Towards Comparative Mining of Web Document Objects with NFA: WebOMiner System

C. I. Ezeife, School of Computer Science, University of Windsor, Windsor, ON, Canada

Titas Mutsuddy, School of Computer Science, University of Windsor, Windsor, ON, Canada

ABSTRACT

The process of extracting comparative heterogeneous web content data which are derived and historical from related web pages is still at its infancy and not developed. Discovering potentially useful and previously unknown information or knowledge from web contents such as "list all articles on 'Sequential Pattern Mining' written between 2007 and 2011 including title, authors, volume, abstract, paper, citation, year of publication," would require finding the schema of web documents from different web pages, performing web content data integration, building their virtual or physical data warehouse before web content extraction and mining from the database. This paper proposes a technique for automatic web content data extraction, the WebOMiner system, which models web sites of a specific domain like Business to Customer (B2C) web sites, as object oriented database schemas. Then, non-deterministic finite state automata (NFA) based wrappers for recognizing content types from this domain are built and used for extraction of related contents from data blocks into an integrated database for future second level mining for deep knowledge discovery.

Keywords: *Non-Deterministic Finite Automata (NFA), Object Oriented Mining, Web Content Mining, Web Data Integration, Wrappers*

INTRODUCTION

World Wide Web (WWW) is growing exponentially over the years and web documents constitute some of the largest repositories of information (Kosala & Blockeel, 2000). Web content usually refers to the information that a user sees on a web document. It also includes some hidden information which help users interact with web contents. Web contents are heterogeneous in nature and may be in different forms like text, image, hyper-link, metadata, audio, video and others with combinations of these content types as well. A complete classification of all these different types of web contents does not exist. Web content data are updated frequently, volatile and not historical (Bhowmick et al., 1999; Dung, Rahayu, & Taniar, 2007). The creation and maintenance of a data warehouse based on the web content data

DOI: 10.4018/jdwm.2012100101

is needed for effective derived and historical querying of web content data. Some researchers adopted the web data extraction system in virtual approach without creating physical data base and warehouse (Bornhövd & Buchmann, 1999) but may have difficulty with contents like images. There are some other information or block in the web pages such as advertisement, attached pages, copyright notices. These are also web contents and usually not considered as part of the primary page information. These unwanted information in a web page are called noise information, and usually need to be cleaned before mining the web contents (Gupta et al., 2005; Ezeife & Ohanekwu, 2005; Li & Ezeife, 2006). Borges and Leven (1999) categorized web mining into three areas: web structure mining, web usage mining and web content mining. Web usage mining processes usage information or the history of user's visit to different web pages, which are generally stored in chronological order in web log file, server log, error log and cookie log (Buchner & Mulvenna, 1998; Ezeife & Lui, 2009; Priya & Vadivel, 2012).When any mechanism is used to extract relevant and important information from web documents or to discover knowledge or pattern from web documents, it is then called web content mining. Traditional mechanisms include: providing a language to extract certain pattern from web pages, discovering frequent patterns, clustering for document classification, machine learning for wrapper (e.g., data extraction program) induction, and automatic wrapper generation (Liu & Chen-Chung-Chang, 2004; Muslea, Minton, & Knoblock, 1999; Zhao et al., 2005; Crescenzi, Mecca, & Merialdo, 2001; Liu, 2007). All these traditional mechanisms are unable to catch heterogeneous web contents together as they strictly rely on web document presentation structure. Existing extractors also are limited with regards to finding comparative historical and derived information from web documents. Creating more robust automatic wrappers for multiple data sources requires incorporating efficient techniques for automatic schema (attribute) match, some of which techniques are presented in Lewis and Janeja (2011). Methods for testing the quality of extracted and integrated information can also be incorporated in the future (Golfarelli & Rizzi, 2011). Some sample queries that may not be accurately answered by existing systems are:

1. Provide a comparative analysis of products including sales, comments on four retail store web sites in the past 1 year.
2. List all 17" LCD Samsung monitor selling around Toronto with price range less than $200.

In Annoni and Ezeife (2009), a model for representing web contents as objects was presented. They encapsulate web contents in object-oriented class hierarchies which would enable catching heterogeneous contents together in a unified way without strictly relying on web page presentation structure, using six web content data types.

Contributions

This paper proposes a two level mining process based on object oriented data modeling of web contents and focuses on the first level mining. The first level mining extracts and classifies web content data using content object hierarchies defined by (Annoni & Ezeife, 2009) in their OWebMiner system, and stores these data into database table. Second level mining is similar to traditional data mining process in an object oriented dataset and is outside the scope of this paper. This paper focuses on the first level mining and proposes an architecture called WebOMiner (standing for Web Object Miner) for extraction and mining of web contents using object-oriented model similar to those defined in the OWebMiner sytem of 2009. Our architecture has 4-modules: crawler module, cleaner module, extractor module and miner module. We developed algorithms for crawler module, used the freeware software "HTMLCLEANER-2.2" (Sourceforge.net, 2010) for cleaner module, extend and complete the algorithms for the extractor module initially proposed by OWebMiner system of 2009, by using an innovative algorithm for miner module that generates and uses non-deterministic finite state automata (NFA) for mining web content objects to enable automatic content extraction, discovery of web object hierarchies and scalability. The five specific contributions made by the WebOMiner system that were not handled by the OWebMiner system of 2009 are:

1. Identification of data blocks and regions from DOM trees of extracted html files of web pages which are to be integrated and mined for content, using only the tag information of the DOM tree with no vision-based context structure, is a contribution of this paper. The initial draft proposal by OWebMiner system of 2009 relies on DOM tree of web document and uses "vision based context structure" for data x-coordinate and y-coordinate location of webpage features, web document zone, data's width, height and center location, and data presentation features (such as style, type, fonts and spaces) to identify data blocks. Extracting the x-coordinate and y-coordinate information from DOM tree for automatic extraction is not easily feasible. Within DOM tree all related data are structured as data block but in flat array data structure used by OWebMiner of 2009, content data lose their relationships. It is important to extract related data together or create clear separation between data blocks and data regions which contain the blocks.
2. The work in the 2009 system did not discuss use of the separator element in content or presentation object extraction. Our work in this paper defines the use of separator element for identification of data block and data region in our problem context.
3. They did not define the object classes, size of object classes, object class hierarchy, object class dependencies, and functionalities of object classes. They only classify the web content elements (such as text, list, forms, product) but did not associate object types with contents, nor discuss how to control the creation of expensive objects. In this work, we generate NFA for each content type so that can be used to identify specific contents in the DOM tree during tuple extraction.
4. They did not address the issue of associating leaf level tags with specific contents. A leaf level tag contains important information about the associated content. It is important to associate leaf level tags before assigning an object to a content type. For example in a data block there are three image tags as shown:

*<img**id= "line"**src= "http://................"**alt =** **"line"** />* (1)
*<img**id= "monitor"**src=* *"http://..............."**alt = "monitor"** />* (2)
*<img src = "http://........................"**alt=** **"add"**/>* (3)

Here, the HTML tags at lines (1) and (2) have three tag attributes: "id," "src," and "alt." Line (3) has two attributes: "src" and "alt." Tag attributes are variable inside a tag and each attribute should have a value. First image tag of line (1) is a line separator as identified from the value of attribute "id" and "alt," second image tag of line (2) is for "monitor" as identified from "id" and "alt" attribute and the third image tag is for "Add to Cart" hyperlink identified from "alt" attribute. If we do not care about tag attribute of a source image, we will not be able to identify the image we want. We resolve this problem by analyzing tag attribute in this paper.

5. They did not address the issue of preventing noisy data entry into database table. Their algorithm does not refine contents before entering into the database table. We address this issue by cleaning noises from data tuple.

Outline of the Paper

First, we present the related work while we then present the proposed WebOMiner architecture and the NFA generator-based miner algorithm.

Afterwards we present the experimental results. Conclusions and future work are presented at the end of the article.

