be extracted. But if the difference is in HTML atrribute simply ignore it. If it is the other case then the web pages does not have the similar structure hence pass it to other extraction process. Thus data can be extracted from detailed and list pages by identifying their diffrences. Short comings of the Paper include 1)Ontology is largley a manual process and expensive often not compatable. 2)certain attributes can be of infinite domain. Such as the product names.

2.3 Vision Based Approaches:

2.3.1 LIXTO: Visual Web Info Extraction:

(Baumgartner, Flesca, & Gottlob, 2001)

Lixto is an interactive tool for data extraction built with a logic based language called Elog. Elog uses a data log like logical syntax and semantics. Which translates relevant html strings into XML. The LIXTO system provides a user interface to select the data of user's interest. Thus selected data is translated into xml. There is no necessity for the user to have an idea over ELOG the underlying language. Elog is wrapper programming language, it generates wrappers for data extraction by taking input as the user selected data from the web page. It is a collection of data-log like rules containing special extraction conditions in their bodies. Once a wrapper is generated it can be automatically extract relevant information from a permanently changing page. The lixto system associates the user selected instance with a generalized tree path in the HTML parse tree, there by identifies similar rules. For example, user selects pattern <ITEM> followed by sub pattern <PRICE

>, this sub pattern relationship expresses that an instance of <PRICE> should occur within a pattern <ITEM>. Then the system searches for such occurred patterns. User can also add conditions like after/before/ not after/range etc. Setting such filters can fetch perfectly desired information. Pattern creation is done by User explicitly providing the patterns their parents, can specify any attributes required then highlights all the objects in the provided example already. For example if the user selects a <TALBE> as a pattern, LIXTO searches for the occurrence this tag in the entire documents also highlights the <TR> tags. A rule is constructed for generating a rule to extract all the target data items. The patterns generated along with the hierarchical relationships are translated to XML by using their names as default <XML> tags. Extraction language Mechanisms is as follows, Head predicates are defined in Elog as record(S, X) where S indicates the parent pattern for instance <TABLE> in our running example. X looks for the sub elements that qualify as S. Thus defined head predicates are the patterns that Elog defines and extracts data according to the web page. Lixto offers two mechanisms of data extraction- string extraction and tree extraction. In the tree extraction mechanism elements are identified by their tree paths. An example of tree path can be *table.*.tr here * indicates can be called a wildcard, it indicates anything can occur. For correctly extracting only target data, attribute conditions can be applied. An attribute condition is a triple specifying a required name, value and a corresponding regular expression. The second extraction method is based on strings. In the parse tree leaf nodes indicate the stings. Regular expressions are used to capture them. Elog atoms correspond to special predicates with well-defined semantics as follows, In Elog the function mapping a given source S to a set of elements matching an *epd* is treated as relation subelem(S, epd, X). Subelem(s, epd, x) evaluates to true iff s is a tree region, *epd* is an element path definition and x is a tree region contained in s where the root of x matches *epd*. The extraction definition predicates specify a set of Extraction instances, the context condition predicates specify that some other sub tree or text must (not) appear after / before a desired extraction pattern. For example, on a page with several tables, the final table could be identified by an external condition stating that no table appears after the desired table. The internal condition predicates are semantic conditions like isCountry(x) or Date(x). Another class of predicates are the pattern predicates that indicates to patterns and their relationship their parents. A standard rule in Elog is as quoted here New(S, X) ← Par (-, S), Ex(S, X),

Co(S, X, . . .)[a, b], where S is the parent instance variable, X is the pattern instance variable, Ex (S, X) is an extraction definition atom, and the optional Co(S, X) are further imposed conditions. A set of such rules form a pattern. All the rules in a pattern uses the same information like the name of the pattern, its parent etc. Such patterns form an extraction program which can be applied to similar pages. For Example, A wrapper for the above quoted example every item is stored in its own table extracted by <record> , all such patterns are defined in the same record. The price attribute uses concept attribute, namely isCurrency which matches with the string like $ and the bids pattern uses a reference to price. The final two patterns are strings.

2.3.2 *Simultaneous record detection and attribute labeling in web data extraction*

(Zhu, Nie, Wen, Zhang, & Ma, 2006)

This paper proposes a method called hierarchical conditional labeling, A novel graphical approach to mutually optimize record recognition and attribute naming. Major contributions of the paper involves 1)This approach extracts data from list web pages as well as detail web pages. 2) Uses a vision based tree construction mechanism. 3) The problem data record detection is viewed as assigning data record labels to the block in the vision tree. Attributes labeling is considered as assigning labels to leaf blocks, and both the tasks can be performed simultaneously. 4) Efficiently incorporates all useful features and their importance weights. Web pages are represented as vision trees by using page layout features such as font, color, size separators are the horizontal and vertical lines that do not cross any node visually. A block in a web page is represented as a node in the vision tree Conditional Random Fields are markov random fields, If x, y are 2 random variables then G= (V, E) where x=label over observation

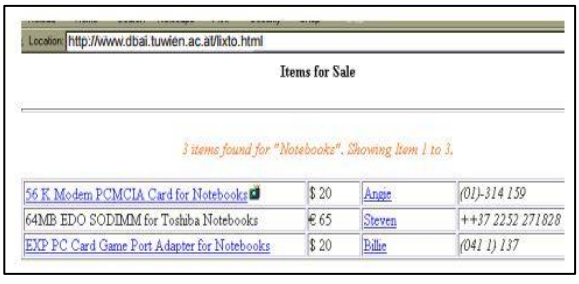


Figure 25: A sample HTML page

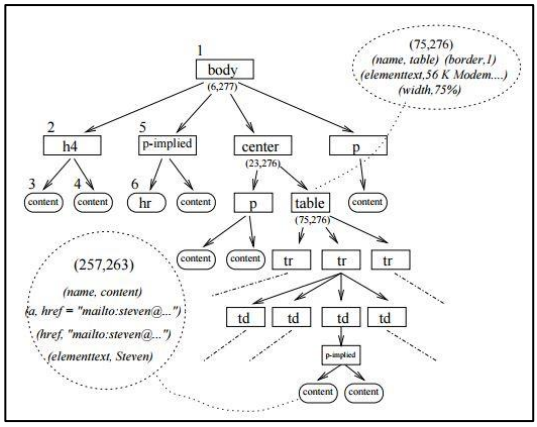


Figure 24: HTML parse tree for the page

to be labelled and y is the variables over corresponding labels. Here, y could be a linear chain.

Now the problem of web data extraction can be narrowed as finding the maximum conditional probability of y over x, where x are the features of all blocks , y are the possible label assignments.

Figure 26: Piece of data from sample web page

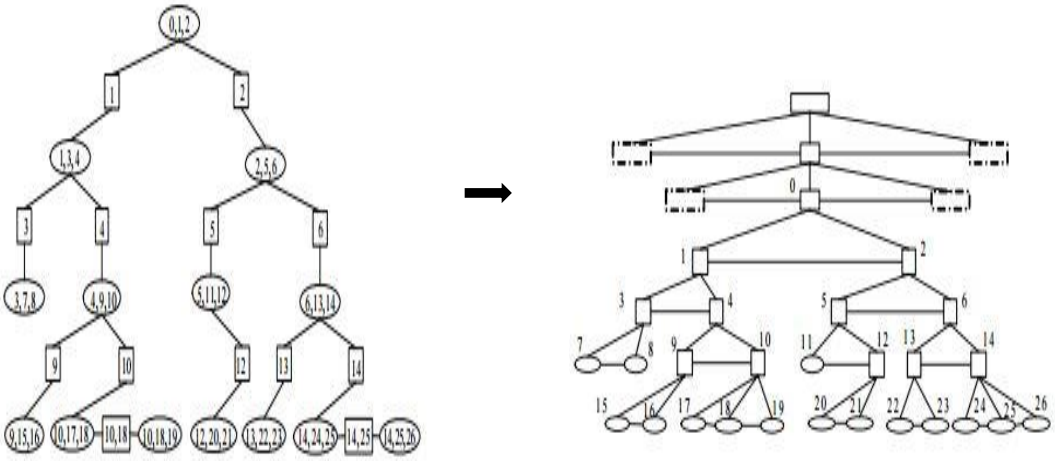


Figure 27: Junction tree for the subgraph starting from 0

Based on the hierarchical representation of data a model can be constructed by denoting inner nodes as rectangles and leaf nodes as ellipses. A random variable Y is associated with each label. Based on the model graph a junction tree is constructed by mapping ellipses a clique nodes and rectangles as separators. Then the algorithm selects the arbitrary clique node as a root node. The figure 2 represents the junction tree of the aforementioned example. The value in each internal node are the values of that particular node along with

its child nodes. The value in the root is thus {0,1,2}.This is followed by two phases. Using HCRFmodel for web extraction it is necessary to define the leabel spaces followed by seperation between the variables at leaf nodes and those at inner nodes, because we need to extract the attribute value of the laf node where a for the inner node has to be identified wether it is a data record or not. There is a necessity to explicitly define lef label space and inner label space each for its own pupose. Forexample the leaf label space for the extraction of products data can be {Product name, product id, prcice, description etc}.

How ever inner label space has 2 partitons object _type independent an dobject tpe dependent.There are certain labels like page head, nav bar , page tail etc, that links to different parts of web page. Such objects are independent of Object_type. The intermediate labels such as data reocrs are dependent on Object_type. The feaures of Elements and Blocks in HCRFS are for each element , the algorithm extracts both vision and content features, all the information can be obtained form the vision tree features. The tree distance of two blocks is defined as the edit distance of their corresponding sub-trees. To exploit the similaruty between blocks the shape an dtype distamce can also be computed.They alo consice that rpeated information in web pages odes not give useful information, as an example they quote that the “add to cart” button and such types occur repeatedly in each page which are not useful.

CHAPTER 3 : PROPOSED SYSTEM THE WEBOMINER_S

Problem Definition :

Given any Business Customer website *W* of a domain *d* consists of several product list pages that are generated by a common templae *T* from the underlying data bases with Schema *S* with tuples α. Identifying S from the webpage there by extracting α to store in a database and further in a datwarehouse for comparative mining.

As discussed in 2.1.4 (Ezeife & Mutsuddy, 2012) proposes object-oriented data model for extraction and mining of heterogeneous web contents. They gave the framework for extracting web data in a B2C webpage and an algorithm (called WebOMiner) for extraction of web objects. Though it achieved the purpose it was very complex and their wrapper have to be updated in case it encounters a new structure. We studied their work and propose two-level mining process for knowledge discovery. This thesis develops the architecture (we call it WebOMiner_S) for web content and structure mining using object-oriented model. It develops, extends and modifies necessary algorithms for WebOMiner_S system. It also discovers database schema and gives guidelines for data integration in the second phase. This thesis addresses the following problems in (Ezeife & Mutsuddy, 2012) work towards development of WebOminer_S System.

3.1 Problems Addressed

1. In the WebOMiner system they have divided the input web page into 3 zones the header, body and the footer zone using the series 1 and series 2 pointers created by their observation that a set of five or more sibling <a> tags mark the start of the body zone marked by series 1 while the last set of 5 or more <a> tags indicates the end of body zone recorded by series 2 pointer. This was done in order to eliminate the noise and narrow down the search for target data which is the product data to the body zone. This Assumption might not always work as expected since every web site has it's own way of design and presentation. We have eliminated this whole process by identifying the repeated block level tags.
2. They have classified the data on the web page into 6 types of content objects, product, list, form, text, noise and singleton. They have discovered structures for each and built NFA's to identify them, this is very time consuming and semi-supervised because their Product NFA has to be trained before identifying the new data. We have overcome this tedious process with our frequent structure finder algorithm in chapter 3.

3. In their work (Ezeife & Mutsuddy, 2012) has used NFA's to identify similar tuples, however the web data is very susceptible to change so with time many new structures of product data records might evolve and their NFA has to update the new structure every time. We have overcome this problem in chapter 3.
4. They have proposed the framework to extract and store in database but the discovery of schema has been pending, we have addressed this in our work in chapter 3.

3.2 Problem Domain

For the specific domain of B2C websites, we have selected to mine all the data rich web pages, i.e., the product list page.). From the common B2C webpage structure shown in figures 05 and 06 (page 20 and 21), product list webpage is commonly a data rich page. We observed that, a product list page usually contains brief list of all or specific types of products. There is a set of product list pages in a B2C website. We define a product list page as follows:

Definition 1: *If 'w' is a B2C website and 'p' is a webpage in 'w' such that* $w = \bigcup_{j=1}^{n} p_j$ *,*

then a page $p_j \in w$ *where* $j \geq 1$*, is a product list page iff '*p_j*' contains a set of tuples* τ *of type* α*, where* $\alpha \geq 1$*, having distinct instance type.*

Definition 2: *The data block contains specific information about the specific product such as product name, price, brand, size, etc.*

If 'p' is a web page it consists of different sets of tuples 't' of type α*, where each* α *is a set of similar list of objects and each object having a finite set of attributes r, then each object with its corresponding attributes forms a data block.*

Example : In the figure shows a web page p it consists of different sets of tuples of type product, advertisements, navigations, header etc., and each type has a similar list of objects like the product type has 8 objects with a set of attributes like price, color, brand, hard disk etc.,

These data blocks has to be stored into a database as a data tuple. In the html Dom tree a data block is a sub-tree of the Dom tree having the same or different parent node. The tags that are used to form a data block are ,<table>,<tr>,<td> etc., as per w3 consortium.

