Annals of University of Craiova, Math. Comp. Sci. Ser.
Volume 31, 2004, Pages 109–121
ISSN: 1223-6934

Classification and comparison of information structures from a web page

Mirel Cosulschi, Nicolae Constantinescu, and Mihai Gabroveanu

Abstract. This paper is closely related to the web structure analysis based on visual representation direction, a new and promising one, from which many web application such as information retrieval, information extraction can take advantage of. Page segmentation can be used to improve the query process in information retrieval, to simplify the work of automatic wrappers and to eliminate irrelevant data such as navigational bar and advertisement. Simple DOM based segmentation did not show satisfactory results, but the combination of DOM features with visual characteristics lead to a better partitioning.

2000 Mathematics Subject Classification. 68P20,68U99.
Key words and phrases. web information retrieval, visual features, association graph .

1. Introduction

The vast amount of information on World Wide Web cannot be fully exploited due to its main characteristics: web pages are designed with respect to the human readers, who interact with the systems by browsing HTML pages, rather than to be used by a computer program. The semantic content structure of web pages is the principal element exploited by many web applications: one of the latest directions is the construction of wrappers in order to structure web data using regular languages and database techniques. A subsequent problem arisen here is the necessity to divide web documents into different information chunks. The mere segmentation based on DOM tree representation [1] does not have enough power to semantically decompose a web page.

A HTML page may contain many types of information presented in different forms, such as text, image or applets (programs written in Java and executed, better said interpreted, inside a *virtual machine* - Java Virtual Machine, browser integrated). Hyper Text Markup Language [2] is a language designed for data presentation, and was not intended as a mean of structuring information and easing the process of structured data extraction. Another problem of HTML pages is related to their bad construction, language standards frequently being broken (i.e. improper closed tags, wrong nested tags, bad parameters and incorrect parameter value).

Web pages from commercial websites are usually generated dynamically using different scripts and data stored in a back-end DBMS. The visitor can easily notice the usage of a few templates in the same site, with slight differences between them. Page-generation process can be seen as the result of two activities:

- the execution of some queries targeted at the support database

Received: September 20, 2004.

- the source dataset is intertwined with HTML tags and other strings in a process that can be called *codification*. Eventually, URL links and banners or images can also be inserted .

A *wrapper* is a program whose scope is to extract data from various web sources. In order to accomplish this task the wrapper must identify data of interest and put them into some suitable formats, and eventually store back into a relational database.

The problem of generating wrapper for Web data extraction can be stated as follows. Given a web page S containing a set of input objects, determine a mapping W that populates a data repository R with the objects in S. The mapping W must also be capable of recognizing and extracting data from any other page S' *similar* to S [16].

In the past years various tools for data extraction have been proposed in the literature with the goal of efficiently solving this problem. Data published in the pages of very large sites and the higher rate of newcomer sites increased demand for semi-automatic or automatic systems. The cost of a reliable application whose maintenance requires human intervention and which will provide information with a high degree of precision increases linearly with the number of wrapped sources.

For example, RoadRunner [12] generates a schema for the data contained in many *similar* pages during an iterative process in which the current schema at step k, T_k is tested and eventually updated against a new web page S, resulting a new schema $T(k+1)$. The algorithm compares HTML tag structure of two or more pages from the same class (they are generated by the same script and are based on the same template). If the pages are not part of the same class the result will be a very general schema, which can extract data contained only in common structures, while the elements from a singular structure remain unknown because they cannot be identified. Based on the schema T a grammar is constructed capable of recognizing among other things, nested-structured data objects with a variable number of values corresponding to their attributes.

1.1. Related Work. Xiaoli Li et al [20] proposed a similar method for segmenting a web page: they introduced the notion of micro information units (MIU). A HTML page is decomposed in many MIU elements by merging adjacent paragraphs, exploiting text font property and term set similarity.

Our approach is more similar to Gu's approach [15]: they rely on breaking out the DOM tree and compare similarities among all the basic DOM nodes.

Other approaches that combine DOM structure and visual cues can be encountered in [22],[21].

The motivation of our work is to decompose a web page into fine grained and also simpler parts, elements that can be used as inputs for automatic wrappers (by example [12]). In the paper [21], the authors demonstrate that the problem of schema inference from many web pages in the presence of nullable attributes, belongs to the NP-complete class of problems. The size reduction of input data for the program that constructs the schema during an inference process thus becomes a must.

