Webpage Menu Detection Based on DOM *

Julian Alarte, David Insa, and Josep Silva

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n
E-46022 Valencia, Spain
{jalarte,dinsa,jsilva}@dsic.upv.es

Abstract. One of the key elements of a website are Web menus, which provide fundamental information about the topology of the own website. Menu detection is useful for humans, but also for crawlers and indexers because the menu provides essential information about the structure and contents of a website. For humans, identifying the main menu of a website is a relatively easy task. However, for computer tools identifying the menu is not trivial at all and, in fact, it is still a challenging unsolved problem. In this work, we propose a novel method for automatic Web menu detection that works at the level of DOM.

Keywords: Information Retrieval, Web Template Detection, Menu Detection

1 Introduction

A webpage menu (in the following just menu) is a fundamental component in a website whose main objective is providing navigation among the main webpages that form the website. In this paper, we focus on HTML-structured webpages (ignoring those webpages built with alternative technologies such as Flash, or those whose structure is constructed by means of JavaScript). From an engineering perspective, a webpage is a set of Document Object Model (DOM) nodes. Thus, a menu is a subset of those nodes, and it provides essential information about the structure of a website, including its main sections, and implicit information about the sitemap of the website.

Our approach to menu detection is based on the DOM [4] structures that represent webpages. Roughly, given a webpage of a website, (1) we first compute

* This work has been partially supported by the EU (FEDER) and the Spanish *Ministerio de Economía y Competitividad* under grant TIN2013-44742-C4-1-R and TIN2016-76843-C4-1-R, and by the *Generalitat Valenciana* under grant PROMETEO-II/2015/013 (SmartLogic). Sergio Pérez was partially supported by the Spanish *Iniciativa de Empleo Juvenil* Programme (MINECO) and the European Social Fund in collaboration with the *Sistema Nacional de Garantía Juvenil* under the grant PEJ-2014-A-24709 (*Promoción de Empleo Joven e Implantación de la Garantía Juvenil 2014*, MINECO).

the relevance (a weighting) of each DOM node in the webpage, and then, (2) we use the relevance to identify those DOM nodes that are more likely to be part of the menu. (3) Finally, we analyse recursively the parents of those nodes to infer the complete menu. In practice, we input a webpage and we output the set of DOM nodes that correspond to the main menu.

2 Related Work

Menu extraction is a topic directly related to *template extraction*. Template detection and extraction are key topics due to their direct application to Web mining, searching, indexing, and Web development. *Content Extraction* is a discipline very close to template extraction. Content extraction tries to isolate the pagelet that contains the main content of the webpage. It is an instance of a more general discipline called *Block Detection* that tries to isolate every pagelet in a webpage. There are many works in these fields (see, e.g., [5, 12, 3, 6]), and all of them are directly related to menu extraction.

Menu extraction techniques are often classified as page-level or site-level. In both cases, the objective is the same, detecting the menus of a given webpage; but they use different information. While page-level techniques only use the information contained in the target webpage, site-level techniques also use the information contained in other webpages, typically of the same website.

In the areas of menu extraction and template extraction, there are three main different ways to solve the problem: (i) using the textual information of the webpage (the HTML code) [12, 9, 7], (ii) using the rendered image of the webpage [2, 8], and (iii) using the DOM tree of the webpage [1, 13, 11].

A webpage menu is a pagelet [1]. Pagelets were defined as a region of a webpage that (1) has a single well-defined topic or functionality; and (2) is not nested within another region that has exactly the same topic or functionality. One of the first template extraction approaches [1] proposed two algorithms that rely on the identification of identical pagelets occurring in a densely linked page collection.

Some works try to identify template pagelets analyzing the DOM tree with heuristics [1]. However, others try to find common subtrees in the DOM trees obtained from a collection of webpages of the website [13, 11]. None of these methods tries only to isolate the menu pagelet, but they also try to find the whole template of the webpage. The main goal of [10] is not template extraction, but it is a webpage segmentation method based on detecting the layout of the webpage. Detecting the layout may help to detect the menu because the webpage is divided into functional blocks (i.e., header, footer, sidebar, main content, advertisements, images, etc.).

