2012

Mining Multiple Web Sources Using Non-Deterministic Finite State Automata

Mohammad Harun-Or-Rashid

Mining Multiple Web Sources Using Non-Deterministic Finite State Automata

By

Mohammad Harun-Or-rashid

A Thesis
Submitted to the Faculty of Graduate Studies
Through Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2012

Mining Multiple Web Sources Using Non-Deterministic Finite State Automata

by

Mohammad Harun-Or-Rashid

APPROVED BY:

_____

Dr. Ronald Barron
Department of Mathematics and Statistics

_____

Dr. Subir Bandyopadyhay
School of Computer Science

_____

Dr. Christie Ezeife, Advisor
School of Computer Science

_____

Dr. Ziad Kobti, Chair of Defense
School of Computer Science

14th September, 2012

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

Existing web content extracting systems use unsupervised, supervised, and semi-supervised approaches. The WebOMiner system is an automatic web content data extraction system which models a specific Business to Customer (B2C) web site such as "bestbuy.com" using object oriented database schema. WebOMiner system extracts different web page content types like product, list, text using non deterministic finite automaton (NFA) generated manually.

This thesis extends the automatic web content data extraction techniques proposed in the WebOMiner system to handle multiple web sites and generate integrated data warehouse automatically. We develop the WebOMiner-2 which generates NFA of specific domain classes from regular expressions extracted from web page DOM trees' frequent patterns. Our algorithm can also handle NFA epsilon(ε) transition and convert it to deterministic finite automata (DFA) to identify different content tuples from list of tuples. Experimental results show that our system is highly effective and performs the content extraction task with 100% precision and 98.35% recall value.

Keywords: Web mining, regular expression, non deterministic finite automata, B2C, frequent pattern, deterministic finite automata, regular expression, DOM tree, Web schema.

DEDICATION

This thesis is dedicated to my parents and family, for their support and encouragement throughout my graduate studies.

This work also desiccated to Dr. C.I. Ezeife for her support, appreciation and generosity during my graduate career.

ACKNOWLEDGEMENTS

I would like to thank all those people who made this thesis possible and an unforgettable experience for me. First of all, I would like to express my deepest sense of gratitude to my supervisor Dr. Christie Ezeife who offered her continuous advice and encouragement throughout the course of this thesis. I thank her for the systematic guidance and great effort she put into my thesis. I am thankful to Dr. Christie Ezeife also for research assistantship support and encouragement whenever I was in need.

I would like to express my very sincere gratitude to my internal reader Dr. Subir Bandyopadyhay and external reader Dr. Ronald Barron for the support to make this thesis possible. I acknowledge my gratitude to my committee chairman Dr. Ziad Kobti for the absolute support for the thesis.

I am thankful to my colleagues Sabbir Ahmed, Yanal Ahmed, Tamanna Mumu and Gunjon Soni for the help and support and their technical assistance to my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 - Introduction

The World-wide-web has become the source of a huge amount of information on the internet due to explosive growth and popularity (Embley et al., 1999). Day by day internet users are also increasing. There are different kinds of users including customers, retailers, service companies, etc. They use the web for gathering detailed information. And these are displayed by different web sites which are in heterogeneous formats. By collecting and organizing this information, it is possible to produce metadata for many applications. It is beneficial to create a huge collection of historical and derived data on the products from different domains (business to Customer, research, library, etc.). These historical and derived data could be used for different purposes such as shopping comparisons, detecting user intention and further knowledge discovery. For example, there is a huge number of online stores (B2C) selling their products through internet. It is hard for customers to retrieve, analyze and compare products or their prices from online stores. In order to get a product with the special attribute (for example: 56" LCD TV) and lowest cost compared to other similar products, a user has to go through all the online stores, which takes a lot of time. Annoni and Ezeife (2009) proposed an approach called OWebMiner that represents web content as objects. They identified six object types which include text, list, image, form, separator and structure. Mutsuddy and Ezeife (2010), Ezeife and Mutsuddy (2013) proposed an approach called WebOMiner that is an automatic web content data extraction technique which models web sites of a specific domain as object oriented database schemas. For Business to Customer (B2C) web sites such as "Bestbuy.com", "Futureshop.com" ,"CompUSA.com", the WebOMiner system is able to extract different types of web page contents like product, list, text and

advertisement information from multiple sources using content non-deterministic finite automaton (NFA) it generated.

In this thesis, we study the problem of extracting information from web pages of different domains (B2C, research, library etc.). The WebOMiner (Mutsuddy and Ezeife, 2010; Ezeife and Mutsuddy, 2013) extracts web contents from B2C web sites. It uses NFA to identify tuple from list of tuple that extracted from web pages DOM tree. There is still need for automatic NFA generation for the object oriented schema for each web site or data sources. Other shortcoming of their work is that they didn't handle ambiguity and epsilon (ε) transition of NFA. This thesis extends the automatic web content data extraction techniques proposed in the WebOMiner system. Our proposed approach is able to generate automatic source database schema NFA of domain classes from frequent patterns extracted from web pages DOM trees. By converting NFA into deterministic finite automata (DFA), our algorithm handles ambiguity and epsilon (ε) transition and able to identify different tuples from list of tuples. Our proposed approach is also able to generate integrated data warehouse schema automatically by using regular expression generated from frequent pattern extracted.

In the following sections of this chapter, we introduce the problem of information extraction from the web. The information extraction problem is discussed in detail in section 1.1, section 1.2 describes data type, section 1.3 explains the information extraction problem in the context of e-commerce, section 1.4 describes document object model, section 1.5 explains finite automata, section 1.6 explains the idea of object-oriented web content extraction, section 1.7 defines the problem statement of this thesis.

The contributions of this thesis are briefly explained in section 1.8. We conclude the chapter with the organization of this thesis in section 1.9.

1.1 Information Extraction

Information extraction (IE) is defined by Peshkin and Preffer (2003) as the task of filling in template information from previously unseen text which belongs to a pre-defined domain. Lerman et al. (2004) identify that IE task is defined by its input and its extraction target whereas input can be categorized into two types, namely unstructured document and semi-structured document. The main goal of information extraction is to extract information automatically from data source such as entities, relationships between entities and attributes describing entities from structured and semi-structured documents. This information can be stored into the database for further knowledge discovery, shopping comparisons and detecting user intentions. In this thesis, we consider information extraction from semi-structured documents that are present on the web.

1.1.1 Types of web page

The World-wide-web can be called a vast repository of information. Data stored on the web can be accessible by the user through search form or dynamically generated web page. The World-wide-web consists of huge amounts of web pages which represent different products (Buttler et al., 2001). Two different types of web pages can be distinguished.

1.1.1.1 Unstructured pages: Also called free-text documents, unstructured pages are written in natural languages. There is no predefine template can be found, and only

information extraction (IE) techniques can be applied with a certain degree of confidence. For example, figure 1 represents an unstructured web page from "Wikipedia.org" which contains only text and there is no template.

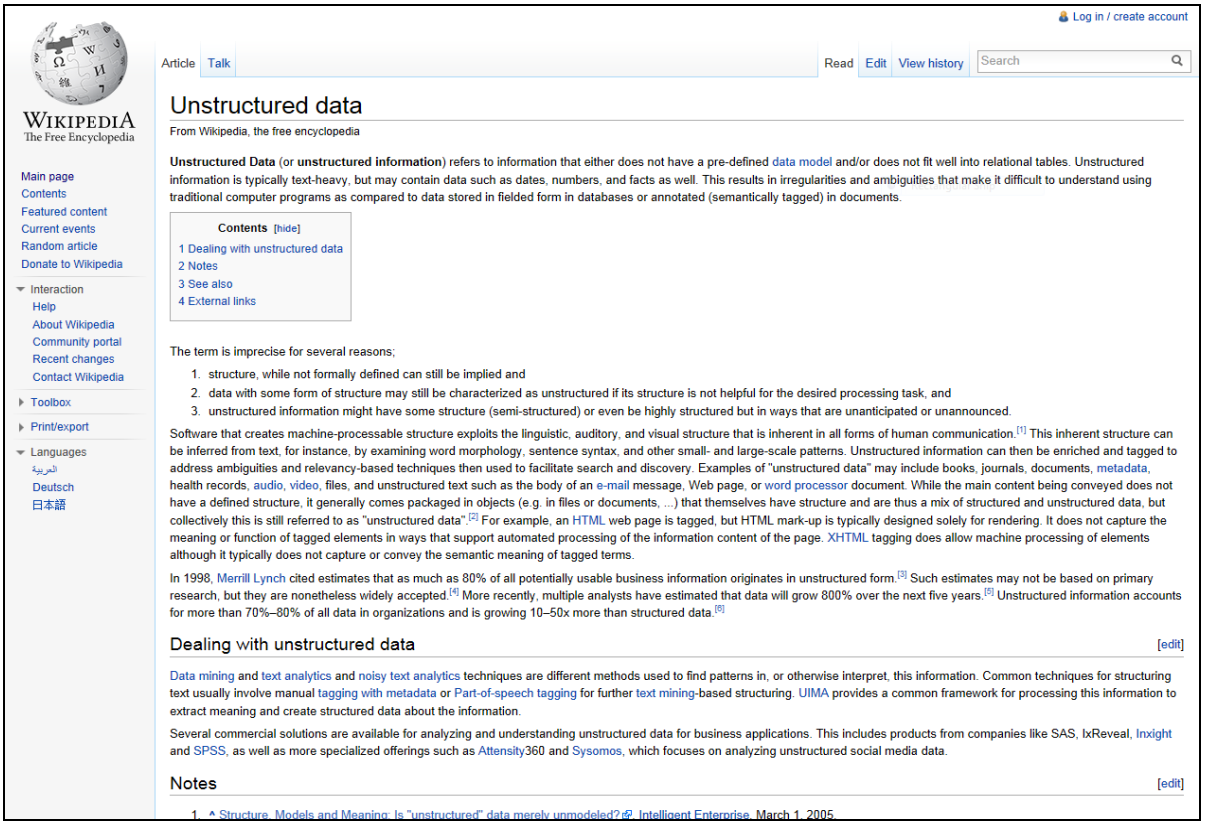


Figure 1: An unstructured web page (wikipedia.com)

1.1.1.2 Structured/semi-structured web page: Structured/Semi-structured pages are normally obtained from a structured data source, e.g., a database, and data are published together with information on structure. The extraction of information is accomplished using techniques based on wrapper generation, rule generation and automatic approaches. Most of the web document formed by structured data such as text, image, hyperlink, structured record such as list, table, and database generated content. This type of web page generated by a program that access structured data in a local database and embeds

them in a HTML template. For example, Figure 2 represents a structured/semi-structured web page which is a product list page from “Bestbuy.com”.

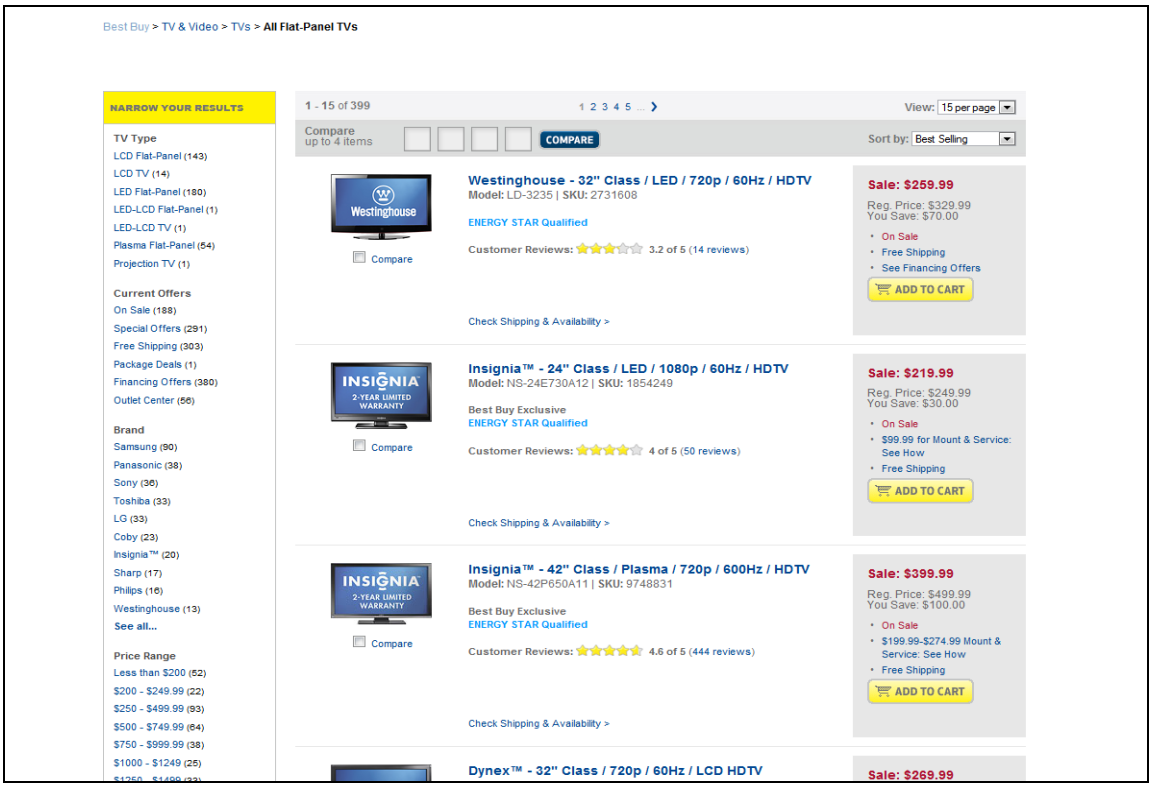


Figure 2: A structured web page (bestbuy.com)

1.1.1.3 List page: List pages are web pages that contain several structured records. Generally, online stores display their different products with list pages. The data record of list pages is important to create a historical and derived data warehouse with the extracted data from it. For example, Figure 2 represents a list page from “Bestbuy.com”. There are two types of list pages, vertically labeled list page and horizontally labeled list pages. Another type of list pages called result page produced by search engines with a user query which is presented as a list or a table on an automatically generated page. For example, Figure 3 represents a web result page which is generated by google search engine. These list pages are called dynamic web pages because they are generated by

using their own template. The template is generated using html tag. There is a query working in the backend to fill the template and display data in the browser.

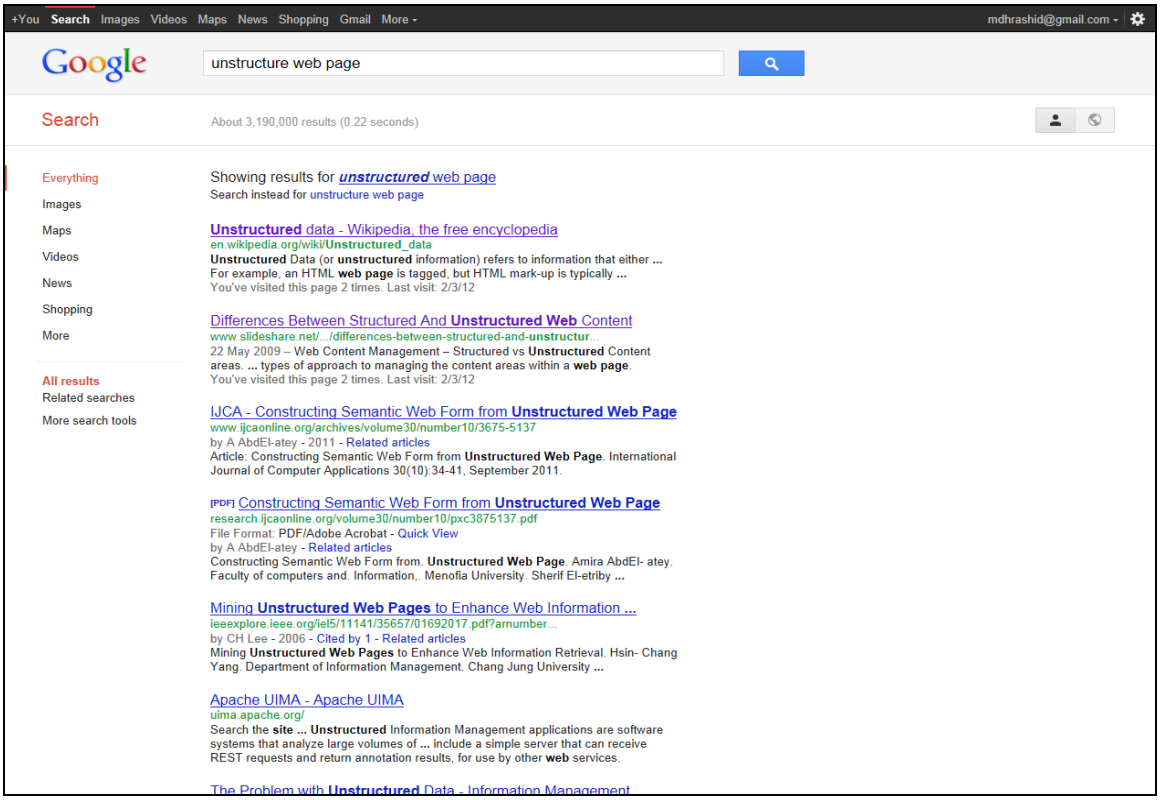


Figure 3:Result page generated by google search engine(google.com)

When a page is generated using a template, a common structure which is created with html tag can be found in the source code of the page. For example, on a page displaying 10 laptops from different brands and prices, their presentation structures are the same through the entire page. This structure can be assumed as a schema of the product. Different domain specific web sites use different structures to display their products. For example, the structure of a B2C web site is different from the structure of a library web site. So, it is a big challenge to create a historical and derived data warehouse with different domain specific web sites.

1.2 Data types in data warehouse

1.2.1 Historical data: Historical data is the data from previous time periods, in contrast to current data (Singhal and Seborg, 2001). It is used for comparisons to previous periods and trend analysis. The past information about a company can be extracted using historical data, and it can be used to help forecast the company's future, for example, "Given a product type and business name, output all promotion price offered by the business within last 10 years". This query extracts all the information about the product until today's date. And from it user can get idea about the promotional trend of this product. Using historical data, we are able to extract information about a specific trend about the sale price of a product which is sold by a B2C company. For example, "Bestbuy.com" offers promotion on their product once a month. If we have a database with historical database we are able to find the trend of the promotional price of any particular product. The advantage is that user can decide about the right time to buy a product.

1.2.2 Derived Data: Derived data types are those that are defined in terms of other data types, called base types (Botzer and Etzion, 1996). Derived data types contain attributes, element or mixed content. They exist in data warehouse as built-in or user-derived. Base types can be derived data types or primitive types. Restriction facets and extension are used to create derived data types. A table exists in database can have derived columns, which values are computed, based on the values of other table columns. It is called a derived table, if all columns are derived. Derived data exists in the database as aggregates such as count, sum, average, minimum, maximum which can be computed from web content data. For example, "What is the average number of laptop on sale each

month or how many times a company offer promotion on their product each year". By getting these two results user can get the idea about the promotional trend of a business to a particular product.

1.2.3 Metadata: Metadata is structured information that describes an information resource and makes it easier to retrieve, use, or manage an information resource (Mize and Habermann, 2010). Metadata is often known as information about information or data about data. How the data is formatted and how and when and by whom a particular set of data was collected, is described by metadata. Metadata helps to understanding information stored in data warehouses. The telephone book is an example of metadata that we are very familiar with, where we search for a telephone number using name or location. Another example of metadata is the catalogue in a library, where we search for information using "Subject", "Title" and "Author".

1.3 Application of Information Extraction

In recent years, a large number of architecture has been proposed for extracting information from web pages. Three approaches are used, which are unsupervised (automatic), supervised (manual) and semi-supervised. In this section we describe different applications used in information extraction from web pages.

1.3.1 Wrapper

The process of information extraction uses a program which is called extractor or wrapper (Adleberg, 1998). The information integration system considers a wrapper as a component which has a single uniform query interface to access multiple information

sources. An information source (e.g., database server, web server) is wrapped by a program and the integration system able to access that source without changing its core mechanism. When the web server is considered as information source, then, the wrapper uses HTTP protocols to query the web server to collect the resulting pages, extract content from HTML documents, and after that integrates with other data sources. Wrappers are used in information extraction from web sites and consist of a series of rules and some codes to apply those rules and are specific to a source. Some examples of existing wrappers are WIEN (Kushmerick et al., 1997), SOFTMEALY (Hsu and Dung 1998) STALKER Muslea et al., 1999) etc.

1.3.2 Traditional Information Extraction versus Web Information Extraction

Information extraction from web is different from traditional information extraction (IE) (Appelt and Israel, 1999). In traditional IE, data are extracted from totally unstructured free texts that are written in natural language. But information extraction from web processes structured data which are online documents and the server-side application program generates it automatically. Usually, Web IE task is performed by machine learning approach (Soderland, 1999), pattern or rule mining techniques (Chang and Lui, 2001; Wang and Lochovsky, 2002; Chang et al., 2003) to extract the data from different sources.

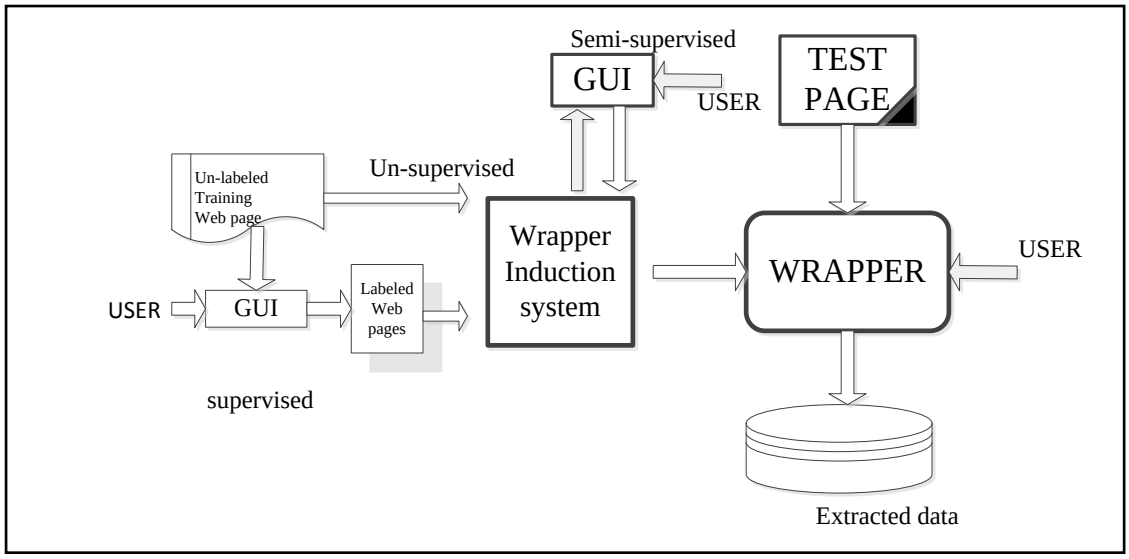


Figure 4: A general view of Information extraction System

1.3.3 Applications of web information extraction

The wrapper Induction (WI) generates wrapper (Wang and Lochovsky, 2002) or information extraction (IE) systems (Kushmerick et al., 1997). Wrapper works as pattern matching procedure (e.g., a form of finite-state machine) which depends on a set of extraction rules. A wrapper generated by wrapper induction is used to extract the information from target resources. In the earlier system, wrapper generation was manual process where programmer was involved in writing extraction rule whereas later systems are automatic rule generalization process based on machine learning. In wrapper induction process, user writes extraction rules manually to labeling target extraction data. Current wrapper induction systems are created with unlabeled training example. The wrapper induction system can be categorized into four groups, include manually-constructed IE system (Hammer et al.,1997;Crescenzi and Mecca,1998; Arocena and Mendelzon, 1999; Liu et al.,2000; Saiiuguet and Azavant, 2001), supervised IE Systems (kushmerick et al., 1997;Califf and Mooney.,1998; Muslea et al., 1999; Soderland, 1999),

Semi-supervised IE Systems(Chang and Lui.,2001; Chang and Kuo,2004; Hogue and Karger, 2005), and unsupervised IE Systems (Arasu and Gracis-Molina,2003; Wang and Lochovsky 2003; Zhai and Liu, 2005).

1.3.3.1 Manually-Constructed IE Systems

In manually-constructed IE system, user creates wrapper program using general programming language like Perl or special-designed language for each web site. This approach is considered as time consuming and labor intensive procedure because it requires the user to have strong computer and programming background. Some existing systems are TSIMMIS (Hammer et al., 1997), Minerva (Crescenzi and Meca, 1998), WEBOQL (Arocena and Mendelzon, 1998), W4F (Saiiuguet and Azavant, 2001) and XWRAP (Liu et al., 2000). Methods in this approach simplify the construction of data extraction system by using some languages. In this approach the user are involved with manually construct data records pattern for the extraction target. For example, TSIMMIS is one of the first approaches that build web wrappers manually (Hammer et al., 1997). In this approach, a wrapper takes a specification file as input that declaratively states by programmers, where the data to extract is located on the pages and how the data should be grouped into objects. Since manual approaches are not scope of this thesis, we do not discuss more details. Reader interested may refer to Minerva (Crescenzi and Meca, 1998), WEBOQL (Arocena and Mendelzon, 1998), W4F (Saiiuguet and Azavant, 2001) and XWRAP (Liu et al., 2000).

1.3.3.2 Supervised IE System

Supervised systems extract data by using a set of web pages labeled with examples and generate wrapper. As an example, the extraction rule for the book title is shown in Figure 5, which contains words “Book”, “Name”, and “”, and immediately followed by the word “”. The title consists of at most two words that were labeled as “nn” or “nns” by the POS tagger is specified by the “Filler pattern”. User provides an initial set of labeled examples and the system may require additional pages for the user to label. Instead of programmers, the user can be trained the system to reduce the cost of wrapper generation,

Extraction rule of Book Title:

Pre-filter pattern	Filter pattern	Post-filter pattern
(1) Word: Book	list: len: 2	Word:
(2)Word: Name	Tag: [nn, nns]	
(3) Word:		

Figure 5 : Labeled training example

Such systems are RAPIER (Califf et al., 1998), WHISK (Soderland, 1999), SRV (Kushmerick et al., 1997) and STALKER (Muslea et al., 1999). The methods of this approach use machine learning techniques to learn and construct wrappers from human labeled examples.

Extraction rule of rating

Length(=1)
Every (numeric true)
Every(in_list true)

Figure 6: SRV rule.

The supervised system SRV (Kushmerick et al., 1997). is a top-down relational algorithm that generates single-slot extraction rules. It considers IE as a kind of classification problem. First, it tokens the input documents and all substrings of continuous tokens such as text fragments are labeled as either positive examples or negative examples. SRV generates rule which are logic rules that based on a set of token-oriented features or predicates. These features are two types: simple and relational. A simple feature describes a function that maps a token with some discrete value such as length, character type (e.g., numeric), orthography (e.g., capitalized) and part of speech (e.g., verb). A relational feature maps a token to another token, for example, the contextual tokens of the input tokens. The learning algorithm works as FOIL, starting with entire set of examples and adds predicates greedily to cover as many positive examples and as few negative examples as possible. Supervised approaches are not scope of this thesis. Reader interested refer to RAPIER (Califf et al., 1998), WHISK (Soderland, 1999) and STALKER (Muslea et al., 1999).