Definition 3: *All the data bocks in the webpage together form the data regions and a webpage can have a finite set of data regions depending on the type of data it presents.*

If 'p' is a web page it consists of a set of data regions of type β *where each* β *is a set of data blocks and are finite.*

Example: In the figure given p is a web page and it consists of a set of data regions of type product, navigational, header, advertisements et., where each has a set of data blocks. The product data region is formed by the 8 data blocks that are similar with common attributes.

When extracted a data region into a relational database forms a product data base table with each product's information as a data tuple. In the html Dom tree a data region is a group of similar data blocks with same parent node and are siblings.

With Such web pages having several data regions with data blocks there is no easy way to pick our target information which is the product data. Also, this set of key information is similar for almost every product list pages but their schemas may be different. On observing various B2C websites we have proposed our automatic web data extraction system based on these observations.

Observation 1: All the B2C websites present and style their content using External CSS (discussed in section 1.4). Where all the product data blocks belong to the same category displayed in a page are styled similar by grouping them using the tag and assigning same styles from the CSS file using the class attribute.

We made use of this observation to recognize the product data regions without performing laborious calculations and classifications on the page which makes the whole task inapplicable.

However, now that we have recognized all the data regions one of the other hard job is to identify the product data region among them.

Observation 2: In the Dom tree representation of web page, the most repeated set of tags are the product data block tags. i.e., the underlying html source code of a product rich web page consists of tags that represents all the data that is being displayed and in them we have observed that the tags that are used to represent the product data blocks are similar to each other except the change in the content.

Thus identifying the set of continuous tags as a structure all over the web page narrows down our search for the product data region.

Observation 3: The Structure of the product data blocks is the most repeated set of html tags with maximum length all over the web page.

Thus our final observation lead us to pointing the product data region with all the product blocks. However we do not assume that all the data blocks are continuous thus we only look for blocks that

use the same class name but we do not compare all the tags to each other like most of the existing systems did.

3.3 Proposed WebOMiner_S Architecture and Algorithm:

We have developed the architecture for extracting product data records from B2C websites automatically, we address it WebOMiner_Simple which is as shown in the figure 30 below:

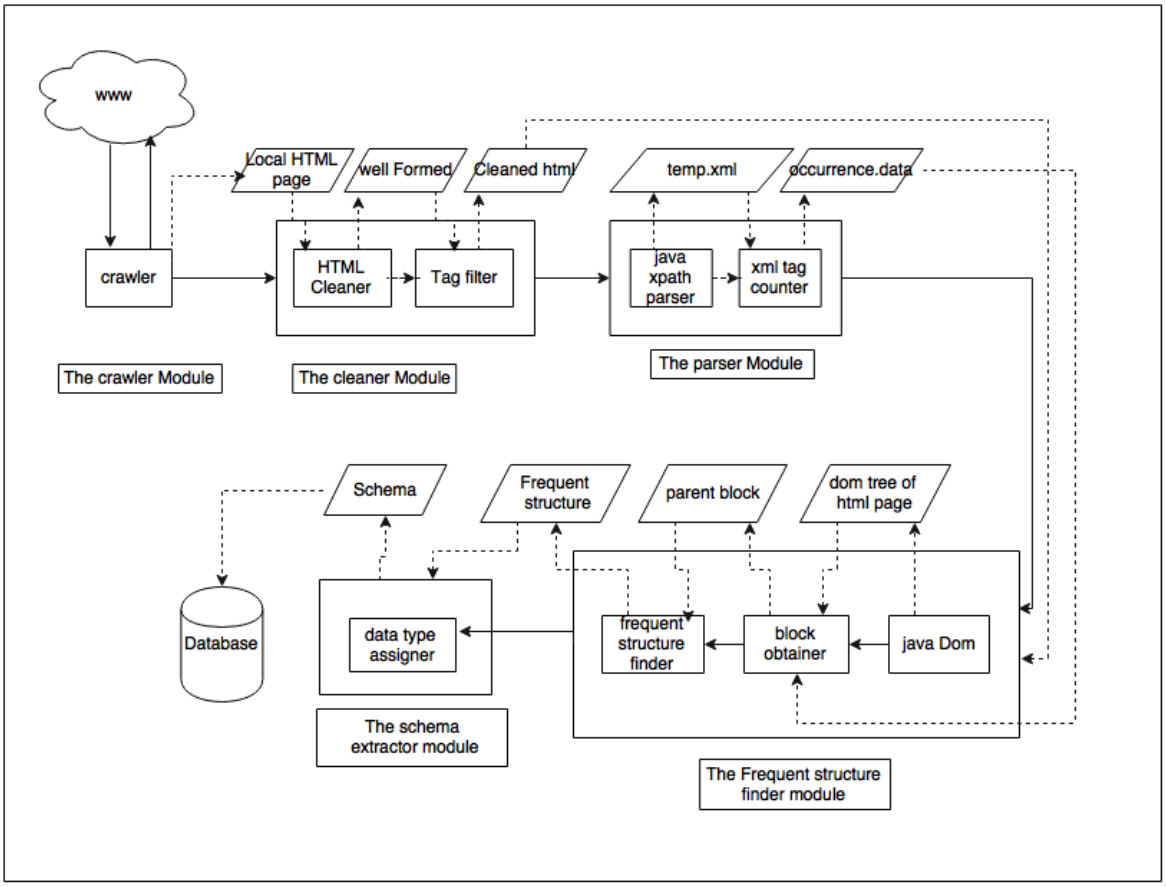


Figure 28: WebOMiner_S Architecture

The architecture in the figure 31 has 5 modules 1) The Crawler Module 2) The Cleaner Module 3) The Parser Module 4) The Frequent Structure Miner Module 5) The Schema Extractor Module. All of these modules are called sequentially by our main algorithm WebOMiner_S as shown in figure 30

Algorithm WebOMiner_S()

Input: Web addresses of two or more product listpages (Weburls).

Output: Set of Table Schemas (S) for each Web document.

Other: Set of HTML files (WDHTMLFile) of web documents,

WDHTMLFile# (set of clean HTML files),TempXMLFile, tagoccur file

Begin

(A) for each web address (Weburls)

Begin

(1) WDHTMLFile = Call the Crawler() to extract webpage (Weburls) HTML into local directory.

(2) WDHTMLFile_ = Call HTMLCLEANER-2.2 to clean-up HTML code

(3) Call the Parser() to first create temporary XML file of WDHTMLFile_ as Temp XML File, and next use the regular expressions on Temp XML File to summarize all tags's number of occurrence in a tagoccur file

(4) Call the Frequent Structure() finder to first create a DOM tree with DocumentBuilderFactory() using clean html file WDHTMLFile_ . Then, it uses the tagoccur file with the DOM tree to identify block tags and its class attribute which has the most frequent similar repeating sequence of tags.

(5) Call Schema Extractor() which uses the frequent structure and the DOM tree to associate each data retrieved with data type and to create the discovered table schema in a database and insert data into it.

End for

Figure 29: WebOMiner_S Main Algorithm

Working of the algorithm and all the modules will be discussed and explained below.

3.3.1 The Crawler Module:

Our crawler module is responsible for extracting the HTML source code of the given product web page from the World Wide Web (WWW). This module takes as its input the web page address (URL), using which it extracts and stores its HTML source code in a file in the local computer directory as uncleaned.html. Our crawler module is similar to the WebOMiner's . They (the WebOMiner system) have proposed a mini-crawler algorithm which crawls through the WWW to find targeted web page, streams entire web document including tags, texts and image contents and

```
Algorithm Crawler()

Input: Web addresse of a product list page (Weburl).

Output: Html page of the input URL as uncleaned.html in local directory

Begin

    (1) URL url = new URL(inputUrl)          //Create URL object and assign the input url
    (2) URLConnection urlConnection = url.openConnection() //Create a url connection
        object
    (3) BufferedReader bufferedReader = new BufferedReader(new
        InputStreamReader(urlConnection.getInputStream()))
                                                 //Wrap the url connection in a buffered
                                                 reader
    (4) while ((line = bufferedReader.readLine()) != null)
          {
           content.append(line + "\n")
         }
                                                 //Read the URL connection via buffered
                                                 reader
    (5) BufferedWriter writer = new BufferedWriter(new FileWriter(uncleaned.html)
           If writer is not NULL
           Writer.write(content)
        //Write content into file uncleaned.html using bufferedwriter
    (6) Save File uncleaned.html.
```

Figure 30: Algorithm crawler

it then creates a mirror of original web document in the local computer directory. It dumps the comments from the html document. Figure 34 shows a sample input web page to our crawler, which is a product rich web page from the bestbuy.ca website of the URL www.bestbuy.ca/laptops.html

which is the input to our crawler module. Our algorithm creates a URL object and assigns the input URL to it and then creates a URLConnection object to establish connection to the server using the method OpenConnection(). As the server responds to the connection with the html page our algorithm wraps it into a buffered reader and reads all the contents received from the server into a variable and finally writes it into a file called uncleaned.html saving it in the local directory. With this step we have our html source code to process. Figure 35 shows the html source code of the input web page starting from the tag <html> and ends with </html> in between all the contents of the web page are in respective tags.

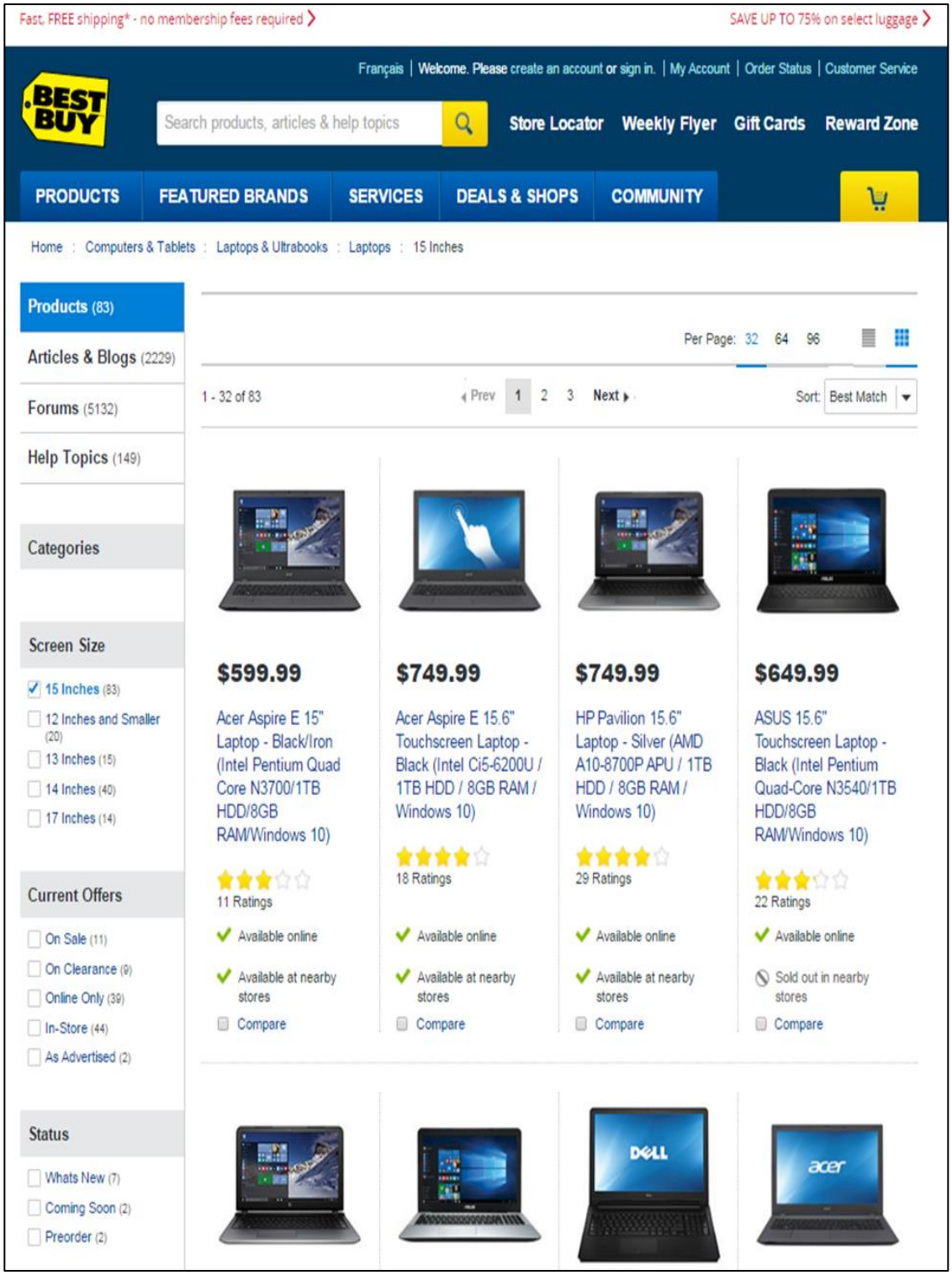


Figure 31: Input web page

```
<html>
        <head>
                <title>bestbuy</title>
        </head>
        <body>
        ……
        ……..
 <li class="top-lvl menu-item parent-item parent-services"><a href="javascript:void(0)" class="link-top link- services">SERVICES</a>
    <ul class="sub-nav mega-menu menu-list">
      <li class="menu-item">
        <div class="services-content">
          <ul class="column-4">
            <li class="menu-item first">
              <a class="icon icon-gs" href="/en-CA/GeekSquad.aspx?NVID=Services;Geek%20Squad;im;en">Geek Squad</a>
              <p>From computer glitches to smartphone mishaps, Geek Squad can save the day. Our Agents are ready and waiting to set up, support, protect and repair.</p>
              <a href="/en-CA/category/geek-squad/22042a.aspx?NVID=Services;Geek%20Squad;im;en">Learn more &gt;</a>
               <div class="prod-saving">
        ……..
        ……..
        ……..
        ……
        ……..
        </body>
</html>
```