RELATED WORK

In order to mine data on web documents, the HTML files are usually converted to pre-parsed documents in a DOM (document object model) tree by many systems (Zhao et al., 2005; Annoni & Ezeife, 2009). In a DOM tree, tag and tag attributes are represented with nodes that specify the hierarchical relationships between tags. IEPAD (Chang & Liu, 2001) is one of the first information extractor systems that generalizes extraction patterns from unlabeled web pages. It is a learning wrapper that discovers repetitive patterns from web pages that are well encoded using a data structure called PAT trees, which is a binary suffix tree. Wargo system (Raposo et al., 2002), internally relies on two wrapper programming languages: Navigation SEQuence Language (NSEQL) for specifying navigation sequence and Data EXTraction Language (DEXTL) for specifying extraction pattern. DEByE (Laender, Neto, & da Silva, 2002) is an interactive tool that receives as input a set of example objects taken from a sample web page and generates extraction patterns. Web Data Extraction System (WICCAP) (Zhao & Ng, 2004) uses Web Data Extraction Language (WDEL), a scripting language. Main problem with wrapper language-based extraction system is their reliability on diversified, non-standard, non-popular wrapper languages. There are two existing approaches for building DOM tree (Jupiter Media, 2007); using tags alone and using tags along with visual cues (Baumgartner, Flesca, & Gottlob, 2001; Chriisment et al., 2004). HTML is a flexible mark-up language and page designer's error in using tag is mostly accepted. So, DOM tree building by using the tag alone requires HTML code cleaning to ensure the HTML page is well formed (that is, has all opening and closed tags) before building the tree.

The OWebMiner System Related to Proposed System WebOMiner

Annoni and Ezeife (2009) proposed the idea of encapsulating heterogeneous web contents into object class hierarchies to extract and mine web contents in a comprehensive, derived and historical way. All papers discussed so far in this related work of web content extraction rely on the web content presentation tree structure and extract only limited targeted facts from the web page. In this 2009 paper, an object-oriented paradigm to model web data to capture both content and presentation objects of a web document was proposed with these two major contributions for web content extraction: (A) They define and give a framework of object-oriented data model and (B) They give the idea of how to extract web objects from the web page. They give a high level algorithm called OWebMiner() for web object extraction and an algorithm called ProcessPresentationSibling() for presentation (e.g., web page tag structure) object extraction process. Their anticipated use of presentation objects is to associate them with content objects for mining process. Their proposed framework for object-oriented data model is based on the following concepts:

1. Related documents share same space and web page presentation tag structure. The web document segmentation work uses DOM tree, data location features (e.g., WebZoneCenterX and Y values, width, height) and data presentation features to distinguish data blocks.
2. Proposed to not evaluate all HTML tags because all HTML tags are not always meaningful. They observed that main HTML tags (e.g., non-empty tags such as < *table* >, < *link* >, < *form*> tags) have impact in content and presentation and pre-formatting and in-line tags such as < *pre* >, < *br/* > should not be evaluated. They rely on DOM tree of web document and use "vision based context structure" for defining data x-coordinate and y-coordinate location of webpage features,

web document zone, data's width, height and center location, and data presentation features such as style, type, fonts and spaces to identify data blocks. They define the web document zone to represent the entire web document as an object named WebZone object, which is divided into three zones: HeaderZone, BodyZone, and FootZone. They classify WebElement into six web content types of Text, Image, Form, Plug-in content, Separator element, and Structure element all of which are in an object class hierarchy. Their main algorithm OWebMiner() basically takes a set of webpages (WDHTMLFile) and for each WDHTMLFile, line (A) of the algorithm extracts all the content and presentation objects into two separate object arrays according to their DOM hierarchical dependencies. Line (B) stores web objects into the database. Line (C) mines the extracted contents from the database. Process begins with the root of the DOM tree of the web page, "<html>." When it hits series-1, it calls algorithm PrecessContentSibling() to start extraction of content objects and continues until it hits series-2. ProcessContentSibling() algorithm inputs DOM tree, a pointer called "TTag" which indicates current tag to process in the DOM tree, ContentObjectArray[] and a variable "indTag" which is a global index for labeling content objects per zone. The algorithm recursively traverses DOM tree block-level tags by depth-first search until it hits non-block level tag and resets "TTag" pointer to represent current processing tag. If depth-first search hits a non-block level tag, it processes all its siblings into an array called "tagArray." 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 algorithm finally returns the ContentObjectArray[] with full content objects from body zone of web page. This related work layed down conceptual object model and initial algorithms but detailed and scalable implementation of this automatic content extractor was still needed.

THE PROPOSED WEBOMINER SYSTEM

The proposed WebOMiner system aims at developing 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 problem to be solved and the proposed solution are given.

Problem Definition 3.1.: Given a number of product list web pages of retail stores (e.g., Figure 1) as defined in Example 1 in the previous sections, extract the information specified into the database for comparative mining and querying.

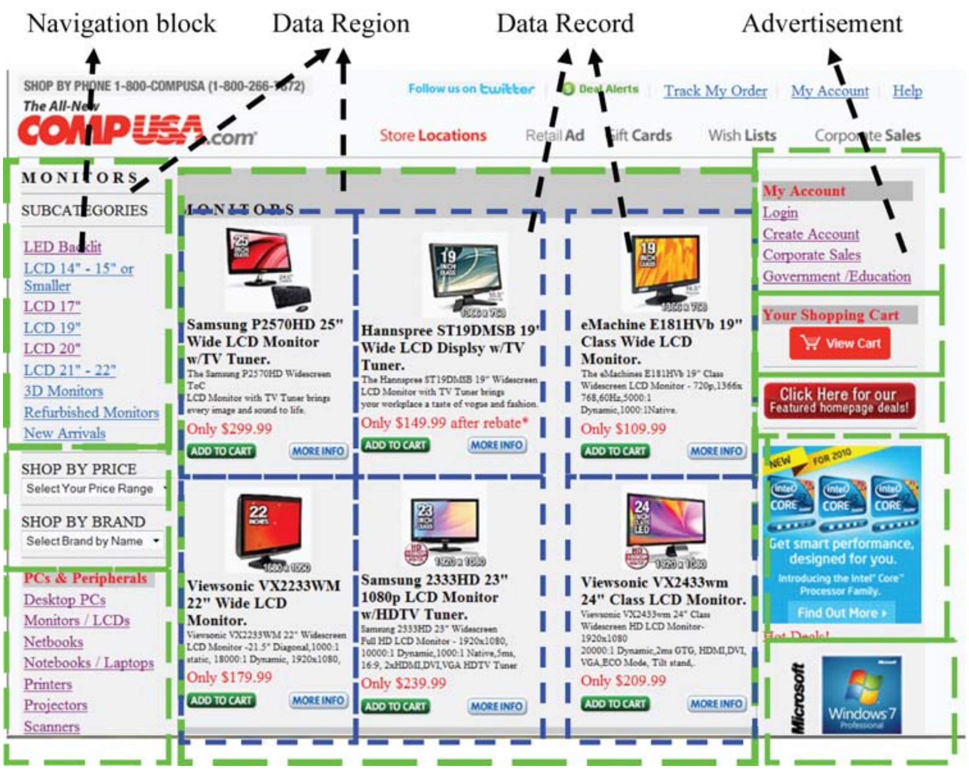
Proposed Solution 3.1.: For the specific domain of B2C websites, we have selected to mine the most common data-rich web page, the product list page which usually contains a brief list of all or specific types of products. There is a set of product list pages in a B2C website, and such a product list page is defined as follows:

Definition 3.1.: A B2C web site w of a domain d *is made up of a number of product list pages where each product list page (e.g., Figure 1), p is defined as a page that contains a set of tuples t of type α identified as one of the instance types in domain d. Product list page, p, contains a set of data blocks that are arranged in different data regions including product, navigation, and noise blocks.*

Definition 3.2. **A Data Region r:** *usually contains similar categories of data. Advertisement data region contains hyperlinks with a set of services. Main product data region contains a set of data blocks.*

Definition 3.3. Each Data Block B: *is hyperlinked (by "MORE INFO") with separate product details page and contains some*

Figure 1. Data blocks and data regions of a product list page



Samsung P2570HD 25" Wide LCD Monitor w/TV Tuner.
The Samsung P2570HD Widescreen ToC
LCD Monitor with TV Tuner brings every image and sound to life.
Only $299.99

Hannspree ST19DMSB 19" Wide LCD Displsy w/TV Tuner.
The Hannspree ST19DMSB 19" Widescreen LCD Monitor with TV Tuner brings your workplace a taste of vogue and fashion.
Only $149.99 after rebate*

eMachine E181HVb 19" Class Wide LCD Monitor.
The eMachines E181HVb 19" Class Widescreen LCD Monitor - 720p,1366x 768,60Hz,5000:1 Dynamic,1000:1Native.
Only $109.99

Viewsonic VX2233WM 22" Wide LCD Monitor.
Viewsonic VX2233WM 22" Widescreen LCD Monitor -21.5" Diagonal,1000:1 static, 18000:1 Dynamic, 1920x1080,
Only $179.99

Samsung 2333HD 23" 1080p LCD Monitor w/HDTV Tuner.
Samsung 2333HD 23" Widescreen Full HD LCD Monitor - 1920x1080, 10000:1 Dynamic,1000:1 Native,5ms, 16:9, 2xHDMI,DVI,VGA HDTV Tuner
Only $239.99

Viewsonic VX2433wm 24" Class LCD Monitor.
Viewsonic VX2433wm 24" Class Widescreen HD LCD Monitor- 1920x1080
20000:1 Dynamic,2ms GTG, HDMI,DVI, VGA,ECO Mode, Tilt stand,.
Only $209.99

key information like image of the product, product name, product number, brand or manufacturer, product price and a hyperlink for shopping the product. This information is defined as instances or objects of a distinctive type.