1.3.3.3 Semi-supervised IE systems

Semi-supervised system accepts a rough example from users to generate extraction rule. Semi-supervised IE systems include IEPAD (Chang et al., 2001), OLERA (Chang et al., 2004) and THRESHER (Hogue and Karger, 200). In this approach, the user uses GUI to specify the extraction targets because no extraction targets are specified for such systems after the learning phase. User's supervision is involved in this approach. The IEPAD (Chang et al., 2001) is one of the first semi-supervised IE systems that generalize extraction patterns using unlabeled web. It does not require any labeled training page and it requires only post effort from the user to choose the target pattern and indicate the data

to be extracted. This method developed based on the observation is that if a web page contains multiple data records to be extracted, they are often rendered regularly using the same template for good visualization. IEPAD generate wrappers by discovering repetitive patterns. IEPAD discovers repetitive patterns in a web page using a data structure called PAT tree which is a binary suffix tree. The suffix tree only records the exact match for suffixes, IEPAD aligns multiple strings which start from each occurrence of a repeat and end before the start of next occurrence by applying center star algorithm. Semi-supervised approaches are not scope of this project. Reader interested refer to IEPAD (Chang et al., 2001), OLERA (Chang et al., 2004) and THRESHER (Hogue and karger, 2005).

1.3.3.4 Unsupervised IE Systems

The labeled training examples are not required in unsupervised IE systems. It extracts information from the web by training the system with example. It also does not have any user interactions to generate a wrapper. Unsupervised IE systems include RoadRunner (Crescenzi et al., 2001), EXALG (Arasu and Garcis-Molina, 2003), DeLa (Wang and Lochovsky, 2003) and DEPTA (Zhai and Liu, 2005). The RoadRunner and EXALG solve page-level extraction task, while DeLa and DEPTA are related to record-level extraction task. The difference between supervised and unsupervised system is that the extraction targets are specified by the users in supervised system and the data that is used to generate the page or non-tag texts in data-rich regions of the input page is defined as extraction target. In this approach the schema is choose by the users. Since all data are not needed, the user needs to do the post-processing work to select relevant data and give each piece of data a proper name. Methods in this category are related to automatic

pattern discovery. The main advantages of these methods include methods do not require separate training, validation and application phases (Breuel, 2003). And these methods can be divided into two categories including based on string matching and based on HTML tree matching. It has been shown that automatic pattern discovery methods based on HTML tree matching are outer perform than the string matching approaches (Yeonjung, 2007). We describe technical details of the above approaches in section 2.

1.4 Document Object Model

The Document Object Model (DOM) is an application programming interface (API) for well-formed XML documents and valid HTML documents (Marini, 2002). The logical structure of XML or HTML documents is described by DOM and also defines the way a document is accessed and manipulated. XML represents many different kinds of information that may be stored in diverse systems and presents this data as documents, and the DOM may be used to manage this data. With the Document Object Model, it is possible to represent a HTML file as documents, navigate their structure, and add, modify, or delete elements and content. The elements of an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model. The HTML DOM represents an HTML document as a tree-structure. The tree structure is viewed as a node-tree and all nodes can be accessed through the tree. The contents of node tree can be modified or deleted, and new elements can be created. The node tree is shown in figure 7, the set of nodes, and the connections between them. The root node is the starting point and branches out to the text nodes at the lowest level of the tree. The nodes in the node tree consists a hierarchical relationship to each other. The relationship

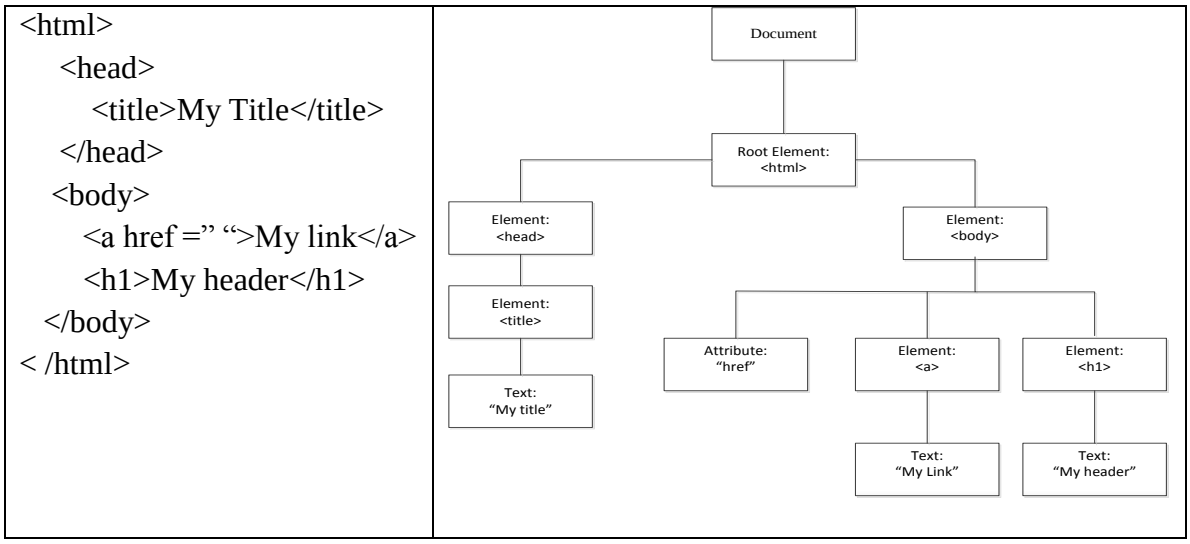


Figure 7: (a) Sample HTML file and (b) Graphical representation of sample HTML file

is described using the terms parent, child, and sibling. Parent nodes have children node and Children on the same level are called siblings. The top node of node tree is called the root. Every node has exactly one parent node except the root node. A node can be contained any number of children. A leaf node does not have children. Siblings are nodes which reside under the same parent. Figure 8 is shown the parent-child relationship.

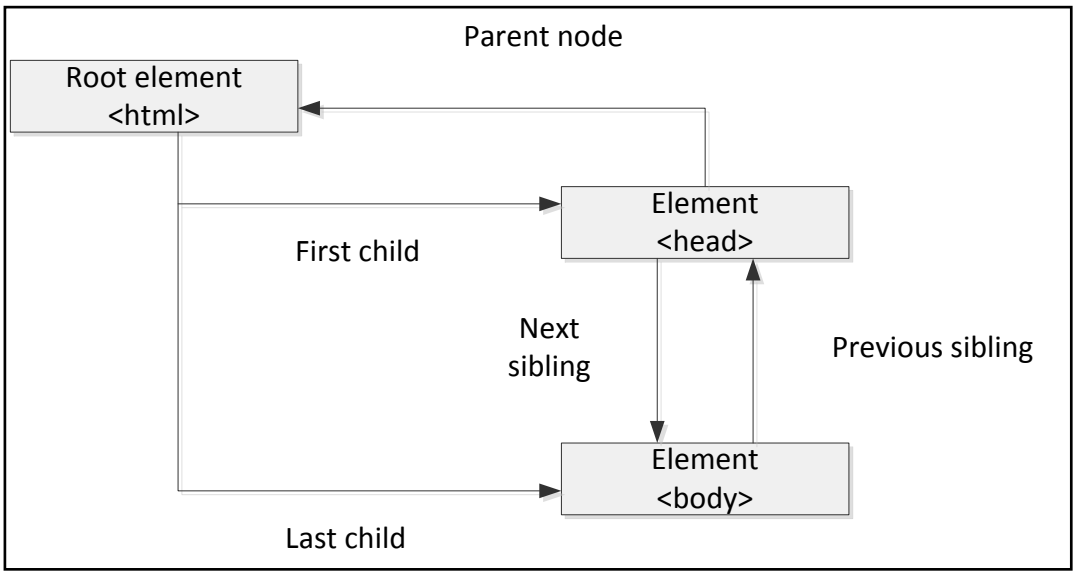


Figure 8: Graphical representation of parent-child relation of the DOM tree

1.5 Finite Automata

Finite automata is a mathematical model which is a combination of 5 tuples (Q, q_0, A, $\sum$, δ) (Cormen, 2009), where Q represents a finite set of states, the start state is $q_0 \in$ Q, A $\subseteq$ Q is a distinguished set of accepting states, $\sum$ is a finite input alphabet, δ represents a function from Q x $\sum$ into Q, called the transition function. The finite automaton starts with state q_0 and it visits to the next state by reading input string one at a time. For example, if the location of the automata is in state q and reads input character "a", it moves from state q to state δ(q, a) by making a transition. The generated finite state machine *M* accepts an input string if it reached at final state. Otherwise input is rejected. A finite automata *M* defines a function called ϕ (finite state function) from $\sum$ to Q such that $\phi(w)$ is reached at the state *M* after reading the string "w". Thus, the finite automata *M* accepts a string *w* if and only if $\phi(w) \in$ A. The function ϕ scan input string recursively by using transition function. Finite automata are two types include deterministic finite automata (DFA) and non-deterministic finite automata (NFA). These are described below

1.5.1 Deterministic finite automata

A deterministic finite state automaton (DFA), also known as deterministic finite state machine that accepts or rejects finite strings of symbols (Cormen, 2009). It produces a unique computation of the automaton for each input string. The word 'Deterministic' refers to the uniqueness of the computation. The states of DFA are fixed and state can be visited one state at a time. An example of deterministic finite state automata is shown in Figure 9, where q_0 is the initial state and final state.

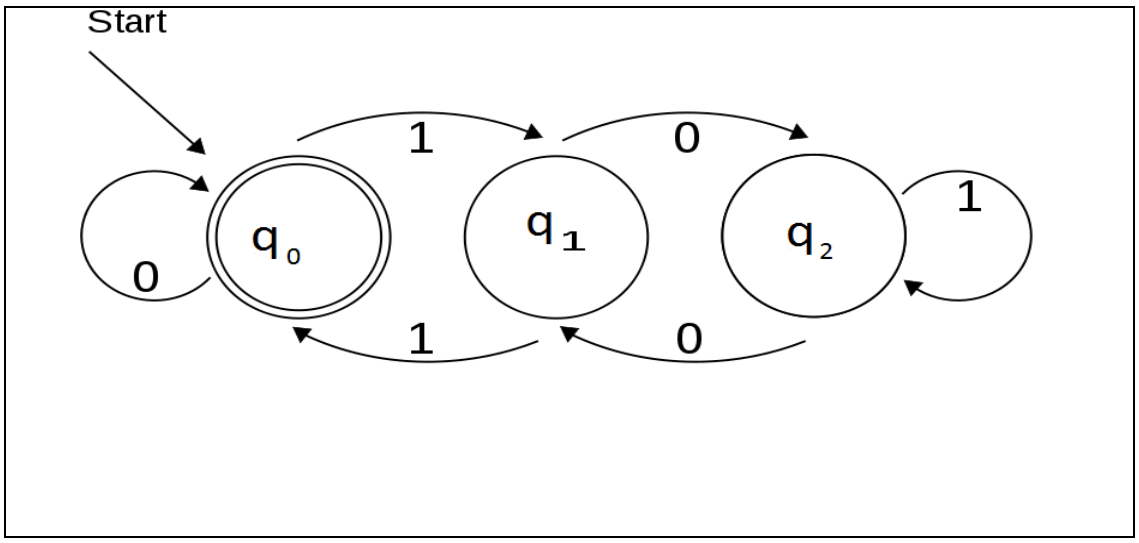


Figure 9: Deterministic finite automata

Figure 9 illustrates a deterministic finite automaton using state diagram. In the automaton, there are three states: q_0, q_1, and q_2 denoted by circles. The automaton takes 0s and 1s as input. For each state, there is a transition arrow leading out to a next state for both 0 and 1. A DFA jumps deterministically from a state to another by following the transition arrow by reading input symbol. For example, if the automaton is currently in state q_0 and current input symbol is 1 then it deterministically jumps to state q_1. A DFA has a start state which is denoted graphically by an arrow coming in from nowhere where computations begin, and a set of accept states which is denoted graphically by a double circle which help define when a computation is successful.

1.5.2 Non-deterministic finite Automata

A non-deterministic finite automaton (NFA) is a finite state machine where from each state using a input symbol the automaton may jump into several possible next states (Cormen, 2009). The difference between deterministic finite automaton (DFA) and non-deterministic finite automata is that next possible state is uniquely determined in DFA but not in NFA. Although the DFA and NFA are not similar by definition, a NFA can be translated to equivalent DFA using power set construction, which means that the constructed DFA and the NFA recognize the same formal language. Both of them recognize only regular languages.

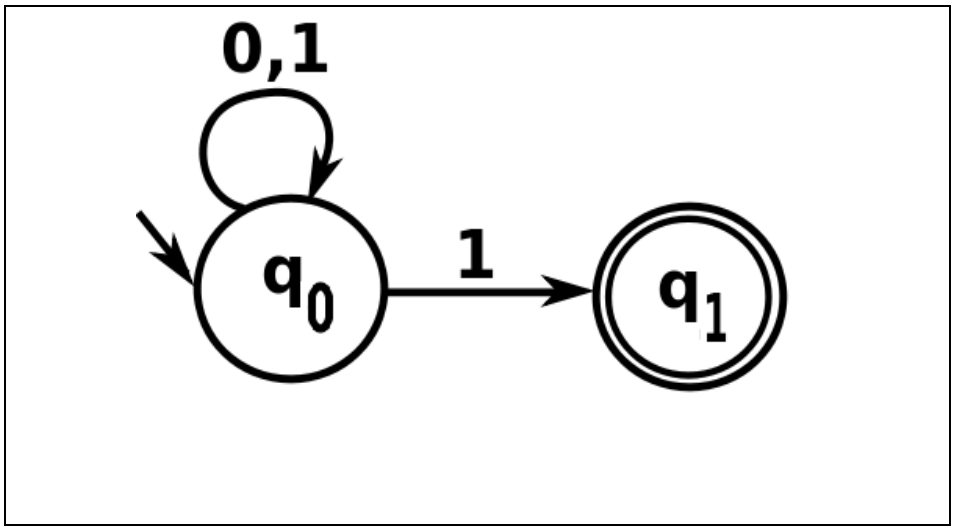


Figure 10 : Non deterministic finite automata

A non-deterministic finite automaton is shown in Figure 10 using state diagram. Here q_0 is the initial state and q_1 is the final sate. It is non-deterministic because the state q_0 has more than one state to move. For every NFA, there is a deterministic finite automaton (DFA) can be found that accepts the same language. Therefore it is possible to convert an existing NFA into a DFA for the purpose of implementing a simpler machine.

1.5.3 Regular Expression

A regular expression provides a concise and flexible means to specify and recognize strings of text, such as particular characters, words, or patterns of characters. The regular expression can be considered as compact notation for describing string. The regular expression follows some rules, which are given below:

- ε (Epsilon) is a regular expression that denotes {ε}, the set containing empty string.
- If '*a'* is a symbol in Σ, then '*a*' is a regular expression that denotes {*a*}, the set containing the string *a*.
- Suppose *q* and *r* are regular expressions denoting the language L(*q*) and L(*r*), then
  - (*q*) | (*r*) is a regular expression denoting L (*q*) ∪L(*r*).
  - (*q*)(*r*) is regular expression denoting L (*q*) □ L(*r*).
  - (*q*) * is a regular expression denoting (L (*q*))*.
  - (q) is a regular expression denoting L (*q*).

1.6 Modeling Web Documents As Objects For Automatic Web Content Extraction

Annoni and Ezeife (2009) proposed a framework called OWebMiner which presents web document as object-oriented web data model to represent web data as web content and web presentation objects. The proposed framework is able to mine complex and structured data as well as simple and unstructured data in a unified way. They identified

three main zones of a web document as instances of specialized classes include HeaderZone, BodyZone and FooterZone. They state that two types of objects exist in these zone including web content objects and web presentation objects. They classified the web content into six categories among them four have sub-content type including text element, image element, form element, plug-in element, separator element and structure element. Text element has two sub content-types including raw text and list text. Raw text has three sub content types including title, label, and paragraph. List text has two sub-content types including ordered list and definition list. Image element has two sub-contents like image and map. Form element has three sub content types which are form select, form input and form text area. The authors classified the web presentation object into six categories which are banner, menu, interaction, legal information, record and bulk. Their proposed algorithm takes a set of HTML files as input. The algorithm works in two sub algorithms. In the first part, it extracts web presentation object and web content object sequentially. And in the second part, it stores the extracted objects into the database. The first part of this algorithm is divided into three steps. In the first step, a DOM tree is generated from the HTML file using DOM parser. In the second step, web zones are identified on the web document from the DOM tree. In the third step, web content object and web presentation objects are extracted. In this algorithm, block level tag include table, division, heading, list, form, block quotation, paragraph and address and non block level tag include anchor, citation, image, object, span, script are considered to extracted object. Two search approaches used in OWebMiner to explore the DOM tree. Depth-first search is executed through block-level tag until it finds non

block level tag. Breadth-first search is executed to parse non block level tag. We discuss technical details of their approach in section 2.

1.7 Mining Web Document Objects with Non-deterministic Finite Automata

Mutsuddy and Ezeife (2010), Ezeife and Mutsuddy (2013) proposed a system for extraction and mining of structured web contents based on object-oriented data model. Their work extended the work of Annoni and Ezeife (2009). They developed the architecture called WebOMiner using object oriented model for extraction and mining of web contents. They introduced an approach of generating and using non deterministic finite state automata for mining web content objects. They defined data block and data regions to ensure consistency between related data. They addressed to relate HTML tag attribute information with related contents to ensure identification of contents, to assign objects and other information together. They defined schema matching to unify similar contents from different web site. They identified noise contents in data blocks and prevent them entering into database table. They also implemented and materialize object-oriented data model for web content and extract heterogeneous related web content together. They defined a mining algorithm that identifies data block and generates non-deterministic finite state automata based wrapper for extraction of related contents. They classified all data blocks of a web page according to their type and check minimum occurrence count based on observed pattern to ensure data consistency before entering them into database. For example, minimum occurrence count for a ‘list’ content can be 3, which means that to be accepted as list tuple record in the DOM tree it should have at least 3 consecutive elements in its block. Their proposed WebOMiner system is an

automatic object oriented web content extraction and mining system for integrating, mining heterogeneous contents that are also derived, historical and complex for deeper knowledge discovery. The WebOMiner extracts information from a given web page including data records (e.g. product image, product brand, product id, short description and price of the product), navigation information (e.g. link URL, link id or name), advertisement (e.g. product advertised, image, URL links to related website). After extraction, WebOMiner stores this information into database for comparative mining and querying. Their proposed approach contains four modules include Crawler Module, Cleaner Module, Content Extractor Module and Miner Module. The proposed crawler module crawls the WWW to find targeted web page given as input. This module creates a mirror of original web document after streaming the entire web document including tags, texts and image contents. The comments are discarded from the HTML document by this module. The cleaner module converts the generated HTML file to well formed by inserting the missing tag, removing inline tag (e.g.
, <ht/>), insert missing "/" at the end of unclosed <image> tag, clean up unnecessary decorative tags. The content extractor module creates DOM tree from HTML page and contents are extracted from the DOM tree in this module. This module assigns respective class object type as per pre-defined object class to the content. It also puts objects into Array List after setting information into objects. Data regions and data blocks are identified by this module and it segments the respective data of a data block from other data blocks by using separator objects. This module also generates Seed NFA pattern for data blocks. It stores identical tuples after extracting objects of all tuples by matching with the refined NFA. It stores the objects into the database after checking the accepted minimum occurrence count for all tuple

categories (e.g., for list, form, product, text etc. tuples). We describe technical details of this approach in section 2.

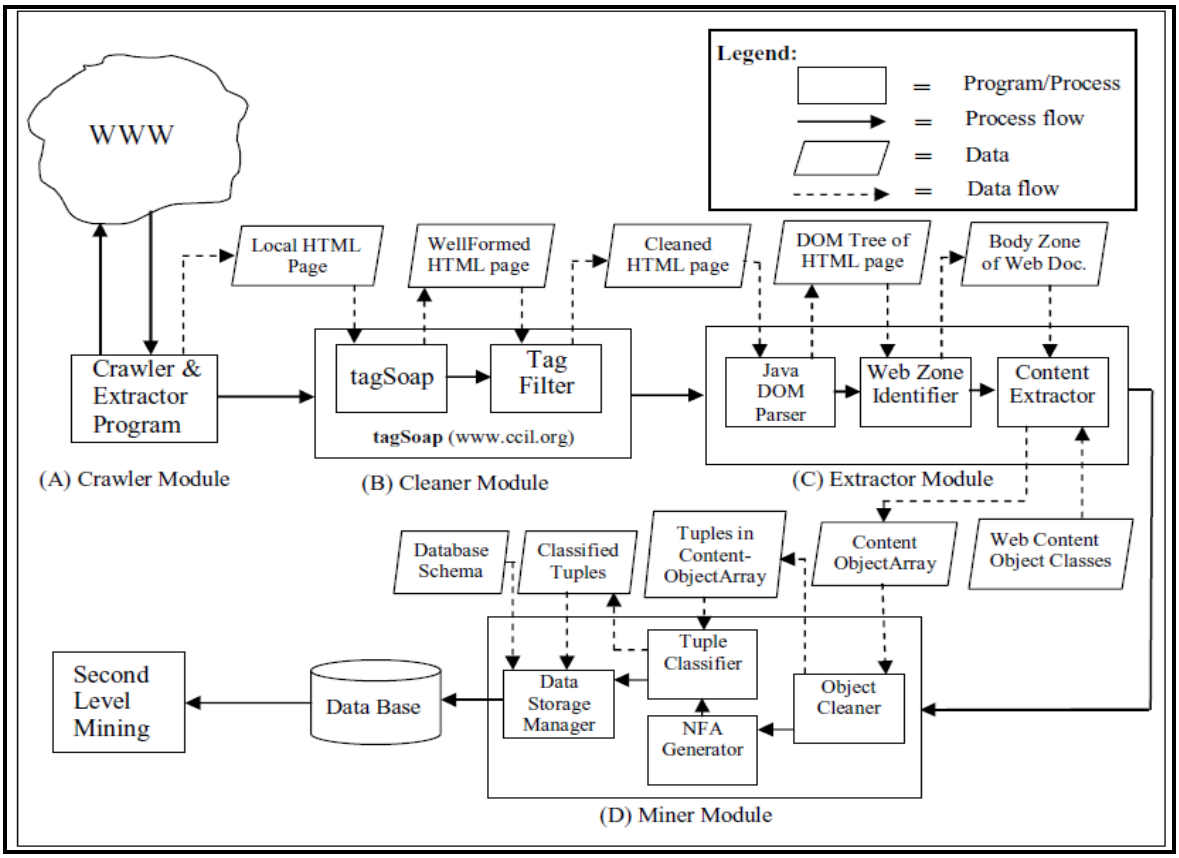


Figure 11: Architecture of WebOMiner

The shortcoming of this approach is that although this method uses an NFA algorithm for identifying tuples of web objects, the mechanism for using NFA algorithm for automatic identification of different content types is not fully integrated in the current system. Also, their NFA is built based on manual observation of ten different B2C web pages for identifying content types from content list. They use manually generated database schema for storing content into database. The authors didn't mention how the "ε" transition or ambiguity was handled. That means the authors didn't mention how the NFA were used

for identifying tuple without convert it to DFA. In this approach, the database schema generation is manual process which is labor intensive and not efficient. The authors didn't define schema integration for different domain specific website.

1.8 Thesis problem statement

This thesis addresses the limitation of WebOMiner (Mutsuddy and Ezeife, 2010; Ezeife and Mutsuddy, 2013) which are given below:

1. The mechanism for using NFA algorithm for automatic identification of different content types is not fully integrated in the current system.
2. Existing NFA is built for identifying content types based on manual observation of ten different B2C web pages.
3. The "ε" transition or ambiguity of NFA was not handled.
4. In the current system, the database schema generation is manual process which is labor intensive and not efficient.
5. Current system does not define schema integration for different domain specific website.

1.9 Thesis Contribution

In this research we developed an algorithm that extends the WebOMiner (Mutsuddy and Ezeife, 2010; Ezeife and Mutsuddy, 2013).

1. Our proposed algorithm finds the frequent pattern matching structure from DOM tree of the HTML page. It iteratively continues the discovery process to find all

matching structure to discover repeated objects (list, product, text etc.) in the page. Finally, RE (regular expression) is formed from the discovered pattern structure and NFA wrappers are generalized from RE. Our proposed framework handles the "ε" transition by converting NFA to DFA (deterministic finite automata) and produces DFA to the miner module of the WebOMiner to identify objects tuple from the list of objects.

2. Our proposed algorithm also generates database schema automatically to store different types of web content objects (list, text, product etc.) into the database. Database schema is generated using generated regular expression based on the frequent pattern matching structure exists on the DOM tree of the list web page.
3. Since different B2C web source follows different sequence pattern to represent content object. Our algorithm generalizes DFA from different web sources.

1.10 Outline of thesis proposal

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 chapter, we survey the literature related to the shortcoming addressed in this thesis include web content mining and how to bring web content from different web sites into a unique format. There are several studies has been made on web content mining. They are categorized into four categories: manual approach, supervised approach, semi-supervised and unsupervised approach. Supervised or manual approach uses wrapper which is generated using a set of web pages labeled with examples of the data to be extracted. Wrapper generation requires a set of data extraction rules which are generated manually from labeled pages. Manual labeling of pages is labor intensive and time consuming because different templates exist in different sources. Semi-supervised approach accepts a rough training example from user and generates extraction rule. Unsupervised or automatic approach generates wrapper without much user interaction. Since unsupervised approach performs better than other three, we only consider unsupervised approaches in this section. And our thesis extends the work of Mutsuddy and Ezeife (2010), Ezeife and Mutsuddy (2013) which is an unsupervised approach. We present previous studies which are broadly related to unsupervised or automatic web content mining. We categorized these techniques based on the techniques for data area identification and record segmentation including comparison-based, grammar-based and separator-based.

2.1 Comparison Based Approaches

These approaches find commonalities and identify records by comparing page fragments. MDR (Liu et al., 2003) is a comparison based approach that relies on string edit distance.

Another comparison based approach called DEPTA (Zhai and Liu, 2005; Zhai and Liu, 2006) which uses a tree edit distance. NET (Liu and Zhai, 2005) uses a tree edit distance as well but collapses shared subtrees. We describe technical details of these approaches in the following section.