Figure 32: uncleaned.html

3.3.2 The Cleaner Module:

Html is not a structured language, tags not being closed or missing tags, improper ordering are very common in the html source code of web pages. It is not possible to build Dom tree from ill-formatted html code, so the raw html source code in the local directory as uncleaned.html has to be cleaned before any further processing. The idea of cleaning the source code had been inherited from the WebOMiner, however we have used a much advanced cleaner for this purpose. JTidy is an open source html parser in java that is used to clean the malformed html to insert missing tags, reorder tags, attributes or text, remove comments and convert it into a well formed Xhtml.

```
.....

....

<div class="welcome">

        <div class="Banner">

         <div class="deal">

           <p>Hot clearout DEALS are <br /> going fast</p>

           <a href="/en-ca/clearance-outlet.aspx?NVID=deals;clearance;lnk;c1;r3;en" class="link">Shop
Now</a>

         </div>

        </div>

        <div class="Banner">

         <div class="deals-dropdown-image-clearance deals-dropdown-image"></div>

        </div>

       </div>

       <div class="row  deals-dropdown-list-item bby-style-wrapper">

        <div class="small-8 columns">

          <div class="deals-dropdown-list-text one-line-of-text">

           <p>DEALS delivered to your inbox - never miss out</p>

           <a href="/en-CA/secure/newsletter-signup.aspx?NVID=deals;newsletter;lnk;c1;r4;en"
class="link">Sign Up Now</a>

          </div>

        </div>

         <div class="small-4 columns">

         <div class="deals-dropdown-image-newsletter deals-dropdown-image"></div>
```

Figure 33: Clean Html File

The input to the cleaner module is the uncleaned.html file, it is processed by JTidy according to the properties set and the cleaned Xhtml file will be saved in the local directory as cleaned.xhtml. Figure 36 highlights the snippet cleaned Xhtml file that has been generated after cleaning the source html file.

3.3.3: The Parser Module:

There is need to convert our clean html file from step 2 to a temporary XML file containing only block level tags so that we can evaluate the tags to retrieve their number of occurrences. The parser module takes as input the cleaned html file from the previous module. A temporary xml file (such as Figure 39) is created by recording all the block level tags such as , <table>, <tr>, , <li>, <ul> etc., from the input html page. In the running example, tags such as

, etc., from the cleaned.xhtml file are recorded into temporary xml file. While creating the temporary xml file, all the attributes in these block level tags like “id”, “src” etc., were excluded and only the class attribute is retained such as , in the running example as the class attribute holds the information regarding which particular class of CSS is being used by this tags which indeed helps us find the block that has been occurring frequently with the same style. Figure 39 shows the temporary xml file for the sample input page from the previous modules, where all the block tags were recorded with only class attributes. The algorithm now calls the search_string() method which uses java regular expression parsing to check the occurrence of each tag with the corresponding class attribute in the entire temporary xml file. Thus, the summary of found repeated tags (those occurring at least 3 times since it is uncommon to have B2C web sites listing less than 3 products) along with the number of occurrences are stored into a tag occurrence file. In the running example has occurred 1 time, the tag >div class= “banner”> has occurred 2 times in the temp.xml file followed by the other repeated tags. Figure 40 shows the tag occur file for the running example with all the block tags and their occurrences.

```
Algorithm parser()
{
Input cleaned html file
Output Temporary xml file with block tags
        Occurrence Data file with count of tag occurrences
Begin
1) Arraylist occurnodelist[ ]
2) For each line in cleaned html file
3)      IF line is a block tag
4)      Write line into temporary xml file
5)      End IF
6) End For
7) For each line in temporary xml
8)      Call function searchString() by passing temporary xml file and line as  parameter
9)      it returns no of occurrences of that line
10)     If occurnodelist[ ] does not contain line
11)     Add line to occurnodelist[ ]
12)     Write line and its number of occurrences into a file occurrence.data
13)End For
End
```

Figure 34: Algorithm parser

Algorithm Searchtring()

Input: line from temporary xml file

Output: no of occurrences of the given line

Begin

1) Create object to Java regular expression pattern matcher()

2) Create a matcher object

3) For each line in temporary xml

4) If Line not in occurnodelist[]

5) Using matcher object check if the current line in temporary xml

Matches the input line

6) Increment count

7) End If

8) Return count;

9) End

Figure 35: Algortihm search_string

```
<divclass=“welcome">
<divclass=“bannar">
<divclass="supporting-info">
<divclass=“product”>
<divclass=“deal”>
<divclass= “prodwrap”>
<divclass=“prod-im”>
<divclass=“p-img”>
……..
……..
……..
<divclass=“review”>
```

Figure 36: Temporary Xml file

```
<divclass=“product">1
<divclass=“bannar">2
<divclass="supporting-info">4
<divclass=“welcome”>1
<divclass=“deal”>4
<divclass= “prodwrap”>12
<divclass=“prod-im”>12
<divclass=“p-img”>12
……..
……..

……
……..
<divclass=“review”>9
```

Figure 37: Occurrence data file

3.3.4: The Frequent Structure Finder Module

The aim of the frequent structure finder module is to find the most frequently repeated blocks with more number of children in the web page that are styled similar. These features are represented in html source code as finding the frequently repeated block tags with same class name in the entire Dom tree which are our product data blocks. The nodelist struct_list is used to hold the candidate product blocks, nodelength holds the number of occurrences of the candidate product block, no of childs holds the number of children for the product block and initially both the values are zero. The NodeList data_tuples holds the Final Product block. First, Dom tree is built from the cleaned.xhtml file generated in the second module. This Dom is the tree representation of the entire html source code discussed in section 1.6 and a model Dom tree is used to present the algorithm in an understandable way in figure 41. Another input to the fs_finder algorithm is the tag occur file from the parser. The tag occur file holds all the block tags with their class name and number of occurrences.

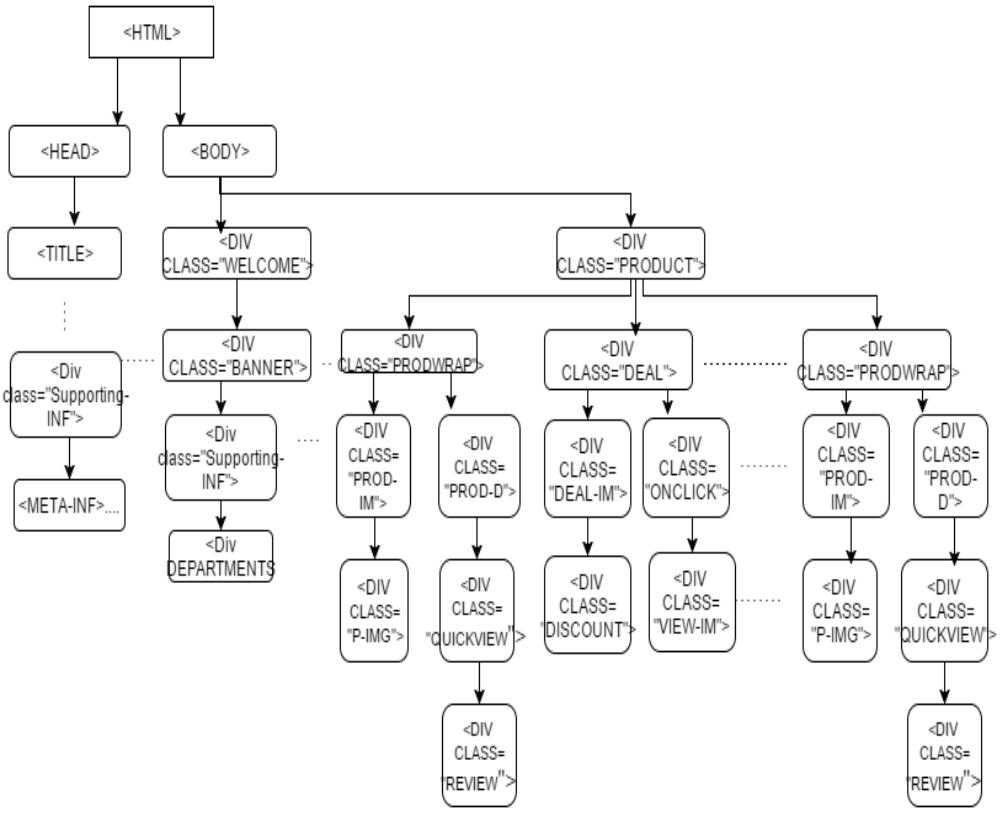


Figure 38: Dom tree

For every tag in the tag occur file that has number of occurrences at least 3(assuming a product list web page has at least 3 products), a dynamic Xpath expression is generated and the Dom tree is queried to find the occurrences of this tag. For the running example the tag <divclass=“product">

is omitted as its count is less than 3, so is . The tag <div class=” supporting-info”> has occurred 4 times, as it satisfied the condition the Xpath expression is generated as path = "//div[@class='supporting-info']" and every occurrence of this tag all over the Dom tree is found and assigned into a nodelist(figure 42) to find the number of occurrences using the method getlength() and number of child nodes using the method getchildnodes() while they are assigned to nodelength =4 and no_of_childs= 1 respectively. Note that tag has occurred 4 times all over the Dom tree, as we cannot show the entire tree only 1 occurrence of it is visible in the snippet Figure 43

<div class= ” supporting-info”>	<div class= ” supporting-info”>	<div class= ” supporting-info”>	<div class= ” supporting-info”>

Figure 39: Nodelist struct_List

The next tag <divclass=“deal”> is also processed in the same way, by generating an Xpath expression path = "//div[@class='deal']" every occurrence is stored in a NodeList and the values of nodelength is 4 while number of childs is 4 (as shown in figure 44) these values are greater than the previous so the data_tuples is updated to . The next tag is <divclass= “prodwrap”> has nodelength= 12 and number of childs= 5 (figure 45) so the data_tuples is now updated to . The next is <divclass=“prod-im”> with nodelength=12, number of childs = 1 our condition that product block should have maximum children and should be repeated maximum number of times will not be satisfied with this tag because though the number occurrences in nodelength is equal the number of childs is lower so this tag is not considered and values are not updated. The same process continues until the end of tag occur file thus finding the most frequently repeated block with more number of children and the discovered product block in the NodeList data_tuples. In the running example the product block discovered by the algorithm is having 5 Childs and occurrences 12.