Data Region and Block Identification

A data region contains data blocks and is enclosed by one too many block level tags. There is no easy way to identify these data regions and blocks. In this paper, we define the following for purposes of identifying data regions and data blocks of a web page stored as a DOM tree. If T represents a DOM tree of an entire web page tags including contents, then, it contains a set of data regions. The data regions are disjoint and are sub-trees of T. We observe that a data region or data block can be within any block level mark-up tag but usually lies within tags like $< div >$, $< table >$, $< tr >$, $< span >$. This set is not complete and intersects with non-block level tags. Observation of positive page tag structure is helpful to identify the region and data block. We denote the region and data block by the set notation '{', '}.' In our case, we use $< div >$ and $< table >$ tags as region and data block tags respectively. A data region in DOM tree consists of a set of data records (defined as tuple in this paper) and all data records in a region, in general, represent similar set of data and are contiguous in a data region. In the context of web content, a data block B_i is usually a text string, image-file, price as string (of type long) representing distinctive related instance. For example, a product data record can be represented in a nested tuple as:

Product (title: string, image: image-file, difSize (product number: integer, brand: string, price:real)). This product data format is not unique and can be different in mark-up encoding and in nested formation for different data regions and web page structures of the same B2C domain. There may also be some additional noise contents like 'Add- to-Cart-' in data blocks which may need to be cleaned up. For example, the same product data can be in the following format in the web page:

{{< *image* >{}{< *title* >,{}, < *number* >, < *brand* >, <*price* >}}}

We define a data block as data tuple when a data block's nested relation is collapsed to a flat relation and any unwanted instances like separator object for nesting are removed (cleaned) inside a data block. For example, when we collapse the instances of the data block and clean-up the internal nested noise block, image block and any cascading block, the resultant data tuple is:

<< *image* >, < *title* >, < *number* >, < *brand* >, < *price* >> We used the notation '<' and '>' to denote a data tuple t. We define the data tuple as:

Definition 3.4: A tuple t *:is a domain content type dom(t) which consists of a set of distinct related instances of atomic or basic type,* $B = B_1, B_2, B_3, \ldots B_k$ *in flat mark-up encoding relation. A mark-up encoding is a pair of mark-up tags open-tag '<>' and close-tag '< / >' respectively. Mark-up encoded data instances reside in the leaf level of tree type encoding and each instance or attribute of a tuple can be encoded differently to distinguish them, or unrelated catalyst instance (e.g., decorative < img >) may be used to highlight the importance. A tuple t denoted by notation can be written as* $t = B_1, B_2, B_3, \ldots B_k$. *In the context of web content,* B_i *is usually a text string, image-file, price representing distinctive instance.*

Thus, the nested data block product given can be converted to a flat tuple with the schema:

Product < *title*: *string, image*: *image–f ile, productnumber*: *integer, brand*: *string, price*:*real* >.

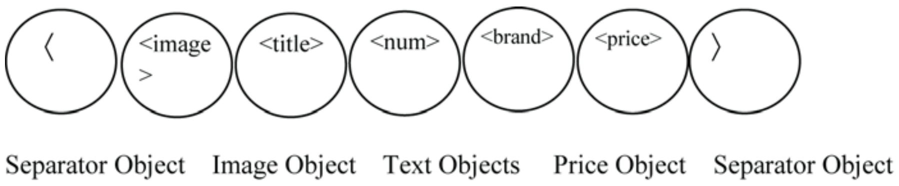
Tuple Formation from Data Block

We observe that a product list webpage contains six basic types of content data blocks, which are Product data block, List or Navigation data block, Form data block, Text data block, Decorative/Singleton data block and the Noise /Advertisement data block. We need to identify data tuples from these content data blocks. Product data block is an important data block in product list page. Related information of a typical product data block are: an image of the product, the name or title of the product, product number, brand, and price. Additional information like rebate in tagged price, brief description of the product, etc. may exist and not necessarily all pages of the domain contain all the information. These information or elements are found as either ordered or un-ordered list and in flat or nested HTML tag encoded relation. The set format of product data block in nested relation is denoted as:

{< *image* >,{< *title* >, < *number* >, . . ., < *brand* >, < *price* >}}. Some pages may contain less information like: {< *image* >, < *title* >, < *brand* >, < *price* >}.

These information need to be identified and assigned respective object (i.e., product object for product element, text object for text element, etc.). We redefine the use of separator object to identify data regions and data blocks. Therefore, in object view, placed in content objects, proposed product tuple (e.g., a flat product data block after cleaning) looks like Figure 2. We used separator element/object and classified separator element in two categories: open-separator, denoted by set notation symbol

Figure 2. Content objects of a product data block



'{' and close-separator denoted by '}' symbol and the product data can be represented as:

$\{< image > \{< title >, < specification >, < price >\}\}$.

For tuple formation of this data block, it needs to flatten/collapse and keep all object instances at the same level. So, interior set notation should be deleted and the outer set notations are replaced by the tuple notations '<' and '>.' It also needs to clean up the noise block, null block, cascading set notations within the data blocks to build it as a tuple. For tuple identification, we use a Non-deterministic Finite state Automata (NFA) based approach of pattern matching.

Extracting Contect Tuple Types Using NFA

An NFA is a finite state machine where the number of states is finite and for each pair of states (q_i) and input symbol (s_i), there may be several possible next states resulting from consuming input symbol or without consuming any input symbol but due to epsilon or null input transition (ϵ). In order to build an NFA for correctly extracting product tuples from all B2C product list pages, a comprehensive list of representative web page product schemas drawn from what can be seen as a set of positive pages for this B2C domain is needed. Thepositive sites for the domain (such as those of Future shop, Best Buy, CompUSA, etc.) are the only suggested input from the user and the system would crawl the relevant parts (e.g., product list pages) of these web sites to discover the object database schemas for the objects or entities of interest. In this case, the discovered object database schema consists of the different product schema representations for a product list page of a B2C web site. For example, the initial ten representations discovered from structures of the product list pagefrom such web sites are:

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

Thus, to extract product tuple instances from the DOM tree representation of any B2C web site, a product tuple NFA has to be built to recognize any of these schemas and any such matching tuple is found and sent to the product content object class. Similarly, and secondly, a List tuple NFA is built to extract any list tuple conforming to the $<< link >, < title >, < link >, < title >, < link >, < title >,$

< *link* >, < *title* > . . . >. The third NFA is for the Form object NFA for recognizing and extracting form tuples that may conform to the schema: << *form* >, < *text* >, < *text* >, < *text* > . . . >. The fourth NFA generated is the Text NFA for extracting Text tuples, which may contain raw texts in the web page or a bag of text describing something. The text tuple may contain a set of text objects with the following schema: << *text* >, < *text* >, < *text* >, < *text* >, < *text* > . . . >> . The segmentation of this text instance needs further research in case of problem domain that contains bulk text or text corpus. The fifth NFA generated is the Noise/Link NFA for recognizing and extracting noise tuples which are a set of hyperlinks with image and have the schema: << *link* >, < *image* >, < *link* >, < *image* > . . . >>. The sixth NFA is for identifying singleton tuples. A Singleton tuple can be anything for presentation purpose. Sometimes, some stand alone attractive images with or without links are used in web pages for better representation or to make the presentation attractive. This tuple can be represented by << *image* >> or << *link* >, < *image* >>. So, a singleton tuple may have some intersection with Noise/Link tuple. The NFAs for recognizing all these six content types are shown in Figure 3. These six NFAs are built from a set of rules that generally apply to entire identified positive pages of the domain being mined and are continuously refined with new discovery. Figure 3 shows the six automatically generated NFAs for recognizing i) product tuple, ii) list tuple, iii) form tuple, iv) text tuple, v) noise tuple and vi) singleton or separator tuple. The algorithm for generating the NFAs is given as Figure 4. An example execution of the NFA algorithm along with the WebOMiner system is given in the previous sections with solution to Example 1.