3 Menu detection

Our technique inputs a webpage and outputs its main menu. It is a page-level technique, thus, it only needs to analyze the information contained in the target

webpage. Because we work at the level of DOM, and due to the DOM tree properties, a menu can be represented with just one DOM node: the node whose subtree contains the menu. In our approach, we first identify a set of DOM nodes that are more likely to be the menu and then we analyze them to select the best candidate. Our approach is divided into three phases:

1. The algorithm visits some DOM nodes in the webpage and, for each node excluding the leaves, it computes and assigns the node a weight. Then, it selects a set of DOM nodes with the higher weight. The webpage menu is a node of this set or an ancestor of it.
2. For each node in the set, we check its ancestors and we evaluate its weight. If the weight of an ancestor is higher than a computed value, we replace the node in the set obtained in phase 1 with that ancestor.
3. The menu node (the node that represents the menu) is extracted by comparing the set of selected DOM nodes. Recall that some of the nodes may be ancestors of the original DOM nodes selected in phase 1. For each node in the set, we analyse the weight of its descendants. The one with the best weight is the menu node.

These three phases are explained in the following sections.

3.1 Rating DOM nodes

This section proposes a metric applied to DOM nodes that helps to identify a set of nodes that probably belong to the menu. Roughly, we explore the DOM tree of the webpage and we assign a weight to each DOM node that meets the following criteria: (1) It is not a leaf of the DOM tree. (2) It is an element node. Any other type (e.g., text nodes, comments, etc.) are not considered.

In order to provide a definition of menu, we first need to state a formal definition of webpage, website, hyperlink and node's hyperlinks.

Definition 1 (Webpage). *A webpage P is a pair (N, E) with a finite set of nodes N. Every node contains either an HTML tag (including its attributes) or text. The root node is the node corresponding to the* `body` *tag. E is a finite set of edges such that $(n \rightarrow n') \in E$, with $n, n' \in N$ if and only if the tag or text associated with n' is inside the tag associated with n, and not exists an unclosed tag between them. E^* represents the reflexive and transitive closure of E.*

Given a node n in a webpage, we often use $descendants(n)$ to refer to those nodes that belong to the subtree of n. We use $characters(n)$ to refer to the total number of characters in $descendants(n)$ excluding those characters that belong to hyperlinks. And we use $target(n)$ to refer to the webpage pointed by the hyperlink of node n.

Definition 2 (Hyperlink). *An hyperlink is a non-empty sequence of words joined by juxtaposition $w_1 w_2 w_3 ... w_n$, where each word finishes with a slash: $h = (dir/)^+$.*

Definition 3 (Node's hyperlinks). *Given a webpage $P = (N, E)$ and a node $n \in N$, $hyperlinks(n)$ is the set containing all the hyperlink nodes in $descendants(n)$. $hyperlinks(P)$ is the set containing all the hyperlink nodes in N.*

A website is composed of webpages such that all of them are reachable from the main webpage. Hence, all of them (possibly except the main webpage) are pointed by at least one hyperlink in another webpage of the website (i.e., all webpages have an indegree greater than 0).

Definition 4 (Website). *A website S is a set of webpages such that $\exists\ P \in S\ :\ \forall\ P' \in S, P \neq P'\ :\ (P, P') \in hyperlinks(S)^*$, where $hyperlinks(S) = \cup_{Q \in S}\ hyperlinks(Q)$.*

We can now define webpage menu. Roughly, a menu is a DOM node whose subtree is the smallest subtree that contains at least two hyperlinks pointing to webpages on the same website and, moreover, because a menu provides navigation to the website, the same menu must appear in at least another webpage of the website. Formally,

Definition 5 (Webpage menu). *Given a website S, and a webpage $P = (N, E) \in S$, a webpage menu of P is a node $n \in N$ such that*

- $\exists\ n', n'' \in N \mid (n, n'), (n, n'') \in E^+ \wedge n', n'' \in hyperlinks(P) \wedge target(n'), target(n'') \in S$, *and*
- $\nexists\ m \in N \mid (n, m), (m, n'), (m, n'') \in E^+$, *and*
- $\exists\ P' = (N', E') \in S \mid n \in N'$.

Clearly, the webpage menu cannot be a leaf node because it must contain nodes with hyperlinks to different pages of the website. On the other hand, the `element` node is the only kind of DOM node that can contain enumerations of hyperlinks. Thus, a webpage menu must be an internal DOM node of type `element`.