2.1.1 MDR: Mining web records from web page

Liu et al. (2003) addressed the problem of mining data records in web page using the existing approaches. They identified three types of existing approaches include manual, supervised learning and automatic techniques. They mentioned that manual approach is not useful for large number of pages, supervised techniques need training data which are prepared manually and for that reason it requires substantial human effort. They also mentioned that existing automatic approach provides unsatisfactory result due to their poor performance. The authors propose a new approach called MDR which is based on two observations. Firstly, a group of data records form a contiguous region of a page and within the data region the data records have similar tag tree structure. Secondly, the data records of a data region have the same parent node. The proposed technique works on three steps. In the first step, it builds a tag tree with HTML tag of the source page. For example, a tag tree of HTML page is shown in Figure 12. In the second step, it mines every data region that contains data records. In this step, first it mines generalized nodes which form a data region based on two properties include the nodes all have the same parent and the nodes are adjacent. For example, Figure 12 presents two generalized nodes where the first 5 TR nodes of the TBODY consist by the first data region and second the data region contains rest of the TR nodes. The proposed algorithm identifies the data

region by string comparison between generalized nodes. For example, Figure 13 represents an artificial tag tree to explain different kinds of generalize nodes and data region. The generalized node is represented by the shaded area. The nodes 5 and 6

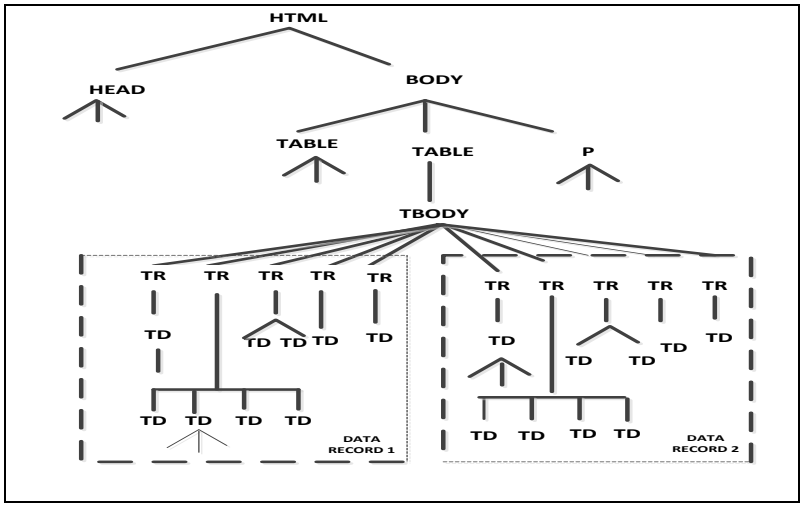


Figure 12: A tag tree representation of a HTML page

formed the data region labeled 1 and their length is 1. The nodes 8, 9 and 10 formed the data region labeled 2 and their length is 1. The pairs of nodes (14, 15), (16, 17) and (18,19) formed the data region labeled 3 and their length is 2. These generalized nodes formed the data region based on the edit distance properties which is that the normalized edit distance between adjacent generalized nodes is less than fixed thresholds.

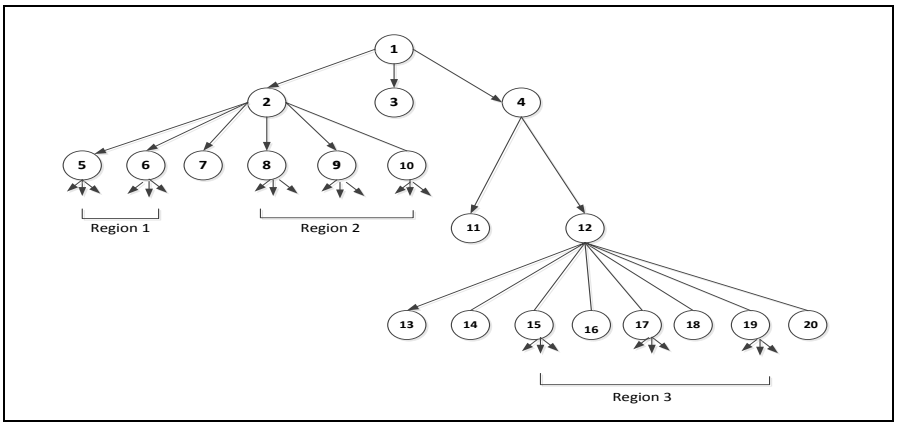


Figure 13: Artificial tag tree

To identify the data region, the proposed mining algorithm finds the first generalized node of a data region and it is possible when starting from each node sequentially. For example, node 8 is the first node of data region 2 in Figure 13. It also needs to find the number of components that a generalized node contains. For example, each generalized node of data region 2 has one component in Figure 13. The algorithm finds the tag nodes or components does a generalized node in each data region have by doing one node, two node combination, ….., K node combination. It starts from each node and perform all 1-node string comparisons, all 2-node string comparisons and so on. Then the comparisons result is used to identify each data region. The comparison process is shown in Figure 14, where it contains 10 nodes with the parent node *p*. The algorithm starts from each node and continues with all possible combinations of component nodes. The following string comparisons are computed:

- (1,2), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10)
- (1-2,3-4), (3-4,5-6), (5-6,7-8), (7-8,9-10)
- (1-2-3, 4-5-6), (4-5-6, 7-8-9)

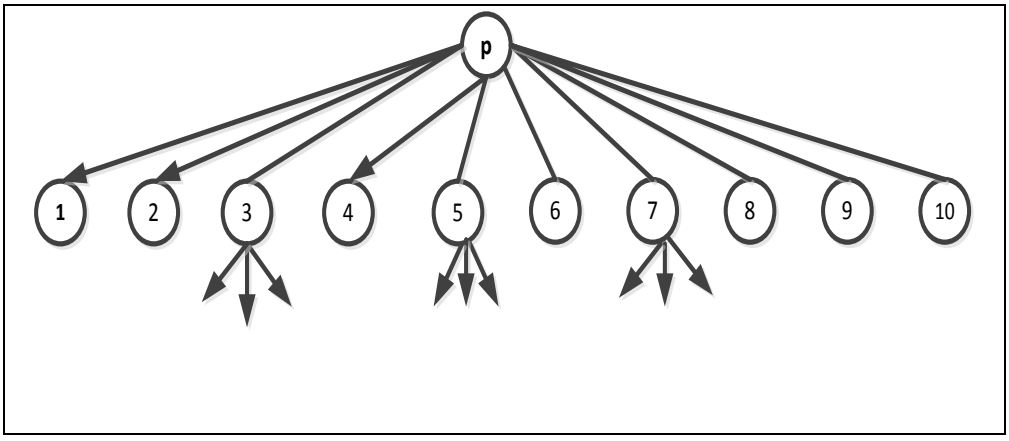


Figure 14: Comparison and combination of node

The pair node (1, 2) describes that tag string of node 1 is compared with tag string of node 2. The tag string includes all the tags of the sub-tree of the node. For example, the tag string for the second TR node (Figure 12) of TBODY is <TR TD TD……..TD TD>. Here, the substring of sub-tree below the second TD nodes is denoted by "….". The pair node (1-2, 3-4) describes the comparison between the combined tag string of node 1, 2 and combined tag string node 3, 4. After doing all the string comparisons, the MDR algorithm identifies each data region by finding its generalized node. Basically, the algorithm finds similar children node combinations to identify candidate generalized nodes and data region of the patent node by using the string comparison results at each parent node. After that MDR identifies the data records in each region. MDR identifies data records based on the assumption is that if a generalized node is the combination of two or more data records then these data records contain similar tag strings. The authors claim that the proposed method which is able to extract web data automatically. Also their proposed method able to discover non-contiguous data records which didn't handle with existing system because the proposed method is developed based on nested structure and presentation feature of web pages. The authors conducted an experiment of their proposed approach with 18 pages from OMNI's web site and a large number of other pages from different domain like books, travel, software, auctions, jobs, shopping and search engine result. They also used a number of training pages to build their system and identify their default edit distance threshold. The authors obtained the result from their experiment is that MDR has 99.88% recall and 100% precision where other system OMINI and IEPAD only have a recall of 39%. They also mentioned that some data records which are consider correct for OMINI and IEPAD, if these data are not consider

as correct then the recall of OMINI and IPEAD reduce to 38.3% and 29% respectively. In that case the precision value they obtain is that 56% for OMINI and 67% for IEPAD. The authors claim that their approach doesn't need any human effort and it mines data records in a page automatically. They also claim that their algorithm able to extract non-contiguous data records. The shortcoming of this approach is that MDR is designed to handle tables tag only. It failed to extract data from web page which contains records that have complex and nested structure. Reis et al. (2004) described that the limitation of this approach is that the proposed algorithm works each time in a single page, so it does not compare the page trees. Although achieving good results, the algorithm only works with multi-record pages and therefore cannot be applied to on-line news page, that are almost exclusively single-record pages. Miao et al. (2009) identified the limitation of this approach is that it does not handle nested data objects.

2.1.2 DEPTA: Web data extraction based on partial tree alignment

Zhai and Liu (2005) addressed the problem of data record extraction from web page. They state that the machine learning approach is time consuming and needs human effort because it requires manually labeling of many examples from each web site for data extraction and this approach is not able to expand to cope with large number of pages. They also found that existing automatic approaches related to pattern discovery are developed based on many assumptions and provide inaccurate results. The authors propose a new architecture for automatic data extraction from web pages. The proposed architecture is called DEPTA. It has three steps to extract data automatically. In the first step, the method builds a HTML tag tree using visual information. The observation behind this step is that each HTML element made with start tag, optional attribute,

optional embedded HTML content and an end tag, is rendered as a rectangle in a web browser. The DEPTA builds tag tree based on the nested rectangle. For doing this, it uses embedded parsing and rendering engine of a browser to find the 4 boundaries of the rectangle of each HTML element and then it checks whether one rectangle is contained inside another rectangle by detect the containment relationship within the rectangles. For example, Figure 15 represents the HTML code on the left which is a table with two rows and right side represents the boundary coordinated produced by the browser for each HTML element shown in the right side.

```
1   <table>
2     <tr>
3         <td>.....</td>
4         <td>.....</td>
5     </tr>
6     <tr>
7         <td>.....</td>
8         <td>.....</td>
9     </tr>
10  </table>
```

LEFT	RIGHT	TOP	BOTTOM
100	300	200	400
100	300	200	300
100	200	200	300
200	300	200	300
100	300	300	400
100	200	300	400
200	300	300	400

Figure 15: A HTML code segment and boundary coordinates

The tag tree is shown in Figure 16 is build based on the visual information which is the sequence of opening tag and also done by the containment check. After building the tag tree DEPTA mines data region in a page that contain similar data records. The DEPTA identifies the data region using string comparison between generalized nodes. For example, Figure 17 represents an artificial tag tree to explain different kinds of generalize nodes and data region. The generalized node represents by the shaded area. The Nodes 5 and 6 formed data region labeled 1 and their length is 1. The nodes 8, 9 and 10 formed the data region labeled 2 and their length is 1. The pair of nodes (14, 15), (16, 17) and

(18, 19) from the data region labeled 3 and their length is 2. These generalized node formed the data region based on the edit distance properties which is that the normalized

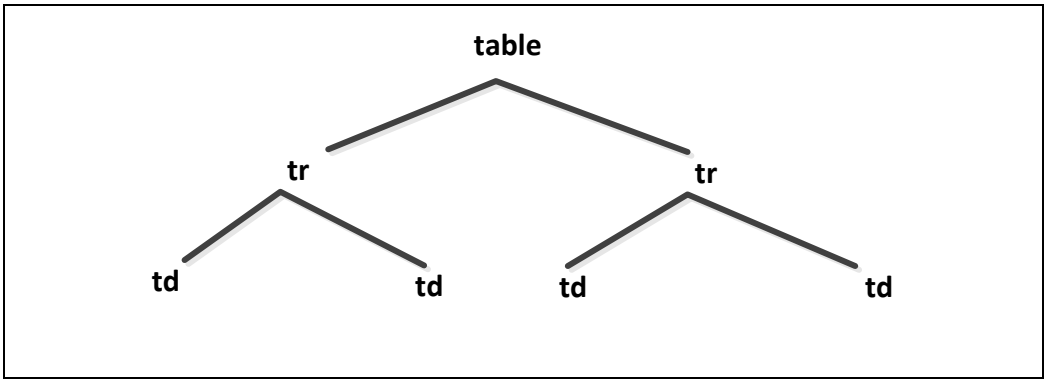


Figure 16: Tag tree for HTML code in Figure 15

edit distance between adjacent generalized nodes is less than fixed thresholds. To identify the data region, the mining algorithm finds the first generalized node of a data region. And it is possible when starting from each node sequentially. For example node 8 is the first node of data region 2 in figure 17.

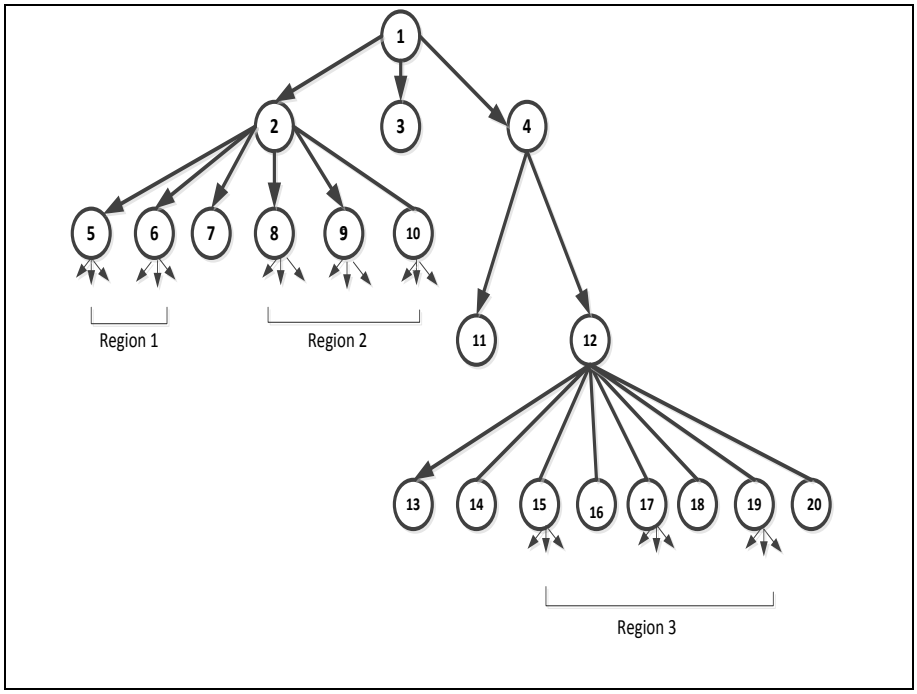


Figure 17: Artificial tag tree

It also needs to find the number of components that a generalized node contains. DEPTA identifies data records from generalized node after all the data regions are identified. DEPTA used the same technique as MDR to identify the contiguous and non-contiguous data records. After identifying the data records, DEPTA extracts data from data records using partial tree alignment technique.

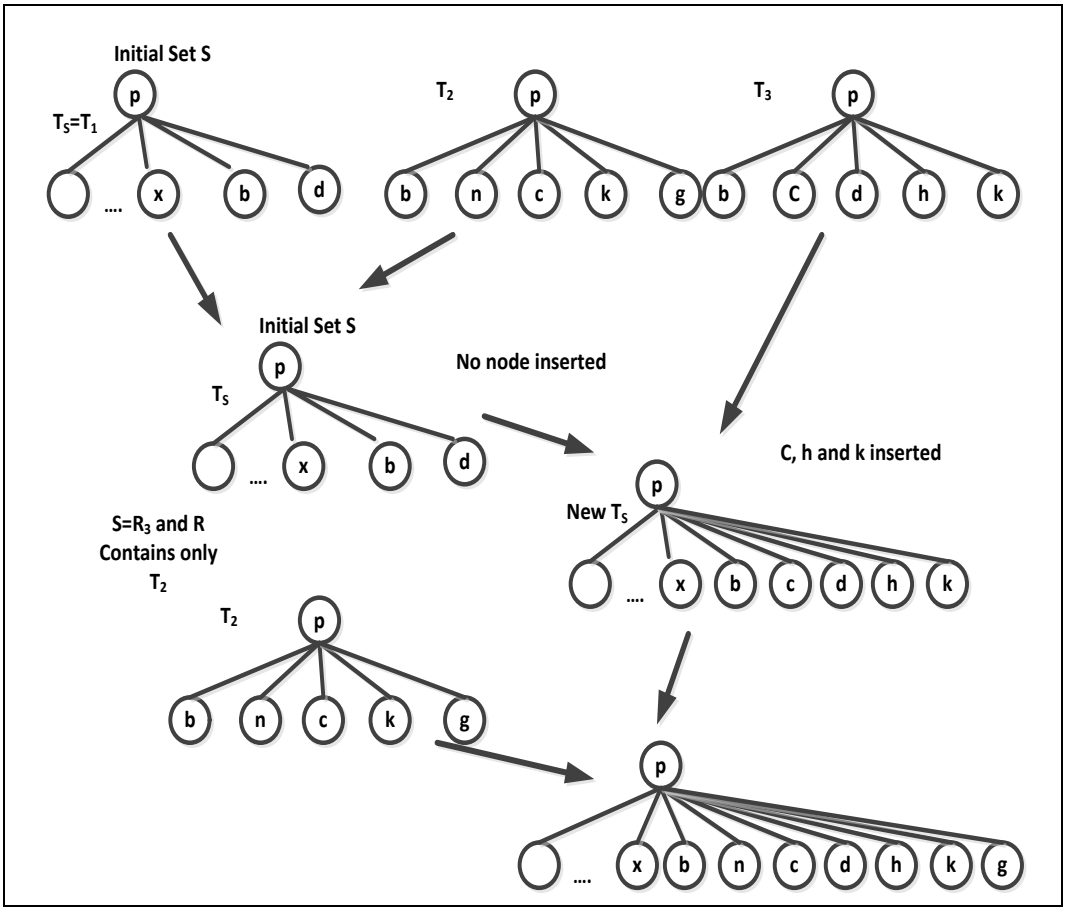


Figure 18: Iterative tree alignment with two iterations

DEPTA grows a seed (tag) tree denoted by T_s to align multiple tag trees. The seed tree T_s initially picked based on the maximum number of data fields. Figure 18 represents an example how the seed tree T_s is build. At first, the algorithm of DEPTA finds the tree that contains most data item. In the Figure 18, T_1 is the seed tree. After that T_2 and T_3 are aligned with T_s to generate the unaligned tree. Then DEPTA do the tree matching and by

using the matrix results it finds all the matched pairs. It is found that T_s and T_2 has one match node *b* where node n, c, k and g are not matched to T_s. Now DEPTA attempt to insert them into T_s to satisfy the partial tree alignment requirement. But it is found that none of the n, c, k and g in T_2 can be inserted into T_s due to the unique location. Then T_2 is inserted into R which means that these nodes need to be further process. When DEPTA compares T_3 with T_s and it finds unmatched nodes c, h and k can be inserted into T_s. For that reason T_3 doesn't need to insert into R. After completion this step R is picked to process again. R only contain node of T_2 and it is matched with the T_s in the next step. For complete the matching process every node of T_2 are matched or inserted. At the end DPETA follows the alignment procedure to produce the data item from each tree. Each un-matched data will generate a single column itself, if there are any unmatched nodes with data still available. The authors performed an experiment in which they tested their method with 49 different web sites which consists 72 pages. These pages are collected randomly. They compared the step-1 of DEPATA with MDR. In their previous work, they checked the performance of the MDR compare to other existing system and proved that MDR perform well better than those systems. In this approach, the authors proved the performance of the DEPA is better than MDR. They obtained that the precision and recall of DEPTA is 99.82% and 98.27% respectively for step 1 where recall and precision of MDR is 86.64% and 97.10%. They also obtain that the precision and recall of DEPTA is 99.68% and 98.18% respectively for step 2. The authors claim that DEPTA can segment data records and extract data from web page very accurately. They also claim that new version of MDR which is MDR-2 is able to handle nested date records due to nested similarity comparison. They also claim that the partial tree alignment technique is

able to align data items in nested record. The authors also claim that more robust tree can be build by using visual information. And it is possible to find more accurate data region with the help of visual information. The shortcoming of this approach is that DEPTA did not consider semantic label in data extraction where they only use tree regularities. DEPTA failed to extract nested data records. Another limitation of DEPTA is that it is only for list page that contains multiple data records. Senellart et al. (2008) mentioned that this approach is less accurate than supervised approach.

2.1.3 Net: A system for extracting web data from flat and nested data records

Liu and Zhai (2005) addressed the problem in extracting data from web page. They identified the problem of wrapper generation for data extraction from web. They mentioned that wrapper generation requires a set of data extraction rule which are generated from manually labeled page. But manually labeling is related to labor intensive and time consuming because different page uses different template and for this reason manual labeling has drawbacks for the large amount of pages. The authors proposed an approach for data extraction which is based on tree edit distance and visual cues. They design their algorithm to traverse the tree from post order (bottom-up) to extract nested data record since nested data records are found at a lower level on repeating pattern. They define a method called Traverse() to traverse the tree and it traverse the tree with the depth is greater than or equal to 3 because the authors observation is that tree with depth 2 or 1 do not contain any data record. They also define a method called Match() to match two child subtree of Node. They also define a sub-method called TreeMatch() to match two child subtree under node and it applied on every pair on child nodes to ensure every

data matches are captured. The method AlignAndLink() align and links matched data items. The method TreeMatch() finds the repeated pattern from list of data record by using restricted tree matching algorithm called simple tree matching (STM). For example, A = <R_A, A_1, A_2,…, A_m> and B=<R_B, B_1, B_2,…, B_n> be two trees, with the root R_A and R_B respectively. And A_i, B_j are the i^{th} and j^{th} first-level sub-trees of A and B respectively. The algorithm identifies maximum matching between A and B is $M_{A,B}$+1, when R_A and R_B match . In the simple_Tree_Matching algorithm, first it compares the roots of A and B. After that the algorithm recursively finds the maximum matching between first-level sub-trees of A and B if the roots match and used W matrix to save it. Based on the W matrix the algorithm finds the number of pairs in a maximum matching between two trees A and B by using a dynamic programming scheme. The authors applied visual based condition to make sure that A and B has no visual conflict. For example, trees are unlikely to match based on the visual information, if the width of A is much larger than that of B. These rules perform better in match results and also computation is reduced significantly.

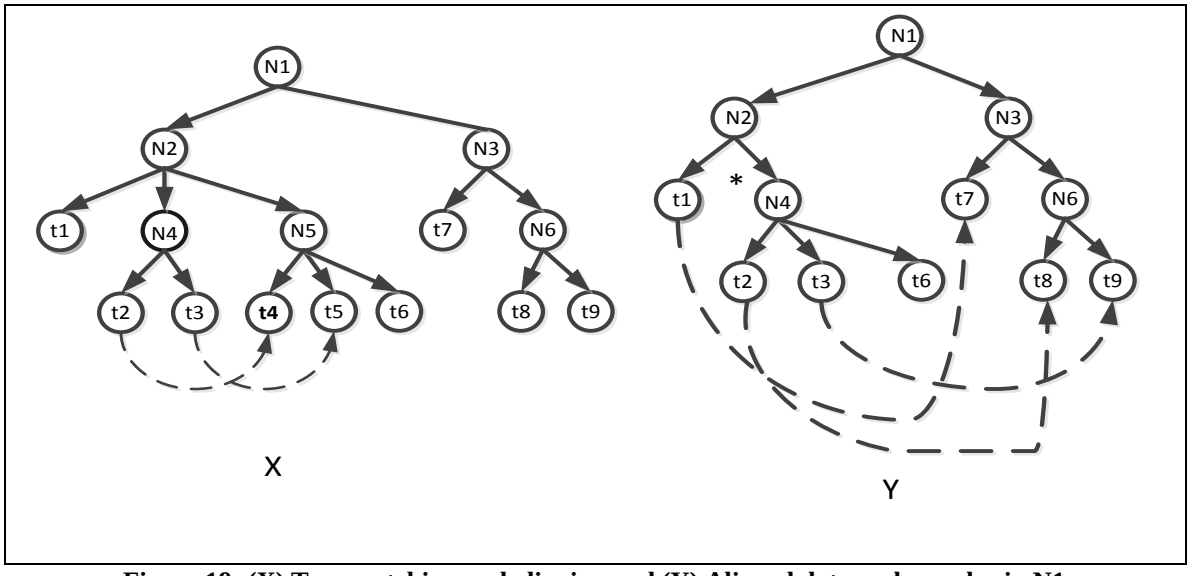


Figure 19: (X) Tree matching and aligning and (Y) Aligned data nodes under in N1

The aligned data items are then linked directionally where an earlier data item will point to its next matching data item. Figure 19 gives an example, a terminal (data item) node is represented by t_i, and a tag node is represented by N_j. Since the algorithm follows post-order traversal, at the level N_4-N_5, t_2-t_4 and t_3-t_5 are matched and they are aligned and linked. It is found that N_4 and N_5 are data records of N_2 as nested and t_6 is optional. The method TreeMatch() will only match N_4 subtree and N_6 subtree at the level of N_2- N_3. It is found that t_2- t_8 and t_3- t_9 are linked and t_1 and t_7 are also linked as they match (Figure. 19). Since N_5 has the same structure as N_4, the subtree at N_5 is omitted in Figure 19(Y) and N_4 is marked with a "*". In Figure 19, as it is turned into a prototype data record by enPrototypes(). The node t_6 is inserted into N_4 as an optional node, denoted by "?". A standard/typical data record containing the complete structure so far is represented by prototypes. The linked data items are inserted into the table using PutDataInTables() (Figure 20). A table is a linked list of one dimensional array, which represents columns. All linked data items are put in the same column. If an item is being pointed to by another item in an earlier column then a new row is started.

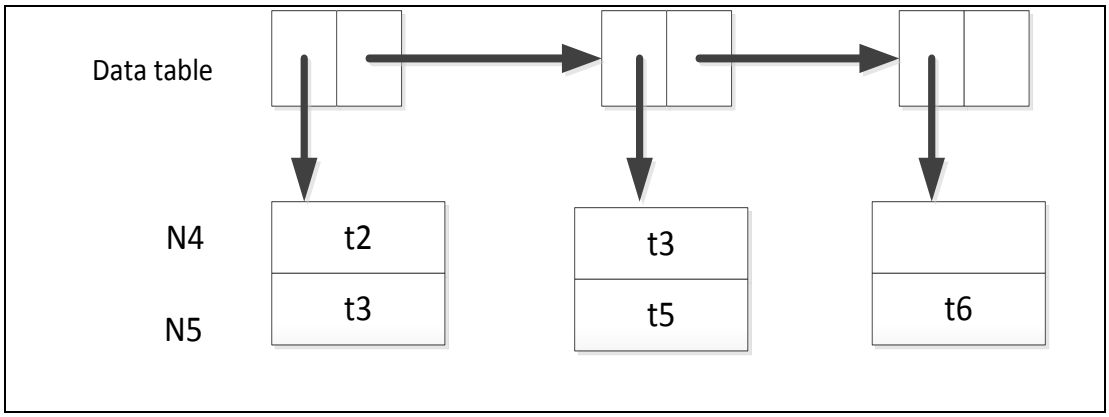


Figure 20: Data table for N4 and N5

For example, for node N_2 in Figure 19, the method putDataInTables() produces the *DataTable* in Figure 20. For node N1 in Figure 19, it produces the *DataTable* in Figure

21. The method produces prototypes after putting data into tables. The tree structure followed by GenPrototypes() based on the first data record (e.g., N_4 in Figure 20) and tree paths which represents optional items not in the first data record are inserted, but in other data records. The optional items occupy some columns that do not have data items in the first data record and for that reason they are deleted from the table. In the example of Figure 21, an optional item $t_6^?$ is added to N_4 which gives * N_4 (the prototype). In Figure 21, it is found that $t_6^?$ is attached to *N4.