In the paper [23], the authors motivate their attempt to decompose a web page in small items called *pagelets* that simplify the page structure, by the fact that templates reduce precision, and navigation bar and paid advertisement contradict Hypertext IR Principles.

2. Method outline

Obtaining some meaningful results with the proposed method depends on using as input data some web pages with a *similar* structure (they were generated with the same script and share a common template). How can web pages with such property be obtained? The first approach implies the participation of a human operator who will classify the pages, while the second one is an automatic one: in the last years there were some attempts to cluster web pages of a web site taking into consideration structure and links. The link analysis starts with the following assumption: if there is a link between two pages then there is some relationship between the two whole pages. This assumption is more general than the reality, in most cases a link from page P_1 to page P_2 indicates that there may be some relationship between a certain part of P_1 and a certain part of P_2. Resulting classes of a specific clustering algorithm which uses page structure and/or link analysis (see [13]) it is supposed to have homogeneous elements with respect to the clustering criteria.

A web page must undergo some intermediary transformations in order to be able to apply the proposed algorithm:

- HTML->XHTML transformation (basically a cleaning-up process)
- extraction of bounding box for each element

2.1. HTML->XHTML transformation. World Wide Web consortium has warmly recommended usage of stricter standard Markup Languages, such as XHTML and XML, in order to reduce errors resulted in the process of parsing various web pages created by disobeying the basic rules. Despite this recommendation, there still remains a huge quantity of web pages that do not respect the new standards, and with whom, the parser of search engine must cope.

We transform each HTML page into a *well-formed* XHTML page. Of the applications that can be involved in this process, we enumerate two of them:

- JTidy, a Java tool based on HTML Tidy [4] that is a W3C open source software
- Neko HTMLParser [3]

These are complex programs, with a great degree of generality, trying to satisfy overall demands. During their usage we encounter situations when the result was not totally satisfactory.

We devised a simpler algorithm called *GoodDom* which performed well with respect to our necessities. We implemented it in Java as part of the whole system developed.

>From a constructed XHTML file, the DOM tree representation used in the next step of our process, can be created with no effort.

2.2. XHTML. XHTML (Extensible HyperText Markup Language) [5] represents a family of document types which extends HTML4 language. In other words, XHTML is a redefinition of HTML 4.01 standards with the help of XML. Its goal is to replace the HTML language in the future and obtain cleaner documents.

- documents must be well-formed: all elements must be nested inside on unique root element <**html**>, any element can have children elements; children elements must be correctly **closed** and **properly nested**.

```
<html>
  <head>...</head>
  <body>...</body>
</html>
```

- tag names must be in **lowercase**

```
<body>
  <p> Numele unui tag trebuie scris cu litere mici </p>
</body>
```

- empty elements must be **closed**
  `Textul va fi separat de o linie orizontala. <hr />`
  `Iar imaginea`
  `trebuie inchisa <img src="poza.gif" />`
- all XHTML elements must be **closed**
  `Wrong: <p> Un text`
  `Correct: <p> Un text </p>`
- XHTML elements must be **properly nested**
  `Wrong: <b><i> Bold, italic urmat de bold, italic</b></i>`
  `Correct:<b><i> Bold, italic urmat de italic, bold</i></b>`
- attribute names must be in **lower case**
  `Wrong: <table WIDTH="100%">`
  `Correct: <table width="100%">`
- attribute values must be **quoted**
  `Wrong: <table width=100%>`
  `Correct: <table width="100%">`
- attribute minimization is **forbidden**
  `Wrong: <frame noresize>`
  `Correct: <frame noresize="noresize" />`
- the **id** attribute replaces the **name** attribute
  `Wrong: <img src="poza.gif" name="poza" />`
  `Correct:<img`
  `src="poza.gif" id="poza" />`
- the **lang** attribute applies to almost every XHTML element. It specifies the language of the content within an element.
- the XHTML DTD defines **mandatory** elements: the 'html', 'head' and 'body' elements must be present, and the 'title' must be present inside the head element; all XHTML documents must have a DOCTYPE declaration