One of the main objectives of a template is to provide navigation to the webpage, thus almost all menus provide a large number of hyperlinks, shared by all webpages implementing the template. Hence, to locate menus we identify those DOM nodes with a high concentration of hyperlinks among their descendants. These nodes very likely belong to the webpage menu.

However, a high hyperlink density is only one of the properties of menus but, often, it is not enough to identify them. We now propose several other properties that must be taken into account to properly detect menus. All these properties are objectively quantifiable and, appropriately combined, they form a weighting that can be used to uniquely identify menus.

Definition 6 (Node properties). *In a webpage $P = (N, E)$, every node $n \in N \wedge descendants(n) \neq \varnothing$ is rated according to the following properties:*

Node amplitude: *The amplitude of a node n is computed considering its number of children: $children(n) = |\{n' \in N \mid (n, n') \in E\}|$. It is defined as:*

$$Node\ amplitude(n) = 1 - (1/children(n))$$

Link ratio: *The link ratio of $n \in N$ is computed with the following function:*

$$Link\ ratio(n) = \begin{cases} 0 & \text{if } |hyperlinks(n)| < 2 \\ |hyperlinks(n)| + descendants(n)/2 * descendants(n) & \text{if } |hyperlinks(n)| \geq 2 \end{cases}$$

Text ratio: *It is computed considering the amount of characters of a DOM node and its descendants:*

$$Text\ ratio(n) = 1 - (characters(n)/\sqrt{characters(P)})$$

UL ratio: *It checks whether the HTML tagName of the node is "ul" or not.*

$$UL\ ratio(n) = \begin{cases} 0 \text{ if } n.tagName \neq \text{“}ul\text{”} \\ 1 \text{ if } n.tagName = \text{“}ul\text{”} \end{cases}$$

Representative tag: *It evaluates some attributes of the node:*

$$UL\ tag(n) = \begin{cases} 1 \text{ if } n.tagName = \text{“}nav\text{”} \\ 1 \text{ if } n.className = \text{“}menu\text{”} \\ 1 \text{ if } n.className = \text{“}nav\text{”} \\ 1 \text{ if } n.id = \text{“}menu\text{”} \\ 1 \text{ if } n.id = \text{“}nav\text{”} \\ 0 \text{ otherwise} \end{cases}$$

Node position: *The position of n in the webpage P is evaluated using the function:*

$$Node\_position(n) = 1 - (position(n)/|N|)$$

where function $position(n)$ is the position of node n in P, if all nodes are sorted with a depth first traversal.

The *Node amplitude* property takes into account the amount of children of a DOM node. The more children a node has, the higher is the probability that the node belongs to the menu. Usually, the menu nodes have a large amount of children that can be either 'link' nodes or 'element' nodes whose descendants contain 'link' nodes. We defined a function that promotes the nodes with more children and penalizes the nodes with less children. A node with a high amount of children will have a *node amplitude* value close to 1. However, a node without or with few children will have a *node amplitude* value close to 0.

The *Link ratio* property counts the amount of hyperlinks a DOM node and its descendants have. The more hyperlinks, the higher is the link ratio. We define a metric that examines all the descendants of a DOM node and counts the number of hyperlink nodes.

The *Text ratio* property evaluates the amount of text the descendants of a DOM node have in comparison to the total amount of text in the webpage. Usually, menu nodes do not contain text between their descendants except for the text of the hyperlinks. Therefore, we do not consider the text of the hyperlinks

when counting the amount of text. The text ratio metric penalizes the nodes with more text in its descendants.

The *UL ratio* property is used to promote those nodes that use the *UL*[1] HTML tag because webpage menus are usually lists of links constructed with this HTML tag. In particular, we observed than more than 60% of the websites in our benchmarks use the *UL* tag for the node containing the menu.

The *Representative tag* property is used to promote the use of other particular HTML tags. We also observed in our benchmarks some other attributes that are frequently used in the webpage menu nodes. We have not considered them together with the *UL tag* because they are not as frequent. These HTML attributes are:

- *Nav* tag: HTML5 defines the *nav* tag to represent a set of navigation links.[2]
- Node's *id*: Some nodes representing the menu have the *nav* or *menu* identifier. For instance, *id="nav"* or *id="menu"*.
- Node's *classname*: Some webpages represent the menu with a node whose classname contains the *menu* or *nav* classes.