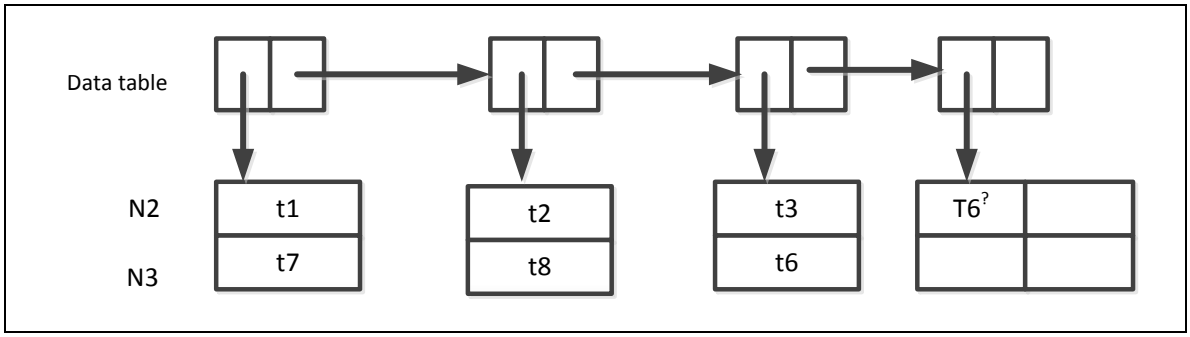


Figure 21: Data table for N2 and N3

The shortcoming of this approach is that Net proposed a greedy approach based on similarity match. It employs expensive approach due to bottom-up traversal with edit distance comparison. It requires full scan from bottom to root. Net does the all-pair tree comparisons within its children during each visit of a node in the traversal. Alim et al. (2009) describes the limitation of Net is that wrappers generated by NET are not efficient though because the programmers have to find the reference point and the absolute tag path of the targeted data content manually. This requires one wrapper for each web site since different sites follow different templates. The effects are increased time consumption and effort from the programmer.

2.2 Separator-based approach

In this approach, a tools searches for tags, tag-sequences or trees as separators to segment a data area into records. This approach was taken by early tools, namely BYU-Tool (Embly et al., 1999) and Omini (Buttler et al., 2001), but much more recent by ViNTs (Zhao et al., 2005), OWebMiner (Annoni and Ezeife, 2009) and WebOMiner (Mutsuddy and Ezeife, 2010; Ezeife and Mutsuddy 2013)

2.2.1 BYU-Tool: Conceptual-Model-based data extraction from multiple-record pages

Embly et al. (1999) addressed the problem of unstructured data on the web which makes the searching difficult and database querying impossible. They identified that most of the web data is unstructured and traditional query language can't be used for query. The authors proposed a new approach which is based on manually constructed domain-specific ontology. Their proposed model also relies on structural encoding properties. This model is considered as a fully automated and parameterized by domain specific ontology. The ontology which is used in this model is based on concept, relationship and specialization relations where concepts are either lexical or non-lexical, relationship between concepts have optional participation constraints and the specialization relationships allow to specify the concept as specialization of other concepts. In this model, each concept is associated with a data frame to link ontology concepts with proceed documents. A regular expression is contained by the data frame to describe all possible encoding concept instances in lexical concept. A normalization rule is applied to extract the instance in a normalized form. The data frame is used to describe the context keywords in both lexical and non-lexical concepts which indicate the presence of a

corresponding object instance. The authors conducted two experiments. In the first experiment, the ontology is applied to a limited corpus of test obituaries from two different sources. They conducted the second experiment of greater quantitative and qualitative scope in order to demonstrate the robustness of the approach and the general applicability of this ontology. They collected a new corpus of obituaries which exemplified wider variability in style and content for the second experiment. As a test data they took 38 obituaries from a web page provided by the Salt Lake Tribune and 90 obituaries from a web page provided by the Arizona daily star. In the result they found that the average recall is 90% and the average 75% precision for names and average 95% precision elsewhere was a pleasant surprise. The authors claim that the model describe fully automates wrapper generation for web documents that are rich in data, narrow in ontological breadth and have multiple records in single page. They mentioned seven items as future work include finding and classifying web pages of interest for a given application ontology using an ontological approach, enhancing the approach for unstructured record identification, indentifying records of interest both within a page or on a set of related pages using the application ontology, improving the model to identify attribute-value pairs and construct database tuples, adding richer data conversion to the data frames, inferred data as well as extracted data can be inserted into the database by providing a means to do inference and use more extensive quality metrics. The limitation of this approach is that it requires human effort because ontology for different domains must be constructed manually by an expert.

2.2.2 OMINI: A fully automated extraction system for the world wide web

Buttler et al. (2001) addressed the problem of information extraction from web using wrapper. They mentioned that programmer needs to understand the specific presentation layout or specific content web page to construct the wrapper and it is labor intensive and error prone because web site information are changed very frequently. They also mentioned that it is hard to maintain additional or new content into the existing integration framework. The authors propose a new approach called OMINI which is a fully automatic object extraction system. The OMINI uses tree structure to parse the web pages. It performs the object extraction from web pages into two stages. In the first stage, the location of interest object contained by the smallest subtree is searched by the subtree extraction algorithm. In the second stage, it finds the correct object separator tags. Both the stages perform their task automatically. OMINI uses the standard derivation (SD) technique, repeating pattern heuristics (RP) for minimal subtree extraction and object boundary identification which is also used by Embly et al. (1999). The system OMINI takes a URL as input and returns extracted list of objects from the given web page as output. The OMINI works on three phases include preparing web document for extraction, locating objects of interest in a web page and extracting objects of interest in a page. In the first phase, it prepares the web document for extraction by taking URL from end user or an application and performs the tasks including fetch web site of the given URL from the remote site. It makes the web document well formed by using the syntactic normalization algorithm and converting the web document into tag tree representation based on the nested structure of start and end tags. In the second phase, OMINI locates objects of interest in a web page and does this part into two steps. In the first step, it

extracts the object-rich subtree which is the minimal subtree that contains all the objects. In the second step, it extracts object separator which finds the object separator tag that separates the objects. For example, given a web document, object discovery phase identifies the primary content region from the document which is converted to the tag tree. The target of the object rich subtree discovery is to locate the object of interest contained by the minimal subtree of T. Here the objects means which need to be extracted in the search result which is presented in the web document by the twelve tables at the right side of the tree. Here the subtree heuristic obtain the tag node HTML[1].body[2].form[4] which is minimal subtree that contains all the news objects of interest. In the object separator extraction phase, OMINI uses a set of individual algorithms include standard deviation heuristic (SD) which measures the standard deviation in the distance between two consecutive occurrences of a candidate tag and based on their standard deviation it ranks the list of candidate tag in ascending order, the repeating pattern heuristic (RP) which counts the number of occurrences of all pairs of candidate tag that have no text in between by choosing the object separator. The identifiable path separator tag heuristic (IPS) which ranks the candidate tags of the chosen subtree according to the list of system supplied IPS tag(most commonly used object separator tags for different types of subtrees in web document), sibling tag heuristic (SB) which counts the pairs of tags that are immediate sibling in a tag tree and partial path heuristic (PP) which lists the paths from a node to all other reachable nodes and counts the number of occurrences of each path. Each of them independently identifies a ranked list of object separator automatically to decide how to separate data objects from each other. After finding the object separator, OMINI extracts the object of

interest in a page. And it is two steps processing include candidate object construction and object extraction refinement. The objects are extracted from the raw text data of the web document in the process of candidate object construction. Object separator which is extracted in phase 2 is used in this process by choosing the objects needed to be extracted from the components of the chosen subtree. At the end, the objects that do not conform to the set of minimum criteria are eliminated in the process of object extraction refinement. This process is involved to remove those objects that are not are the same structure like objects that are missing a common set of tags or objects that have too many unique tags. Also the objects that are too big or too small are removed in this process. The authors conducted a series of experiments over 2000 web pages from 50 popular web sites. They first generated a random list of 100 words from the standard unix dictionary to retrieve the pages automatically. After that they fed each word into a search from of the 50 web sites. They discarded the page with no result after retrieving the page. They used manual approach for the static web page which do not have search interface. They conducted the experiment with all local version of the page to ignore overload web sites to obtain consistent result overtime. The authors obtain the result where a recall ratio in the range of 93%-98% and precision ratio of 100% in these experiment. They also compare their result with Embley et al. (1999) where the heuristics only achieved a success rate of only 59% and the success rate of author's approach is 93%. The authors claim that their approach is fully automated to extract object from web. They mentioned that they have interest to include the automation of evaluation process and incorporation of evaluation feed-back refinement of object extraction in the future. Also they have a plan to do the integration with query optimization and semantic interoperability software

system in future. The limitation of this approach is that the author did not address how to precisely locate the data object instances in the separated parts and how to extract them by their specific structure. The separator contains only one HTML tag which is insufficient. Wang and Lochovsky (2003) described the limitation of OMINI is that this approach only good for segmenting web pages into parts, possibly containing data object instances. Liu et al. (2003) identified the limitation of this approach is that this model performs poorly on some web pages, the description of one data objects may intertwine with the descriptions of some other objects.

2.2.3 ViNTs: Fully automatic wrapper generation for search engine

Zhao et al. (2005) addressed the problem of manually generating program that extracts record from dynamically generated search result pages due to the response of submitted query in search engine. They state that manually approach is costly, time-consuming and impractical. They identified that search engines require manual maintenance of the extraction program due to frequently changing their result display format. They also state that it is time consuming to construct a wrapper manually for each search engine if an approach aims to connect to hundreds of thousands of search engines. The authors propose the new approach called VINTs. The main focus of this approach is wrapper generation. First, it identifies some candidate result record from each sample result page by analyzing the types such as link or text and the position of all the rendering boxes. Then it builds some initial wrapper based on these records and a hypothesis about the general format of the SRR wrapper. After that generated wrappers are refined to indentify the boundaries which separate different types of records. In the next step, VINTs uses additional visual feature to select most promising wrapper for the result page from the

refined wrapper. In the final step, the generated wrappers are integrated to produce the final wrapper for the search engine. The VINTs describes the block as a sequence of content line types and indentations. The content line types contain the link which is basic types, text, link-text, the head variants link-head, text-head, and link-text-head and the types hr-line and blank. The distance of the starting point of the line from the left hand side of the screen is measured by the indentations. At first, ViNTs compares two blocks by computing the sequence of line types and the sequence of identifications. The normalization of the both sequences is done separately. ViNTs uses the technique called modified shape code of a block which generates from subtracting the minimum positions in the sequence from all occurring position which is the normalized position sequence. Three distance measures on these sequence is applied by ViNTs include type distance, shape distance and the position distance. ViNTs arranges the blocks into candidate group after dividing the lines of page into blocks such as the grouped blocks are similar according to the three distance measure. ViNTs also builds individual wrapper based on a common result template for search result.

R#	Tag path
1	<HTML>C<HEAD>S<BODY>C<IMG>S<CENTER>S <HR>S<B>S<HR>S<DL>C<DT>C<STRONG>C
2	<HTML>C<HEAD>S<BODY>C<IMG>S<CENTER>S <HR>S<B>S<HR>S<DL>S<DL>C<DT>C<STRONG>C
3	<HTML>C<HEAD>S<BODY>C<IMG>S<CENTER>S <HR>S<B>S<HR>S<DL>S<DL>S<DL>C<DT>C <STRONG>C
4	<HTML>C<HEAD>S<BODY>C<IMG>S<CENTER>S <HR>S<B>S<HR>S<DL>S<DL>S<DL>S <DL>C<DT>C<STRONG>C

Figure 22: Tag path extracted from web document

The VINTs generates initial wrapper with the tag path of the records in each sub-groups and the hypothesis about the format of the wrapper (prefix (X) (separator1 |

seperator2|…))[min, max]). There is a possibility that different initial wrapper or no initial wrapper can be generated from the different sub group. In this step, prefix and the separators are identified. The parameter min and max is indentified in the next step which is refinement step. Figure 22 is used to describe this step. First it finds the maximum common prefix PRE of all input tag paths. In the running example, PRE =<HTML>C<HEAD>S<BODY>C<IMG>S<CENTER>S<HR>S<B>S<HR>S<DL>S.

It can be different from the needed wrapper. There is a possibility that PRE contain the correct prefix and also additional path node at the end. By removing PRE from tag path it is possible to indentify the addition path node at the end. For example, Let P_i = path (r_i) – PRE and then compute $Diff_i$= p_{i+1}-p_i where pi is a suffix of $p_{i+1.}$ The separator can be indentified when the differences are the same. In the running example Diff= <DL>S is the separator. After that all the occurrences of Diff are removed from PRE. Here PRE1 is the new PRE and E is the last node of PRE1. Now additional separator is identified by comparing the Diff and E and it is checked that is there any Diff occurs before E. The path node E is indentified as a new separator when both the conditions are satisfied and E also removed from PRE1. This process continues until new separator is identified and the remaining tag path of PRE1 is the prefix of initial wrapper. In the running example, only one separator is identified and the final prefix is <HTML> C<HEAD>S <BODY>C <IMG>S<CENTER>S<HR>S<B>S<HR>S. There are three cases are identified if the Diffs are different. Case 1: There is no common suffix of the Diffs and then the process of wrapper generation fails and process terminated. Case 2: There is a common suffix but it doesn’t have multiple occurrences in any Diffs and then suffix is the separator which is identical to any of the Diffs and subtract from PRE. The remaining PRE is the prefix for

initial wrapper. Case 3: Some Diffs contain common suffixes which have multiple occurrences. Then each Diffs are expanded by taking the structure of the child nodes of the nodes in the Diffs into consideration. The expanded Diffs are used in the second case to indentify the separators. The wrapper building process fails if the separator not found. From the above two step, for the running example the initial wrapper is <HTML>C<HEAD> S<BODY>C<IMG>S <CENTER>S<HR>S<B>S<HR>S(X<DL> [0,∞], where X is a wild card. The initial wrapper is used to extract all matching records from the result page. If it can successfully extract record then the wrapper is accepted from refinement step. If it fails to extract record then the wrapper is incorrect. Then the node in the separator is expanded by their child node in step 2 to find a new separator. If the new wrapper is found then the initial wrapper is revised and above process is repeated. The wrapper building process fails if the new wrapper cannot be accepted or a new separator not found. A tag path is matched by the template as common prefix, leading to the data area and contains a number of separators to segment the data area into records. ViNTs choses the final wrapper based on four criteria include the relevant data areas is large, resides in the middle of the page, contains a large number of records and contains records with a large number of characters. The authors used a commercial tool ICEbrowser for result page rendering and tag tree construction. They used Pentium 4 with 1.7GH PC to generate wrapper for a search engine with 5 sample result pages and 1 no-result page. And the wrapper is build in 3 to 7 seconds. The authors tested ViNTs with three data sets. Data set 1 has 4types of 100 search engine include education, government, medical and general. Data set 2 has 100 search engines which are collected from profusion.com and these are not included in data set 1. For these two data sets, there

are 10 queries are submitted and 10 first result pages are collected manually. They used a non-existent term as a query for collecting no-result page for each search engine. They used data set 3 which is obtained from Omini testbed and it is a collection of more than 2000 web pages from 50 web sites. The author obtained that ViNTs able to generate very high quality wrappers with the precision and recall close to 100% on data set 1 and close to 98% on data set 2. The authors identified the reason of 2% decrease performance due to the failure of ViNts on 2 search engines in date set 2. The authors claim that their approach is fully automated and this technique can be achieved considerably higher extraction accuracy than that of the state of art web information extraction systems. They also claim that their approach implicitly employs a method on search result records on current search engines to extract these records. The authors mentioned that they have plan to improve ViNTs by utilizing additional visual features to further reduce the reliance on HTML tag structure. The limitation of VINTs is that it will fail to separate horizontally arranging data records which will require vertical separator due to fact that it only supplies horizontal separator. Other limitation is that at least four data record have to be present in a web page for wrapper building. Zheng et al. (2007) describe about the limitation of ViNTs is that since this approach based on visual layout information, it is difficult to identify visual information without any assumptions about the target domain. The visual feature used in ViNTs are only limited to the content shape-related features and it is used to identify the regularities between search records. For this reason, ViNTs depends on structural similarities and must generate wrapper for each search engine.

2.2.4 OWebMiner: Modeling web documents as objects for automatic web content extraction

Annoni and Ezeife (2009) identified three main problems in existing data mining approaches from web documents include existing systems failed to focus web search on either web document presentation or content or both, there is no unique framework which able to mine each web object based on their structure type e.g., unstructured, loosely or strictly structured and existing approaches are domain dependent and time consuming process. In this paper, the authors proposed a framework which presents we document as object-oriented web data model to represent web data as web content and web presentation objects to solve the above problems. The proposed framework is able to mine complex and structure data as well as simple and unstructured data in a unified way. They observed that <h1> or <a> tag in HTML is more meaningful than <pre> tag which used for pre-formatted text. The authors defined three main zones of a web document as instances of specialized classes include HeaderZone, BodyZone and FootZone. They state that two types of objects are exists in these zone include web content objects and web presentation objects. The authors classified the web content into six categories among them four have sub-content type include text element, image element, form element, plug-in element, separator element and structure element. The text element has two sub content-types include raw text and list text. A raw text has three sub-content types include title, label and paragraph. A list text has two sub-content types include ordered list and definition list. Image element has two sub-content types like image and map. Form element has three sub content types which are form select, form input and form text area. The authors classified the web presentation object into six categories

which are banner, menu, interaction, legal information, record and bulk. In the first step, DOM tree of a web document is generated from the HTML file using DOM parser. In the second step, web zones are indentified on the web document from the DOM tree (Figure 24). In the third step, web content object and web presentation objects are extracted. In this algorithm, block level tag (e.g., table, division, heading, list, form, block quotation, paragraph and address) and non block level tag (e.g., anchor, citation, image, object, span, script) are considered to extract object. Two search approaches are used in OWebMiner to explore the DOM tree (Figure 24). The depth-first search is executed through block-level tag until it finding non block level tag. The breadth-first search executed to parse non block level tag. In the web content extraction process, the authors define an algorithm which identified web zone from web document. It takes a web document DOM tree (Figure 24) as input and returns an array which contains web zone of the web document. The string comparison technique is used to parse tag sets. The authors identify two tag series called series 1 and series 2 in the web. Series 1 is the set of five or more <a> or <area> sibling nodes. And series 2 is the series which include the keywords ‘copyright’, “private policy”, “about our company”. In the proposed algorithm series 1 and series 2 are searched to identify the web zone objects. The authors also developed sub algorithm of OWebMiner called PresWebObjectScan() and ContWebObjectScan() to extract web presentation object and web content object repectviely. The method PreswebObjectScan() extract object such as “Menu”, “LegalInformation” etc. whereas ContWebObjectScan() extract object such as “TextElement”, “Definitation List”, “pluginserver” etc. Annoni and Ezeife’s (2009) main algorithm OWebMiner() is given in Figure 23:

```
Algorithm: OWebMiner()
Input: A set of HTML files (WDHTMLFile) of web documents.
Output: A set of patterns of objects.
Begin
      For each WDHTMLFile

            1. Extract web presentation objects and content objects
            sequentially with respect to their hierarchical dependencies.

            2. Store the object hierarchies into a database table
      endFor
      3. Mine patterns lying within objects

end
```

Figure 23: OWebMiner() algorithm (Annoni and Ezeife, 2009)

The input of their proposed algorithm (Figure 23) is a set of webpages (WDHTMLFile). The Line (1) of the algorithm will extract all the content and presentation objects for each WDHTMLFile into two separate object arrays according to their DOM hierarchical dependencies. The web objects are stored into database by line (2). Line (3) will mine the extracted contents from the database. They also developed sub-algorithm (1) of their main algorithms OWebMiner() called PresWebObjectScan() and ContWebObjectScan(). The method ContWebObjectScan() uses array data structure ContentObjectArray[] to store content objects. The process starts with the root of DOM Tree node "<html>"(Figure 24). When it finds the series-1, it calls the method ProcessContentSibling() to start extraction process of content objects and continue until it hits series 2. The method ProcessContentSibling() takes DOM Tree (Figure 24) as input, A pointer called "TTag" which indicate current tag to process in DOM Tree. The algorithm uses depth-first search to traverse DOM tree block-level tags until it hits non-block level tag and reset "TTag" pointer to represent current processing tag. It processes all it's siblings into an array called "tagArray", when depth-first search hits a non-block level tag. For all non-block level tags in "tagArray", the algorithm then associates a

content object to tag value. Otherwise it recursively calls itself to advance "TTag" pointer. The ContentObjectArray[] contains all content objects from body zone of web page. The authors stop at this point in their paper and left the remaining mining from the content object array as future work. The authors claim that their proposed object-oriented web data model able to distinguish content from presentation aspects of data e.g. title, label, image etc. They also claim that their algorithm able to extract objects e.g. title, label, image etc. from any given web document of any web application and it is domain independent. The authors also claim that their proposed framework is based on object-oriented approach which mines a web document as a set of object by extracting both the content and presentation views of web documents. The limitation of their framework is that it doesn't evaluate all the HTML tag because their observation is that all the tag in HTML are not meaningful. There are several limitations in this approach. This approach doesn't identify the data block and data region. It is important to identify data region and data block to extract web objects. Their approach based on "vision based context structure" and this is useful when using browser rendering engine. But for automatic extraction process without use of web browser, co-ordinate location of any feature is not possible. Their proposed algorithm did not address the use of separator element for identification of data block and data region. Their approach did not define the object classes, size of object classes, object class hierarchy, object class dependencies and functionalities of object class. They only classify the web content elements but did not associate object types with content, nor discuss how to control the creation of expensive objects. They did not address the issue of preventing noisy data entry into database table.

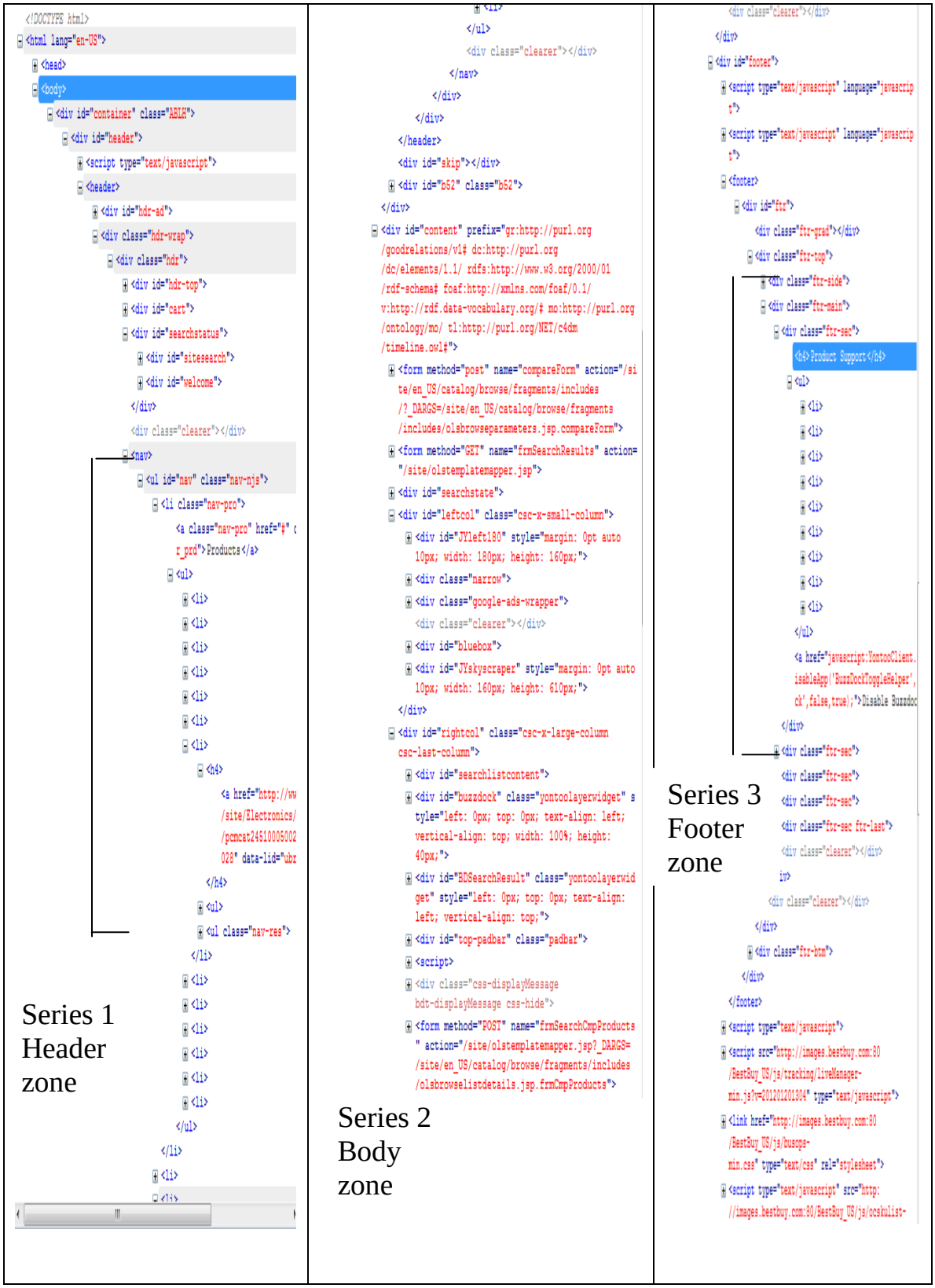


Figure 24: DOM tree representation of positive page from "bestbuy.com"

2.2.5 WebOMiner: Towards Comparative Web Content Mining using Object Oriented Model.

Mutsuddy and Ezeife (2010), Ezeife and Mutsuddy(2013) proposed a system for extracting and mining of structured web contents based on object-oriented data model. They developed the architecture called WebOMiner using object oriented model for extracting and mining of web contents. They introduced an approach of generating and using automata for mining web content objects. They define data block and data region to ensure consistency between related data. They addressed to relate HTML tag attribute information with related contents to ensure identification of content, to assign objects and other information together. They also defined object class hierarchies according to the problem domain and defined schema matching to unify similar contents from different web sites. They identified noisy contents in data blocks and prevent from them entering into database table. They also implemented and materialized object-oriented data model for web content and extract heterogeneous related web content together. They defined a mining algorithm that identify data block, generates non-deterministic finite automata based wrapper for extraction of related contents. They classified all data blocks of a web page according to their type and check minimum support to ensure data consistency before entering them into database. Their proposed WebOMiner system is an automatic object oriented web content extraction and mining system for integrating, mining heterogeneous contents that are also derived, historical and complex for deeper knowledge discovery. WebOMiner extracts information from a given web page includes data records (e.g., product image, product brand, product id, short description and price of the product), navigation information (e.g., link URL, link id or name), advertisement

(e.g., product advertised, image, URL links to related website). After extraction, WebOMiner stored this information into database for comparative mining and querying. Figure 25 is shown the main algorithm of WebOMiner.

Algorithm Main
Input: Set of HTML files (WDHTMLFile) of web documents.
Output: Set of patterns of objects.
Variable: ContentObjectArray[].