The Proposed WebOMiner Framework

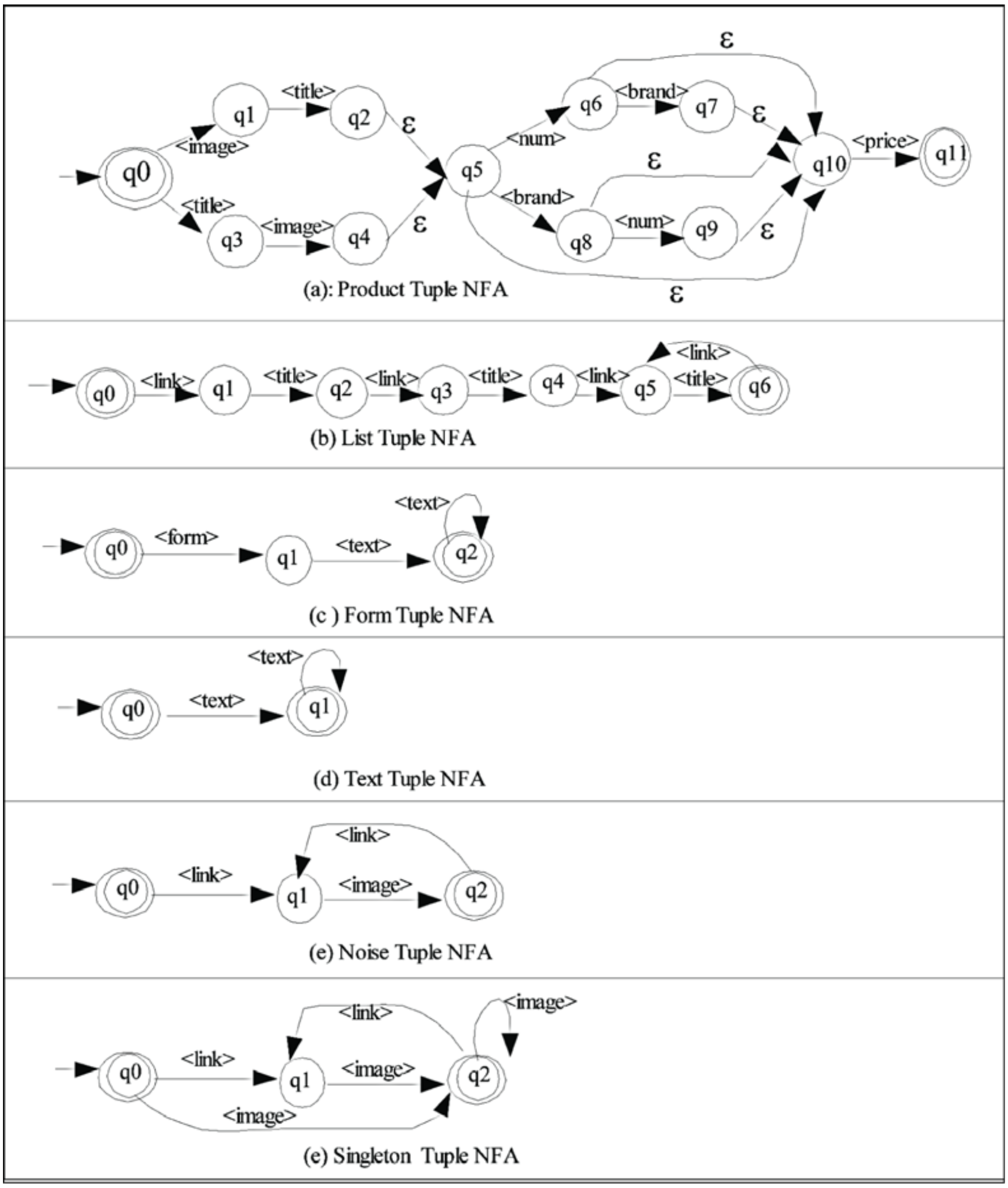
The proposed architecture for extraction and mining of web contents using object-oriented model, which is called "WebOMiner" is shown as Figure 6. The algorithm implementation of this system is given as Figure 5.

Summary of the Modules of the WebOMiner System

The WebOMiner system shown in the algorithm of Figure 5 consists of six modules that are called by the main algorithm sequentially. The modules labeled A to E form the first phase of mining, which begins with input data that is a set of domain web site http addresses or URLs (universal resource locator), and the output is the integrated object database that is mined in phase F for the second level mining. The second level mining consists of answering all types of comparative queries of objects in these web sites and can include specialization or generalization queries along object hierarchies, frequent pattern and sequential pattern mining querying, separate or combined object content querying, historical or derived object content querying. A summary of the functions of each of the modules is discussed next.

a. **The Crawler Module:** We developed a mini-crawler algorithm that crawls through the WWW to find targeted web pages given as input, streams entire web document including tags, texts and image contents and it then creates a mirror of original web document in the local computer in a directory. Our crawler module discards the comments from the HTML document. This crawler module calls the SiteMapGenerator.generate method, which takes a URL string as input and outputs HTML file in local machine and also outputs an ArrayList of Nodes having tags and contents of HTML file. Node information is then written into the output HTML file.
b. **The HTML Cleaner Module:** This module is responsible for making the HTML file from the first step well formed by inserting missing tags at appropriate locations, removing inline tags < *br/* >, < *hr/* >, inserting missing "/" at the end of un-closed < *image* > tags, cleaning up

Figure 3. NFAs for six web content types (product, list, form text, noise, singleton)



unnecessary decorative tags. The result is a refined HTML page in local directory. This module is accomplished with the web based program, HTMLCLEANER-2.2 (Sourceforge.net, 2010).

c. Content Extractor Module creates the DOM tree from HTML page and extracts the contents from the DOM tree, assigns respective class object type as per pre-defined object classes to the contents and sets information into objects and finally puts objects into ArrayList. It also identifies the data regions and data block and uses separator objects to segment the respective

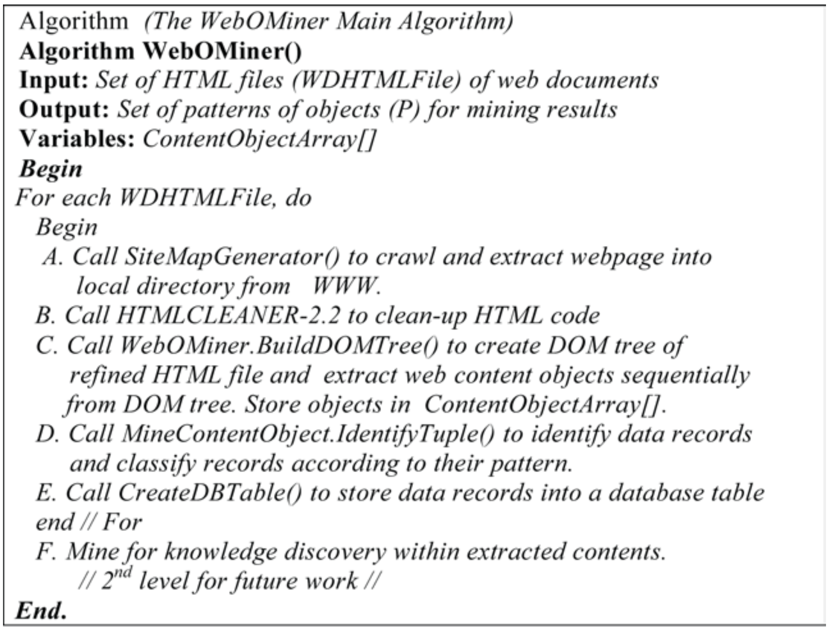
Figure 4. Algorithm GenerateSeedNFA for generating candidate NFA

```
Algorithm GenerateSeedNFA (Enum x)
Input:  Enumeration x  //Pattern Table of specific tuple type x
Output:         Seed NFA of tuple type x
Other: set of states(Q); transitions($\Sigma$); transition ($\delta$);
current state ($q_c$), starting state ($q_0$), final state (F)
Begin
 1. If Seed NFA exist
    1.1  set $q_c \leftarrow q_0$;
         else
    1.2  Initialize data structure for NFA, N= ($Q, \Sigma, \delta, q_0, F$);
         1.2.1  Set $Q \leftarrow \{q_0\}, \delta \leftarrow 0, F \leftarrow 0$;
         1.2.2  Set $q_c \leftarrow q_0$;
2. For each object, 's' in sequence of tuples
    2.1  If $\exists\ \delta(q_c , s) = q_n$ or $\exists\ \delta(q_c , \varepsilon) = q_j$ or $\exists\,\delta(q_j , s) = q_n$ in Seed NFA
         2.1.1  set $q_c \leftarrow q_n$;
    2.2 else if $\exists\,\delta(q_c , s') = q_n$ ; where $s' \neq s$  //To create $\delta$ transition
         2.2.1 Create new state $q_a$ ; $a < c$
         2.2.2 Create transition $\delta(q_a , \varepsilon') = q_c$ ;  // i.e., $\delta \leftarrow \delta\ U\{((q_a , \varepsilon'), q_c)\}$
         2.2.3 set $q_c \leftarrow q_a$ , $Q \leftarrow Q\ U\ \{q_c\}$ ;
         2.2.4 Create transition $\delta(q_c , \varepsilon') = q_j$ , $q_c \leftarrow q_j$ ;   here $c < j$
         2.2.5 Create new state $q_m$ and $q_m$ and $\delta(q_c , s) = q_m$ ;
                                              // i.e., $\delta \leftarrow \delta\ U\{((q_c , s), q_m)\}$
         2.2.6 set $q_c \leftarrow q_m$ , $Q \leftarrow Q\ U\ \{q_m\}$ ;
     2.3 else create new state
         2.3.1 Create new state $q_{c+1}$ and $q_{c+1}$ and $\delta(q_c , s) = q_{c+1}$ ;
                               // i.e., $\delta \leftarrow \delta\ U\{((q_c , s), q_{c+1})\}$
         2.3.2 set $q_c \leftarrow q_{c+1}$ , $Q \leftarrow Q\ U\ \{q_{c+1}\}$ ;
     2.4 If 's' is the last object in tuple
         2.4.1 If ($Q \cap q_{c+1} = 0$) Set $q_c \leftarrow F$;
           else
           Refine Seed NFA to create representation pattern;
 end
```