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Titlu </title>
 </head>
 <body> Continut </body>
</html>
```

2.3. GoodDOM Algorithm description. GoodDOM algorithm consists of the following main steps:

Algorithm 1 Algorithm GoodDOM

1: Parse HTML page obtaining base elements (tags and text strings)
2: Call Forward-Pass
3: Call Backward-Pass

The description of *Forward-Pass* step can be found here 2.

Algorithm 2 Algorithm Forward-Pass

Input: A - the vector contains token list from original HTML page
Output: B - the vector contains the modified token list

```
1: for each item from A do
2:     if (item doesn't have a closing tag OR can be empty) then
3:         add item to B
4:     else
5:         if (item is a tag) then
6:             first test some special cases
7:             if (item is open-tag) then
8:                 push item on Stack
9:             else
10:                if exists on the stack the pair of item element) then
11:                    while (Stack is not empty) do
12:                        pop from the Stack and save into tmpTab
13:                        if (tmpTag ≠ tag) then
14:                            add to B a closed-tag in correspondence to tmpTag
15:                        else
16:                            break
17:                        end if
18:                    end while
19:                end if
20:                add item to B
21:            end if
22:        end if
23:    end if
24: end for
```

After the input page was parsed and decomposed in atomic units, called tokens, the algorithm starts processing those elements using a stack for help. The stack will keep all *open tags*, tags for which their closing pair was not encountered yet. According to the rules of XHTML, a *close tag* must correspond to each element. If such a tag cannot be found on input, then it will be created (line 14).

In lines 2-3, elements which cannot have a closing tag (e.g. <style>, <script>) or can be empty (e.g. <hr>,
, <img>) are directly added to vector B. In line 06 the *special* cases are treated first: we are talking about particular chaining of tags which will be handled differently for each situation. Due to the HTML loose syntax, there is a very high number of improper tag combinations, an exhaustively handling of all these possibilities being difficult either to foresee or to implement. In our implementation we handled the majority of situations that can be encountered in common applications with a high probability and which proved to be adequate for our proposed goal.

If the element is an *open tag* then it will be pushed onto the stack (line 7) and added to the output vector B (line 20). If the current element is a *close tag*, and if his related item (*open tag*) is presented into the stack, then all elements between this item and the stack's top will be extracted (line 12). For each element extracted from the stack, it will be created a corresponding *close tag* subsequently added to the vector B (line 14).

The *Backward-Pass* step is described in Algorithm 3.

Algorithm 3 Algorithm Backward-Pass

Input: A - the vector contains token list resulted from **Forward-Pass** call
Output: B - the vector contains the modified token list

```
1: the order of elements from the vector A is reversed
2: for each item from A do
3:     if (item doesn't have a closing tag OR can be empty) then
4:         add item to B
5:     else
6:         if (item is a tag) then
7:             if (item is close-tag) then
8:                 push item on Stack
9:             else
10:                if exists on the stack the pair of item element) then
11:                    while (Stack is not empty) do
12:                        pop from the Stack and save into tmpTab
13:                        if (tmpTag ≠ tag) then
14:                            add to B an open-tag in correspondence to tmpTag
15:                        else
16:                            break
17:                        end if
18:                    end while
19:                end if
20:                add item to B
21:            end if
22:        end if
23:    end if
24: end for
25: the order of elements from vector B is reversed
```

Looking at the subroutine 3, it can easily be seen that *Backward-Pass* is very similar to *Forward-Pass*: in this situation the stack will be used to keep track of *close tags*. The element list will be run through backwards, from the last element to the first element; to keep things in an unitary way the first operation is to reverse the input vector (line 1).

The vector A is traversed element by element. Each element of vector A which does not have a closing tag or can be empty is added to the result vector B (line 3-4). If item is a *close tag* then it will be pushed onto the stack (line 8) and added to vector B. Lines 10-19 handle the case when an open-tag is encountered to whom must be associated a *close tag*. If there are other elements between stack's top and the position of *close tag*, then for each such element a correspondent *open tag* is created(line 14).

Example. Let us consider the following page:

```
<head>
 <title> First Test Page </title>
</head>
<body>
  <tag1>
    <tag2>
      <tag3>
       Inside Tag3
```