Begin
For each WDHTMLFile
A. **Call SiteMapGenerator()** to crawl and extract webpage into local directory from WWW.
B. **Call tagSoup.html()** to clean-up HTML code.
C. **Call WebOMiner.BuildDOMTree()** to create DOM tree of refined HTML file and extract web content objects sequentially from DOM Tree. Store objects in ContentObjectArray[].
D. **Call MineContentObject.IdentifyTuple()** to identify data records and classify records according to their pattern.
E. **Call CreateDBTable()** to store data records into a database table
endFor

F. Mine for knowledge discovery within extracted contents.
End

Figure 25: Main algorithm of WebOMiner

Their proposed approach contains four modules include crawler module, cleaner module, content extractor module and miner module. The proposed crawler module crawls the WWW given as input to find targeted web page. This module creates a mirror of original web document after streaming the entire web document including tags, texts and image contents. The comments are removed from the HTML document by this module. The cleaner module converts the downloaded HTML file well formed by inserting the missing tag, removing inline tag (e.g.,
, <ht/>), insert missing "/" at the end of unclosed <image> tag, clean up unnecessary decorative tags. The content extractor module converts HTML page into DOM tree and extracts contents from the DOM tree. This

module indentifies respective class object type as per pre-defined object class to the content. Also puts objects into Array List after setting information into objects. Data regions and data block are identified by this module and it segment the respective data of a data block from other data block by using separator objects. This module also generates seed NFA pattern for data blocks. It stores identical tuples after extracting objects of all tuples by matching with the refined NFA. It stores the objects into the database after checking the minimum support for all tuples categories. The shortcoming of this approach is that this method uses NFA for identifying tuples of web objects. But how the NFA is built, it is not defined by the author. And the NFA contains "ε" transition, but the author didn't mention how the "ε" transitions were handled. That means the author didn't mention how the NFA were used for identifying tuple without convert it to DFA. In this approach, the database schema generation is manual process which is labor intensive and time consuming. The author didn't define about schema integration of different domain specific website.

2.3 Grammar-based approach

RoadRunner (Crescenzi et al., 2001) and DeLa (Wang and Lochovsky 2003) describe the common structure shared by, respectively, different pages or different subtrees within the same page by inferring a grammar. Data fields in the grammar are used to identify the data to be extracted.

2.3.1 RoadRunner: Towards Automatic Data Extraction From Data-Intensive Web Site.

Crescenzi, Mecca and Merialdo (2001) begin by stating that there is no existing architecture that generate wrapper automatic for data extraction from web. They state that

since the amount of information in the web growing very fast, it is not easy task to access and manipulate these data through manually generated wrapper. The authors proposed an approach called ROADRUNNER which infers a grammar describe the common structure shared by, respectively, different pages or different subtrees within the same page. With this approach, each HTML page tokenizes and summarizes each text spawn into a single token. Initially, ROADRUNNER takes a page to generate initial wrapper and using this wrapper it parse the other sample pages. By doing this, it assumes that static and irrelevant data are indentified by similarities and dynamically generated and relevant data are identified by dissimilarities. ROADRUNNER generalizes the wrapper and generates a union free regular expression (UFRE) when each mismatch found during parsing the sample page. The relevant data fields identified with the UFRE. The algorithm considers two types of mismatch include text mismatch and tag mismatch. Text mismatch occurs when two text tokens are compared and return different text. In that case, ROADRUNNER assumes that text field contains a database field. The tag mismatch occurs when two different type tokens are compared. In that case, ROADRUNNER assumes that either an iterated or optional pattern causes the mismatch. ROADRUNNER identifies iteration by assuming that repeated pattern ended by the last common tag and that is the mismatch tag starts the pattern. ROADRUNNER searches for candidate pattern by using this assumption and try to match two successive instances of the assumed pattern recursively. ROADRUNNER identifies an optional pattern by assuming if any mismatch tag appears either in the wrapper or in the sample page, it can be skipped until a matching tag appear. ROADRUNNER generates a number of candidate patterns by using this assumption. Data area identification and record segmentation process done in a

single step by the ROADRUNNER. It infers a grammar which contains the corresponding information implicitly, instead of explicitly identifying a data area or record delimiters and the data exists in the record is not align and labeled. When ROADRUNNER generates candidate, that time it also identifies the starting or ending tag of iterated patterns and searches through the wrapper and parse page for a matching instance of the ending tag. The parse page is searched for the mismatching tag from the wrapper to identify potential optional patterns. If any occurrence found, then the algorithm generates a candidate pattern. In the same time, the wrapper is searched for the mismatch tag of the sample page and generates the candidate pattern. ROADRUNNER uses backward moving manner to match the candidate pattern for evaluating the candidate pattern iteration. For do this, the algorithm first compares the last matching tag with the ending tag. After that it compares the tag followed by the ending tag and the last matching tag and so forth. The backward matching process is the recursive process. Finally, the wrapper is generalized and the parsing process is ended. ROADRUNNER uses AND-OR tree for efficient searching of the candidates. In this process, for parsing the given page all the non-recursive mismatch must be resolved first and that time it generates an AND node. And it generates an OR node at the time of tag mismatch to choose the candidate patterns for iterative and optional pattern. The authors performed an experiment of their proposed algorithm on real HTML sites. They developed a prototype using java. They used *JTidy* to clean HTML sources, fix errors and convert the code to XHTML, and also build the DOM tree. They used Intel Pentium III processor working at 450MHz, with 128 Mbytes of RAM, running Linux (kernel 2.2) and Sun Java Java Development kit 1.3 to conduct their experiment. The authors provided two tables to

display the results. They listed the result of independent experiment with table A and with table B, the authors compared their result with other extraction systems include Wien and stalker. In table A they listed the result of class which is a short description of each class, (#w) number of wrapper created by the system, (#s) number of samples matching each wrapper, (extr) outcome of the data extraction process and schema which include (nest) level of nesting, (pod) number of attribute, (opt) number of optional. The element exits in table B include web sites and number of samples, target schema, (pcd) number of attributes, (nest) level of nesting ,(opt) if the page contain option element, if the attribute may occur different order (ord), results based on computing time of the three system, WINE and STALKER refer to CPU time which is required for learning. The authors claim that the proposed approach generate wrapper fully automatically. The generated wrapper is template independent which means that it doesn't depend on any prior knowledge of the target pages and their content. Also it doesn't require any user interaction. The author also claim that their approach is not restricted to flat record and it is able to handle nested structure. The authors also claim that ROADRUNNER can be work without prior knowledge about structure of the page. They state that ROADRUNNER needs lower time to learn the wrapper than WINE and STALKER and also able to handle nested structure. There are several limitations to the ROADRUNNER approach. This approach is based on the assumption is that the input page is generated by the template. But this assumption is not valid for the web site that contains HTML tag within data values. For example, if a web page contains several text paragraphs with <P> and <i> tag inside, ROADRUNNER will either fail to discover any template, or produce a wrong template. Arasu et al. (2003) describe the limitation of ROADRUNNER is that it

assumes that the “grammar” of the template used to generate the pages is union-free. This is equivalent to the assumption that there are no disjunctions in the input schema. The authors of ROADRUNNER themselves have pointed in (Crescenzi et al., 2001) that this assumption does not hold for many collections of pages. Moreover, as the experimental results in (Crescenzi et al., 2001) suggest, ROADRUNNER might fail to produce any output if there are disjunctions in the input schema. Arasu et al.(2003) also identifies the limitation of ROADRUNNER is that when it discovers that the current template does not generate an input page, it performs a complicated heuristic search involving “backtracking” for a new template. This search is exponential in the size of the schema of the pages. It is, therefore, not clear how ROADRUNNER would scale to web page collections with a large and complex schema.

2.3.2 DeLa: Data extraction and label assignment for Web database.

Wang and Lochovsky (2003) addressed the problem of automatic data extraction from web and assigning data with meaningful label. The authors state that existing approaches based on assumption either that schema information provided from web site or the relational schema specified by the user. The authors identified the problem is that it is not guaranteed either that schema information always provided by the web site or every user have experience with the database to define the schema. Also current approaches require human effort which is not efficient in manually providing the label of the extracted data. The authors identified that wrapper generation requires the data-rich section which is relevant data for extraction need to be identified and a pattern that represents structure the data objects in data rich section need to be constructed. To solve the first problem the

authors proposed the algorithm called data-rich section extraction (DSE) which identify the data–rich section in HTML pages. To solve the second problem, the author proposed a new concept called C-repeated pattern which identify plain or repeated nested data structure in HTML pages. The authors observed that web site consist similar structure to organize their content such as the location of advertisement and navigational menus. The authors employ DSE (data-rich Section Extraction) algorithm based on this observation. The DSE algorithm compares two data-rich pages from same web site to identify data-rich section. The basic idea of this algorithm is that it removes the common sections and identifies the remaining ones as data rich section after compare two pages from the same web site. The comparison algorithm used document object model (DOM) to represent the layout of the HTML pages. A depth-first order is used to traverse the DOM trees and compare them node-by-node from the root to the leaves. If two internal nodes from two trees are similar then the algorithm goes one level down to match their children from the leftmost to rightmost one. If the leaf nodes are similar then they removed from the trees. The algorithm returns to their parent and compare the other children if the nodes are not similar. The parent node will be removed if all of their children have been removed. The Figure 26 represents an example of token sequence and its token suffix tree. Square with a number represents the leaf that indicates the starting token position if the suffix. Circle with a number represents each internal node with a number that the position of the token where its children differ. Same parent nodes are sharing by the sibling node which put in alphabetical order. The substring between two token positions of the two nodes has shown as label of each edge between two internal nodes. The token which is internal node of the suffix starting from the leaf node has shown as label of each edge between

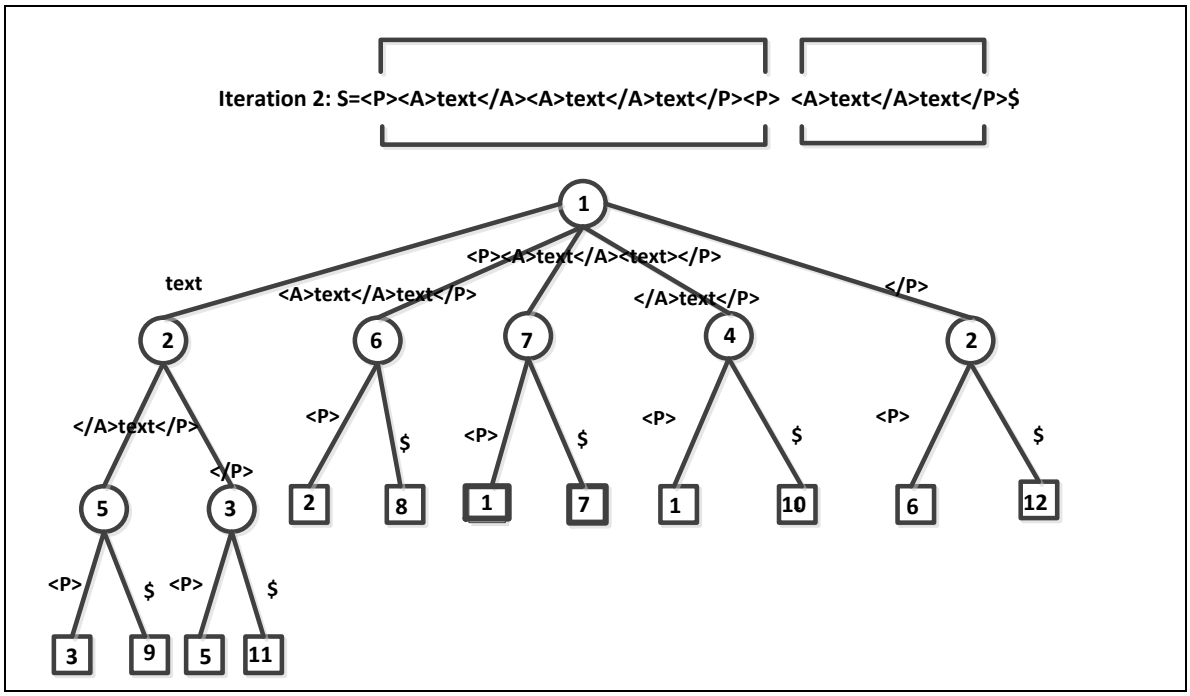


Figure 26: C-repeated pattern

one internal node and one leaf node. For example, <A>text</A> is the edge label between node **a** and **b**. It is the substring which starts from the first token up to, but not including the fourth token. The string "<A>text</A>text</P><P>" can be build by adding the edge label from the root to node which is the unique prefix that indicate the fifth suffix sting "<A>text</A>text</P><P><A>text</A>text</P>. To discover C-repeated patterns, internal node's path label and their prefixes in the token suffix tree works as a candidate. If any two of its occurrences are adjacent then it is a C-repeated pattern for each candidate repeated pattern. It occurs when the distance between the two starting positions is equal to the pattern length. The structure of the suffix tree is simple based on the occurrence retrieval if each repeated pattern. In case of repeated pattern P, it is possible to find the highest internal node in the tree. For Example, <A>text</A> is a repeated pattern in the Figure 26. Because node b is an internal node and the pattern contained by its path label which is prefix.

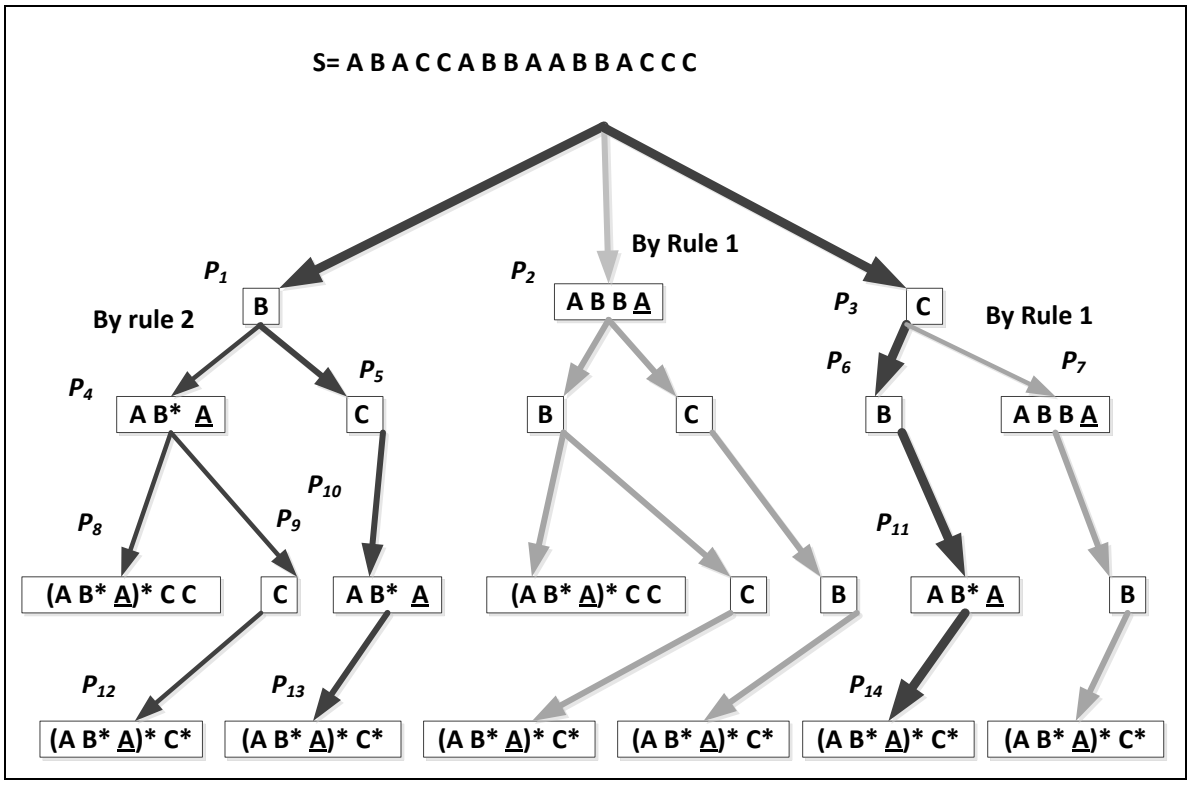


Figure 27: An example of a pattern tree

The pattern with starting position 2, 5 and 11 exists three times in the sequence which is indicated by its leaf node c, e and f. This is a C-repeated pattern since its occurrences are adjacent (5-2=3). After discovering the C-repeated pattern, the proposed algorithm discovers the nested structures from the string sequence of the HTML pages. In that case, hierarchical pattern tree is used to handle this task. For example, after discovering the pattern <A>text</A>, it masks the occurrence from S_2 to S_4 and form a new sequence in the Figure 27. A new suffix tree is build based on new sequence and search for new C-repeated pattern "<P><A>text</A></P>". At the end, a regular expression "<P>(<A>text</A>)*</P> is generated from these two iteration which represents the structure for the two data objects. Here "*" indicate object appears zero or more time. Pattern generated by the iterative discovery process form a hierarchical relationship. Because some pattern's discovery is dependent on some other pattern's discovery. For

example, discovery of pattern “<P><A>text</A></P> depend on discovery of pattern <A>text</A>. It is also possible that some discovered pattern is independent of each other. For example, in the string “text<IMG>text<IMG>text<IMG>text<IMG> text<P>text<P>”, the pattern text<IMG> is not dependent on the pattern text<P> and vise versa. Both the dependence and independence of the C-repeated pattern can be presented by using a Pattern tree. The Above Figure 27 represents a pattern tree where token of the HTML sequence is represented by each character. In the string “ABACCABBAABBACCC”, token is represented by each character and two data objects in this string is (AB*A)*C covered by the structure. The character “*” means the substring may appear zero or more times. To reduce the complexity, the authors employ heuristics to filter out patterns that cross pairs of HTML tags. And to prune some branches of the pattern tree they used three rules. The advantage of this approach is that it can automatically generates regular expression wrapper to extract data objects and able to restore the retrieve data into a table. The proposed approach is able to assign meaningful label to the data attributes and can extract nested data from HTML pages. The limitation of this approach is that since DELA able to generalize optional and alternative pattern only after extracting all candidate patterns, it might miss optional and alternative structure which are nested inside a repetitive structure. Zhai and Liu (2005) describes about the limitation of DELA is that it needs to use multiple pages (which are assumed to be given) that contain similar data records from the same site to find patterns or grammars from the pages to extract data records. Assuming the availability of multiple pages containing similar data records is a serious limitation. Miao et al. (2009) identifies one limitation of DELA is that it is not robust against optional data inserted into records.

CHAPTER 3 - Web content mining using non-deterministic finite state automata

As discussed in section 2.2.5, WebOMiner (Mutsuddy and Ezeife, 2010), Ezeife and Mutsuddy, 2013) used non deterministic finite state automata based algorithm for extraction and mining structured web content data. We studied the approach of WebOMiner and found that this method uses an NFA algorithm for identifying tuples of web objects, the mechanism for using NFA algorithm for automatic identification of different content types is not fully integrated in the current system. Also, they generate NFA for identifying content based on manual observation of product schema from ten B2C web sites which is not efficient and use manually generated database schema to store extracted contents into database. We propose an architecture that generates finite state automata automatically for content mining and it also generates data warehouse schema automatically. This thesis develops the architecture for web content mining. It extends and modifies necessary algorithm from WebOMiner. This thesis addresses the following limitations of WebOMiner.

3.1 Problem Addressed

1. WebOMiner used non deterministic finite automate (NFA) for mining different types of web content. But the mechanism for using NFA algorithm for automatic identification of different content types is not fully integrated in the current system. For example, they manually discovered ten representations database structure of product list page from different B2C web site. They built product NFA manually based on this structure. This thesis solves the limitation of

WebOMiner by generating NFA from the frequent pattern extracted from DOM tree of the web page.

2. WebOMiner used NFA for content mining without converting it to DFA. Because in NFA, the states are not fixed and state can be visited more than one state at a time. But in DFA, the states are fixed and state can be visited one state at a time. So, NFA needs to be converted into DFA for better efficiency. This thesis solves this shortcoming by converting the NFA to DFA by handling ``ε`` transition of NFA.

3. The process of database schema generation in WebOMiner is manual and it is not efficient. The schema of the web can be extract from web page and it can be used to generate database schema automatically. This thesis solves this limitation of WebOMiner by generating the database schema using extracted pattern of different content from DOM tree of the web page.

4. WebOMiner extended ContentWebObjectScan algorithm (Annoni and Ezeife, 2009) to extract product information (e.g. image, brand, text, product number, price). The information is extracted based on the value of "id" attribute of the HTML tag (Ex. div, tr etc.). For example, they assumed that price content will be tagged with the HTML tag (Ex. div, tr etc.) and this tag will contain attribute "id" with the value "price" (ex. id = "price"). This process is not generic and for that reason WebOMiner failed to extract content information (e.g., price, title etc) from every B2C web site since the value of the attribute are not same for every B2C web site.

We discuss about our approach to solve the above limitations in the following sub-section of this chapter as: in section 3.2, we have discussed about different web content objects. In section 3.3, we have discussed about the challenges to solve the above limitations. In section 3.4, we have discussed the thesis problem domain and approach to solution, section 3.5, we have presented the mining technique. In section 3.6, we have presented our proposed architecture and algorithm.

3.2 Web content objects

WebOMiner (Mutsuddy and Ezeife, 2010; Ezeife and Mutsuddy, 2013) uses four content types to extract from web page include text content, image content, form content and plug-in content. Each content type are described below:

3.2.1 Image content

Image contents are embedded into the web documents with the <image> or <map> tag. It contains simple picture which refer to a physical image document of any physical location. For example, the HTML tag "" means that the image "car.jpg" is embedded into the HTML document and its physical location is "photo" folder. Some web page embeds image with the <map> tag to define the mapping of the image. For example, client side mapping uses <map> tag where server side mapping uses <ismap> tag.

3.2.2 Text content

Text content resides in the leaf level of the DOM tree of the HTML document. There are two types of text content include raw text and list text. The raw text exists in the web page with or without alignment where the list text exists in order or unorder form.

3.2.3 Form content

Form content are used to get information from user such as user selection, user's feedback, orders through internet etc. Form contents are embedded into web page using <form> tag and different input formats are used to gather the information. For example, the tag <textarea> is used to get any command or textual information from user, the tag <select> is used for user's selection from a list of choice and the tag <input> is used for one or multiple choice from user.

3.2.4 Plug-in content

The plug-in contents are generated dynamically by server side database or automated calculation by the function or programs. Two types of program generates the plug-in in the web page include client side program and server side program. The client side programs are vulnerable because the controlling computer functions or programs are embedded into the web page. Server-side program generates the dynamic contents because it interacts with another program at server. The visual basics, PHP codes and HTML embedded CGI are the example of plug-in contents.

3.3 Challenges to solve the problem

Web content mining from different domain (e.g., B2C, Research, library etc) is an important problem in web data mining. A web page from specific domain contains different types web contents like product, list, text etc. Given a web page, the problem is to identify the different type web content and store them into the database. Since web

page from different domain contain different types of information based on their attribute, so their schema should be different. So it is another problem to generate the database schema base on the schema of web content objects. For example, web page from "bestbuy.com" contains product information like computer, tv etc. and it contains the product attribute object like title, image, price, model no. etc. And a page from "CompUSA.com" contains information about a product with the attribute like title, image, brand and price. We observe that, two different B2C web sites have two different schemas. So, we need a general schema that can be used to store data from these web sites. Also, "acm.com" is a web site from research domain. If we want to extract information about a journal paper from "acm.com", we have to handle with different schema of web content. Because a journal paper has the schema like title, author(s) name, year of publication, abstract, etc. The database requires different schema to save this web content. Though both the web sites has common web contents objects like List, Text, noise etc.

The above problem clearly identifies the need for finite state automata that identifies different types of objects from given web page and generates database schema automatically. We propose a framework called FA (finite automata) generator to solve the above problem. It has five modules include pattern extractor module, RE (regular expression) generator module, NFA generator module, DFA generator module and Schema generator module. The proposed framework generates DFA for WebOMiner to identify tuple and generate database schema. The architecture of our proposed framework is shown in Figure 28.