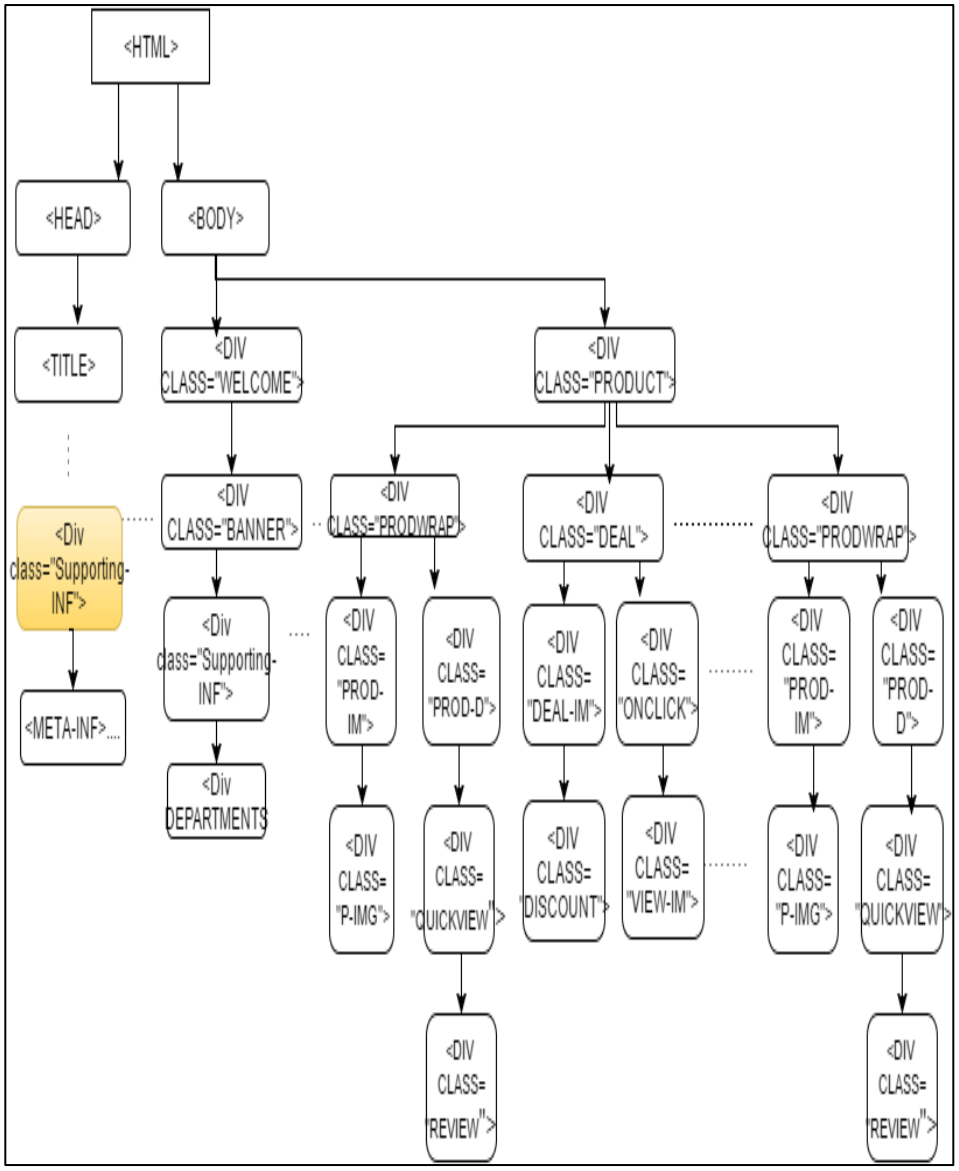


***Figure 40: Dom tree ***

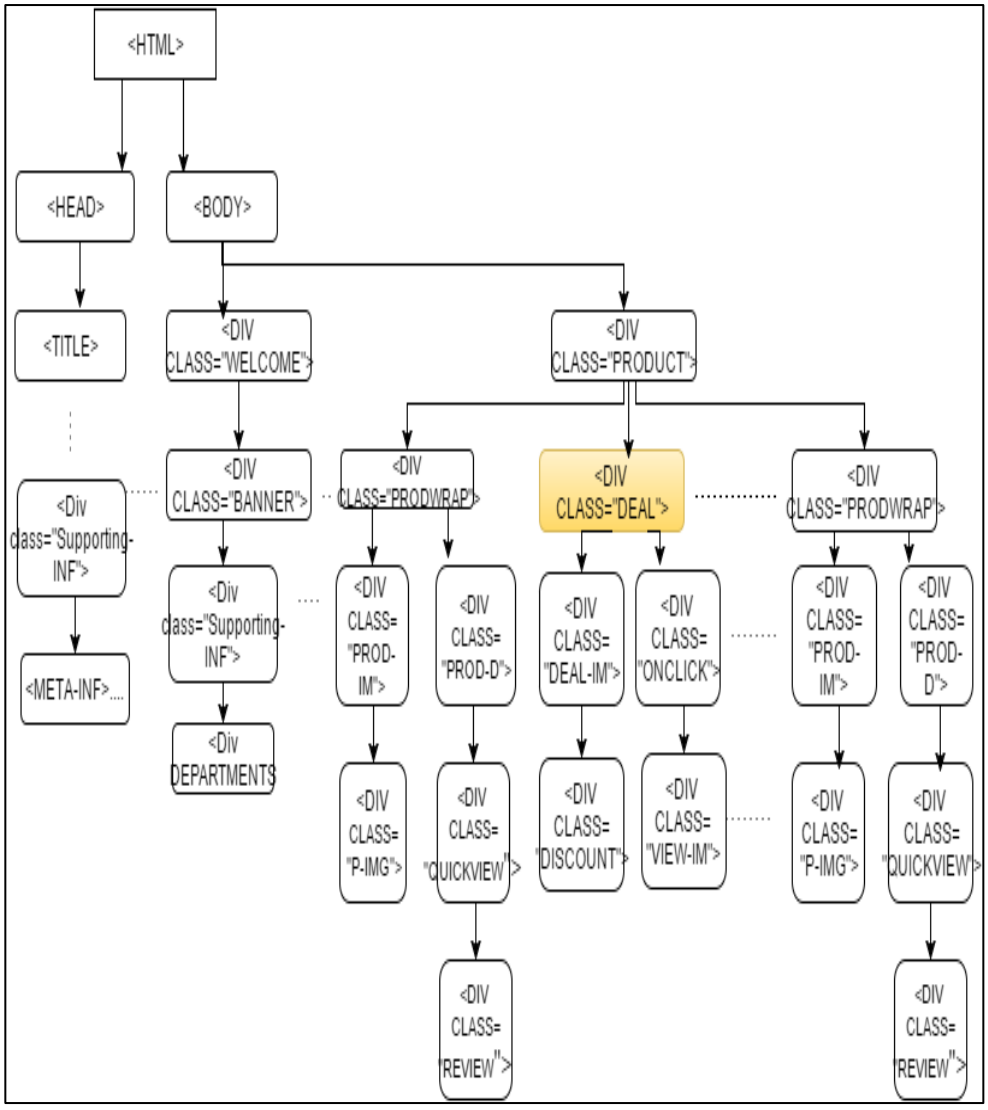


***Figure 41: Dom tree ***

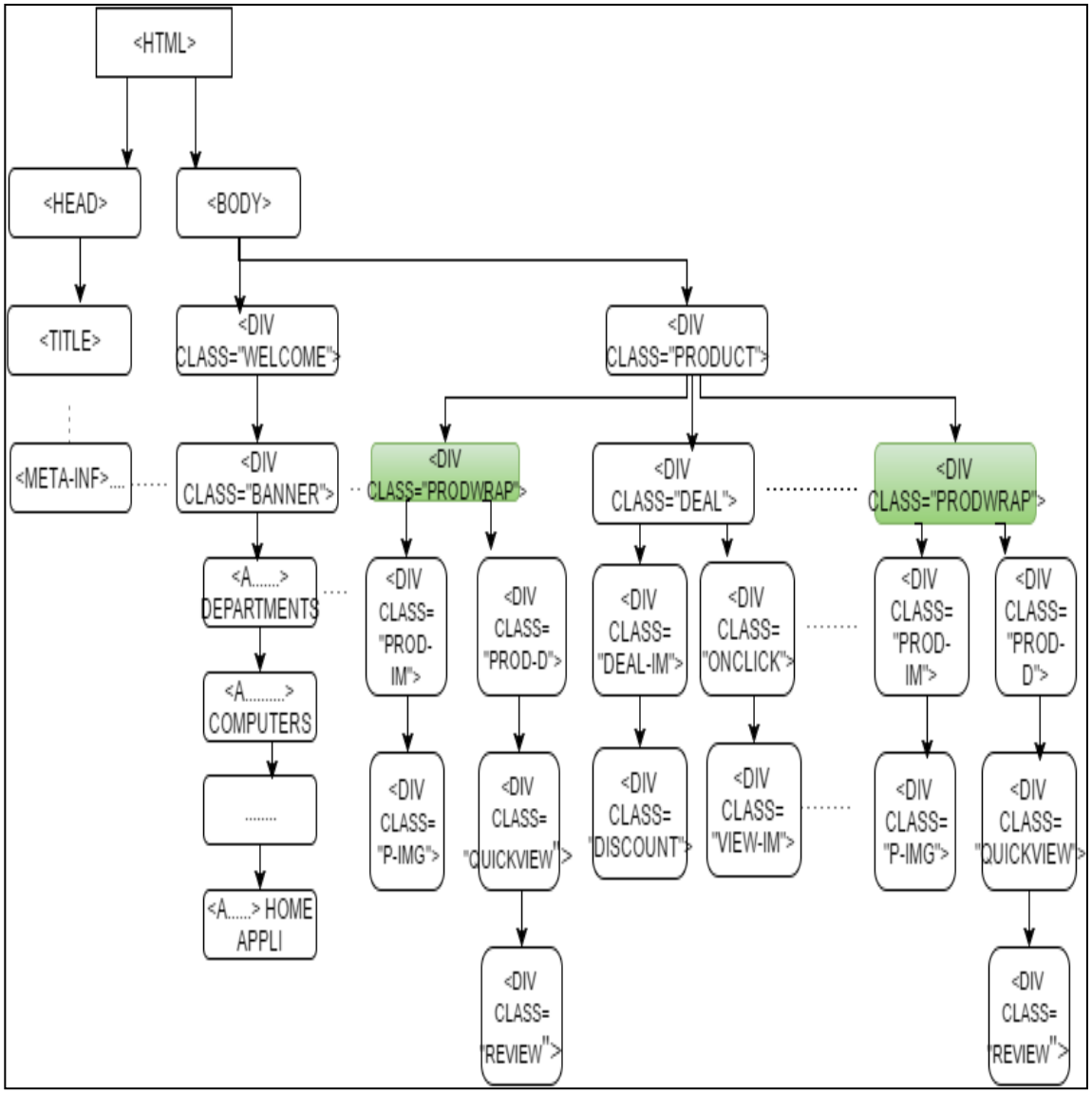


***Figure 42: Dom tree with ***

Using the parent tag its complete block is retrieved from the Dom tree, in our running example the tag all of its children are retrieved and stored in NodeList to discover schema of the product block.

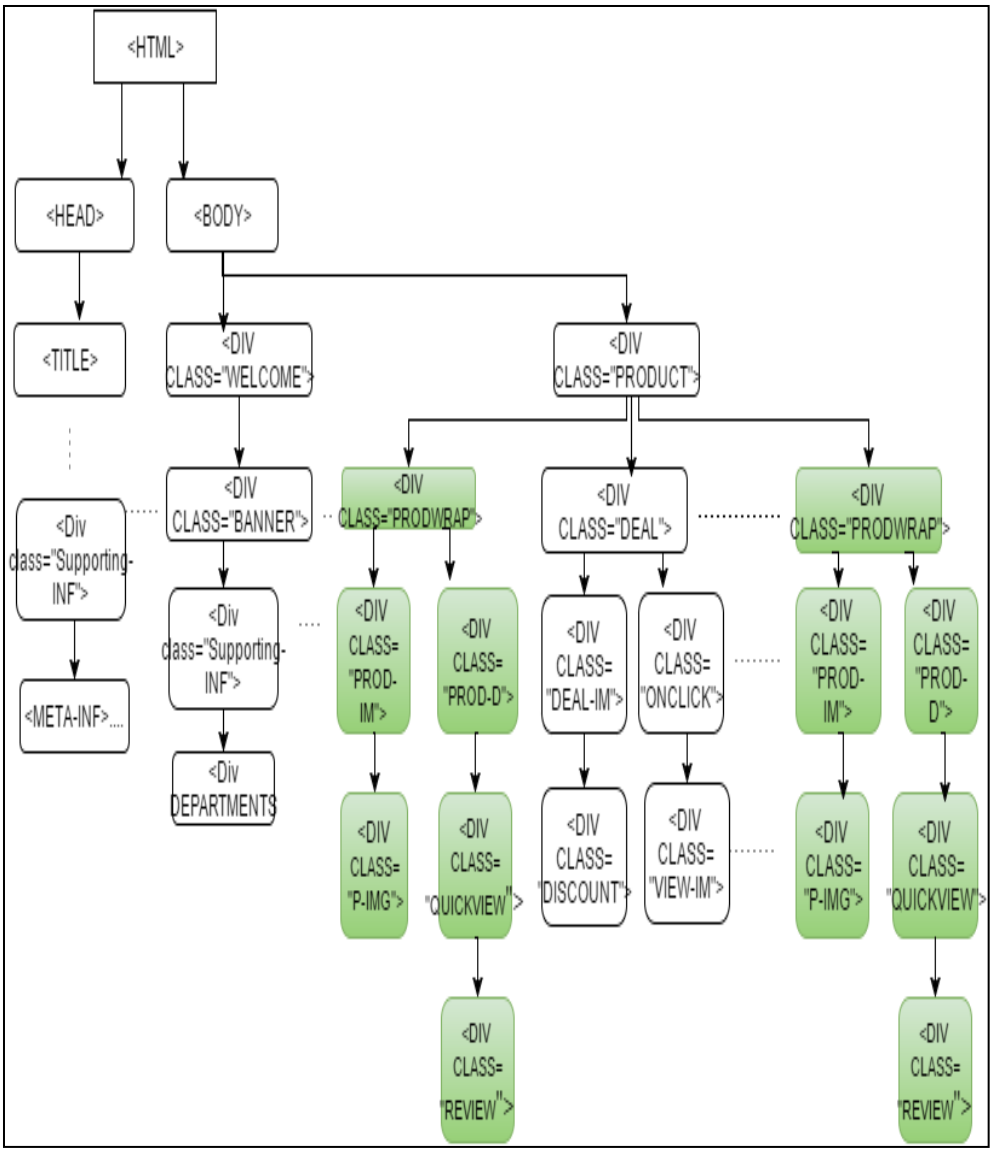


Figure 43: Dom tree product blocks

For the product parent tag all of its children are retrieved in depth first search are as the Dom tree structure represented in figure 47 and are saved in NodeList.