```
      </tag3>
      Outside Tag3, inside Tag2
      <tag2>
        Open second Tag2, Close Tag1
      </tag1>
      Close second Tag2
    </tag2>
    Close first Tag1
  </tag1>
</body>
</html>
```

The page tokenisation result is: <head>, <title>, 'First Test Page', </title>, </head>, <body>, <tag1>, <tag2>, <tag3>, 'Inside Tag3', </tag3>, 'Outside Tag3, inside Tag2', <tag2>, 'Open second Tag2, Close Tag1', </tag1>, 'Close second Tag2', </tag2>, 'Close first Tag1', </tag1>, </body>, </html>.

After the call of *Forward-Pass*, the output vector will contain the following elements: <html>, <head>, <title>, 'First Test Page', </title>, </head>, <body>, <tag1>, <tag2>, <tag3>, 'Inside Tag3', </tag3>, 'Outside Tag3, inside Tag2', <tag2>, 'Open second Tag2, Close Tag1', </tag2>, </tag2>, </tag1>, 'Close second Tag2', </tag2>, 'Close first Tag1', </tag1>, </body>, </html> (new tags are marked with bold).

After the call of the last processing step, *Backward-Pass*, the page will look like:

```
<html>
 <head>
  <title>
   First Test Page
  </title>
 </head>
 <body>
  <tag1> - new
   <tag2> - new
    <tag1>
     <tag2>
      <tag3>
       Inside Tag3
      </tag3>
      Outside Tag3, inside Tag2
      <tag2>
       Open second Tag2, Close Tag1
      </tag2> - new
     </tag2> - new
    </tag1>
    Close second Tag2
   </tag2>
   Close first Tag1
  </tag1>
 </body>
</html>
```

2.4. Element bounding box extraction. The second preliminary step of our algorithm consists of extracting the bounding box corresponding to each element from the DOM tree. The bounding box represents the smallest rectangle that fully covers an element of the DOM tree when it is rendered in a browser along with the whole page to which it belongs.

To efficiently discover bounding box coordinates we tested many *rendering engines* from various browsers. In order to be taken into consideration a web browser must fulfill the following conditions:

- to be able to embed it in our Java project
- to offer an access point to the internal representation of a web page
- to have a minimal documentation

We lead our experiments with the following Java web browsers:

- *Grand-Rapid Browser* (http://www.meyou.com/grandrapid/) provides excellent support for the general web and comes with a variety of customization features, but it is like a black-box for a programmer who likes to use it inside an application.
- *NetClue* (http://www.netcluesoft.com/desktop/) renders standard web sites almost identically to ots elder brother Netscape and IE. The programmers have access to virtually every event and to directly manipulate DOM content.
- *WebRenderer* (http://www.webrenderer.com/) is a wrapper library which has support for the most encountered operating systems. The wrapper gives access to all internal events.
- *WebWindow* (http://www.javio.com/webwindow/) is a web browser developed entirely in Java, with two versions, one for Swing and one for AWT. The renderer is quite fast, presenting nice features for zooming text and images.
- *IceBrowser* (http://www.icesoft.com/products/icebrowser.html) is one of the oldest Java web browsers which reached its maturity. We can say that its performances are excellent from the hackability's point of view.
- *JRex* (http://jrex.mozdev.org) acts like a wrapper for Mozilla, using the full power of its libraries. Jrex has APIs to receive events and access DOM and supports XUL. It can be used as an embedded browser into an application.
- *Flying Saucer* (https://xhtmlrenderer.dev.java.net/) is the newest member of this 'family', being still under development.

The coordinates of a bounding box are obtained from a software module specially written to catch *events* generated by *rendering engine*'s components.

To each element to whom it corresponds a visual representation (inside a HTML page there could be elements without any visual representation, for example comments), we add 4 new attributes. For example, the next tag

```
<IMG    height="40" width="285" border="0"
    src="car08_file/ab41.gif" alt="Yahoo! Autos"/>
```

becomes