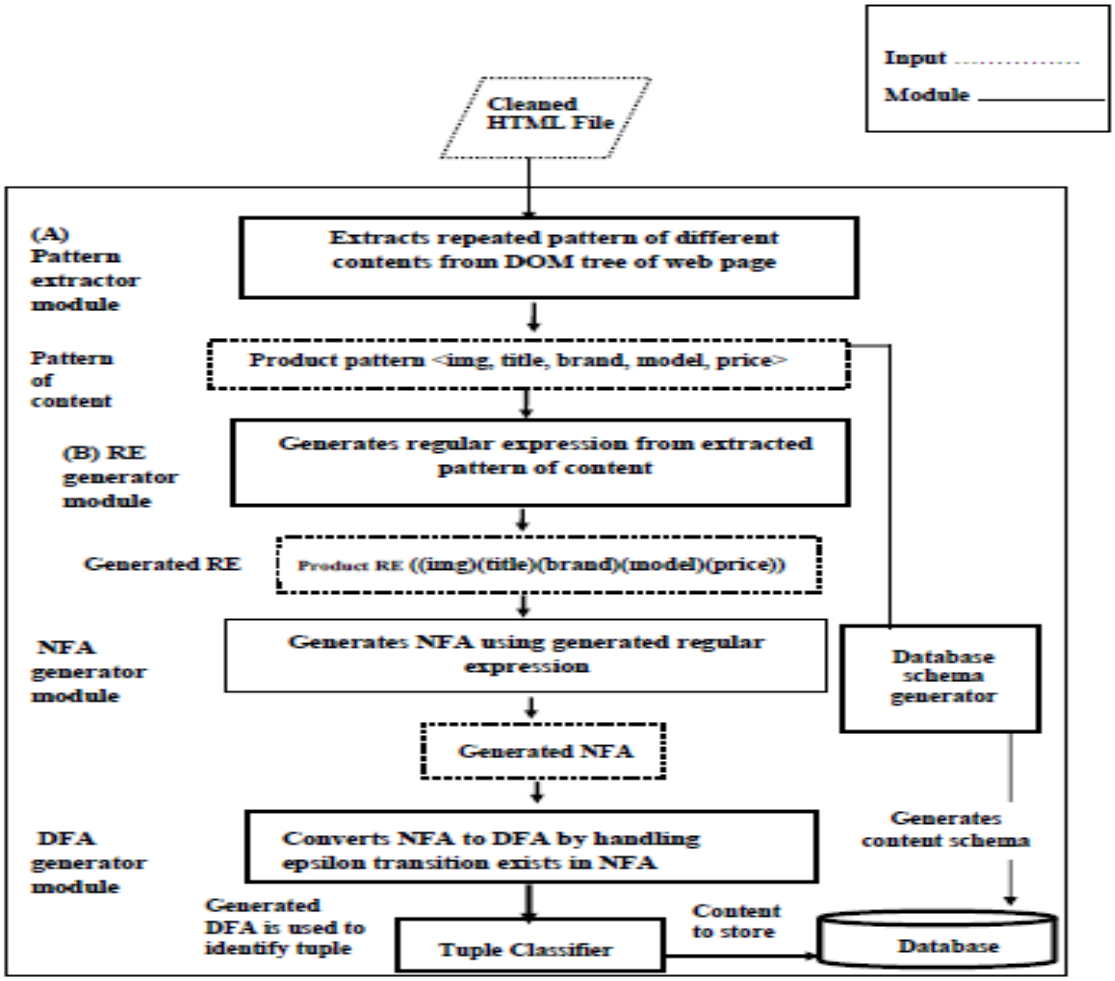


Figure 28: Architecture of FA generator

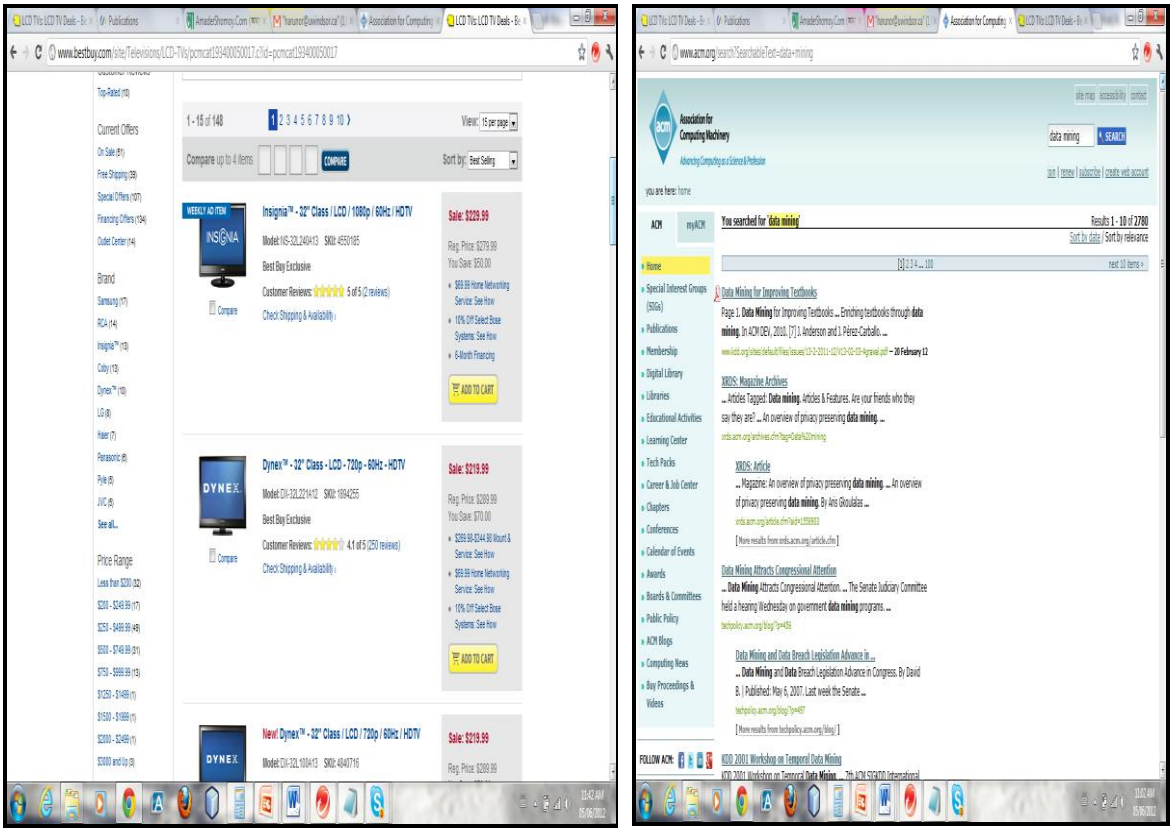
3.4 Problem domain

In this thesis, we have tried to extract content from list page (discussed in section1.1.3) of domain specific web sites. The list page from domain specific web site (e.g., B2C, library) is considered a data rich page. Our observation about the list page is that a list page contains brief list of all or specific types of product (e.g., tv or computer in B2C web site, book in library web site, journal or conference in ACM/IEEE). Since different web site contains different product schema, so our target is to generate non-determinitic finite automata based on the different content type (e.g., list, text, product) to identify

different tuple from content object array that contains extracted content from a list page. Also generate data warehouse schema based on the extracted unified pattern.

3.4.1 Extract pattern of content

In this thesis, we consider to extract web contents from list web page of different domain (B2C, Library, Research etc.). The List web page contains list of information about product, journal, book etc. For example, Figure 29(a) represents a list page from B2C domain specific web site "Bestbuy.com" and Figure 29(b) represents a list page from research domain specific website "acm.org".



(a) (b)

Figure 29: (a) List web page from "bestbuy.com" and (b) List web page from "acm.com"

We observed that, the list page of every domain specific web site is generated using their own template. Also list page contains data objects (information about a product or a journal, list, text etc.) that follows similar format. The structures of data objects appear repeatedly if the list page contains data object instance that is more than one. For example, Figure 29 illustrates two list pages from “Bestbuy.com” and “acm.org”. Both of the pages contain multiple list objects, product objects, text objects, form objects etc. These objects appear with their own format. For example, list object appears with their attributes such as link, text and product object has attributes like image, title, model, text, price etc. Repeated contents in semi structured documents are usually prominent and easily raise a reader’s attention. Most Web data-extraction systems (Arasu et al. 2003; Chang et al. 2001; Crescenzi et al. 2001) assume that repeated contents are important and should be extracted. For example, in Figure 29(a), all monitors have similar formats and most parts of their HTML codes are repeated, like those in the Figure 30.

Figure 30: Product pattern encoded by HTML tag

Our target is to extract the frequent pattern of the content that exists in the web page. For example, the product content occurs frequently with their attributes (e.g., image, title, model, price etc.) in the web page. To generate a data warehouse with the content, it is required to extract different pattern of content from web page.

3.4.2 Regular expression (RE) generation

In this thesis, our goal is to generate non-deterministic finite automata to identify different content types from a content list to store content into the data warehouse. Also we need a unified data warehouse schema to store content with different structure into data warehouse. The NFA can be generated from regular expression. In this thesis, we try to generate regular expression from extracted frequent pattern from web pages. For example, from the above frequent pattern of the “bestbuy.com” web page (Figure 30), the following regular expression need to be generated (Figure 31):

(<titile><image>*<brand><text>*<num><price>)

Figure 31: RE generated from extracted pattern

Since different B2C web sites consist of different patterns for the presentation, so the pattern does not match between them. For example, the RE generated from “futureshop.com” is given below (Figure 32):

(<image><title>*<num><brand><text>*<price>)

Figure 32: RE generated from "futureshop.ca"

If the existing RE and the current RE differs, then it is required to merge them and generates a unified RE. For example, a unified RE is given below (Figure 33):

((<image>|<Title>)*(<num>|<brand>)*<text>*<price>)

Figure 33: Unified RE

But if the current pattern is from different domain then the module only generates the RE for that domain. For example, RE generated from “ACM” web site is given below:

(<Title><Author><published> <year><reference>)

Figure 34: RE generated from "acm.com"

3.4.3 NFA generation

In this thesis, we use the non-deterministic finite automata to identify different tuple from tuple list. We generate NFA using the generated regular expression from frequent pattern extracted from different web site. In this process an NFA is constructed first from a regular expression. To construct an NFA from regular expression (RE), we use Thompson’s construction algorithm. This method constructs a NFA from components of regular expression and using ε-transitions. The ε transitions act as “glue or mortar” for the subcomponent NFA’s. An ε-transition adds nothing since concatenation with the empty string leaves a regular expression unchanged. A Nondeterministic Finite Automata (NFA) has a transition diagram with possibly more than one edge for a symbol that has a start state and an accepting state. The NFA has an accepting state for the symbol. For example, Figure 35 represents the generated NFA from the unified RE (((<image>|<Title>)*(<num>|<brand>)*<text>*<price>)).

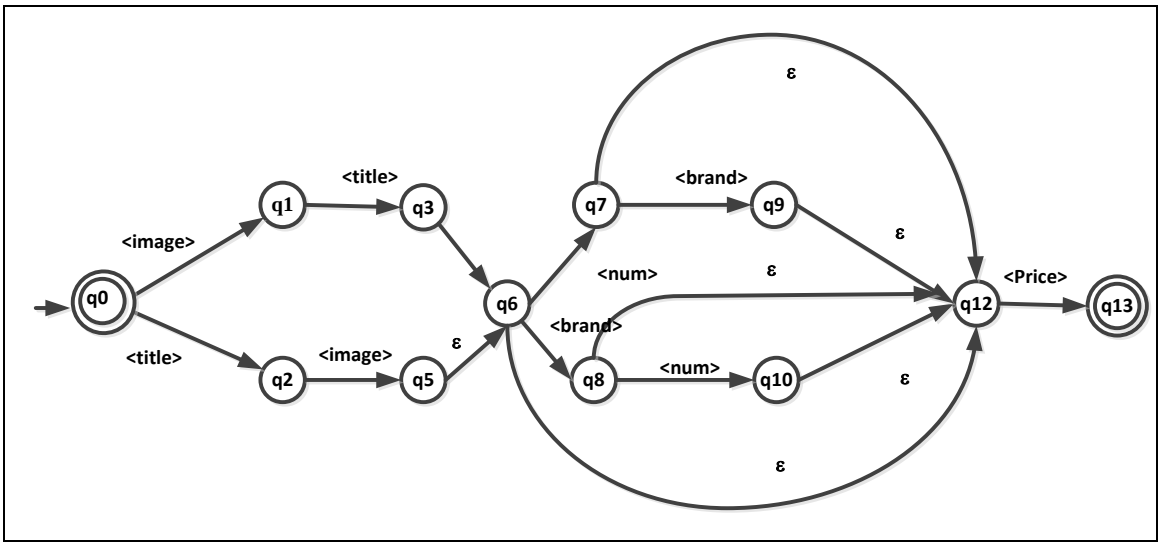


Figure 35: Generated RE is converted to NFA

3.4.4 DFA generation

In this thesis, we also use DFA to identify tuple from tuple list. DFA works more efficiently for tuple identification DFA is generated from NFA by removing ε- transition from NFA and DFA doesn't have any repeated labels on outgoing edges.

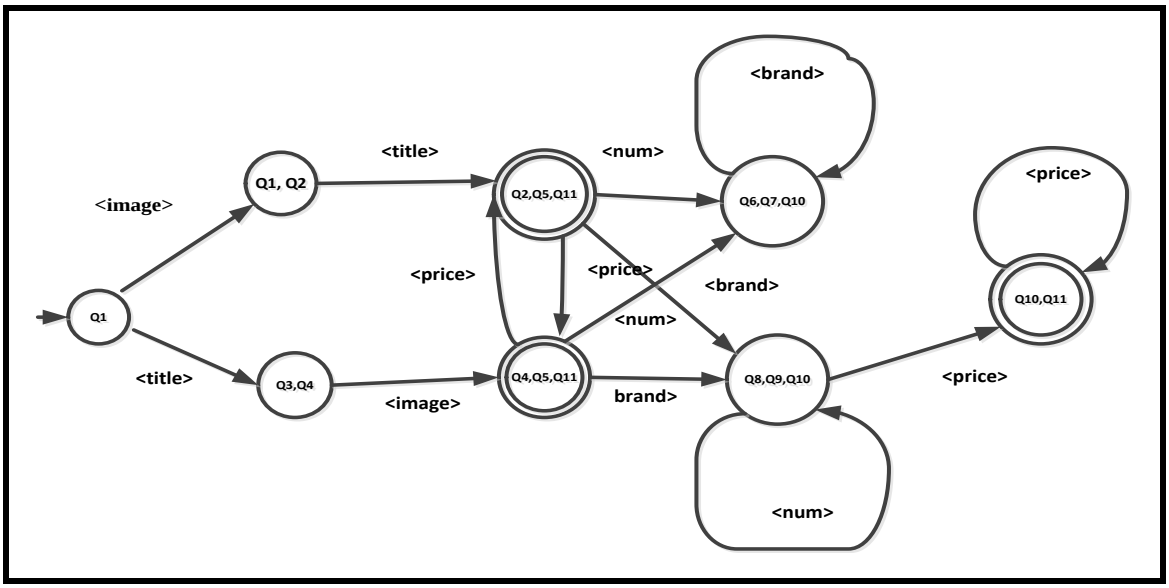


Figure 36: Generated NFA is converted to DFA

We need a finite state machine that is a deterministic finite automaton (DFA) so that each state has one unique edge for an input alphabet element. So that for tuple identification there is no ambiguity.

3.5 Proposed "WebOMiner-2" Architecture and Algorithm

In this thesis, we modified the architecture called WebOMiner proposed by Mutsuddy and Ezeife (2010), Ezeife and Mutsuddy (2013). We named it "WebOminer-2" which is shown in Figure 37.

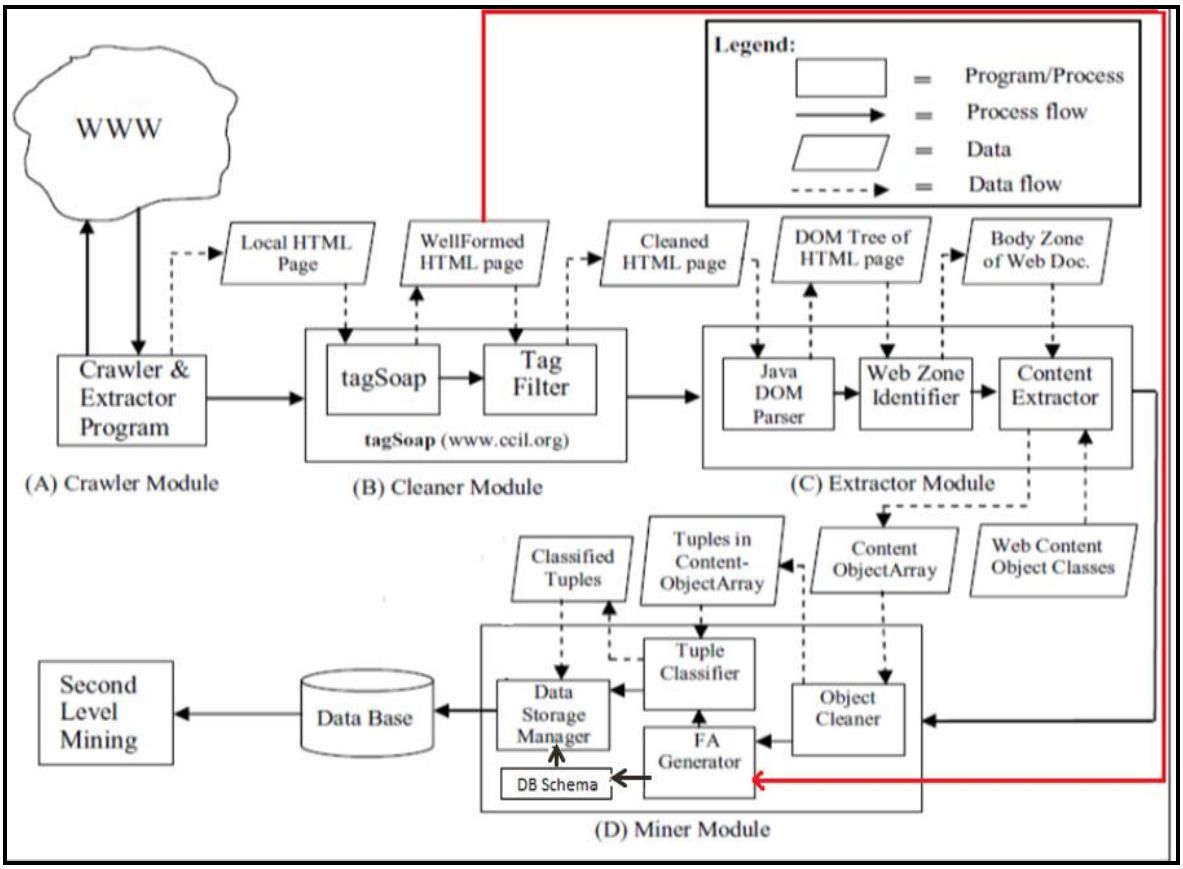


Figure 37: Architecture of WebOMiner-2

We modified the miner module of WebOMiner (Figure 11, page 24). The NFA generator (Figure 11) of miner module generates non deterministic finite automata to identify tuple

(product, list, text) from the tuple list. And NFA is generated based on ten representation of product structure which were discovered manually. Also database schema of product, list, text are created manually in WebOMiner.

Algorithm Main
Input: Set of HTML files (WDHTMLFile) of web documents.
Output: Extracted content from web page is stored into database
Variable: ContentObjectArray[].

Begin
For each WDHTMLFile
A. **Call SiteMapGenerator(url)** to crawl the url and download the webpage into local directory from the url.

B. **Call HTMLCLEANER(downloaded HTML file)** to clean-up HTML code to make it well formed

C. **Call WebOMiner.BuildDOMTree(WDHTML file)** to create DOM tree of refined HTML file and extract web content objects sequentially from DOM Tree. Store objects in ContentObjectArray[].
D. **Call MineContentObject.IdentifyTuple()** use NFA/DFA to identify data records and classify records according to their pattern
E. **Call CreateDBTable()** to store data records into a database table
endFor

F. Mine for knowledge discovery within extracted contents. /* *pending to develop* */
End

Figure 38: Main algorithm of WebOMiner-2

WebOMiner-2 contains four modules include crawler module, cleaner module, content extractor module and miner module. The crawler module crawls the URL given as input to find targeted web page. This module creates a mirror of original web document after streaming the entire web document including tags, texts and image contents. The comments are removed from the HTML document by this module. The cleaner module converts the downloaded HTML file well formed by inserting the missing tag, removing inline tag (e.g.,
, <ht/>), insert missing "/" at the end of unclosed <image> tag, clean up unnecessary decorative tags. The content extractor module converts HTML page

into DOM tree and extracts contents from the DOM tree. This module indentifies respective class object type as per pre-defined object class to the content. Also puts objects into Array List after setting information into objects. Data regions and data block are identified by this module and it segment the respective data of a data block from other data block by using separator objects. This module also generates finite state automata from extracted pattern exists in the DOM tree of the web page. First, the extracted pattern is converted to regular expression using RE generator module. Then the generated RE is converted to NFA using NFA generator module. After that the generated NFA is converted to DFA by handling the "ε" (epsilon) transition. Also this module generates data warehouse schema using generated regular expression. We actually modified the miner module of WebOMiner which generates NFA from the frequent pattern of different objects (product, list, text etc.) which is extracted from the DOM tree of HTML page. We named our framework as "FA Generator". The miner module of WebOMiner-2 calls the "FA generator" to generate NFA and data warehouse schema.

Our proposed architecture has five modules include PatternExtractor module, Regenerator module, NFAgenerator module, DFAgenerator module and Database schema generator. The pattern extractor module takes a set of HTML page as input. It iteratively processes one page at a time and finds the frequent pattern of different content exits in each web page. It produces the frequent pattern of different content as input of the RE generator module. The RE generator module takes the frequent pattern of different content as input and generate the regular expression after analyzing it. The RE generator module produces the regular expression as input for NFA generator module. The NFA generator module converts the generated regular expression using the

Thomson's construction algorithm. The generated NFA is converted to DFA by the DFA generator module. The DFA generator module handles the "ε" (epsilon) transition using subset construction algorithm. The schema generator module generates the database schema automatically using the content attributes from the generated regular expression. These modules are called sequentially.

```
Algorithm: FA Generator

        Input: Set of cleaned HTML files of Web documents
        Output: Generate DFA to identify tuple and create database schema
Begin
        1. Call PatternExtractor(cleaned HTML file) to extract frequent pattern from
        webpage
        2. Call REGenarator(Extracted pattern) to generate regular expression from
        extracted pattern
        3. Call NFAGenerator(generated RE) to generate NFA from regular
        expression
        5. Call DFA Generator(generated NFA) to generate DFA from NFA
        6. Call SchemaGenerator(RE) to generate schema into database.

End
```

Figure 39: Main algorithm of FA generator

In the following section, we will explain each module and will discuss how our algorithm works.

3.5.1 PatternExtractor Module

The purpose of this module is to read the cleaned (well formed) HTML file (or files) of different domain and extract pattern of different objects like (list, product, text etc.). At first, the tags (div,table, tr, td, ul, li) are parsed from HTML file and saved into a file. We considered these tags because these tags are used to embed the content in the web page and these tags define a division or a section in HTML document. After that each tag

occurrence are computed to find the frequent pattern of the different contents. The idea behind this, if any content exists more than once, then its structure is repeated. That means the tag which creates the structure of the content will be repeated.

```
Algorithm: PatternExtractor
Input: Cleaned HTML file
Output: return frequent pattern of different objects (product, list, text etc)
Begin
        1.Call parseHTML(HTML file) - Parse the HTML file to find the div/table/tr/td .
        2. Call PatternExtractor("temp.xml", "occurrence.data") – count every tag occurrence
        from "temp.xml" and save it to "occurrence.data" file
End
```

Figure 40: Algorithm PatternExtractor

PatternExtractor module contains two main methods called parseHTML() and PatternExtractor(). The parseHTML() method parse all the div/table/tr/td tag from the HTML file. For our running example HTML page, parse() method parse "div" tag and save it to "temp.xml" file. In Line 1 of ParseHTML() algorithm creates FileReader object "fstream" by passing cleaned HTML file as parameter. The FileWriter object "fwstream" is created in Line 2 by passing "temp.xml" as parameter.

```
Algorithm : parseHtml(Cleand HTMLfile)

Input: Cleaned HTML file
Variable: String lineread
            FileWriter fwstream
            FileReader fsstream

Output: return a XML file that contains div/td/tr/table tag with class attributes

Begin
        1.  Create FileReader object "fstream" by passing cleaned HTML file as parameter
        2.  Create FileWriter object "fwstream" by passing "temp.xml" file as parameter
        3.  Create BufferReader object  "in" by passing FileReader object as parameter
        4.  Create  BufferWriter object "out" by passing FileWritreObject as parameter

        5   Do
        6       Read line from file
        7       If ( Line read contains div/table/tr/td tag)
        8               write  this line into file "temp.xml"
        9       While(end of file)
        10      return "temp.xml"
End
```

Figure 41: Algorithm parseHTML

Line 3 and 4 creates BufferReader and BufferWriter objects “in” and “out” respectively. From line 5 to 9, there is a loop that reads each line from “Cleaned HTML file” and checks that if this line contains any div/table/tr/td. If the condition satisfies, then the read “line” is written to “temp.xml” file. Line 9 return “temp.xml” file to PatternExtractor algorithm. For our running example HTML page, ParseHTML() reads “HTML” as a first line of the file. This line doesn’t satisfy the conditional statement. So, the loop reads the next line which is “body” tag. It also doesn’t match with the conditional statement. When the loop reads the third line of the HTML file which contains “div” tag, and this line is written to “temp.xml” because this line satisfied with the conditional statement. Thus the loop reads every line until end of the HTML source file and create “temp.xml” file with the tag div/table/tr/td. Figure 42 displays the snapshot of “temp.xml” file.

```
<divitemscope="itemscope"itemtype="http://schema.org/SearchResultsPage"class="ABLH">
<div>
<div>
<div><!----></div>
<divclass="admon"data-arrowdirection="up"data-offsettop="35"data-offsetleft="100"><!----
></div>
<divclass="hdr-wrap">
<divclass="hdr">
<div>
<div><ahref="http://www.bestbuy.com/site/olspage.jsp?id=pcat17005&amp;type=page&amp;pag
eId=pcmcat193400050017&amp;pageType=category&amp;_DARGS=/site/en_US/global/nav/ols
minicart.jsp_A&amp;_DAV="><spanclass="cart-icon">Cart</span><spanclass="cart-
items"><strong>0Items</strong></span></a></div>
<div>
…….

<divclass="clearer"/>
<div/>
<divstyle="background:#fff;"class="b52">
<div>
```

Figure 42: Snapshot of “temp.xml” file

After writing the div/table/tr/td tag from cleaned HTML file to “temp.xml” file, the method tagOccurenc() of pattern extractor module find the every tag occurrence and store

this information into “occurenceCount.data” file. In this step, it reads each line from “temp.xml” file and passes it to “occurenceCount()” function. OccurrenceCount() takes each line as a parameter and return its occurrence. For example, “ line read from “temp.xml” file and passes it to occurenceCount() as a parameter. The “temp.xml” file also passes reference of “occurrence.data” to occurenceCount() method. OccurrenceCount() method accept these parameter and scans the occurrence of “ in “temp.xml” and finds its occurrences in “temp.xml” and return it. Line 1 and 2 of this algorithm creates FileReader and FileWriter object by passing “temp.xml” and “Occurrence.data” respectively as parameter. Line 4 and 5 creates BufferReader and BufferWriter object respectively. The object of arraylist is created in line 3. An integer type variable “count” is declared in line 6. There is a loop from line 7 to line 13 which read each line from “temp.xml” file and passes it to occurenceCount() with the file “temp.xml” file to find the occurrence each tag line. When the method occurenceCount() method return the occurrence of the line, then the occurred line and its count saved to “occurrence.data” file.

```
Algorithm: tagOccurence()
Input: Line read from “temp.xml” file and “temp.xml” file
Output: Return the occurrence of each line of “teml.xml” file.
Begin
        1. Create FileReader object by passing “temp.xml” as parameter
        2. Create FileWriter object by passing “occurrence.data” file as parameter
        3. Create a ArrayList to store each line from “temp.xml”
        4. Create BufferReader object “in”
        5. Create BufferWriter object “out”
        6. Declare  a integer type variable count to store the occurrence of each line
        7.Do
        8.      Reach each line from “temp.xml”
        9.      Pass this line to occurenceCount() as parameter. OccurrenceCount() return count of
                line
        10      If arraylist doesn’t contains this line
        11.             add this line to arraylist
        12.             write this line and its occurrence count into “occurrence.data”
        13. while(end of “temp.xml”)
End
```

Figure 43: Algorithm tagOccurence()

For our running example, tagOccurence() method reads the “temp.xml”(Figure 42) . When it reads the first line from “temp.xml” file which is “<divitemscope="itemscope" itemtype= "http://schema.org/SearchResultsPage"class="ABLH">” and passes it to occurenceCount() method. The occurrenceCount() method checks it’s occurrence in “temp.xml” file and if it’s occurrence is not more than one, then this line is ignored to save in “occurrence.data” file. After that it reads second line from “temp.xml” file which is “ ”. The occurenceCount() finds its occurrence 23 time. Since it’s occurrence is more than one, so this line and its occurrence is saved to “occurrence.data” file. And this way tagOccurrence() finds each line occurrence of “temp.xml” file until end of file and save this information to “occurrence.data” file. Figure 44 displays a snapshot of “occurrence.data” file.

```
<div> 23
<divclass="clearer"/> 23
<div/> 4
<divclass="hproduct"itemscope="itemscope"itemtype="http://schema.org/Product"> 15
<divclass="image-col"> 15
<divclass="compareButton"> 15
<divclass="info-side"> 15
<divitemprop="offers"itemscope="itemscope"itemtype="http://schema.org/Offer"> 14
<divclass="puck"></div> 15
<divclass="info-main"> 15
<divclass="attributes"> 15
<divclass="description"itemprop="description"><b>BestBuyExclusive</b></div> 9
<divclass="rating"> 15
<divclass="availHolder"> 15
<divclass="tooltip-wrapper"data-tooltip-pos="right"data-tooltip-xpos="305"data-tooltip-ypos="-8"> 15
<divclass="tooltip-header">ShippingandAvailability</div> 15
<divclass="tooltip-contents"> 16
<divclass="clearer"></div> 15
<divclass="hr"><hr/></div> 14
<divclass="ftr-sec"> 7
```

Figure 44: Snapshot of “occurrence.data”

The occurenceCount (Readline,”temp.xml) method accept string type variable “readline” which contains line from “temp.xml” file and reference of “temp.xml” as parameter. Line

1 creates scanner object. A arraylist object is created in Line 2. A matcher object is created in Line 3. Line 4 creates a pattern object. Line 6 to 14 is a loop that checks each line occurrence in “temp.xml” file. Line 15 return the count number to tagOccurence() which is line occurrence. For our running example, while the second line “ ” of “temp.xml” pass to occurenceCount() with “temp.xml” file. The occurenceCount() finds that there are 23 occurrences of “ ” in “temp.xml”. And it returns counter 23 to the method tagOccurence. Thus OccurenceCont() finds each line occurrence and return it.

```
Algorithm: OccurenceCount()

        Input: Line read from “temp.xml” and reference of “temp.xml”
        Output: return the counter which contain each line occurrence in “temp.xml”

Begin
        1.      Create an object of Scanner class
        2.      Create an object of arrayList class
        3.      Create an object of pattern class
        4.      Create an object of Matcher class
        5.      Do
        6.              read the line from “temp.xml”
        7.              create matcher object by passing reading line as parameter
        8.              check find() method of matcher object is satisfied
        9.              increment count
        10.     While(end of file)

        11. return count

End
```

Figure 45 : Algorithm OccurenceCount()

3.5.2 Regular Expression Generator

The next module is regular expression generator module. The input of this module is the “occurrenceCount.data” file and the cleaned HTML file. At first, generateRE() method converts the cleaned HTML file into a DOM tree by passing the file into parse() method of DocumentBuilder object. It also creates “Xpath” object instance. The “Xpath” object is used to extract the specified node from the DOM tree of source HTML file.