The *Node position* property is used to consider the fact that menus are usually located at the top or in the top left corner of the webpage. This means that the node containing the menu should be at the beginning of the DOM tree. We established a ponderation where the first nodes of the DOM tree get higher values than the last ones.

With these properties we can assign each node in the DOM tree a weight that represents the probability that this node is the main menu of the webpage. The exact ponderation used to combine all these node properties must be determined with empirical evaluation. Thus we conducted experiments to determine the best ponderation of these metrics. This is discussed in detail in Section 4.1.

3.2 Selection of candidates

Once we have calculated the weight of all the nodes in the webpage, we select the heavier nodes. These nodes are considered as candidates to be elected as the main menu. This process is simple: an algorithm visits all the nodes in the DOM tree, checks their weights, and selects the ones with higher weights. Only the nodes with a weight over a specified threshold are selected. This threshold has been calculated based on experimentation and its value is 0.85.

3.3 Selection of root nodes

The selection of candidates only considers individual information based on the properties of the nodes. The candidates are nodes with a high concentration of

[1] HTML Unordered List.

[2] We consider the *nav* tag because it is the specific tag (and recommendation) in HTML5 for representing menus. However, note that it can be changed if we want to focus on other technologies.

Algorithm 1 Selection of the root node

Input: A DOM node n and a threshold t.
Output: A DOM node *menuNode* representing a candidate to be the whole menu.

```
begin
  menuNode = n;
  currentNode = n;
  baseWeight = n.weight;
  found = false;
  while (∃ currentNode.parentNode ∧ found == false)
     parent = currentNode.parentNode
    nodeCount = |{node | node ∈ parent.children ∧ node.weight > t*baseWeight}|;
     if (2 * nodeCount > |parent.children|)
          currentNode = parent;
          if (parent.children > 1)
              menuNode = parent;
     else found = true
  return menuNode;
end
```

links, with little or no text, etc. but they are not necessarily a menu. In fact, they are very often a part of the menu. For instance, in a menu that contains a submenu "Products" with a high density of hyperlinks, the DOM node representing the menu option "Products" is probably a candidate. Nevertheless, the DOM node representing the complete menu could not be selected as a candidate. The real menu is usually an ancestor of a candidate. It usually combines two or more candidates and possibly other nodes such as, e.g., images. This phenomenon usually happens in complex or large menus. Moreover, in menus with a set of submenus, the selection of candidates process usually detects only one of these submenus.

Algorithm 1 explores the ancestors of a candidate node to find the node that really represents the whole menu. The algorithm recursively explores each of the ancestor nodes of the candidate and checks whether more than half of their children have a *weight* higher than a given threshold t, called *root threshold*. When it finds a node for which this criterion does not hold, the algorithm stops and selects the last ancestor as the menu node.

Example 1. Consider the DOM tree in Figure 1. The "UL" node with a dotted border is one of the candidates because it has a *weight* higher than 0.85. Nevertheless, this node represents only a portion of the main menu, so we have to analyze its ancestors in order to locate the root node of the main menu.

By using Algorithm 1, we first explore its parent node (the "LI" node with grey background) and we check that more than half of its children have a *weight* higher than the *root threshold* multiplied by the weight of the candidate node "UL", so we can continue exploring its ancestors. Next, we explore the parent of the "LI" node, which is the "UL" node with black background. Again, we check that more than half of its children have a *weight* higher than the *root threshold*