Algorithm FSfinder()

Input: occurrence data file (occur), cleaned html file (clean)

Output: Nodelist frequent structure (FRS)=null(for product blocks names), data tuples(for product blocks values)

other: Domtree: file

FINAL nodelength = 0, no of childs = 0, final childs = 0

xmlnodetag: string (for complete block node),

xpathExpression expr: string, path : string

previous product block: integer

Nodelist Tagstructure, struct list, data tuple, product block

xmlnodetag IDCount = 0, nodes length: integer

Packages:Java DOM, Java Xpath, Java transformer

begin

1. Domtree = Call DocumentBuilderFactory(clean)

2. Create object xpath for xpathfactory() to use its methods to retrieve all nodes that match the path expression from Dom tree.

3. for each xml tag in tag occurrence file occur,

    3.1 Retrieved xml is stored into xmlnodetag

    3.2 path = "//div[@class='"+xmlnodetag+"']"; (The class attribute in xmlnodetag is assigned to the variable dynamically in each loop)

    3.3 xpathExpression expr =xpath.compile(path)

    3.4 struct list= xpath.evaluate(Domtree, XPathConstants.NODESET)

    // This retrieves all the nodes that match the path expression using xpath object for this xml

    3.5 node length = struct list.getlength()

    3.6 no of childs= struct list(0).getchildnodes().getlength()

    3.7 If (node length >= FINAL nodelength and no-of-childs > final childs)

    3.7.1 product block = xmlnodetag

    3.7.2 FINAL nodelength=node length

    3.7.3 final childs = no of childs

```
        3.7.4 data tuple=struct list
        end for
        4. if product block has child
5. for each child node in product block
Begin
        5.1 FRS = FRS append child node to the frequent structure
        5.2 Read next node
end For
6. Return  data tuple
end
```

Figure 44: Algorithm fs_finder

```
< divclass = “prodwrap’’ >
      <h4 class="prodTitle">
            <a
id="ctl00_CP_ctl00_C3_ctl00_LT_SL_ctl01_ctl00_CalltoAction"
href="http://www.bestbuy.ca/en-CA/amd-a-series.aspx">Asus 5’6
laptop</a>
       < divclass = “prodImage’’ >
           < divclass = “prod − image’’ >
          <img src=“bestbuy/1043.img”> < /div > < /div >
< divclass = “prodDetails’’ >
   <h4 class=“prod-Ram”>500GB</h4>
    <h4 class=“prod-HDD”>4GB</h4>
     <h4 class=“operat”>Windows 10</h4> < /div >
       < divclass = “ProdPrice’’ >
          <h1>$678”</h1> < /div >
        < /div > < divclass = “clear’’ > . . . < /div > < /div >
```

Figure 45: Frequent Structure

3.*3.5 The Schema Extractor Module:*

This module performs the task of extracting both the table schema and the product tuples from the frequent structure (sequence of tags). It converts the schema of the database table retrieved by the FSFinder algorithm to a data warehouse schema by appending an integration attribute called "storeid", and a history attribute called "time". Those two attributes will enable integration of tuples from different web sites at different timestamps for comparative and historical querying. A product tuple P is a product with attributes a1, a2, a3, . . ., an. This module takes as input both the frequent structure pattern and the nodelist data_tuples that has all the product data blocks which has to be extracted and stored into database. The schema extractor assigns corresponding datatypes based on the tags to the frequent structure pattern for the table schema. If it is an < img > tag the data type image will be assigned, if it is any of < p >, < span >, < li >, < a > tags, data type "string" will be assigned, etc. Finally, the data warehouse schema is discovered from the web page. This can be used to create and update a database table and further creating a data warehouse when an integrative attribute "storeid" and a historical attribute "time" are appended to the table schema. Having a data warehouse schema allows for comparative mining and historical querying. Table 2 shows the table schema extracted from the running example. Once the database or warehouse schema is defined, the actual data tuple instances is extracted from the nodelist data_tuples which holds all the product blocks with their parent nodes. From the Dom tree, each parent node is checked if it has child nodes and contents are updated into the database/warehouse.

```
Algorithm schemaFinder()
Input: Frequent Structure
Output: Data Base table with product records
Begin
        1) String Web_Schema
        2) FOR Each tag IN frequent structure
        3)   IF tag = img
        4)          Assign data type blob and add this to web schema
        5)   ELSE IF tag = text
        6)          Assign data type varchar and add this to web schema
        7)   END IF
        8) END FOR
        9) Create Table web_Schema
End
    1)  For Each product block in data_tuples
    2)      For Each Tag in product block
    3)          If tag = img
    4)                  Content += Retrieve src || data-url
    5)          Else if tag = text
    6)                  Content += retrived text data
    7)          End if
    8)       End For
    9)      Insert content into database table
    10) End For
```

Figure 46: Schema Extractor

Discovered schema for a single page of bestbuyWebsite is below:

Product_table(column_1: String, column_2: String, Column_3: String, Column_4: String, Column_5: String, Column_6: String, Column_7: String, date : date, store_id: String)

Figure 47: Discovered Schema

Store id	**Time**	**Column_1**	**Column_2**	**Column_3**	**Column_4**	**Column_5**	**Column_6**	**Column_7**	**Column_8**
bestbuy_1	31-05-2016 11:00	Asus 5'6 laptop	bestbuy/1043.img	I5 processor	4GB	Looks good	$678	500GB	3
…	….	….	….	…	…	….	…	….	….

Figure 48: Dtabase table generated

CHAPTER4: EVALUATION OF WEBOMINER_S SYSTEM

We have performed the implementation of our algorithms and have presented them in the most robust, scalable and generalized form. More improvement is required in our algorithms of our system to make them robust enough to handle various kinds of complex commercial websites and their structures. Our crawler module needs to be automated such that the positive web pages can be discovered without human intervention. Our system uses JTidy to clean the raw html pages but developing our own focused cleaner to remove noise and clean the webpage can reduce the overload on the followed modules in terms of comparisons and IO. However, we have discussed our preprocessing and limitations in detail in the appendix section 7.0. At this point we have tested our system on the real time web pages from different websites with different structures and domains to showcase our robustness and usability.

4.1. Empirical Evaluations

We have tested our system on 50 real time web pages each belonging to a different website having various products and has different structure. The product data records from each website belong to different domains such as electronics, music, hardware, travel, apparel, accessories etc. Our system is implemented in java programming language. We then run our system in 32-bit windows 10 operating system, 8 GB RAM with an Intel core i5 2.60GHz processor Lenovo ThinkPad Machine. We have used the standard precision and recall to evaluate our system for both number of data records to be retrieved and also their number of columns in schema retrieved. Precision is measured as the average in percentage of the number of correct product blocks retrieved to the retrieved number of product blocks by the system, while the recall is measured as the average in percentage of the number of correct product blocks retrieved to the actual number of product data blocks in the web page. Our results were tabulated in the Table 1 below

Product web page	Actual Product Data Values		Extracted Product Data Values			
	No of Product blocks	Schema fields	Discovered Product Blocks (correct)	Discovered Schema fields	Discovered Missing Schema fields	Discovered Irrelevant Schema fiends
www.bestbuy.ca	32	5	32	7	0	2
www.guess.ca	12	5	12	5	0	0
www.costco.ca	25	7	25	7	0	0
www.tripadvisor.ca	44	6	44	6	0	0
www.musiciansfriend.ca	19	6	19	6	0	0
www.homehardware.ca	15	5	15	5	0	0
www.michaelkors.ca	22	5	22	6	0	1
www.sony.ca	34	8	34	8	0	0
www.toysrus.ca	19	6	19	6	0	0
www.Ebates.ca	7	6	7	6	0	0
www.tobi.com	90	6	90	6	0	0
www.iga.net	20	5	6	6	0	1
www.rietmans.ca	12	6	12	6	0	0
www.shopclues.ca	36	4	36	4	0	0
www.lacoste.com	37	6	37	6	0	0

www.apple.com	15	8	15	10	0	2
www.hollister.ca	27	10	27	11	0	1
www.abercrombie.ca	21	9	21	9	0	0
www.ebags.ca	13	7	13	7	0	0
www.katespade.ca	27	5	27	5	0	0
www.inetvideo.ca	11	10	11	9	0	1
www.shot.tcm.com	25	15	25	16	0	1
www.nationalgeagraphic.com/store.com	13	5	13	5	0	0
www.disneystore.com	96	7	96	7	0	0
www.industrykart.com	12	16	12	16	0	0
www.guitarcenter.com	17	9	17	10	0	1
www.votawtool.ca	100	9	100	9	0	0
www.fossil.ca	20	7	20	7	0	0
www.ninewest.ca	49	7	49	7	0	0
www.dkny.ca	12	5	12	5	0	0
www.shop.ca	11	4	11	4	0	0
www.canadacomps.ca	10	6	10	6	0	0
www.evine.com	23	9	23	10	0	1
www.visions.ca	8	7	8	8	0	1
www.shopmyexchange.com	27	8	27	8	0	0
www.creatronic.com www.rakuten.com	16	7	16	7	0	0
www.sweetwater.com	37	6	37	6	0	0

www.earlymusicshop.com	33	9	33	10	0	1
www.1-800-bakery.com	19	7	19	7	0	0
www.eaglemusicshop.com	14	7	14	7	0	0
www.albertsonsorders.mygrocer.com	26	8	26	8	0	0
www.goboplay.com	19	6	19	6	0	0
www.vistek.ca	162	5	162	5	0	0
www.framesdirect.ca	12	5	12	5	0	0
www.ca.oakley.com	23	7	23	7	0	0
www.ethoswatches.com	103	6	103	6	0	0
www.thegadgetshop.ca	11	6	11	6	0	0
www.shurecanada.ca	23	7	23	7	0	0
www.ilookglasses.com	30	7	30	8	0	1
Total	1489	342	1489	358	0	16
Results	**Precision**				**Recall**	
Product data records	100%				100%	
Schema Fields	95.55%				100%	

Table 1: Experimental Results

4.2 Experimental Results

The purpose of our experiment is to measure the performance of WebOMiner_S system for data record extraction. Table 1 shows large scale experimental results as performance measure for our WebOMiner_S system. We have taken one page per each website for experimentation and the numbers in "Actual data records" column shows the number of product blocks in the web page including the number of columns in the schema per each web page. The other column "extracted

number of data records" shows the number of records that the system has identified as product data records and the number columns of data it has extracted. The row total indicates that when there were 1489 data records the system has extracted all of them exactly and it did not miss out any target data nor did it extract any noise. While the algorithm has also performed well in extracting the schemas in out of 342 columns in total it has extracted all of them, however it has extracted 16 irrelevant columns for 14 webpage schemas that does not have much meaning in the product database.

We observed that the reason for these irrelevant columns is the noise that has been embedded into the product block in the web page, here noise means textual data in the product block itself that does not belong as an attribute to the product object. For example, check box below each product enabling users to compare other products has the text tag value as "comparison", "select comparisons" or "quick-view" etc., which is noise bit as it this is inside the product data block we do not have any methods to remove this detect and remove this noise. For instance as shown in the figure 50.