GenerateRE() method reads line from “occurenceCount.data” file and checks that this line contains any div/table/td/tr/li/ul tag with class attributes. If it finds any, then this line pass to compile() method “Xpath” object to extract all the nodes from DOM which are matched with the class attributes. For example, First line of “occurrenceCount.data” file is “ 23”, since this div tag doesn’t contain any class attribute, so it is ignored to pass. The second line is “ 23”. Since this line contains class attribute, so it passes to compile method. The compile method() returns nodes which is saved to object instance of NodeList. The length of nodelist is counted using method length() and save to a variable called “numofNode”. The node length of next line also counted and compare with previous node length. If both the length are matched, then first element of current node is picked to traverse. While traversing each tag are checked to find the object attributes like (image, model, title, price etc.). For our running example, while the 3rd line of “occurrenceCount.data” is read and passes it to compile() method. The Comiple() method returns its nodelist which length is 15. And while the 4th line is read and its nodelist length found as 15. Then both the lengths found as matched. After that, the first element from nodelist of current line read (4th line) is traversed to find the existing attributes. Here we consider tocompare length of nodelist between two line which has similar occurenec count in “occurrenceCount.data” file. Because if the product block is repeated in the HTML page, then its tags are repeated same number of times. Line 1 of generateRE() method creates the DOM tree of HTML file. Line 2 of this algorithm creates Xpath object instance. There is a loop between lines 3 to line 26 which reads every line from “occurrenceCount.data” file. When loop traverses each node of DOM tree, it also traverses the child of the nodes. For our running example, at first

iteration it scans first line of “occurrenceCount.data” which is “ 23”. It means that “ ” tag is repeated 23 times in the HTML file. The conditional statement checks that does this line contain any class attributes. So, the loop iterates to next line from “occurrenceCount.data” which is “<divclass=”clearer”/>”. Since this line contains a class attributes so this line is processed by the conditional statement block. First, it extracts the value of class attributes and it creates a “path” string object with the value of class (path=”//div[@class=”clearer”]). This “path” variable passes to compile method “xpath” object “expr”. The evaluate() method of “XpathExpression” extract all the nodes from the DOM tree which is matched with “class=clearer” embedded with “div” tag and save this result set to NodeList object. The length of the nodelist is extracted using getLength() and this value is saved to variable “numberofnodes”. There is a integer variable called “previousnumberofNodes” keeps record of node last visited and compared with “numberofnodes”. Initially the value of “previousnumberofnode” set to “0”. During first iteration, the value of number of nodes is 23 and “previousnumberofnodes” is 0. Since value of these two variables doesn’t match, so conditional block is not processed. The loop iterator reads next line which is “ 4”. This line is ignored to process because it doesn’t contain any “class” attributes. The loop iterator reads the fourth line of “occurrence.data” file which is “<divclass=’hpproduct’itemoscope=’itemscope’ itemtype=’http://schema.org/offer”>15”. This time “path” variable set as “//div[@class= hpproduct]” where “hpproduct” is the value of class attribute of the current line read from “occurrence.data”. And this time compile() extracts all the nodes from DOM tree which are matched with the variable “path”. The length of Nodes is computed and saved it to “mumberofnodes”(15). The previous node length saved to “previousnumber-

ofnodes”(38). Since the length of current node and previous node doesn’t match, conditional statement will not process. Then loop iterator reads next line from “occurrenceCount.data” which is “<divclass=img-col>”. The length of this node is 15 which is matched with previous node. Now the current node is traversed in the conditional statement block. The node is traversed through it’s child nodes. For our running example, current node contains two child nodes including <a> and . The <a> tag has one child node including <img> tag. And the tag has three child nodes including “<script>”, <input> and <a> tag. Successive iteration scans regular expression of product block (image, price, title, Model, SKU). We set some criterion for support in for identifying attributes of product which are given below:

Image: We consider that product block contains “image” attribute if there exists an “<img> tag in product block. For our running example, as shown in Figure 46, the node has child nodes including “<a>” and “ ”. The first child node “<a>” has child node called <img>. Algorithm REGenerator() generates regular expression as (image) or (image*) [if more than one image found in the block].

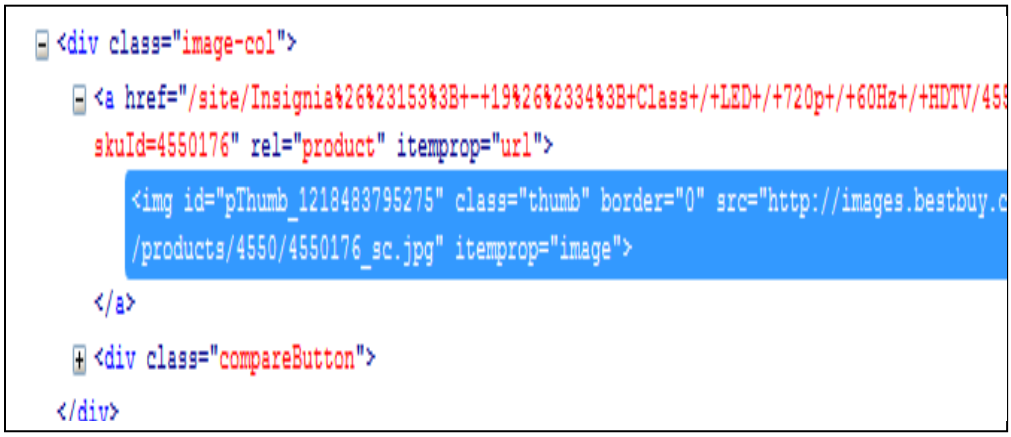


Figure 46 : “<img>” tag in product block.

Title: Every product block contains header and it is embedded with a "<a>" tag. The "<a>" tag contains some attributes and "href" is one of them. The header contains brand name of the product. If the "<a>" tag has "href" attributes and the value of "href" contains the brand name that exists in the header, we consider it a product title. For our running example, as shown in Figure 47 is shown the source code of the product block. We noticed that node has child nodes and <h3> is one of them. The <h3> node contains child node <a> which has child as text "Insignia-32" Class-LCD-720-60Hz-HDTV". This text is shown as header in the product block.

Figure 47: Source code of product header

The product header contains "Insignia" as product brand. And "href" is the attribute of <a> tag. Here the value of "href" contains "Insignia" and we consider it as "title" of the product.

Price: We consider that the product block contains product attribute if there exists a node which has attribute class with the value "price". For our running example, Figure 48 is shown the source code of the product block. The node has child nodes including , <h4>, , <ul>, and <img>. The first child is tag and it has child nodes <link>, and <h4>. The last child of is <h4> which has

class attributes and its value contains string “price”. While traversing DOM tree, if our algorithm finds a node with attribute value is “price”, it will be consider as “price” attribute of the product and generates RE with “price”.

Figure 48: “price” information in product block source code

Brand: We consider that product block contains “Model” attribute, if there exists a node with attribute value “model”. For our running example, as shown in figure 49, the node “< div class = attributes>” has child nodes <h5> . The node <h5> has child node which contains attribute “itemprop” with the value “brand”. While traversing the DOM tree, REGenerator() algorithm consider it as “brand attribute of the product and generates RE with “model”.

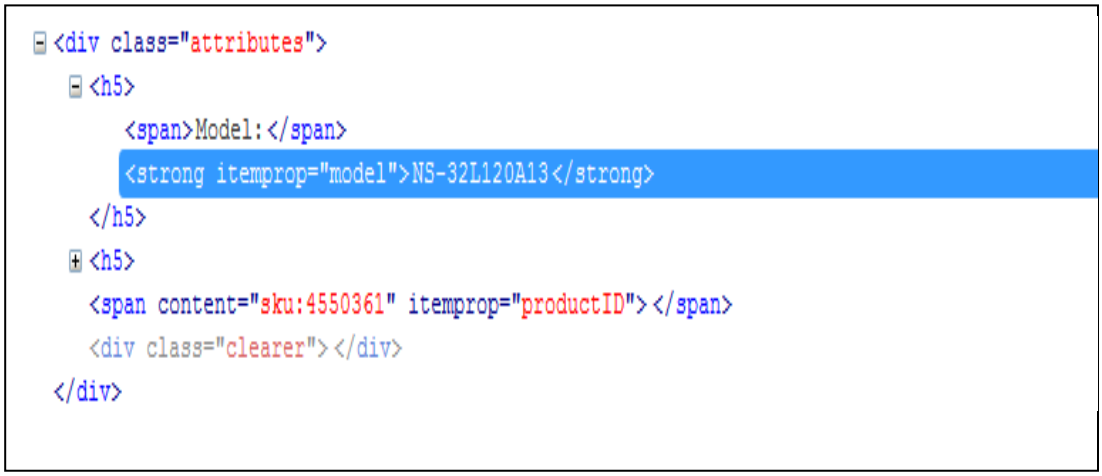


Figure 49: “brand” attribute in product block source code

ProdNumber: We consider that product block contains "prodNum" attributes If there exists a node with attribute which has value as string "SKU". For our running example, as shown in figure 50, the node node "< div class = attributes>" has child nodes <h5> . The node <h5> has child node which contains attribute "class" with the value "SKU". While traversing the DOM tree, REGenerator() algorithm consider it as "ProdNum" attribute of the product and generates RE with "ProdNum".

Figure 50: "ProdNum" attribute in Product block source code

By traversing the DOM tree, The algorithm reGenerator() generates regular expression from different B2C web site which are given below:

B2C web site	Generated RE
Bestbuy.com	(title,image,prodNum,brand,price)
FutureShop.ca	(title,image,prodNum," ", price)
CompuUsa.com	(title,image,brand,prodNum,price)
Walmart.ca	(title,image,brand," ",price)
Walmart.com	(title,image," "," ", price)
Target.com	(image,title,prodNum,brand,price
Sears.com	(image,title,prodNum," ",price)
Tigerdirect.com	(image,title,brand,prodNum,price)
Thesource.ca	(Image,title,brand," ",price)
Ebay.com	(image,title," "," ",price)

Table 1: Generated regular expression from different B2C Website.

After generating regular expression from different B2C web site, algorithm generateRE() unifies the regular expressions to generate non-deterministic finite automata from it. From the above regular expressions, generateRE() unified the following RE:

((img|Title)(title|image)(brand|prodNum)(prodNum|Brand)(price))

```
Algorithm: generateRE()

        Input: Reference of "occurence.data" file which contains tag occurrence in HTML file.
        Output: Return pattern of different object (product,list,text etc.)

Begin
        1 . Create DocumentBuilder Factory object instance– to convert the HTML file into DOM tree

        2.  Create Xpath object instance – xpath used to find specific node from DOM tree

        3.do
        4.      Create NodeList object instance by setting object return from method getChildNodes() of
                node object
        5.      Declare a variable String type "pattern"
        6.      Get the nodename from Node object instance

                7. If node name is "Span"
                8.      Get the childnodes of Node object
                9       Get the value of first child
                10.     If value match with String "model"
                11.             Pattern += "model"
                12. Else if node name is "a"
                13. Create NodeList object to get all the child from instance of Node object
                14. Create String object to same node value from 1st object from the list
                15. Tokenize the value string using String Tokenizer
                16. if node name is 'href'
                17.     if(node value contains the value of firsttoken)
                18.             Pattern+= 'title'
                19else if attributes of node name is "class"
                20.     if(child node name is 'img")
                21.             pattern += image
        22.     get the attributes of nodes and set those to NameNode Map objects instance
        23.     For each attributes
        24.     get the item from NameNodemap Object instance and save it to Node object instance
        25.     If node value is "price"
                        Pattern += price
        26      while(end of "occurrence.data")
End
```

Figure 51: Algorithm generateRE()

3.5.3 NFAGenerator Module

The input of this module is the unified regular expression which is generated by the REGenerator module. This module generates the NFA to identify tuple from tuple list. This method contains two method called generateNFA() and identifyTuple().

```
Algorithm: NFAGenerator
Input: Regular expression, which is generated by Regenerator module
Output: Generate NFA and identify different types of tuple from tuple list using NFA.

Begin
        1. Call generateNFA(regular expression) – to generate the NFA based on regular
        expression
        2. Call identifyTuple(tuple pattern) -  to identify tuple from the tuple list.
End
```

Figure 52: Algorithm NFAGnerator()

This module implemented Thompson's construction algorithm to generate the NFA from regular expression. Thomson's construction builds NFA for each term of regular expression and combines them with "ε". Line 1 of this module calls method generateNFA() which accepts regular expression as input. This method used NFA objects to build the NFA from regular expression. The NFA object has some properties including initial and final state, size and transition_table. It also has some behaviors including is_legal_state(state s), add_transition(int from, int to, String input), shift_state(int shift) , fill_shift(NFA nfa), append_empty_state() and show_NFA(). The method is_legal_state(state s) checks the validity of the state. The range of the state should be 0 to size-1. The method add_trans(int from, int to, String input) insert input into two dimension array called transiotion_table. The method shit_state(int shift) creates a new empty transition table with the new size, copy all the transition to the new table and

update the NFA properties. The method append_empty_state() append a new row and column to the NFA. The class GeneratesNFA has some behaviors to build the NFA from regular expression including generateBasicNFA(String input), generateAlterNFA(NFA nfa1, NFA nfa2), generateConcatNFA(NFA nfa1, NFA nfa2) and generateStarNFA(NFA nfa). The method generateBasicNFA(String input) generates basic NFA with single input. The method generateAlterNFA(NFA nfa1, NFA nfa2) generates an alternation of nfa1 and nfa2. The new generated nfa will contain all the states from nfa1 and nfa2 in addition new initial state and final state. The initial state comes first, then comes states of nfa1, states of nfa2`s comes after state of nfa1`s and at the end comes new final state. This method uses the behavior of NFA including shift_state() to make room for new initial state, fill_state() to make room in new nfa, add_transition() to set new initial state and the transition from it, append_empty_state() to make up state for new final state, add_trand() to set new final state.

The method generateConcatNFA(NFA nfa1, NFA nfa2) generates a concatenation of nfa1 and nfa2. It first generates nfa1 and then nfa2. In this case, nfa2`s initial state replaces with nfa1`s final state. This task is done by the NFA behavior shift_state(). The method new_nfa(nfa2) of NFA generates a new nfa and initialize it with the shifted nfa2. In this case, new nfa formed by the states of nfa1`s. The initial state of nfa2`s is overwritten by the initial state of nfa1`s. This way nfa1 and nfa2 merge automatically which transform nfa2`s initial state from nfa1`s final state.

For our running example, generated regular expression (shown in figure :) is the input of algorithm GenerateNFA(). Line 1 tokenizes the “regex”(img (price|title) (title|model|price) (model|prodnum|Proddesc) (prodNum|price)) based on empty space (“

“), open bracket “(“, close bracket “)” and or “|”. Line 2-6 within a loop that creates a string “states” with the unique state from “regex” (e.g. “img, price, title, model, prodNum, ProdDesc” is created from the above “regex”). Line 7 tokenizes the string “states”. Line 8-10 form a loop that generates NFA for

```
Algorithm: GenerateNFA(regex)

Input: generated regular expression in Regenerator() module.
Output: generatedNFA

Begin

        1. Tokenize the “regex” using StringTokeinzer object
        2. Do
        3.      State = extract token from string tokenizer object
        4.      check for duplicate state
        5.              create string “states” with unique state
        6.while(hasmore token)

        7. Tokenize “states” using stringTokenizer object
        8.do
        9.      generate NFA for each individual attribute (e.g. “img”, “model”,
                “title”, ) exists int the regex.
        10.while(hasmoretoken)

        11. Tokenize the “regex” using string Tokenize object
        12. do
        13      generate NFA for each alternation
        14.     increment counter;
        15.while(has more token)

        16.Tokenize “regex” using StringTokenizer object
        17.do
        18      generate final NFA by doing concatenation between each state.
        19      increment value of I;
        20. while (i<counter)


End
```

Figure 53: Algorithm GenerateNFA()

each state exists in the string “states”. Line 9 is doing this using “generateBasicNFA()”. In this step, individual NFA are created for img , model, title, prodnum, prodDesc, price. Each NFA has it’s own initial state and final state. This task is done by method called “generateBasicNFA()”. Line 11 tokenizes string “regex” using string tokenizer object based on “(“ and “)”. Line 12-15 form a loop to generate NFA for each alternation. For our running example, after tokenize

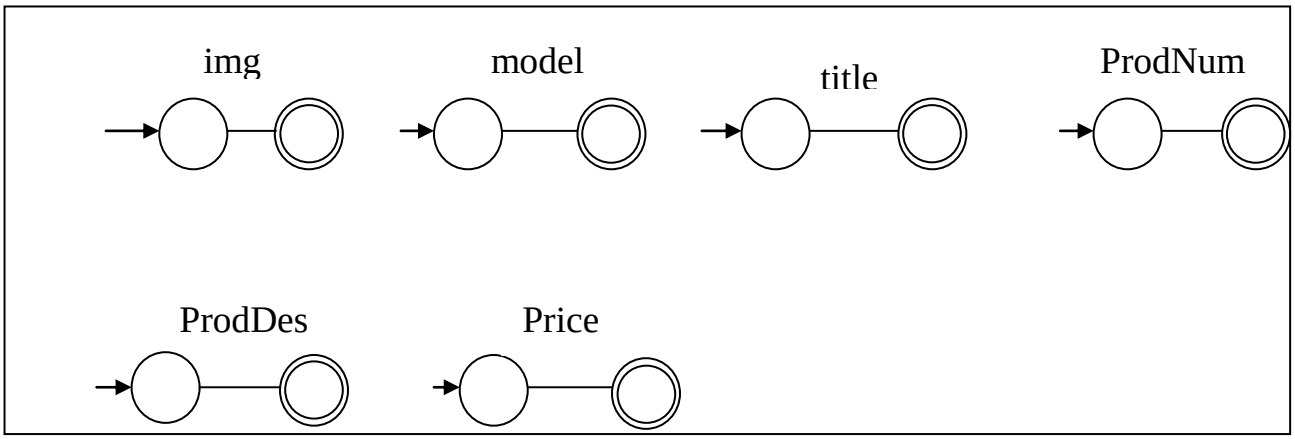


Figure 54:Snapshot of generated NFA of each attribute

“regex” there are four alternations exists in string tokenize object including “price|title”, “title|model|price” , “model|prodnum|ProdDesc” and “prodNum|price”. Line 11-15 generates the four alternation NFA. For example, the algorithm first generates the NFA for “price|title”. It first tokenize the string based on “|” character and extract two attributes which are “price” and“title”.

After that it merge “price” NFA and “title” NFA and generate NFA that is called alternation NFA of “price|title”. It this case, NFA is build with two individual NFA “price” and “title” and additional initial state and final state. The initial and final states connect with “price” NFA and “title” NFA with “” transition. Figure 59 is shown the structural view of alternation NFA generation. Line 12-15 also generates three other

alternation NFA including “title|model|price”, “model|prodnum|ProdDesc” and “prodNum|price”. This task is done by the method called “generateAlterNFA()”.

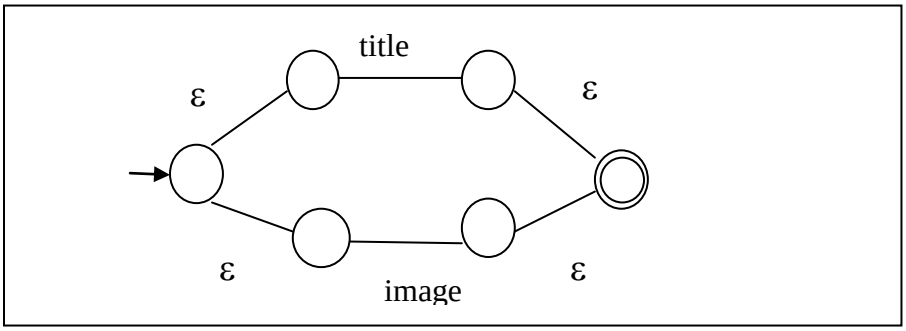


Figure 55: Structural view of alternation NFA of “(title|image)”

Line 17-20 do the concatenation between individual NFA and alternation NFA and generate unified NFA. In this case, initial state of nfa2 and final state of nfa1 is overlapped. This task is done by the method called generateConcatNFA().

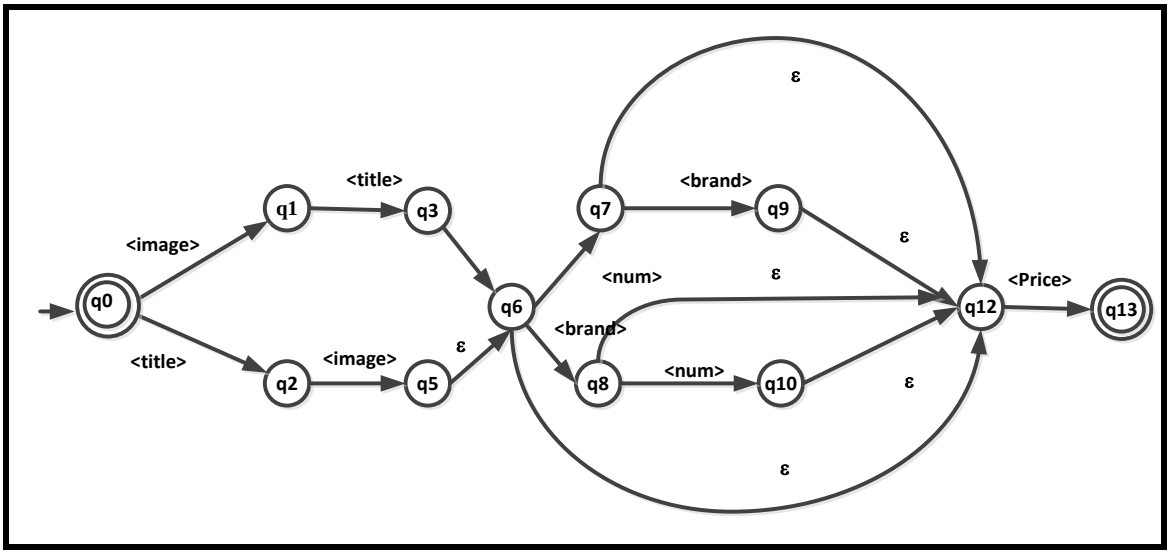


Figure 56 : Generated NFA

3.5.4 DFA generator

In this module, we implement “subset construction” algorithm to convert NFA to DFA to handle “ε” transition. The idea of Subset Construction is to build a DFA that keeps track

where the NFA can be. Each state in this DFA stands for a set of states the NFA can be in after some transition.

```
Algorithm: subset-construction
inputs: N - NFA
output: D - DFA
Begin
        1. add eps-closure(N.start) to dfa_states, unmarked
        2.D.start = eps-closure(N.start)

        3. while there is an unmarked state T in dfa_states do
        4.      mark(T)

        5.      if T contains a final state of N
        6.      add T to D.final

        7.      foreach input symbol i in N.inputs
        8.              U = eps-closure(N.move(T, i))
        9.              if U is not in dfa_states
        10.             add U to dfa_states, unmarked

        11. D.trans_table(T, i) = U
 end
```

Figure 57: Algorithm Subset construction

The algorithm starts by generating the initial state for the DFA. An initial state of DFA is really the NFA's initial state plus all the states reachable by **eps(ε)** transitions from it, the DFA initial state is the **eps-closure** of the NFA's initial state. A state is "marked" when all the transitions from it were visited. A state is added to the final states of the DFA if the set it represents contains the NFA's final state. The rest of the algorithm is a simple iterative graph search. Transitions are added to the DFA transition table for each symbol in the alphabet of the regex. So the DFA transition actually represents a transition to the **eps-closure** in each case. A DFA state represents a set of states the NFA can be in after a transition. For our running example, The algorithm “subset construction” takes

generated NFA(shown in fig 56) as input. The function "eps-closure" returns the states of N which are reachable from T (state set) by "ε" transition. First state T are added to the output. Then each states are checked for "ε" transition and the state with this transition are added to the output. The process proceeds iteratively until no more states can be reachable with "ε" only. For example, When "eps-closure" visit state q_2 , it return state { q_2 , q_5, q_{11} }. Because the state "q_2" have transition into these states with "ε" transition. And the function move(T,A) return information about which states in NFA are reachable from T with input set "A". This function traverse the state set T and looks for transition on the given input and returning the state that can be reached. It doesn't consider "ε" transition as input. The algorithm "subset construction" maintains a transition table to generate the DFA.