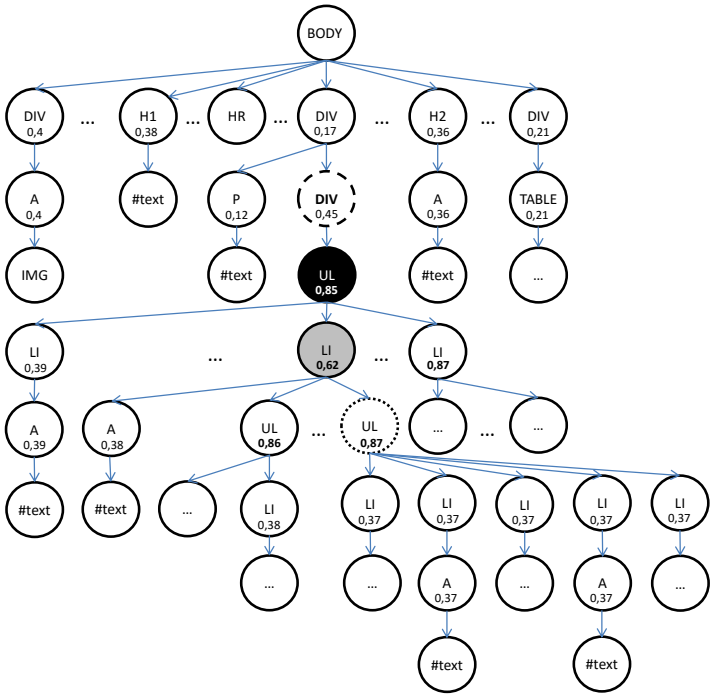

Fig. 1. Example of the selection of root node candidates

multiplied by the weight of the candidate node "UL", so we continue exploring its parent. Then, we do the same with the parent of the "UL" node (a "DIV" node with a dashed shape). In this case, the algorithm continues exploring the parent, but it keeps a pointer to the "UL" node as the menu node because the "DIV" node has only one child. Finally, we check its parent, which is another "DIV" node. It has two children, one is the "DIV" node with a dashed shape and the other is a "P" node. Both nodes have a *weight* lower than the *root threshold* multiplied by the weight of the candidate node "UL", so, in this case, the "DIV" node does not meet the criterion. The last node that satisfies the condition is the "UL" node (black background), thus the algorithm returns it as the root node.

3.4 Selection of the menu node

Algorithm 1 inputs one candidate and outputs a node that could be the main menu. Because we often have more than one candidate, the application of Algorithm 1 over the set of candidates produces another set of nodes. Hence, we need a mechanism to determine which of them represents the real menu of the webpage. This mechanism is implemented by Algorithm 2. For each node in the set, it counts the number of descendants that have a weight over a specified threshold, called *menu threshold*. Then, the average weight of these nodes is

Algorithm 2 Menu node selection

Input: A set of DOM nodes N and a threshold $weight$.
Output: A DOM node $menuNode$ representing the main menu.

begin
  $max = 0$;
  $bestWeight = 0$;
  foreach $(n \in N)$
    $heavyChildren = \{child \mid child \in n.children \ \wedge \ child.weight > weight\}$;
    $nodeCount = |heavyChildren|$;
    $nodeWeight = \sum_{child \in heavyChildren} child.weight$;
    if $(nodeWeight/nodeCount > bestWeight)$
      $menuNode = n$;
      $bestWeight = nodeWeight/nodeCount$;
  return $menuNode$;
end

computed. The node with the highest average weight is selected as the menu of the webpage. We established this criterion because the menu node often has a high concentration of nodes with a high *weight*.

4 Implementation

The technique presented in this paper, including all the algorithms, has been implemented as a Firefox add-on. In this tool, the user can browse on the Internet as usual. Then, when they want to extract the menu of a webpage, they only need to press on the "Extract Menu" button and the tool automatically (internally) rates the DOM nodes, analyzes them, and selects the menu node. The menu is then displayed on the browser as any other webpage.

Example 2. Figure 2 shows the output of the tool with a real webpage. The left image is the main webpage of the **foodsense.is** website. Once the menu is extracted, we can see it in the right image.

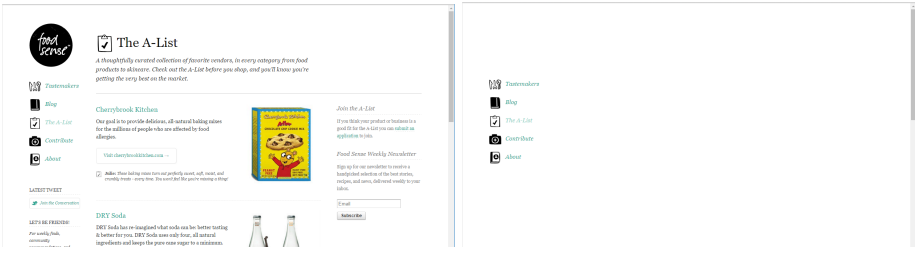

Fig. 2. Example of the detection of a webpage menu

4.1 Training phase: Determining the best parameters through empirical evaluation

In our theoretical formalization, we presented our technique in an abstract way. Some parameters of our algorithms, however, have been left open. In this section, we calculate the value of these parameters based on an experimental analysis. Firstly, we need to define a weighting to combine the properties proposed in Definition 6. Moreover, Algorithm 1 explores the ancestors of a node to determine which of them is the webpage menu. This process depends on a parameter called *root threshold* that sets the stop condition. Another parameter to take into consideration is the one used to select the menu node among all the possible candidates in the set. Algorithm 2 explores the menu nodes in the set and, for each one, it computes the number of children with a weight over a specified threshold called *menu threshold*.