data of a data block from other data blocks. We use Java DOM package to create and parse DOM tree of the webpage. The proposed WebOMiner() algorithm line-C calls OWebMiner.BuildDOMTree() method which is given as Figure 7.

We modified the 2009 OWebMiner's ContentWebObjectScan() algorithm in order to catch body zone content objects according to their hypotheses for identifying the beginning and end of body zone of a web document and to set series-1 and series-2 pointers in the DOM tree but using tags. This is now the ContentObjectArray() function called by the DOM tree builder. The 2009 OWebMiner suggests using two tag series (a set of at least five or more <a> or <area> sibling tags to distinguish the boundaries between the header, the body and the foot zones of a web document. An <a> or <area>

Figure 5. The WebOMiner main algorithm



```
Algorithm (The WebOMiner Main Algorithm)
Algorithm WebOMiner()
Input: Set of HTML files (WDHTMLFile) of web documents
Output: Set of patterns of objects (P) for mining results
Variables: ContentObjectArray[]
Begin
For each WDHTMLFile, do
  Begin
   A. Call SiteMapGenerator() to crawl and extract webpage into
        local directory from   WWW.
   B. Call HTMLCLEANER-2.2 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
   end // For
   F. Mine for knowledge discovery within extracted contents.
        // 2nd level for future work //
End.
```

Figure 6. WebOMiner architecture for object-oriented web content mining

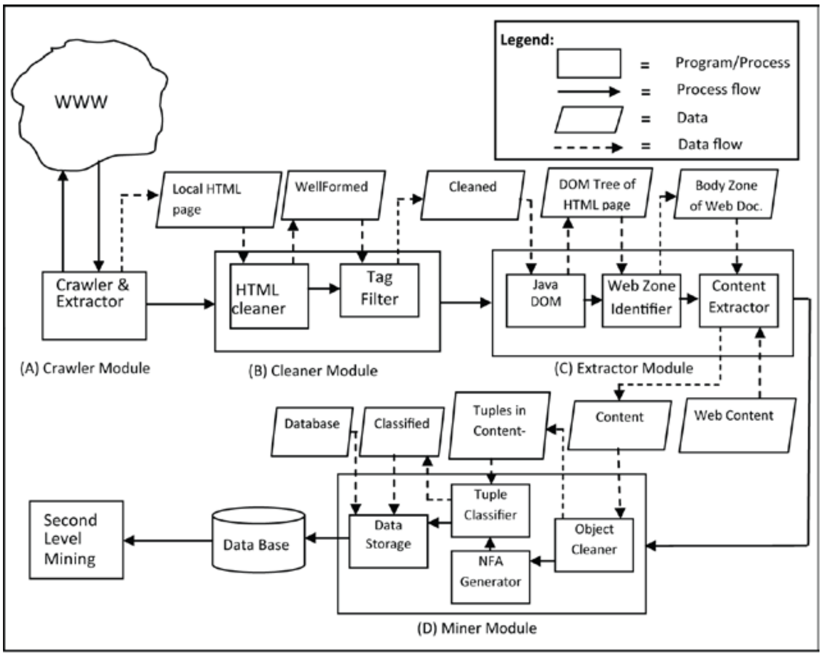


Figure 7. OWebMiner. BuildDOMTree algorithm

```
Algorithm OWebMiner.BuildDOMTree()
Input:   Refined HTML file of web documents.
Output:  Populated ContentObjectArray[].
 Begin
    1. Use Java DOM Package to create DOM Tree
    2. Call ContentObjectArray() to identify series-1 and series-2
 End
```

tag in an HTML file represents navigation URL. They observed that a set of first five or more sibling <a> tags indicate the starting of body zone and is called series-1. The last set of five of more sibling <a> tags indicate the end of the body zone and is called series-2. Moreover, they indicate that if the search for series-1 goes over half of the DOM tree size, then, the web document does not have a header zone and series-1 is empty. Also, if the search for series-2 from the first half of the DOM tree to its end returns null, the web document does not have a foot zone. The ContentObjectArray() algorithm calls ProcessContentSibling() for identifying the data regions and data blocks, which is also a modified version. Our modification of this algorithm is to reflect the identification of data regions and data blocks using separator/singleton element. The result of this module is the extracted ContentObjectArray[] of the web page DOM tree.

d. **Call MineContentObject.IdentifyTuple():** This module takes as its input ContentObjectArray[] from the module C and extracts different tuple types to place in their separate containers. This algorithm generates Seed NFA pattern for data blocks from positive pages. It extracts objects of all tuples by matching them with the refined NFAs and storing identical tuples (e.g., product record object tuples, list tuples) into appropriate TupleList. It then counts tuples and checks requirements for all tuple categories and if they satisfy the minimum requirement count for that block type, it stores the objects into the database. The MineContentObject() algorithm inputs the entire populated ContentObjectArray[] and outputs categorized content objects using separator objects and set minimum requirement counts for correctly identifying data blocks. For example, for product block, the minimum requirement count can be set to 3, meaning that data block is expected generally to have at least 3 product tuples. Tuple squeezing is also performed in this module before storing it in the database. Tuple squeezing is used to generalize tuples of the same data block so that pattern of any tuple containing varying length pattern can be represented in the same category. We use LinkedList data structure to squeeze this tuple without disturbing its data representation order.

e. **Call CreateDBTable():** This module is responsible for using the data warehouse star schema approach to integrate and store data records into a database table. It accepts as its input the generated tuple ArrayList from module D with company names and creates an integrated data warehouse fact table with a set of dimension tables. The algorithm for this module is shown as Figure 8. Once the integrated databases are built the second level mining of step F can be done at a later stage. This paper focuses on the first level mining involving steps A to E.

Figure 8. Algorithm CreateDBTable.insertData()

```
Algorithm CreateDBTable.insertData()

Input:  ArrayList, String CompanyName
Output:         Populated data tables
Begin,

        1. Register Oracle driver and create Oracle connection.
        2. PrepareStatement for different Data Tables(e.g., Product, List).
        3. Check tuple type and company name data is coming from
           and set primary key
        4. for each object in a tuple
               4.1. Check object type and retrieve data from object.
               4.2. SetString to PrepareStatement list.
               4.3. Insert data into respective data table.
               end
        end
```

APPLICATION OF THE WebOMiner SYSTEM

Example 1

Given a product list web page of a retail B2C web site like CompUSA sample with 6 products given as Figure 1, using the WebOMiner system, extract all types of information like: (i) Those related to data records such as product image, product brand, product id, short description, product price. (ii) Navigation information such as link URL, link id or name. (iii) Advertisements such as product advertised, image, URL links to related website. The extracted information will be stored in the database for future comparative mining and querying.