State/input	(img) ε*	(title) ε*	(Prodnum) ε*	(brand) ε*	(price)ε*
Q_0	Q_1, Q_2	Q_3, Q_4	ϕ	ϕ	ϕ
Q_1, Q_2	ϕ	Q_2, Q_5, Q_{11},	ϕ	ϕ	ϕ
Q_3, Q_4	Q_4, Q_5, Q_{11}	ϕ	ϕ	ϕ	ϕ
Q_2,Q_5, Q_{11}	ϕ	ϕ	Q_6,Q_7, Q_{10}	Q_6,Q_7, Q_{10}	ϕ
Q_4,Q_5, Q_{11}	ϕ	ϕ	Q_6,Q_7, Q_{10}	Q_8,Q_9, Q_{10}	ϕ
Q_6,Q_7, Q_{10}	ϕ	ϕ	ϕ	Q_6,Q_7, Q_{10}	ϕ
Q_8,Q_9, Q_{10}	ϕ	ϕ	Q_8,Q_9, Q_{10}	ϕ	Q_{11}, Q_{10}
Q_{11}, Q_{10}	ϕ	ϕ	ϕ	ϕ	Q_{11}, Q_{10}

Table 2:Transition table of DFA

The main idea of "subset construction" algorithm is that it removes the "ε" transition from NFA. And eliminate the state that has two outcomes to go to other state. It reform

the NFA by converting it to finite state automata which has set of state with one possible outcome. The generated DFA is shown in figure 60.

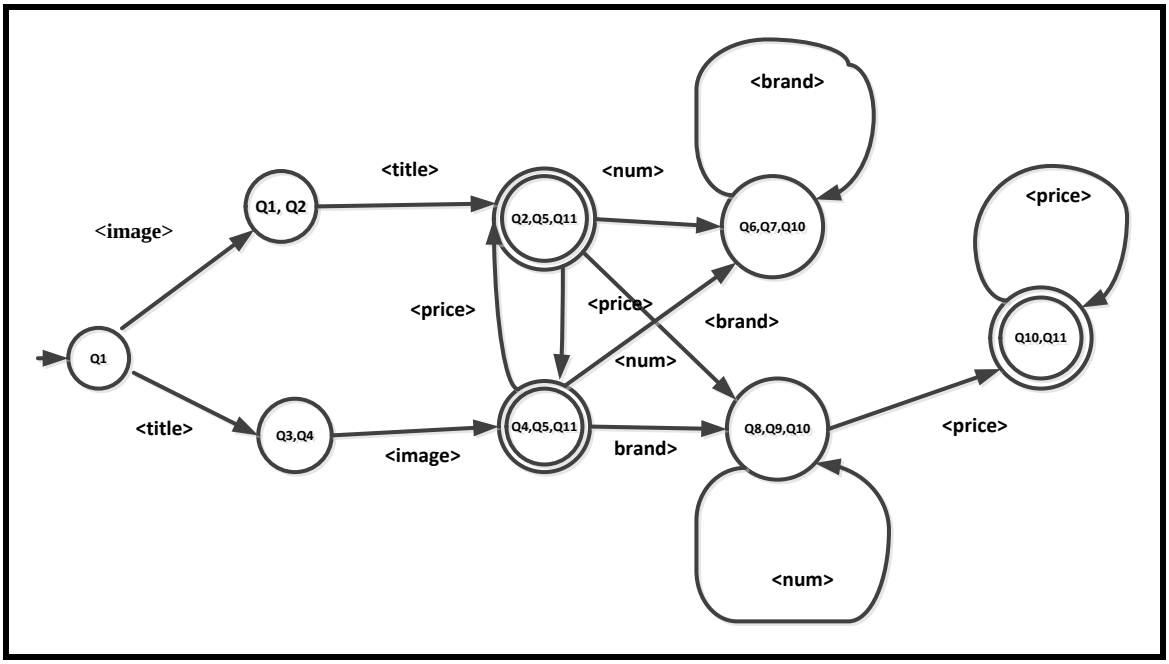


Figure 58: Generated DFA

The DFA generator module has function called “DFA simulation” that accepts or rejects the input string. This function is used by the “tuple classifer” module of “WebOMiner-2”. The tuple classifier module extracts the web content pattern from “contentObjectArray” and call function “DFA simulation” to verify the pattern with the generated DFA. If the pattern match with the DFA , the function return “accept”. Otherwise return “reject”. The “DFA simulation” algorithm is shown in Figure 59.

```
Algorithm dfa-simulate
inputs: D - DFA, I – Input string as content type
output: identified or not identified
Begin
 1.          x = start state of  D
 2.          y = get next input character from I

 3.          while not end of I do
 4.          x = state reached with input i from state s
 5.          y= get next input character from I
 6.   end

 7.   if x is a final state
 8.          return (identified)
 9.   else
 10          return (not identified)
```

Figure 59: Algorithm DFA simulation

3.5.5 DatabaseSchema generator

This module generates the database schema automatically using the unified RE that is generated by Regenerator module. This module takes unified RE string as input. First it call the connect() method to connect to the database. Then it calls the generateSchema() method to generate the schema based on the unified RE string which is passed into as a parameter.

```
Algorithm: DatabaseSchema generator
        Input: Object "Pattern" extracted in pattern extracted module
        Output: Create database Schema
Begin
        1. Call connect() – to create the connection with the databse
        2. Call generateSchema(unified RE)- to create the database schema into the database.
End
```

Figure 60 : Algorithm SchemaGenerator()

The “generateSchema()” method accepts the unified RE String <Image,image,title,model,ProdNumeber,price,price> as a parameter. Then it uses StringTokenizer object to tokenize the string. Then it creates the prepare statement object to create the database schema. It sets the column type based on the token. For example, when it scans the token “image” it set the column type as “bolb”, for the other token it sets data type as “Char” or “varchar”. With the above pattern, it generates the prepared statement which is given below:

String Schema= “Create table Product (id number, company_name, image1 bolb, image2 bolb, title char(50), Prod_Number char(15), Price1 char(5), Price2 Char(2));

After creating the “schema” string, it passes to the preparedStatement() method of Connection object. Then executedUpdate() method of connection is called to create the schema into the database. The generateSchema() method first checks the existing schema in the data warehouse. For this, it first fetches the existing schema of a specific object (e.g. product) from the database. Then it compares the schema with the pattern. If it finds any additional attribute then it update the database schema by adding the additional attribute as a column to the existing schema.

CHAPTER- 4 Evaluation of WebOMiner-2 System

The implementation phase of our algorithm has been completed and need some modification to make our system more scalable and robust. Since pattern recognition and generation of regular expression using finite automata is a new approach to data mining, a valid comparison in performances with other techniques do not exists.

Further improvement is needed to generate finite state automata from other domain contexts. The Crawler module of WebOMiner-2 needs further improvements to identify positive list pages (e.g. Figure 02) automatically. The miner module also needs improvements to extract information from detail pages that contain more information about the product (e.g. product specification).

4.1 Strength of WebOMiner-2

In this thesis, we developed a system called WebOMiner-2 which is a novel approach for web content mining using the object-oriented model. We developed an unsupervised system for web content mining using non-deterministic finite state automata. Existing web content extracting systems use the unsupervised, the supervised, and the semi-supervised approaches. The supervised and manual approaches use wrapper which is a set of web pages, labeled with examples of the data to be extracted. Wrapper generation requires a set of data extraction rules which are generated manually from labeled pages. Manual labeling of pages is labor intensive and time consuming because different templates exist in different sources. The semi-supervised approach accepts a rough training example from user and generates extraction rules. The unsupervised or the automatic approach generates wrappers without much user interaction. Since

WebOMiner-2 system is an automatic web content data extraction system, we compare our system with other unsupervised or automatic systems. The comparative analysis is given below:

Existing unsupervised approaches are able to extract only textual contents from the web. Most of them consider extracting product information from list pages and some of them extract information from the search engine results. These systems do not consider extracting heterogeneous web content like image or any other multimedia contents from the web page. Our WebOMiner-2 system able to extract heterogeneous data because the tag attributes are analyzed during the DOM tree traversal. Therefore images are identified effectively from the data block.

The unsupervised system MDR (Liu et al., 2003) is developed based on two observations. The first observation is that a group of data records form a contiguous region of a page. The second observation is that the data records have similar tag tree structure within the data region and the data records of a data region have the same parent node. The MDR is designed to handle web pages which generated by <table> tags. It failed to extract the data from the web pages which contain records that have complex and nested structures. The MDR works each time in a single page, so it does not compare the page trees. Although it achieving good results, the algorithm only works with multi-record pages and therefore cannot be applied to on-line news pages, that are almost exclusively single-record pages. In our WebOMiner-2 system we used the observations for data record identification. Our observation is that all objects of a data record are adjacent in a DOM sub-tree and each data record is separated from the others. Therefore, the DOM tree contains a single parent node which represents the sub-tree of an entire

data record. This parent node is identified by our system for each data record. Our system is unsupervised and is automatic because it does not depend on the browser rendering engine.

The DEPTA (Zhai and Liu, 2005) did not consider semantic label in data extraction where they only use tree structure. The DEPTA failed to extract nested data records. Our system able to extract nested data records because it traverses each node of the DOM tree and extract each record from the data block. The DEPTA use excel table to store extracted web content data. Excel table cannot be considered as a functional database because it is a data grid. The DEPTA stores similar tag encoded contents into same excel columns. Our system is able to generate a database schema automatically to store the extracted web content from each web page. Because it is able to identify the content type during the traverse of the DOM tree and extracts the pattern of the content to generate the database schema to hold the content information into the database.

The NET (Liu and Zhai, 2005) proposed a greedy approach based on similarity match. It employs an expensive approach due to a bottom-up traversal with edit distance comparison. It requires a full scan from bottom to root. The NET does the all-pair tree comparisons within its children during each visit of a node in the traversal. The Wrapper generated by NET is not efficient though because the programmers have to find the reference point and the absolute tag path of the targeted data content manually. This requires one wrapper for each web site since different sites follow different templates and it is labor intensive and time consuming. Our system does not depend on any templates and it does not employ an expensive approach because it is top to bottom traversal approach.

The limitation of OMINI (Buttler et al., 2001) is that it doesn't address how to precisely locate the data object instances in the separated parts and how to extract them by their specific structures. The separator contains only one HTML tag which is insufficient. The OMINI is good for segmenting web pages into parts, possibly containing data object instances. OMINI performs poorly on some web pages, the description of one data objects may intertwine with the descriptions of some other objects. On the other hand, WEBOMINER-2 is able to extract data from all regions from body zone include list, product, text, advertisement etc. Our system generates a NFA from regular expression of different objects existing in the web page. This NFA is used to identify different object from object list.

The VINTs(Zhao et al., 2005) fails to separate horizontally arranging data records which will require vertical separators due to fact that VINTs only supplies horizontal separator. The VINTs needs at least four data record exist in a web page for wrapper building. Since VINTs is based on visual layout information, it is difficult to identify visual information without any assumptions about the target domain. The visual feature used in VINTs are only limited to the content shape-related features and it is used to identify the regularities between search records. For this reason, VINTs depends on structural similarities and must generate a wrapper for each search engine. Our system is able to extract the web content from both horizontally and vertically arranging data records because it doesn't depend on horizontal or vertical separator. It traverses the DOM tree and extracts each data record from the each node. Since our system doesn't generate any wrapper, our system doesn't require any training web page. Our system able to extract web contents from web page by identifying the data type while extracting.

4.2 Empirical evaluation:

This thesis developed an architecture which is a combination of 5 modules and generates finite automata by processing the web content and generates data warehouse schema. The empirical evaluation of our system is done by the experiment with 5 website including "bestbuy.com", "bestbuy.ca", "futureshop.ca","compUSA.com" and "walmart.com". We run our system on a 64 bit operating system at Interl® core™ i3-2350 CPU @2.30 GHz 4GB RAM Toshiba machine for each these web sites for empirical evaluation of our system. We use the standard precision and recall measures to evaluate the results of our system. Precision is measured as average in percentage for the number of correct data retrieved, divided by the total number of data retrieved by the system. Recall is measured as average in percentage for the total number of correct data retrieved divided by the total number of existing data in the web document. The results of the retrieval by our WebOMiner-2 system is tabulated in Table 3 below:

Website	Data records					Data record extraction WebOMiner				Data record Extraction WebOMiner-2			
	Product	List	Noise	Text	Total	Correct	Wrong	Missing	Failed	Correct	Wrong	Missing	Failed
Homedepot.com	12	11	5	0	28	16	0	0	12	28	0	0	0
Shopxscargo.com	16	15	4	2	37	21	0	0	16	37	0	0	0
Target.com	9	10	3	1	23	14	0	0	9	23	0	0	0
Sears.com	13	8	2	0	23	21	0	2	0	21	0	2	0
Factorydirect.com	15	12	4	0	31	30	0	1	0	30	0	1	0
Bestbuy.com	12	10	3	1	26	14	0	0	12	26	0	0	0
Recall						69.1%				98.3%			
Precision						100%				100%			

Table 3: Experimental results is showing extraction of record from web pages.

4.3 Experimental Results

The purpose of our experiment is to measure the performance of WebOMiner-2 for data record extraction. The Table 3 shows small scale experiment results as performance measure for our WebOMiner-2 system. We compare our system with WebOMiner. We have taken one page per web site for experiments. These are the six different B2C web site which didn't choose to generate the NFA for WebOMiner (The WebOMiner system generates NFA based on manual observation of ten different B2C web sites). The number of "Data record" column shows different type of data records (product, list, text, noise) exist in those page. The Total column shown total number of data records for each pages. The column "correct" means that the system able to identify the contents correctly. For example, WebOMiner extracts 16 contents correctly from 28 contents from "HomeDepot.com". The column "failed" means that the system is failed to identify contents. For example, WebOMiner failed to identify 12 product contents. The column "wrong" means that the system wrongly identified the contents. The column "missing" means that the contents are missing due to different structure. For those pages WebOMiner-2 system is able to identify data records correctly. But the WebOMiner failed to extract product data information from four web site (homedepot.com, shopxscargo.com, target.com and bestbuy.com). Because NFA generated by WebOMiner failed to identify the product tuple from these web page. There are no wrong data records are extracted because our system is not based on the prediction. It missed 3 data records out of total 168 data records in all six web pages from different websites. From the above table we found that WebOMiner-2 performs better than WebOMiner in data record extraction.

We observed the reason for missing attributes. All of those missing are in List type data records and because of mixing object type in data tuple. The WebOMiner defines a List data tuple as a set of (<link> <text>) pair and there should be at least 3-pairs in the tuple to be qualified as List tuple. But those missing tuples are pair of <image> and <text> and therefore did not satisfy the criteria.

CHAPTER 5 - Conclusion and Future Work

This thesis extends work of Mutsuddy and Ezeife (2010), Ezeife and Mutsuddy(2013) to generate finite automata to mine related content from specific domain context. We modified the NFA generator module of WebOMiner to generate finite automata from regular expression which generated from repeated pattern of web content. Our algorithm able to generate database schema automatically using automata pattern. We named our architecture as WebOMiner-2. Our architecture has 5-modules includes pattern extractor, regular expression generator, NFA generator, DFA generator and schema generator. We developed algorithms to extract pattern of different objects (product, list, text, etc.) from HTML page. The pattern extractor module extracts repeated pattern from web page. The regular expression generator module generates regular expression using pattern extracted in pattern extractor module. The NFA generator module implements thompson's construction algorithm to built NFA from regular expression. We implemented subset construction algorithm to convert NFA into DFA. We modified the WebOMiner architecture to mine different contents from web page using finite automata. We also developed "Schema generator" module that generates database schema into the database based on pattern extracted from web page.

5.1 Future Work

The generation of regular expression from repeated pattern of web content and Pattern recognition using finite state automata is a new approach in data mining. So, this approach has many scopes for improvement. Our proposed approach able to generate finite automata of related contents from specific domain context. Further improvement is needed to generate finite automata from other domain context. The Crawler module of WebOMiner-2 needs further improvement to identify positive list page (e.g., Figure 02) automatically. The miner module also needs improvement to extract information from detail page that contains more information about the product (e.g. product specification).

REFERENCES

1. Alim, S., Abdul-Rahman, R., Neagu, D., Ridley, M. 2009. Data retrieval from online social network profiles for social engineering applications. Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for , Vol., No., pp.1-5, 9-12 Nov. 2009.

2. Arasu, A. and Garcia-Molina, H. 2003. Extracting structured data from web pages. In proceedings of the ACM SIGMOD International Conference on Management of Data. 337-348.

3. Annoni, E. and Ezeife, C.I. 2009. Modeling web documents as objects for automatic web content extraction. In proceedings of the ACM/LNCS sponsored 11th International Conference on Enterprise Information Systems (ICEIS 09), pp. 91- 100, May 6-10, Milan, Italy.

4. Arocena, G.O. and Mendelzon, A.O. 1998. WebOQL: Restructuring documents, databases, and webs. Proceedings of the 14th IEEE International Conference on Data Engineering (ICDE), pp. 24-33, Orlando, Florida

5. Appelt, D.E. and Israel, D. J. 1999. Introduction to information extraction technology. A Tutorial Prepared for IJCAI-99.

6. Algur S.P. and Hiremath, P. S.2006. Extraction of flat and nested data records from web pages. In AusDM '06: Proceedings of the fifth Australasian conference on Data mining and analystics. Australian Computer Society, Inc., pp. 163–168. Darlinghurst, Australia

7. Adelberg, B. 1998. NoDoSE: A tool for semi automatically extracting structured and semi-structured data from text documents. SIGMOD Record, Vol. 27, No. 2, pp. 283-294,

8. Buttler, D., Liu, L., and Pu, C. 2001. A fully automated extraction system for the World Wide Web. Distributed Computing Systems, 2001. 21st International Conference on. , Vol., No., pp.361-370, Apr 2001.

9. Botzer, D.; Etzion, O. 1996. Optimization of materialization strategies for derived data elements. Knowledge and Data Engineering, IEEE Transactions on , Vol.8, No.2, pp.260-272, Apr 1996.

10. Breuel, T.M. 2003. Information extraction from HTML documents by structural matching. Proceedings of the 2nd International Workshop on Web Document Analysis (WDA2003) PARC, Inc.,Palo Alto, CA, USA.

11. Chang, C., Kayed, M., Girgis M.R. and Shaalan, K.F. 2006. A Survey of Web Information Extraction Systems. IEEE TKDE, Vol 18, No. 10, pp.1411-1428, Oct. 2006.

12. Chang, C.H., and Lui, S.C. 2001. IEPAD: Information Extraction Based on Pattern Discovery. Proceedings of the 10th Int'l Conf. World Wide Web (WWW), pp. 223-231, 2001.

13. Chang, S.C., Hsu, C.N., and Lui, S.C. 2003. Automatic Information Extraction from Semi-structured Web Pages by Pattern Discovery. Decision Support Systems J., Vol. 35, No. 1, pp. 129-147, 2003.

14. Chang, C.H. and Kuo, S.C. 2004. OLERA: A semi-supervised approach for web data extraction with visual support. IEEE Intelligent Systems, 19(6):56-64, 2004.

15. Califf, M. and Mooney, R. 1999. Relational learning of pattern-match rules for information extraction. In Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence (AAAI '99/IAAI '99). American Association for Artificial Intelligence, pp328-334. Menlo Park, CA, USA.

16. Crescenzi, V. and Mecca, G. 1998. Grammars have exceptions. Information Systems, 23(8): 539-565, 1998.

17. Crescenzi, V. and Mecca, G. 2004. Automatic Information Extraction from Large Websites. Journal of the ACM 51, 5, 731-779.

18. Liu, L., Pu, C., and Han, W. 2000. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. Proceedings of the 16th IEEE International Conference on Data Engineering (ICDE), San Diego, California, pp. 611-621, 2000.

19. Crescenzi, V., Mecca, G., and Merialdo, P. 2001. ROADRUNNER: Towards automatic data extraction from large web sites. In Proceedings of the 2001 International Conference on Very Large Data Bases, pp. 109–118.

20. Embley, D., Campbell, D., Jiang, Y., Liddle, S., Lonsdale, D., Ng, Y.-K., and Smith, R.1999. Conceptual-model-based data extraction from multiple-record web pages. Journal on Data & Knowledge Engineering 31, 3, pp. 227-251.

21. Ezeife, C.I. and Mutsuddy, T. 2013. Towards Comparative Mining of Web Document Objects with NFA: WebOMiner System, to appear in the International Journal of Data Warehousing and Mining (IJDWM), a 2013 volume.

22. Hammer, J., McHugh, J. and Garcia-Molina, H. 1997. Semi structured data: the TSIMMIS experience. In Proceedings of the 1st East-European Symposium on Advances in Databases and Information Systems (ADBIS), pp. 1-8, 1997, St. Petersburg, Russia.

23. Hogue, A. and 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 (WWW), pp. 86-95, 2005,Japan.

24. Hsu,C.N. and Dung,M.T. 1998. Generating finite-state transducers for semi-structured data extraction from the Web. Information Systems, Volume 23, Issue 8, Pages 521-538, December 1998.

25. Hong, J. L., Siew, E.-G., and Egerton, S. 2010. Information extraction for search engines using fast heuristics. Data and Knowledge Engineering 69, 169-196.

26. Kayed, M. and Chang, C.H. 2010. FiVaTech: Page-Level Web Data Extraction from Template Pages. IEEE Transactions on Knowledge and Data Engineering 22, 2, pp.249-263.

27. Kushmerick, N., Weld, D., and Doorenbos, R. 1997. Wrapper induction for information extraction. Proceedings of the Fifteenth International Conference on Artificial Intelligence (IJCAI), pp. 729-735, 1997.

28. Lerman, K., Getoor, L., Minton, S., and Knoblock, C. 2004. Using the structure of Web sites for automatic segmentation of tables. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD '04). ACM, pp.119-130. 2004, New York, NY, USA.

29. Liu, B., Grossman, R., and Zhai, Y. 2003. Mining data Records in web pages. In Proceeding of the 9th ACM SIGKDD conference on Knowledge Discovery and Data Mining. 601-605.

30. Liu, B. and Zhai, Y. 2005. NET: A system for extracting web data from flat and nested data records. In Proceeding of the 6th conference on Web Information Systems Engineering, pp. 487-495.

31. Liu, W., Meng, X., and Meng, W. 2010. ViDE: A vision-based approach for deep web data Extraction. IEEE Transactions on Knowledge and Data Engineering 22, 3, 447-459.

32. Liu, L., Pu, C., and Han, W. 2000. XWRAP: an XML-enabled wrapper construction system for web information sources. Data Engineering, 2000. Proceedings 16th international conference on , Vol., No., pp.611-621, 2000.

33. Lo, L., Ng, V. T.Y., Ng, P., and Chan, S. C. 2006. Automatic Template Detection for Structured Web Pages. In Proceedings 10th international conference on Computer Supported Cooperative Work in Design. vol., no., pp.1-6, 3-5 May 2006.

34. Muslea, I., Minton, S., and Knoblock, C. 1999. A hierarchical approach to wrapper induction. Proceedings of the Third International Conference on Autonomous Agents (AA-99), pp. 190-197,Seattle,Washington,USA, 1999.

35. Marini, J. 2002. The Document Object Model, Processing Structured Documents, McGraw-Hill.

36. Mize, J. and Habermann, R.T. 2010. Automating metadata for dynamic datasets. *OCEANS 2010*, pp.1-6, 20-23 Sept. 2010.

37. Miao, G., Tatemura, J., Hsiung, W., Sawires, A., and Moser, L.E. 2009. Extracting data records from the web using tag path clustering. In Proceedings of the 18th international conference on world wide web (WWW '09). ACM, pp. 981-990, New York, NY, USA.

38. Mutsuddy T., and Ezeife C., I. 2010. Towards comparative web content mining using object oriented model. Unpublished M.Sc., University of Windsor, ON, Canada

39. Peshkin, L., and Pfeffer, A. 2003. Bayesian information extraction network. In IJCAI' 03: Proceedings of the 18th international joint conference on Artificial intelligence, pages 421–426,Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

40. Reis, D.C., Golgher, P.B., Silva, A.S., and Laender. A.S. 2004. Automatic web news extraction using tree edit distance. In Proceedings of the 13th international conference on World Wide Web (WWW '04). ACM, 502-511. New York, NY, USA.

41. Singhal, A., Seborg, D.E. 2001. Matching patterns from historical data using PCA and distance similarity factors. American Control Conference, 2001. Proceedings of the 2001 , Vol.2, No., pp.1759-1764 vol.2, 2001.

42. Saiiuguet, A. and Azavant, F. 2001. Building intelligent Web applications using lightweight wrappers. Data and Knowledge Engineering 36(3), pp. 283-316, 2001.

43. Simon, K. and Lausen, G. 2005. ViPER: Augmenting Automatic Information Extraction with visual Perceptions. In Proceedings 14th ACM Conference on Information and Knowledge Management, pp. 381-388.

44. Senellart, P., Mittal, A., Muschick, D., Gilleron, R., and Tommasi, M. 2008. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceedings of the 10th ACM workshop on Web information and data management* (WIDM '08). ACM, pages 9-16, New York, NY, USA.

45. Soderland, S.1999. Learning information extraction rules for semi-structured and free text. Journal of Machine Learning, 34(1-3): pages. 233-272, 1999.

46. Su, W.,Wang, J., and Lochovsky, F. H. 2009. ODE: Ontology-Assisted Data Extraction. ACM Transactions on Database Systems Vol.34,No. 2, pp. 12:1-12:35, July,2009.

47. Wang, J., and Lochovsky, F.H., 2002. Data-rich section extraction from HTML pages. In Proceedings of the 3rd Conference on Web Information Systems Engineering. Pages 313-322.

48. Wang, J., and Lochovsky, F.H. 2002. Wrapper Induction Based on Nested Pattern Discovery. Technical Report HKUST-CS-27-02, Dept. of Computer Science, Hong Kong, Univ. of Science & Technology, 2002.

49. Wang, J. and Lochovsky, F. H. 2003. Data extraction and label assignment for Web databases, Proceedings of the Twelfth International Conference on World Wide Web (WWW), pp. 187-196, 2003. Budapest, Hungary.

50. Wu, B., Cheng, X., Wang, Y., Guo, Y., and Song, L. 2009. Simultaneous product attribute name and value extraction from web pages. In Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, pages 295–298, 2009.

51. Yeonjung, K., Jeahyun, P., Taehwan, K., dan Joongmin, C. 2007. Web information extraction by HTML tree edit distance matching. In Proceedings of the International Conference on Convergence Information Technology (ICCIT.2007), pp. 2455-2460, 2007,Washington, DC, US.

52. Zhai, Y. and Liu, B. 2005. Web Data Extraction Based on Partial Tree Alignment. In Proceedings of the 14th International World Wide Web Conference on World Wide Web (WWW '05). ACM, pages 76-85. York, NY, USA.

53. Zhai, Y. and Liu, B. 2006. Structured Data Extraction from the Web Based on Partial Tree Alignment. IEEE Transactions on Knowledge and Data Engineering Vol. 18, No.12, pp.1614-1628.

54. Zheng, S., Song, R., and Wen., J. 2007. Template-independent news extraction based on visual consistency. In AAAI'07, volume 22, pages 1507–1513, 2007.

55. Zhao, H., Meng, W., Wu, Z., Raghavan, V., and Yu, C. 2005. Fully Automatic WrapperGeneration For Search Engines. In Proceedings 14th International World Wide Web Conference, pages 66-75.

VITA AUCTORIS

NAME:	Mohammad Harun-or-Rashid
PLACE OF BIRTH:	Dhaka,Bangladesh
YEAR OF BIRTH:	1978
EDUCATION:	Dhaka City College, Dhaka, Bangladesh, 1995
	University of Windsor, B.Sc., Windsor, ON, 2004
	University of Windsor, M.Sc., Windsor, ON, 2012