```
<IMG    dy="44" dx="285" ly="14" lx="127"
    height="40" width="285" border="0"
    src="car08_file/ab41.gif" alt="Yahoo! Autos"/>
```

Attributes dx, dy, lx, ly have the following semantics:

- lx and ly represent the coordinates of the upper-left corner of the bounding rectangle

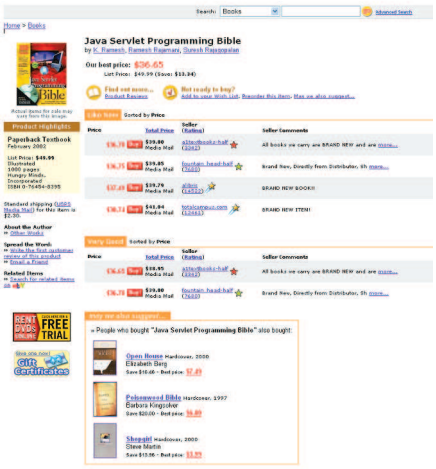

FIGURE 1

- dx and dy represent the *width* and *height* of the bounding rectangle

In the figure 1 the reader can see the result of rendering a HTML web page inside a web browser (IceBrowser).

3. Spatial relationships

A page description can be made using the information related to its content and to objects positions (the coordinates of associated bounding box). This assumption leads us to the fact that the rebuilding of a web page can be made of two kinds of features:

- geometric features
- content-related features

Geometric features of an object can be stored in a vector, thus every object can be assimilated to a point in a multi-dimensional space, each dimension corresponding to a feature. Among geometric parameters which define an object aiming to describe a whole cathegory of diverse documents, we can process the ratio of bounding rectangles length and width, the ratio of rectangle area and page area, the style and font size of the paragraph text, the length of the text.

Besides previous descriptive elements which characterize the document from a *static* point of view, another type of geometric relations is necessary to capture *dynamic* interactions related to absolute objects positioning inside a web page and also to their relative positioning.

Having as a starting point the interval relationships [6] defined in the context of temporal intervals, the paper [19] describes an extension for bidimensional space.

When considering the OX axis and two intervals $u = (x_1^u, x_2^u)$ and $v = (x_1^v, x_2^v)$, we have the following seven situations:

Definition 3.1
- u ***precedes*** v: $x_2^u < x_1^v$
- u ***meets*** v: $x_2^u = x_1^v$
- u ***overlay*** v: $x_1^u < x_1^v$ *si* $x_1^v < x_2^u < x_2^v$

- u ***start*** v: $x_1^u = x_1^v$ *si* $x_2^u < x_2^v$
- u ***during*** v: $x_1^u > x_1^v$ *si* $x_2^u < x_2^v$
- u ***finishes*** v: $x_1^u > x_1^v$ *si* $x_2^u = x_2^v$
- u ***equals*** v: $x_1^u = x_1^v$ *si* $x_2^u = x_2^v$

4. Object correspondence

The main part of the presented method consists of constructing a pairs list. A pair is composed of two objects, between which there is a correspondence (relation), each object representing a node in the DOM tree created as a result of parsing a XHTML file.

Problem description. Let us suppose we have two DOM trees, TreeA and TreeB, associated with two XHTML pages, pageA and pageB. The goal is to determine a list with elements in this format $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ where $a_1, a_2, \ldots, a_n \in TreeA$, $b_1, b_2, \ldots, b_n \in TreeB$, and each pair (a_i, b_i) must fulfill certain conditions.

4.1. Association graph.

Definition 4.1. *In the following, let A and B be two finite, nonempty sets, having cardinalities n and m respectively. We will call the following structure an* ***association graph*** *$G = (V, E)$, where:*

- *V is the set of vertices, $V = A \times B, V = \{(x, y)|x \in A, y \in B\}$*
- *E is the set of edges, $E \subseteq V \times V, E = \{(u, v)|u, v \in V\}$*

G is an undirected graph. The edge $e = (u, v) \in E$ if and only if the vertices u and v are **compatible**. Two vertices are said to be **compatible** if their association respects some conditions which can differ from a certain problem to another.

$|V| = n\text{x}m$ and $|E| \leq n^2\text{x}m^2$

After creating the **association graph** the initial problem, related to constructing a correspondence mapping, becomes the problem of determining an optimal mapping between two well structured models. It is worth noting that optimality means here maximality. The solution for our problem is represented by **maximal clique**.

The problem of determining a **maximal clique** is not a new one, this subject being intensely studied in the past two decades. It is not an easy problem, and it belongs to the class of NP-hard problems from the complexity point of view, a class with other famous representatives: the problem of minimum cover set, the problem of maximal independent set.

Many approaches for solving this problem have been proposed, every method attempting to introduce new conditions and optimizations to the huge amount of computations.

The process of finding **maximal clique** is modeled by one of the newest optimized algorithms [18].

5. Algorithm outline

MatchViews procedure (Algorithm 4) has as input variables two sets A and B composed by *visual* elements (View type) and returns a subset of cartesian product $A \times B$.

For "NodeA and NodeB respect visual conditions" we will employ Allen's relations [6] or an extension to Allen's interval relations called *Thick Boundary Rectangle Relations - TBRR* [19].

Algorithm 4 Algorithm MatchViews