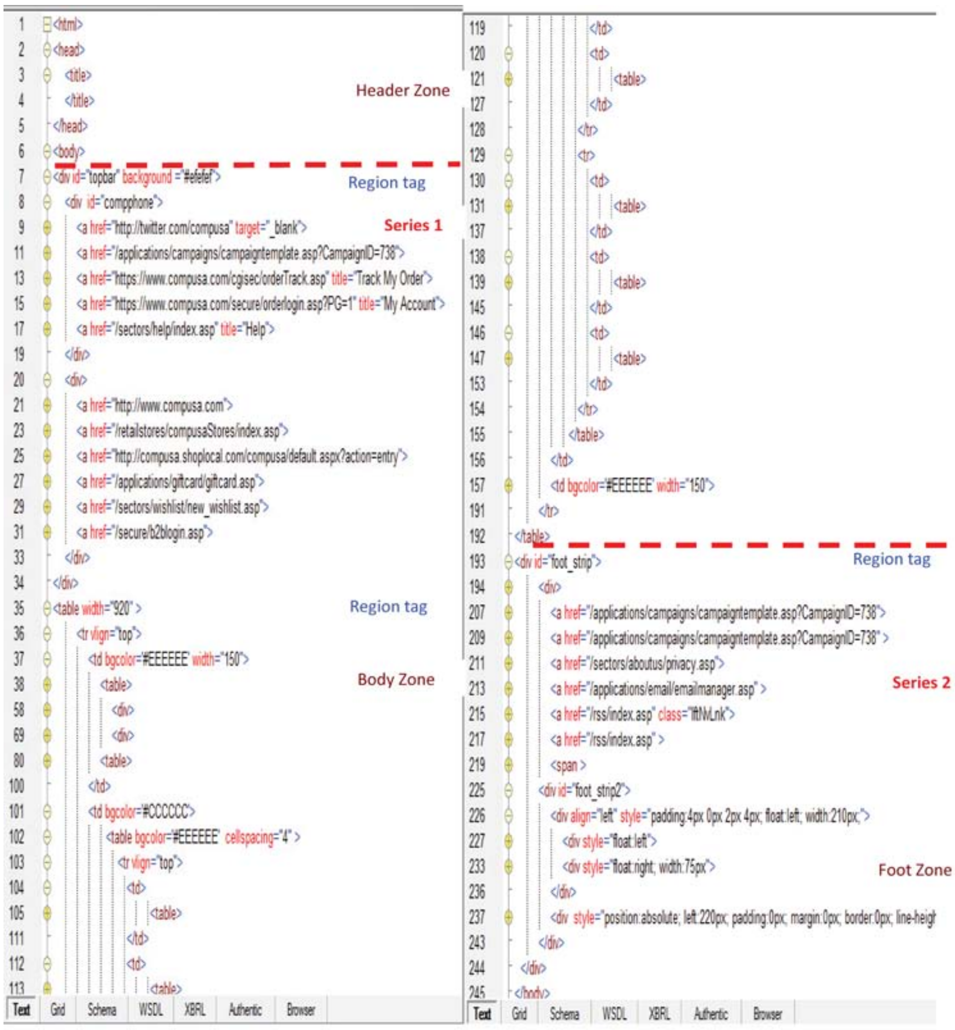
SOLUTION 1: The proposed WebOMiner system 1 implemented in JAVA runs in both Unix and Windows based environment (with NetBeans or Eclipse) to store the extracted data from the web page in a DBMS like Oracle tables. The directory where the system is run requires having the following files:WEBOMINER.jar, htmlcleaner-2.2.jar, ojdbc6.jar and the downloaded and cleaned web page to be extracted which is stored as cleanHTML#_.html. The algorithm would download the pages when provided the link and the cleaned page clean-HTML#_.html was obtained after running steps A and B on Figure 1. Note that all cleaned web pages to be extracted are currently stored in cleanHTML#_.html (e.g., CompUSA_.html). The Unix command for initiating the extraction of the cleanHTML#_.html page is given:

```
java–cp WEBOMINER.jar:htmlcleaner-2.2.jar:ojdbc6.jar webominer.Main.
```

This command has the effect of going through steps C to E of the WebOMiner algorithm using the input file cleanHTML#_.html. Step C would build the DOM tree from this clean HTML file as well as build the ContentObjectArray[] of this page. Figure 9 shows the generated DOM tree of the running example (i.e., of the web page of Figure 1).

Next, the algorithm traverses the DOM tree of Figure 9 to generate the ContentObjectArray[] as follows. This algorithm starts storing content objects into the ContentObjectArray[] until it hits the Foot zone by identifying series 2. Here, series-1 is set to TTag (current pointer at DOM tree) at line 7, which is a "<div>" tag (region node). The algorithm at some point, calls CheckTagObject() which creates an OpenSeparator object and stores it into the

Figure 9. DOM tag tree of CompUSA.com web document for Figure 1



```
1   <html>
2   <head>
3     <title>
4     </title>
5   </head>
6   <body>
7   <div id="topbar" background ="#efefef">
8     <div id="compphone">
9       <a href="http://twitter.com/compusa" target="_blank">
11      <a href="/applications/campaigns/campaigntemplate.asp?CampaignID=738">
13      <a href="https://www.compusa.com/cgisec/orderTrack.asp" title="Track My Order">
15      <a href="https://www.compusa.com/secure/orderlogin.asp?PG=1" title="My Account">
17      <a href="/sectors/help/index.asp" title="Help">
19    </div>
20    <div>
21      <a href="http://www.compusa.com">
23      <a href="/retailstores/compusaStores/index.asp">
25      <a href="http://compusa.shoplocal.com/compusa/default.aspx?action=entry">
27      <a href="/applications/giftcard/giftcard.asp">
29      <a href="/sectors/wishlist/new_wishlist.asp">
31      <a href="/secure/b2blogin.asp">
33    </div>
34  </div>
35  <table width="920" >
36      <tr vlign="top">
37        <td bgcolor='#EEEEEE' width="150">
38          <table>
58            <div>
69            <div>
80          <table>
100       </td>
101       <td bgcolor='#CCCCCC'>
102         <table bgcolor='#EEEEEE' cellspacing="4" >
103           <tr vlign="top">
104             <td>
105               <table>
111             </td>
112             <td>
113               <table>
119             </td>
120             <td>
121               <table>
127             </td>
128           </tr>
129           <tr>
130             <td>
131               <table>
137             </td>
138             <td>
139               <table>
145             </td>
146             <td>
147               <table>
153             </td>
154           </tr>
155         </table>
156       </td>
157       <td bgcolor='#EEEEEE' width="150">
191     </tr>
192 </table>
193 <div id="foot_strip">
194     <div>
207       <a href="/applications/campaigns/campaigntemplate.asp?CampaignID=738">
209       <a href="/applications/campaigns/campaigntemplate.asp?CampaignID=738" >
211       <a href="/sectors/aboutus/privacy.asp">
213       <a href="/applications/email/emailmanager.asp" >
215       <a href="/rss/index.asp" class="lftNvLnk">
217       <a href="/rss/index.asp" >
219       <span >
225     <div id="foot_strip2">
226       <div align="left" style="padding:4px 0px 2px 4px; float:left; width:210px;">
227         <div style="float:left">
233         <div style="float:right; width:75px">
236       </div>
237       <div style="position:absolute; left:220px; padding:0px; margin:0px; border:0px; line-heigt
243     </div>
244   </div>
245 </body>
```

ContentObjectArray[]. TTag is then set to the next child tag "<div>" at line 8 (data block node) and similarly stores another OpenSeparator object into ContentObjectArray[]. The TTag is again set to its child node "<a>" at line 9 and the algorithm recursively calls itself. Since it is a non-block level tag, the algorithm stores respective "<link>" followed by "<image>" objects for all five siblings (line 9 to line 17) into ContentObjectArray[]. Line 19 ends a data block and the algorithm stores a closing separator object into the ContentObjectArray[]. Similarly, the algorithm starts another data block which ends at line 33. Line 34 ends this data region. Line 35 starts with another data region that ends at line 192. Line 58 and 69 are two text data blocks "SHOP BY PRICE" and "SHOP BY BRAND" as shown in left pane of Figure 1. These embedded tags and contents are hidden in Figure 1. Similarly, lines 105, 113, 121, 131, 139, and 147 are six monitor data blocks embedded into hidden tables of the figure.

When the algorithm hits line 193, it gets series 2 pointer and returns the populated ContentObjectArray[] to the main algorithm

WebOMiner(). Two partial snapshots of this ContentObjectArray[] are given in Figure 10.