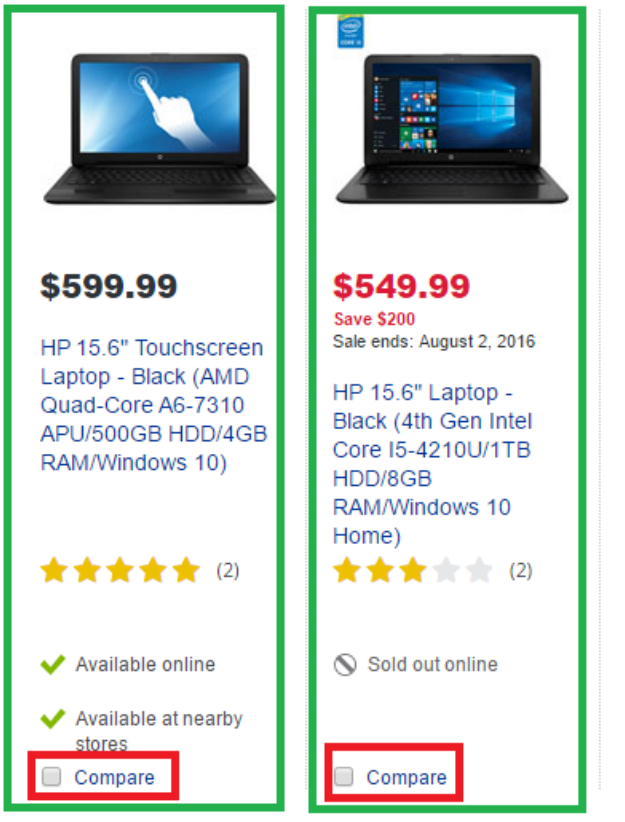


Figure 49: noise extracted

4.3. Comparison with the WebOMiner

Our system performs better than the WebOMiner in terms of scalability and extendibility which are the two key aspects of web data extractors. Scalability is very important to a data extraction system since the World Wide Web is massive and mining target data from it should be possible in feasible amount of time also being accurate.

1. The WebOMiner sequentially compares each NFA with the content object in the content object array and it has 6 types of NFA's for a single webpage their system is highly complex and takes computationally more amount of time compared to WebOMiner_S. Also the WebOMiner performs the unnecessary computations as it does not just extract the product tuples but also the other 5 content type objects which are not at all fetched by our system. We have tested performance of both the systems on a large e-commerce website (www.bestbuy.ca) which has over 16 categories of products (baby, electronics, clothing etc.) each category with again 15 subcategory(electronics -> laptops, cellphones etc.) with an average of 80 products each web page there are around 20,000 products. Our system took 838 milliseconds to identify, extract and create data warehouse table for a web page of 32 products whereas the WebOMiner took 1276 milliseconds to update already created database. In the next case to check if

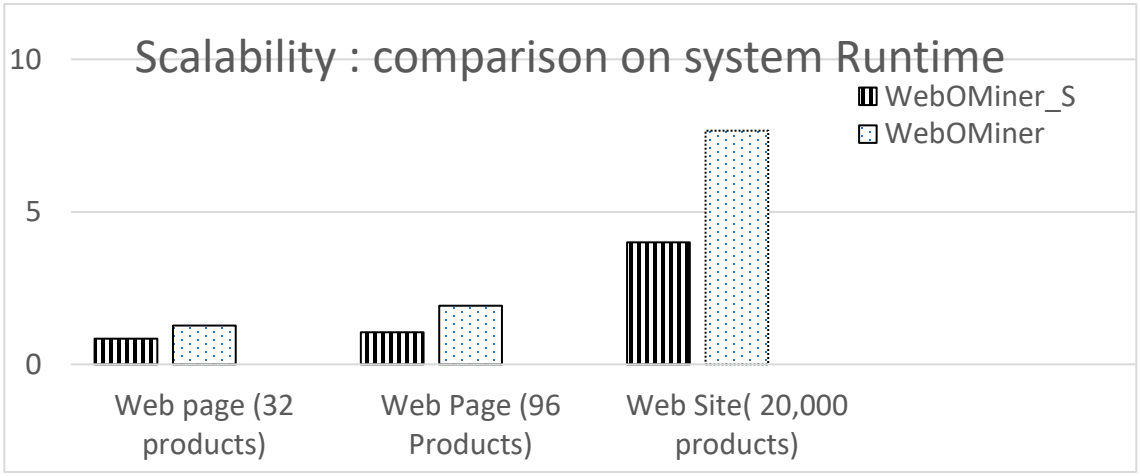


Figure 50: Comparison on system runtime

the presence of more products on one web page increases the run time we have tested both systems on web page with 96 products, the results have shown only a slight increase in runtime in our system while almost double increase on WebOMIner runtime. Finally, to test the scalability we have ran both the systems on an entire large website (www.bestbuy.ca with 20,000 products) proving our system is highly scalable than the WebOMiner(7.66 minutes) as it takes nearly only half the time 4.00 minutes.

2. Dependency in the process of extraction, The WebOMiner trains its NFA's to identify the product tuples thus creating a dependency between the previously fetched product data and the new ones which makes it inapplicable to varied domains but the proposed system is completely independent and dynamic in data extraction.

3. In terms of performance and correctness system has achieved 100% precision and recall in identifying product data blocks. On the other hand our schema discovered has 95.55% precision and recall 100% in discovering schema of the web pages where as WebOMiner is not capable of fetching schema automatically rather their schema is predefined and same for every product.

4.4 Limitations

1. The product tables generated dynamically does not have appropriate column labels or specific ordering of data columns rather for each web page based on their frequent structure their schema is discovered. It is limitation because, in order to integrate product data from multiple tables (bestbuy, Costco) the specific columns are needed (e.g. price from bestbuy and price from Costco) cannot be discovered automatically to integrate as the system will not know which column is price. In order to overcome this column names has to be given appropriately. This problem can be solved using some ad-hoc features such as 1) If all the data items in the column contains "$" sign name the column as "price" and assign datatype integer. 2) If the length of the data in the column is greater than 18 name it as "description". If the data contains one or two digits without any $ sign name it as "ratings".

2. Our algorithm retrieves in block noise which is, the information present in the product block that is not an attribute of product. Example "compare" in bestbuy web page for each product. Such in-block noise are same for every product data record. Which means for every product data record the data of the column will be same "compare". In order to remove this using regular expressions to compare each data item in that column and if they are same it indicates that this column is noise and it has to be deleted from the table.

CHAPTER 5: CONCLUSION AND FUTURE WORK

This thesis proposes the system WebOMiner_Simple that uses an easy to understand and less complex technique to improvise the WebOMiner which uses the technique of using NFA's to extract product data from b2c websites. Also the WebOMiner could not address the issue of complete automation in web data extraction systems which the proposed system has solved by dynamically discovering the schema for each given web page and does not need any training data. Our architecture has 5 modules, the crawler, cleaner, parser, frequentstructure_finder and the schema extractor module. We have modified algorithms for crawler initially given by the WebOMiner, modified freeware software "JTidy" for the cleaner module and have developed algorithms for the parser, frquentstruture_finder module and the schema extractor module. In this thesis we have used the web content and web structure mining techniques to find the frequently repeated blocks of html tags to discover the product data and their schemas. In contrary to the WebOMiner we propose not to evaluate any of the tag attributes to find the product content instead finding the frequently repeated set of tags with maximum children and maximum repetition are our target product data blocks. In order to find this we have used the XPATH language to find out the block level tags with the same class name occurring all over the Dom tree of the web page. Once we discover product blocks using the FS_finder our schema extractor now automatically discovers the schema of the web page and stores in the database by appending the integration and historical attributes. We are currently working on development of the web application which can answer complex historical queries and can mine the data by integrating the tables generated which will be available to the public for performing comparative mining using the GUI.

5.1 Future Work

We feel that our effort of mining the product data records automatically in a simplistic way has a lot of room for improvement.

1. Our algorithms are only capable of extracting the meta data available on the "product list page", it can be modified and extended to extract complete product specifications and details from the "detail product page". The detail page can be identified by finding the link to it(<a> tag that leads to detail page) in the frequent structure generated by our fourth module and extending our algorithm to extract data from it.
2. Another improvising to our system can be in the crawler module by automatically identifying the target product page using a focused crawler. Web crawling and target page identification has its own vast radius in research so developing an automatic product page crawler would be quite challenging yet very useful. In order to identify the positive product list web page the number of in links and out links of the page can be counted and the positive

page is the one with more number of out links as each represents a detail page of each product.

3. We are using the open source software JTIDY to clean the raw html this cleaning phase is more generalized and does not give much scope for us to remove unnecessary noise at this phase itself, modifying this module to achieve such purposes will lead to further better system.
4. One of the main limitation of our system is, We have implemented our system on static html pages to try to achieve the goal of data extraction in simplistic way now this can be extended to “onclick” and “onload” html pages that are built on the users click using technologies like ajax and jquery etc., This is one of the major improvement that the system needs. In order to achieve this an embedded browser component can be used. The embedded browser can be plugged into the current system it them loads web page requested by the user including all the java-script and ajax functions, scriplets and the html code generated after calling the functions can be found from the embedded browser component. Now this html code can be given as input to our system to extract product data. Some of the well-known embedded browser components are “web developer toolbar”, “firebug” etc.,
5. Now that we have created the data warehouse of products further mining and querying functionalities are to be extended to analyze the database tables and to perform the comparative analysis and also to discover potentially useful patterns based on the historical data and thus to make predictions on future data thereby being able to recommend users with appropriate products on the time.

REFERENCES

Adelberg, B. (1998). NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. *ACM Sigmod Record.* ACM.

Annoni, E., & Ezeife, C. (2009). Modeling Web Documents as Objects for Automatic Web Content Extraction-Object-oriented Web Data Model. *ICEIS (1),* (pp. 91-100).

Arlotta, L., Crescenzi, V., Mecca, G., & Merialdo, P. (2003). Automatic annotation of data extracted from large Web sites. *WebDB* (pp. 7-12). Citeseer.

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). Visual web information extraction with lixto. *VLDB,* (pp. 119-128).

Bhavik, P. (2010). A survey of the comparison shopping agent-based decision support systems. *Journal of Electronic Commerce Research*, 178.

Chakrabarti, S. (2000). Data mining for hypertext: A tutorial survey. *ACM SIGKDD Explorations Newsletter*, 1-11.

Chang, C.-H., & Lui, S.-C. (2001). IEPAD: information extraction based on pattern discovery. *Proceedings of the 10th international conference on World Wide Web* (pp. 681-688). ACM.

Chang, C.-H., Kayed, M., Girgis, M., & Shaala, K. (2006). A survey of web information extraction systems. *Knowledge and Data Engineering, IEEE Transactions on*, 1411-1428.

Cooley, R., Mobasher, B., & Srivastava, j. (1997). Web mining: Information and pattern discovery on the world wide web. *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on* (pp. 558-567). IEEE.

Crescenzi, V., Mecca, G., & MerialdO, P. (2001). Roadrunner: Towards automatic data extraction from large web sites. *VLDB,* (pp. 109-118).

Ezeife, C., & Mutsuddy, T. (2012). Towards comparative mining of web document objects with NFA: WebOMiner system. *International Journal of Data Warehousing and Mining (IJDWM),* 1-21.

Hammer, J., McHugh, J., McHugh, J., & Garcia-Molina, H. (1997). Semistructured Data: The TSIMMIS Experience. *In Proceedings of the 1st East-European*, 1-7.

Hammer, Joachim, J., McHugh, J., & Garcia-Molina, H. (1997). emistructured Data: The TSIMMIS Experience. *Advances in Databases and Information Systems.*

Hogue, A., & Karger, D. (2005). Thresher: automating the unwrapping of semantic content from the World Wide Web. *Proceedings of the 14th international conference on World Wide Web* (pp. 86-95). ACM.

Hsu, C.-N., & Dung, M.-T. (1998). Generating finite-state transducers for semi-structured data extraction from the web. *Information systems*, 521-538.

Kosala, R., & Blockeel, H. (2000). Web mining research: A survey. *ACM Sigkdd Explorations Newsletter*(1), 1-15.

Laender, A., Ribeiro-Neto, B., & da Silva, A. (2002). DEByE–data extraction by example. *Data & Knowledge Engineering*, 121-154.

Laender, A., Ribeiro-Neto, B., da Silva, A., & Teixeira, J. (2002). A brief survey of web data extraction tools. *ACM Sigmod Record*, 84-93.

Liu, B. (2007). *Web data mining: exploring hyperlinks, contents, and usage data.* Springer Science & Business Media.

Liu, B., & Chen-Chuan-Chang, K. (2004). Editorial: special issue on web content mining. *Acm Sigkdd explorations newsletter*, *6*(2), 1-4.

Meng, X., Hu, D., & Li, C. (2003). Schema-guided wrapper maintenance for web-data extraction. *Proceedings of the 5th ACM international workshop on Web information and data management* (pp. 1-8). ACM.

Muslea, I., Minton, S., & Knoblock, C. (2001). Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 93-114.

Muslea, I., Minton, S., & Knoblock, C. (1999). A hierarchical approach to wrapper induction. *Proceedings of the third annual conference on Autonomous Agents* (pp. 190-197). ACM.

Novotny, R., Vojtas, P., & Maruscak, D. (2009). Information extraction from web pages. *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web*

Intelligence and Intelligent Agent Technology-Volume 03 (pp. 121-124). IEEE Computer Society.

Srivastava, J., Cooley, R., Deshpande, M., & Tan, P.-N. (2000). Web usage mining: Discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter, 1*(2), 12-23.