```
Input:  firstSet - an array formed by View references, representing
        the first set of elements
        secondSet - the second set of views
Output: a PairView array
Local:  graph - a Graph object
```

1: **for** (each element *ViewA* in *firstSet* of Views) **do**
2: **for** (each element *ViewB* in *secondSet* of Views) **do**
3: **if** (PairConditions for *ViewA* and *ViewB* is satisfied) **then**
4: graph.addGraphNode(new GraphNode(new PairView(ViewA, ViewB)))
5: **end if**
6: **end for**
7: **end for**

▷ adds edges

8: **for** (each element *NodeA* belonging to the graph nodes set **do**
9: **for** (each element *NodeB* belonging to the graph nodes set) **do**
10: **if** (*NodeA* and *NodeB* respect visual conditions) **then**
11: graph.addEdge(NodeA, NodeB)
12: **end if**
13: **end for**
14: **end for**
15: **return** graph.getMaxClique()

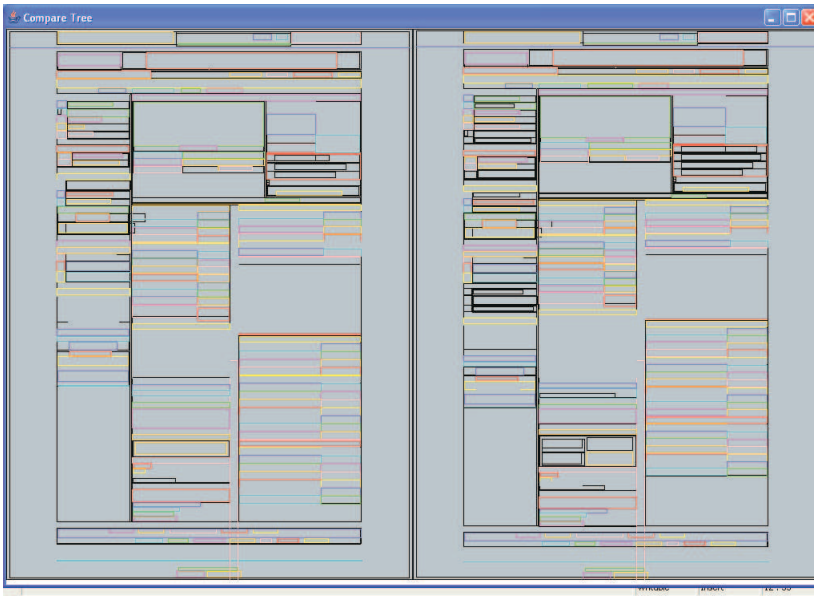

FIGURE 2

The results of a working prototype of algorithm **Match** applied to two similar pages with the one presented in figure 1 are captured in Figure 2.

The proposed method can be applied in different ways:

(1) The algorithm can be applied only at the level of *leaves*, e.g. only to the terminal vertices from the two trees.
The number of *leaves* vertices from a DOM tree associated with an average HTML page, can exceed 500, and thus the number of nodes in the **association graph** can exceed $500x500 = 250.000$.
(2) In the first phase, there are computed the pairs of corresponding nodes by inspecting those at depth 1, after that the nodes at depth 2, and so on.
The motivation for this approach is: the corresponding leaves are embedded inside more complex structures, so called *containers*, which, in their turn, must be able to be correspondent.

6. Conclusions

In this article, the principal problems encountered during segmentation of two web pages and their blocks correspondence were presented. Various solutions to overcome the situations were presented and analysed in detail. Our future work will mainly consists in investigating other heuristic conditions that can improve the selection process. For web pages that have many elements at the same level of the DOM tree the time of response can become unfeasible, running time of algorithm for finding maximal clique heavily depending on the size of inputs. So the main concern is the size reduction on the number of elements for candidate sets.

7. Acknowledgements

The first author would like to express his gratitude to Professor Paolo Merialdo and Valter Crescenzi from University Roma Tre, for their kindly support and many interesting discussion about the subject.

References

[1] Document Object Model (DOM) Level 1 specification. W3C Recommendation, October 1998. http://www.w3.org/TR/REC-DOM-level-1.
[2] HTML 4.01, http://www.w3.org/TR/
[3] Neko HTML Parser, http://www.apache.org/ andyc/neko/doc/html/index.html
[4] W3C, *HTML Tidy*, http://www.w3.org/People/Raggett/tidy
[5] *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*, http://www.w3.org/TR/2002/REC-xhtml1-20020801
[6] James F. Allen, *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM, 26:832-843, 1983.
[7] A. Arasu, H. Garcia-Molina, *Extracting structured data from web pages*, ACM SIGMOD *2003*, 2003.