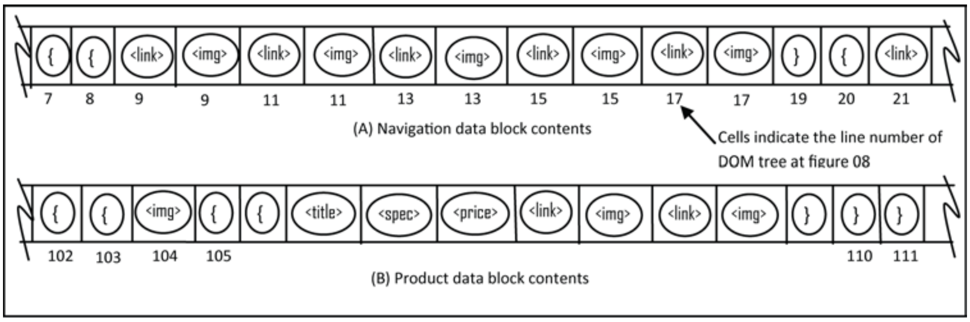
Line-D of our main WebOMiner algorithm mines populated ContentObjectArray[] for identification of data blocks and their classification to make contents ready for database entry. Line-D starts with calling our mining algorithm MineContentObjects(). It inputs populated ContentObjectArray[] and outputs a set of content patterns ready to input into database table for content integration. It scans ContentObjectArray[] for open and close separator object and identifies candidate tuples by matching key objects and minimum required count. It then refines separator objects by deleting themselves. At the same time, this algorithm generates Seed NFA pattern for data blocks.). GenerateSeedNFA() automatically generates candidate NFA by second pass iteration through all objects of an identified effective extraction of data from ContentObjectArray[] and wide range of other pages from WWW. It inputs all data structure of algorithm IdentifyTuple() as global and works with satisfied Enumeration type to create its seed NFA. The algorithm identifies the tuple type from PatternTable and looks for any existing Seed NFA for that tuple type. If it does not exist, it starts creating a new NFA by scanning objects and creating NFA state along with appropriate transition between states as per NFA generation algorithm of Figure 4. In case of our running example data tuple shown in Figure 10 of the ContentObjectArray snapshot, since the existing Seed NFA is null, the algorithm creates the starting state 'q_0' and refers 'q_c'(current state) to 'q_0' as per line 2. For <link> object at cell 9 it creates another state '$q_{1,}$' which it refers to as next to the header state '$q_{0.}$' It stores <link> object into 'q_0' and refers 'q_1' as 'q_c' according to line 2.3. This ensures a transition from 'q_0' to 'q_1' for <link> object. In the second iteration it scans <image> object of cell 9, creates state '$q_{2,}$' refer it as next to '$q_{1,}$' store object into 'q_1' and refer 'q_2' as 'q_c' per line 2.3. This process continues until the last <image> object of the tuple at cell 17. Since it is the last object, the last state is denoted by 'F' as per line 2.4.1. The next step is the refinement of the generated Seed NFA as per definition of the NFA patterns for different tuple types shown in Figure 3.

Line 2.0 of our mining algorithm MineContentObjects() uses a function SqueezeTuple() which basically squeezes the tuple length to represent their general pattern. For example our running example nevigation tuple is as follows:

$\langle$ <link>, <img>, <link>, <img>, <link>, <img>, <link>, <img>,<link>, <img>, $\rangle$

The length of this tuple is unknown with a set of repeated tags <link> and <img>. These repeated tags follow a general pattern. We can squeeze these tuples with their common pattern of a link tag followed by a title text tag as $\sum_3^n (< link >, < img >)_j$. Lines 2.1, 2.2, and 2.3 extract objects of all tuples by matching

Figure 10. Snapshot of ContentObjectArray[]


{
{
<link>
<img>
7
8
9
9
11
11
13
13
15
15
17
17
19
20
21
(A) Navigation data block contents
Cells indicate the line number of DOM tree at figure 08
<title>
<spec>
<price>
}
102
103
104
105
110
111
(B) Product data block contents

with refined NFA and storing identical tuples into TupleList. The MineContentObject() counts tuples and checks required support count for all tuple categories and if they satisfy the support, stores objects into relational database. This leads to extracting web content data into six database tables stored in Oracle Sqlplus. The six created tables are: 1. Product(product id, brand, image, prod number, price, text, company name); 2. Company(company id, company name); 3. List type (List type id, list type); 4. List (list id, link, text, company name, list type); 5. Text (text id, text, company name); 6. noise (noise id, link, image, company name). All these data have been extracted by the WebOMiner system so that when several web pages are extracted, some comparative analytical queries and mining could be performed. The WebOMiner.jar contains currently about 36 .JAVA files including Main.JAVA (for algorithm 1) and those for steps A to E of this algorithm.

FUTURE WORK ON THE CURRENT SYSTEM'S IMPLEMENTATION

This is the very first effort for mining web contents using object oriented model. There is plenty of room for improvement of our system in the future. Limitations of the current system and future work as identified are stated as:

- **Crawler Module:** Current crawler module can take one URL string at a time for extraction and mining of web contents. Further improvement is required in the future for automatic identification of positive web pages from the web. Present implementation of this module is designed aiming to work for basic functionalities as crawler with the functionality to download data stream from the targeted web pages into the local computer and cleaning of the web page comments from it. For robustness and scalability, we need to improve the current crawler module to handle all kinds of situations from the web. We are currently extending it to generate object schemas of positive B2C web pages automatically.
- **Cleaner Module:** Currently, we are using open source software "HTMLcleaner-2.2" for cleaning the web pages. Development of an independent cleaner module may improve the systems performance and usability in future.
- **Extractor Module:** One major problem with the extractor module is its inability to handle long tag attribute values. A reasonable way to handle long HTML tag attribute value is needed that are currently blasting the DOM Tree creation. We need to find out a way to reduce the tag attribute value length without loss of resources from it. A reasonable solution by finding any alternative way to create DOM from other platforms may solve the problem. Another noticeable limitation of this system is its limited capacity to handle noise contents from the web page. More research is required to handle noise from data tuples.
- **Miner Module:** We introduced the idea of using NFA for mining web contents in this paper. This NFA has two fold uses: Generation of extraction pattern for contents and generation of database schema, cardinalities to create tables and to store contents into relational database. In this paper, we generated extraction pattern but generation of database schema from the generated NFA is pending to develop. Another limitation is the use of pattern table for classification of tuples from the ContentObjectArray. Implementation of any automatic classification using co-sign or other similarity algorithm will eliminate the use of semi-automatic use of pattern table and squeezeTuple algorithm from the miner module.

EXPERIMENTS AND PERFORMANCE ANALYSIS

We created simplified mirror of six popular B2C web sites (e.g., futureshop.ca, compUSA.com, bestbuy.ca, walmart.ca, shopping.com, dell.

com as of July 2010) for empirical evaluation of our system using different page structures. Our system is implemented in Java programming language. We then ran our system in 32-bit Windows Vista Home Premium operating system at Intel Due Core 2.26 GHz, 3.00 GB RAM Sony machine for each of these mirror web sites for empirical evaluation of our WebOMiner system. The source codes for the program are available in the code directory and the document USER_MANUAL.txt provides information on how to compile and execute the system. We use the standard precision and recall measures to evaluate the web content retrieval accuracy and effectiveness of our system. Precision is measured as the 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 system is tabulated in Table 1. From Table 1, it could be seen that the WebOMiner is effective for correctly extracting detailed data from web documents since the precision for measuring accuracy of extraction (total records retrieved correctly / total records retrieved) is 176/176 or 100%. The recall is total records retrieved correctly/total records in the page, which is 176/185 or 96%.

CONCLUSION AND FUTURE WORK

We argue that there is need for a system capable of per- forming deeper knowledge discovery consisting of comparative analysis of such product features as prices, answering historical and derived queries about products and other data on web pages. We propose an approach for solving this complex data extraction that is based on the object oriented data model, which would utilize six class content types that can be linked through class inheritance hierarchies. We propose a system called WebOMiner (Web Object Miner) for this purpose, which has introduced some good features that would allow for future robustness and scalability. We have demonstrated that this first implementation phase of our system is effective for extracting web contents and storing them in database tables for querying and mining. Future improvement on the proposed system includes: the crawler module needs to create the functionality for more automatic selection of the targeted documents from the web, create automatic object database schemas of each web site as well as the object data warehouse schema of integrated web site schemas, cleaner module needs to handle long tag attributes (Chaudhuri et al., 2003). Complex attributes can be handled with content type "structure" and the NFA for the structure content type can be included for this type. The object oriented data model has been implemented with

Table 1. Accuracy of web content extraction from web pages

Website	Data Records					
	Product	List	Noise	Text	Total	Correct
www.futureshop.ca	10	13	4	0	27	27
www.compUSA.ca	18	21	8	2	49	47
www.bestbuy.ca	7	10	4	1	22	21
www.walmart.ca	2	4	2	-	8	8
www.walmart.ca	2	4	2	-	8	8
www.shopping.ca	40	4	4	-	48	47
www.dell.ca	14	13	4	-	31	28

relational database management system through nested relations and JAVA classes. Future work should also consider implementing this in an object oriented DBMS. Another DOmilestone wrapper generation system DEPTA (Zhai & Liu, 2006, 2007) builds M tree to analyze web document and uses single web page for wrapper generation like our WebOMiner system. Future extensions of the system are ongoing research.