Valter, C., & Giansalvatore, M. (1998). Grammars have exceptions. *Information Systems*, 539--565.

Wang, J., & Lochovsky, F. (2003). Data extraction and label assignment for web databases. *Proceedings of the 12th international conference on World Wide Web* (pp. 187-196). ACM.

Wu, S.-T., Li, Y., Xu, Y., Pham, B., & Chen, P. (2004). Automatic pattern-taxonomy extraction for web mining. *Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on* (pp. 242-248). IEEE.

Zaiane, O., & Han, J. (1998). Webml: Querying the world-wide web for resources and knowledge. *Proc. ACM CIKM'98 Workshop on Web Information and Data Management (WIDM'98).* Citeseer.

Zhai, Y., & Liu, B. (2005). Web data extraction based on partial tree alignment. *Proceedings of the 14th international conference on World Wide Web* (pp. 76-85). ACM.

Zhu, J., Nie, Z., Wen, J.-R., Zhang, B., & Ma, W.-Y. (2006). Simultaneous record detection and attribute labeling in web data extraction. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 494-503). ACM.

A P PE N D I X – A

WebOMiner_S System Manual

Developed by Bindu Peravali

Table of Contents

1.0 System Architecture

Our WebOMiner_S system architecture consists of five modules: Crawler module, Cleaner module, Parser module, Frequent Structure Finder module and Schema Extractor module. The overall architecture of the WebOMiner system is shown in figure 1 below:

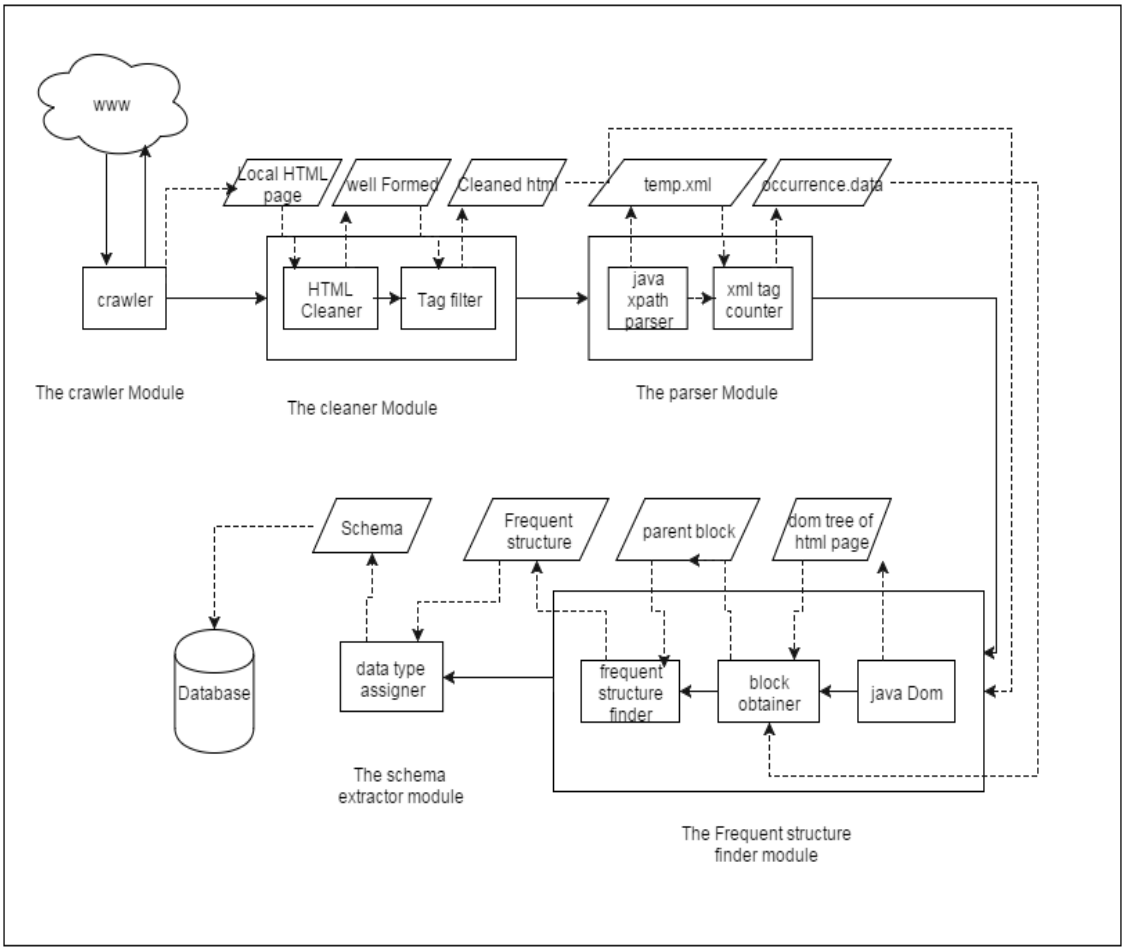


Figure 51: Architecture of WebOminer_S

Here, the crawler module crawls through the WWW to find the given web page, streams entire web document creates a mirror of original web document in the local computer. Cleaner module uses the freeware software JTidy, it first looks for ill-formatted HTML tags and missing end-tags and insert missing tags at appropriate location. It then filters inline tags to conform structural relationship of text contents and reorders tags if necessary. The resultant of the cleaner module is a web page in local directory which is well-formed and cleaned. Parser module generates a temporary xml file by recording every block level tag into it. From this temporary xml file the occurrence count of each block tag with the same class name is written into occurrence.data file. This is passed as input to the frequent structure finder module, this module creates the Dom tree for the cleaned html file from the second module. Then, a dynamic xpath expression created for every block tag with the corresponding class name. This Xpath expression is run over the Dom tree to obtain every occurrence of the block tag with a particular class name in the entire Dom tree. The obtained result is stored in a NodeList and iteratively compared with the previous NodeList to get the most occurred block tag with more number of children. Thus found frequent structure is given as input to the next

module. The schema extractor module, retrieves the complete block of the frequent structure by using the parent node. Foe each node in the complete structure, if it is a text node the datatype String is assigned, if it is an IMG node the source attribute is retrieved and stored. Then, using the schema a data warehouse star schema table is generated by appending the store id and time followed by inserting the data values into it.

2.0 User Interface

The GUI allows the user to input URL of the web page he wants to extract. The entered URL is the only input to the system. The extracted data can be viewed in the database connected to the system. After the program being successfully imported in the java environment such as eclipse and being connected to the MySql database the user has to call the URL "http://localhost:8080/WebOMiner_s/" from a web browser such as google chrome which opens

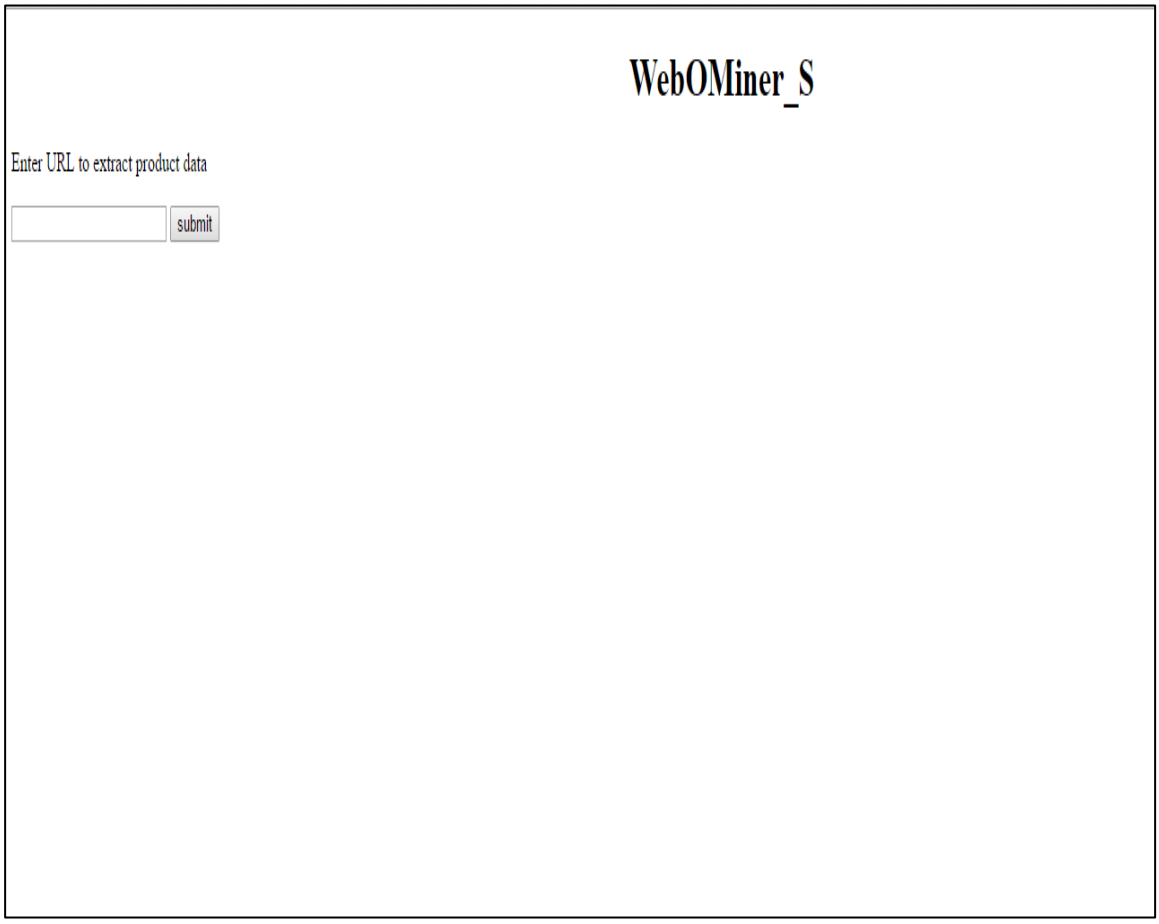


Figure 52: GUI for the user

up the GUI for user is as shown in figure 54.

2.1 How to debug, run the program on windows using ide and get the results:

To debug and run the system we need to confirm the following first:

1. Conform operating system requirement described in section 3.0.

2. Install java SDK 1.7 or later version and set the class path / path in environment variable. Setting an environment variable is discussed in "forums.sun.java" in the flowing link:

   http://forums.sun.com/thread.jspa?threadID=5450340

3. For debugging and running the system from IDE we need to down load and install Eclipse IDE as described in section 5.1.

4. Import "webominers" project in Eclipse IDE by using following steps:

   File → Import Project →General→ File System → select "webominers" from your computer → ok. The "webominers" project will be imported to Eclipse IDE.

5. Install My Sql latest version as described in section 5.3.

6. Connect to mysql using username and password then create two tables blockcount and tablecount for storing information such as the number of columns in each table and number of tables generated so far respectively. Using command "CREATE TABLE blockcount (colcount,int (11));" and "CREATE TABLE tablecount (count ,int (11));

7. Install JDeveloper IDE as described in section 5.4.

8. Make sure all the jar files needed 1. mysql-connector-java5.1.38-bin.jar 2. commons-lang3-3.4.jar 3. commons-lang3-3.4-javadoc.jar 4. guava-16.0.1.jar 5. org-apache-commons-lang.jar` driver file is in "libraries" folder under "webominers" package.

9. Prior to any experiment or enhance / modification of the system. First set the "FilePath" variable in the servelet in line 74 to any desired location on your computer for the program to use the space to create and read/write into files such as the uncleaned.html, cleaned.xhtml, occurrence.data, temporaray.xml.

10. Go to Debug button in Manu bar of the Eclipse IDE, click it and select and click Debug (webominers) button as shown in figure 3. If the program is imported properly a new window will be opened in the web browser prompting the user to enter URL of the product list web page to extract.

11. To look at the populated database tables,

Go to start Manu →All Programs → My Sql workbench→ Connect database → Enter username and password in My Sqlworkbench login

16. After login to MySql database, go to SQL Menu → select SQL Command→Enter command.

17. From the new window enter SQL command and click on “Run” button. The results will be shown in “Results” window below as shown in section 6.0.

2.2 How to compile, run the program on Linux from terminal and get the results:

To debug and run the program we need to confirm the following first:

1. Install java SDK 1.7 or later version as described in section 5.1.1

2. Install Oracle 11g or latest version as described in section 5.3

3. Connect to sqlplus using username and password then create a table tablecount for storing information such as number of tables generated so far respectively. Using command “CREATE TABLE tablecount (count int);

4. The project named “webominersunix” is the main folder containing all the jar files needed given in step 5 and it has two sub folders “src” and “bin”. The src folder contains package folder “webominersunix” which has the source code .java file “WebominersUnix.java” contains all the methods sequentially called by the main method inside it.

5. Make sure all the jar files needed are inside the main project folder webominersunix. All the jar files needed can be downloaded from the web links given in section Java Tools or just by copy pasting the jar file names in google.