[8] D. Cai, S. Yu, J.-R. Wen, M.-Y. Ma, *Extracting content structure for web pages based on visual representation*, 5th Asia Pacific Web Conference, Xi'an China, 2003.
[9] D. Cai, S. Yu, J.-R. Wen, M.-Y. Ma, *VIPS-a vision based page segmentation algorithm*, Microsoft Technical Report, MSR-TR-2003-79, 2003.
[10] Chia-Hui Chang,*IEPAD: Information extraction based on pattern discovery*, in Proceedings of the tenth international conference on World Wide Web, 2001.
[11] M. Cosulschi, M. Gabroveanu, *Information retrieval from web pages*, in Proceedings of 4th National Conference on Artificial Intelligence and Digital Communications, Craiova, 2004.
[12] V. Crescenzi, G. Mecca, P. Merialdo, *RoadRunner: Automatic Data Extraction from Data-Intensive Web Sites*, International Conference on Management of Data and Symposium on Principles of Database Systems (SIGMOD02), 2002.

[13] V. Crescenzi, G. Mecca, P. Merialdo, *Wrapping-Oriented Classification of Web Pages*, Symposium on Applied Computing (SAC02), 2002.
[14] V. Crescenzi, P. Merialdo, P. Missier, *Fine-grain web site structure discovery*, Fifth Workshop on Web information and data management, ACM Press, 2003.
[15] X. Gu, J. Chen, W.-Y. Ma, G. Chen, *Visual Based Content Understanding towards Web Adaptation*, in Second International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH2002), Spain, 2002.
[16] A. Laender, B. Ribeiro-Neto, A. Da Silva, J. Texeira, *A Brief Survey of Web Data Extraction Tools*, ACM SIGMOD Record, 31(2), June 2002.
[17] Z. Liu, Wee Keong Ng, Ee-Peng Lim, *An automated algorithm for extracting website skeleton*, DASFAA 2004, 2004.
[18] Patric J. Ostergard, A New Algorithm for The Maximum-weight Clique Problem, Nordic Journal of Computing, 2001.
[19] L. Todoran, M. Worring, M. Aiello, C. Monz, *Document Understanding for a Broad Class of Documents*, ISIS TR Series, vol. 2001-15, oct 2001.
[20] Xiaoli Li, Bing Liu, et al, *Using Micro Information Units for Internet Search*, ACM Special Interest Group on Knowledge Discovery in Data (SIGKDD02), 2002.
[21] G. Yang, I. V. Ramakrishnan, and M. Kifer, *On the complexity of schema inference fromWeb pages in the presence of nullalbe data attributes*, in ACM Internation Conference on Information and Knowledge Management (CIKM), 2003.
[22] Shipeng Yu, D. Cai, J.-R. Wen, W.-Y. MA, Extracting COntent Structure for Web Pages based on Visual Representation, in The Fifth Asia Pacific Web Conference (APWeb2003), 2003.
[23] Ziv bar-Yossef, Sridhar Rajagopalan, *Template Detection via Data Mining and its Applications*, World Wide Web Conference (WWW02), 2002.
[24] J. Wang, F.H. Lochovsky, *Data-rich section extraction from html pages*, 3rd International Conference on Web Information Systems Engineering (WISE 2002), Singapore, December 2002, Proceedings, IEEE Computer Society.

(Mirel Cosulschi) University of Craiova
Faculty of Mathematics-Informatics
Department of Informatics
E-mail address: mirelc@central.ucv.ro

(Nicolae Constantinescu) University of Craiova
Faculty of Mathematics-Informatics
Department of Informatics
E-mail address: nikyc@central.ucv.ro

(Mihai Gabroveanu) University of Craiova
Faculty of Mathematics-Informatics
Department of Informatics
E-mail address: mihaiug@central.ucv.ro