ACKNOWLEDGMENT

We thank the students Allan Zawada for helping with testing the codes and preparing user documentation and Harun-Or-Rashid for helping with testing parts of the codes. The WebOMiner source codes with user manual and other instructions are available from the IJDWM source codes website which is at http://users.monash.edu.au/~dtaniar/IJDWM/

REFERENCES

Annoni, E., & 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* (pp. 91-100).

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). Visual web information extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases* (pp. 119-128).

Bhowmick, S. S., Madria, S. K., Ng, W. K., & Lim, E. P. (1999). Web warehousing: Design and issues. In Y. Kambayashi, D.-L. Lee, E. Lim, M. Mohania, & Y. Masunaga (Eds.), *Proceedings of the Workshops on Advances in Database Technologies* (LNCS 1552, 93-105).

Borges, J., & Levene, M. (1999). Data mining of user navigation patterns. In *Proceedings of the KDD Workshop on Web Mining*, San Diego, CA (pp. 31-36).

Bornhövd, C., & Buchmann, A. P. (1999, June). A prototype for metadata-based integration of internet sources. In M. Jarke & A. Oberweis (Eds.), *Proceedings of the 11th International Conference on Advanced Information Systems Engineering* (LNCS 1626, pp. 439-445).

Buchner, A. G., & Mulvenna, M. D. (1998). Discovering internet marketing intelligence through online analytical web usage mining. *SIGMOD Record*, *27*(4), 54–61. doi:10.1145/306101.306124

Chang, C., & Lui, S. L. (2001). IEPAD: Information extraction based on pattern discovery. In *Proceedings of the 10th International Conference on World Wide Web*, Hong Kong (pp. 681-688).

Chaudhuri, S., Ganjam, K., Ganti, V., & Motwani, R. (2003). Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, CA (pp. 313-324).

Chriisment, C., Dousset, B., Karouach, S., & Mothe, J. (2004). Information mining: Extracting, exploring and visualising geo-referenced information. In *Proceedings of the Workshop on Geographic Information Retrieval.*

Crescenzi, V., Mecca, G., & Merialdo, P. (2001, September). RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the Conference on Very Large Data Bases*, Rome, Italy (pp. 109-118).

Dung, X. T., Rahayu, W., & Taniar, D. (2007). A high performance integrated web data warehousing. *Cluster Computing*, *10*(1), 95–109. doi:10.1007/s10586-007-0008-9

Ezeife, C. I., & Liu, Y. (2009). Fast incremental mining of web sequential patterns with PLWAP tree. *International Journal of Data Mining and Knowledge Discovery Journal*, *19*(3), 376–418. doi:10.1007/s10618-009-0133-6

Ezeife, C. I., & Ohanekwu, T. E. (2005). Use of smart tokens in cleaning integrated warehouse data. *International Journal of Data Warehousing and Mining*, *1*(2), 1–22. doi:10.4018/jdwm.2005040101

Golfarelli, M., & Rizzi, S. (2011). Data warehouse testing. *International Journal of Data Warehousing and Mining*, *7*(2), 26–43. doi:10.4018/jdwm.2011040102

Gupta, S., Kaiser, G., & Stolvo, S. (2005 May 10-14). Extracting context to improve accuracy for HTML content extraction. In *Proceedings of the International World Wide Web Conference*, Japan (pp. 1114-1115).

Jupiter Media Corporation. (2007). *XML parsers: DOM and SAX put to the test.* Retrieved from http://www.devx.com/xml/Article/16922/1954

Kosala, R., & Blockeel, H. (2000). Web mining research: A survey. *ACM SIGKDD Explorations Newsletter*, *2*(1), 1–15. doi:10.1145/360402.360406

Laender, A. H. F., Neto, B. R., & da Silva, A. S. (2002). Debye-date extraction by example. *Data & Knowledge Engineering*, *40*(2), 121–154. doi:10.1016/S0169-023X(01)00047-7

Lewis, D. M., & Janeja, V. P. (2011). An empirical evaluation of similarity coefficients for binary valued data. *International Journal of Data Warehousing and Mining*, *7*(2), 44–66. doi:10.4018/jdwm.2011040103

Li, J., & Ezeife, C. I. (2006, September 4-8). Cleaning web pages for effective web content mining. In S. Bressan, J. Küng, & R. Wagner (Eds.), *Proceedings of the 17th International Conference on Databases and Expert Systems Applications*, Krakow, Poland (LNCS 4080, pp. 560-571).

Liu, B. (2007). *Web data mining: Exploring hyperlinks, contents, and usage data (Data-centric systems and applications)*. New York, NY: Springer.

Liu, B., & Chen-Chung-Chang, K. (2004). Editorial: Special issue on web content mining. *ACM SIGKDD Explorations Newsletter*, *6*(2), 1–4. doi:10.1145/1046456.1046457

Muslea, I., Minton, S., & Knoblock, C. (1999). A hierarchical approach to wrapper induction. In *Proceedings of the Third Annual Conference on Autonomous Agents* (pp. 190-197).

Priya, R. V., & Vadivel, A. (2012). User behaviour pattern mining from WebLog. *International Journal of Data Warehousing and Mining*, *8*(2), 1–22. doi:10.4018/jdwm.2012040101

Raposo, J., Pan, A., Alvarez, M., & Hidalgo, J. A., & Vina, A. (2002). The Wargo system: Semi-automatic wrapper generation in presence of complex data access modes. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications* (pp. 313-320).

Sourceforge.net. (2010). *Htmlcleaner*. Retrieved from http://htmlcleaner.sourceforge.net/download.php

Zhai, Y., & Liu, B. (2006). NET – A system for extracting web data from flat and nested data records. In A. H. H. Ngu, M. Kitsuregawa, E. J. Neuhold, J.-Y. Chung, & Q. Z. Sheng (Eds.), *Proceedings of the 6th International Conference on Web Information Systems Engineering* (LNCS 3806, pp. 487-495).

Zhai, Y., & Liu, B. (2007). Extracting web data using instance-based learning. *World Wide Web (Bussum)*, *10*(2), 113–132. doi:10.1007/s11280-007-0022-0

Zhao, H., Meng, W., Wu, Z., Raghavan, V., & Yu, C. (2005). Fully automated wrapper generation for search engines. In *Proceedings of the 14th International Conference on World Wide Web* (pp. 66-75).

Zhao, L., & Ng, W. K. (2004, May 24-27). WICCAP: From semi-structured data to structured data. In *Proceedings of the 11th IEEE International Conference and Workshop on Engineering and Computer-based Systems*. Brno, Czech Republic (pp. 86-93).

C. I. Ezeife received her MSc in Computer Science from Simon Fraser University, Canada in 1988 and a PhD in Computer Science from the University of Manitoba, Canada in 1995 following a First class BSc Honors degree in Computer Science in 1982. She has held academic positions in a number of universities including her current University of Windsor where she has been since 1996, and is a Full Professor of Computer Science since 2009. Her research interests include distributed object-oriented database systems, data warehousing and mining. She has authored several technical publications including over 15 comprehensive journal articles in journals such as ACM Computing Surveys*, Springer's* International Journal of Distributed and Parallel Databases *and* International Journal of Data Mining and Knowledge Discovery*, Elsevier's journal of* Data and Knowledge Engineering *and IGI Global's* International Journal of Data Warehousing and Mining. *She is author of two books on Problem Solving and Programs with C by Thomson Learning Publishers, which have been successfully used for teaching hundreds of first year Computer Science students for over twelve years.*

Titas Mutsuddy received his MSc in Computer Science from the University of Windsor, in Fall 2010 in the area of object oriented web content mining, under the supervision of Dr. Christie I. Ezeife. Titas previously graduated with a BSc (Honors) in Computer Science in 2005 from the University of Windsor following a BSc in Civil Engineering from Chittagong University of Engineering and Technology, Cittagong, Bangladesh in 1992. He also received a Master of Science in Civil Engineering in 2008 from Wayne State University, Detroit, USA. He has been employed in the industry working in databases and web application development for a number of years now and is currently with Stantec Consulting Ltd, Windsor.