   1. odbc14.jar : In order to connect to database

   2. guava-16.0.1.jar : Multimap data structure used in fs_finder function

   3. org-apache-commons-lang.jar: To use StringUtils method in database function

   4. commons-lang3-3.4.jar : To support Multimap

   5. commons-lang3-3.4-javadoc.jar: Documentation for the library

6. Prior to any experiment or enhance / modification of the system. First set the “FilePath” variable in the main method in line 74 to any desired location on your computer for the program to use the space to create and read/write into files such as the uncleaned.html, cleaned.xhtml, occurrence.data, temporaray.xml.

7. .For the database inorder to connect to your database give valid credentials such as db name, user id , password in the method datavalues() inline 356 of the java file “WebOminersUnix.java”

8. To compile the program use the command javac –cp ".:jarfile1: jarfile2" programname.java

Which will be $ javac -classpath ".:/home/woddlab/WEBOMINER_S/webominersunix/commons-lang3-3.4.jar:/home/woddlab/WEBOMINER_S/webominersunix/commons-lang3-3.4-javadoc.jar:/home/woddlab/WEBOMINER_S/webominersunix/guava-16.0.1.jar:/home/woddlab/WEBOMINER_S/webominersunix/jtidy-r938.jar:/home/woddlab/WEBOMINER_S/webominersunix/ojdbc14.jar:/home/woddlab/WEBOMINER_S/webominersunix/org-apache-commons-lang.jar" WebominersUnix.java;

In order to compile WebominerUnix.java the current directory of the terminal has to be "src" use cd directory in order to navigate to current directory

9.To run the compiled .class file use the command java –cp . classfilename

Which will be $ java -classpath ".:./home/woddlab/WEBOMINER_S/webominersunix/commons-lang3-3.4.jar:/home/woddlab/WEBOMINER_S/webominersunix/commons-lang3-3.4-javadoc.jar:/home/woddlab/WEBOMINER_S/webominersunix/guava-16.0.1.jar:/home/woddlab/WEBOMINER_S/webominersunix/jtidy-r938.jar:/home/woddlab/WEBOMINER_S/webominersunix/ojdbc14.jar:/home/woddlab/WEBOMINER_S/webominersunix/org-apache-commons-lang.jar" webominersunix.WebominersUnix;

In order to run the class file the current working directory of the terminal should be the parent directory of the package "webominersunix" which is " src" for the system to find both .class file and .java file.

3.0 Operating System

System is developed in 32-bit Windows 10 operating system at Intel i5 2.26 GHz processor, 8.00 GB RAM Lenovo Thinkpad Machine. This system is portable in University of Windsor CDF Solaris Operating System on Unix environment. In that case, only minor changes from Windows based syntax to Unix compatible syntax transformation is needed to compile, execute and run the program.

4.0 Programming Environment

The application is programmed and tested in Java 2 Platform Standard Edition version 1.6.0_16, which is installed in Windows Vista Home Premium edition. The advantages of this environment are as follows:

- Regular expression capability.
- The improved Java Doc.
- Low coupling and usability.
- Relatively easy implementation of Object-Oriented design pattern.

5.0 Installation of the system

5.1. Installation of Eclipse IDE:

5.1.1. Installing the Software Bundle on Microsoft Windows:

To install the software, we must need to have administrator privileges on our system. The installer places the Java Runtime Environment (JRE) software in %*Program Files*%\Java\jre6, regardless of the specified JDK install location.

Note: This installer does not displace the system version of the Java platform that is supplied by the Windows operating system.

Before Installation:

1. We need to verify our system to meet or exceed the following minimum hardware requirements:
    - 800MHz Intel Pentium III or equivalent
    - 512 MB of RAM.
    - 750 MB of free space

    Note: The installer uses the *%USERPROFILE%*\Local Settings\Temp directory to store temporary files.

2. First need to verify that we have administrator privileges on our system.
3. Then download the jdk-6u21-nb-6_9_1-windows-ml.exe installer file.

    Installing the Software:

1. We need to double-click the installer jdk-6u21-nb-6_9_1-windows-ml.exe file to run the installer.
2. At the JDK Installation page specify which directory to install the JDK into and click Next.
3. At the NetBeans IDE Installation page, we need to do the following:
    1. Specify the directory for the NetBeans IDE installation.
    2. Accept the default JDK installation to use with the IDE or specify another JDK location.
4. Review the Installation Summary page to ensure the software installation locations are correct.
5. Click Install to begin the installation. When the installation is complete, we can view the log file, which resides in the following directory: %*USERPROFILE*%\.nbi\log.

5.1 Installation of the MySql software using installer:

MySQL Workbench can be installed using the Windows Installer (.msi) installation package. The MSI package bears the name mysql-workbench-version-win32.msi, where version indicates the MySQL Workbench version number.

To install MySQL Workbench,

1. right-click the MSI file → select the Install option from the pop-up menu, or simply double-click the file.

In the Setup Type window you may choose Complete or Custom installation. To use all features of MySQL Workbench choose the Complete option.

6.0 Data Base Schema and format

Our system creates and inserts the product data into tables automatically there is no requirement of explicit creation schema. The created tables will be in the name of Product_Number, number is a dynamic variable it increments for each extraction so your first page extraction will be

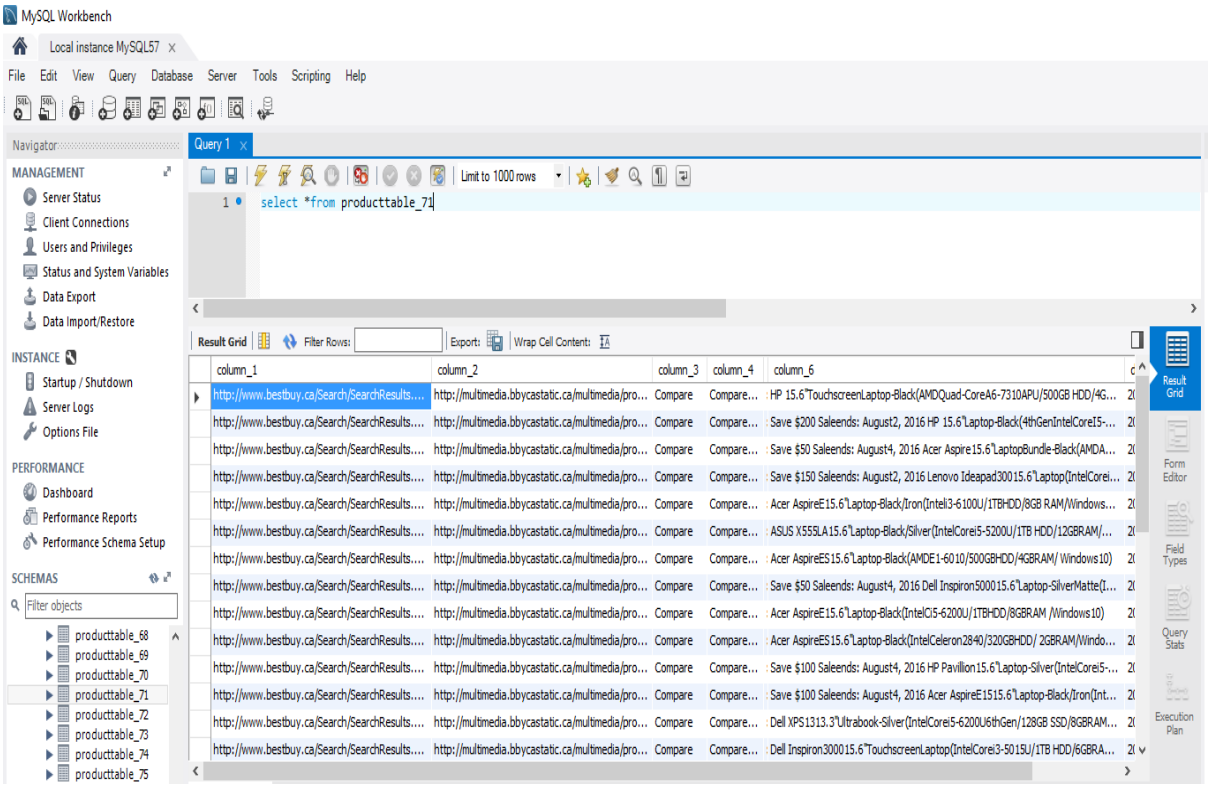


Figure 53: Product table created

PRODUCT_1 table. The number of columns of each PRODUCT table are also determined the program and each column will be named column_1,column_2….

7.0 Limitations of the software

Limitations of the current system and future work as identified are stated as below:

Crawler module: Current crawler module can take one URL string at a time for extraction and mining of web contents. Further improvement is required in future for automatic identification of positive web pages from the web. Present implementation of this module is designed aiming to work for basic functionalities as crawler with the functionality to download data stream from the targeted web pages into the local computer and cleaning or the web page comments from it. For robustness and scalability, we need to improve the current crawler module to handle all kinds of situations from the web.

Cleaner module: Currently we are using open source software "JTidy" for cleaning the web pages. Development of an independent cleaner module may improve the systems performance and usability in future.

Parser Module:

The parser module can be improved by removing the necessity to read/write into a file instead the xml data can be stored into other appropriate data structures to avoid the i/o operations.

Frequent Structure Finder Module:

By discovering the frequently repeated structures in the html code of the web page we are able to identify the target product data from product list web pages. This technique has to be extended to detail pages because they have the complete product information unlike the product list pages having only meta-data. To do that the web link(<a> having the URL to detail page) that lead to the detail page has to be discovered in the frequent structure found in the fourth module and a new algorithm has to be designed to identify contents from it.

Schema Extractor module:

The only disadvantage in our schema extractor module currently is that it assigns data type varchar to all the text tags, as price is also embedded in the text tag itself in html it assigns varchar to price also. The algorithm has to be further improved to assign correct datatype based on the content of the text tag.

8.0 Java Tools

We used a set of java tools for the development of WebOMiner system. Table - A list some important java tools we used for the development of the system and their reference URL's for future developer's reference.

9.0 Web Application

The entire application of WebOMiner_S is available at miner.newcs.uwindsor.ca. To access it as user just type the URL "miner.newcs.uwindsor.ca" in your web browser.

To access it as administrator in order to modify/ improve the existing work login with user name "peraval" and password "changeme" using ssh or ask the network administrator to add you as "sudo user" to the server, so that you can access the contents from your login.

Table – A

Java Tools	Reference URL
BufferedInputStream	http://download.oracle.com/javase/1.4.2/docs/api/java/io/BufferedInputStream.html
Class	http://download.oracle.com/javase/1.5.0/docs/api/java/lang/Class.html
File	http://download.oracle.com/javase/1.4.2/docs/api/java/io/File.html
Multimap	https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html
HttpURLConnection	http://download.oracle.com/javase/1.4.2/docs/api/java/net/HttpURLConnection.html
InputStream	http://download.oracle.com/javase/1.4.2/docs/api/java/io/InputStream.html
InputStreamReader	http://download.oracle.com/javase/1.4.2/docs/api/java/io/InputStreamReader.html
javax.xml.parsers.DocumentBuilder	http://download.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/DocumentBuilder.html
javax.xml.parsers.DocumentBuilderFactory	http://download.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/DocumentBuilderFactory.html
NodeList	http://download.oracle.com/javase/1.4.2/docs/api/org/w3c/dom/NodeList.html
org.w3c.dom.traversal.DocumentTraversal	http://download.oracle.com/javase/1.5.0/docs/guide/plugin/dom/org/w3c/dom/traversal/package-tree.html
org.w3c.dom.traversal.NodeFilter	http://download.llnw.oracle.com/javase/1.5.0/docs/guide/plugin/dom/org/w3c/dom/traversal/class-use/NodeFilter.html
org.w3c.dom.traversal.NodeIterator	http://download.oracle.com/javase/1.5.0/docs/guide/plugin/dom/org/w3c/dom/traversal/class-use/NodeIterator.html
commons-lang3-3.4.jar	http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Process.html
guava-16.0.1.jar	http://www.science.uva.nl/ict/ossdocs/java/jdk1.3/docs/api/java/lang/Runtime.html
mysql-connector-java-5.1.38-bin.jar	http://download.oracle.com/javase/1.5.0/docs/api/java/lang/String.html
org-apache-commons-lang.jar	http://download.oracle.com/javase/6/docs/api/java/sql/package-summary.html
StringTokenizer	http://download.oracle.com/javase/1.4.2/docs/api/java/util/StringTokenizer.html
URL	http://download.oracle.com/javase/6/docs/api/java/net/URL.html

VITA AUCTORIS

NAME:	Bindu Peravali
PLACE OF BIRTH:	Andhra Pradesh, India
YEAR OF BIRTH:	1993
EDUCATION:	Vignan High School, Guntur, AP, 2010
	Bapatla Women's Engineering College, B.Tech., Guntur, AP, 2014
	University of Windsor, M.Sc., Windsor, ON, 2016