Our method to approximate the best combination of values for the two thresholds (*root threshold* and *menu threshold*) and for the weighting of node properties, follows these steps:

1. First, prior to the development of our technique, we constructed a suite of benchmarks prepared for menu detection.[3]
2. Second, we executed our system with a training subset of the benchmarks suite. For this, we evaluated the precision and recall obtained for each different combination of values for the properties and thresholds. We performed more than 1.5 million experiments, with a total computing time equivalent to 85 days in an Intel i7 4770k processor.
3. Third, we selected the best combination of properties and thresholds, and evaluated it against an evaluation subset of the benchmarks suite.

Our suite of benchmarks is the first suite of benchmarks prepared for menu detection. This means that each benchmark has been labelled with HTML classes indicating what parts of the webpage belong to the menu. This allows any technique to automatically validate their recall and precision. Our suite is composed of 50 benchmarks, and it is not only prepared for menu detection, but also for content extraction and template detection (i.e., the template and the main content of the webpages are also labelled with specific HTML classes). This is one of the main contributions of our work. Any interested researcher can freely access and download our dataset from:

`http://www.dsic.upv.es/~jsilva/retrieval/teco/`

The suite is composed of a collection of Web domains with different layouts and page structures. To measure our technique, we randomly selected an evaluation subset and we performed several experiments with a menu detector that implemented our technique. Once the algorithm detected the menu, we compared

[3] We designed and implemented the suite of benchmarks before we constructed our technique to avoid their interference.

it with the real menu, and we computed the precision, recall and F1 scores of the algorithm. For each node, we computed its final weight evaluating different weightings ($weight = A * Node\ amplitude + B * Link\ ratio + C * Text\ ratio + ...$, where $A + B + C + ... = 1$). We repeated all the experiments with these possible values for the weightings used:

Node amplitude:	$[0,00 - 0,20]$ in steps of $0,05$.
Link ratio:	$[0,05 - 0,40]$ in steps of $0,05$.
Text ratio:	$[0,25 - 0,60]$ in steps of $0,05$.
UL ratio:	$[0,00 - 0,20]$ in steps of $0,05$.
Representative tag:	$[0,00 - 0,20]$ in steps of $0,05$.
Node position:	$[0,00 - 0,25]$ in steps of $0,05$.

Moreover, for each possible weighting, we also evaluated the *Root threshold* and the *Menu threshold* with the following values:

Menu threshold:	$[0,80 - 0,90]$ in steps of $0,05$.
Root threshold:	$[0,70 - 0,90]$ in steps of $0,10$.

After having evaluated all possible combinations against all the benchmarks, the fastest combination that produces the best F1 metric is:

Menu	Root	Amplit.	Link	Text	UL	Repres.	Position	Recall	Precision	F1	Time
0,80	0,70	0,20	0,10	0,30	0,20	0,10	0,10	94,10 %	97,91 %	94,09 %	4.77s

Once the best combination was selected in the training phase, we evaluated our technique against a suite of 50 benchmarks (accesible in the website of the project). The technique achieved a precision of 98.21%, a recall of 94.13%, and a F1 of 94.46%. In 74% of the experiments, the menu was perfectly detected (F1=100%). The average computation time was 5.38 s.

5 Conclusions

Menu detection is useful for many systems and tools such as, e.g., indexers and crawlers. It is particularly useful for template extraction because many techniques use the menu to detect webpages that share the template.

This work presents a new technique for menu detection. This technique is a page-level technique and, thus, our algorithms only need to load and analyze one single webpage (the webpage from which we want to extract the menu). This is specially important from the performance point of view, because loading other webpages is costly.

We have proposed a set of features that should be considered in the menu detection process. We empirically evaluated these features to define a weighting that can be used in the menu detection process. The obtained results are quite successful, almost 75% of the experiments perfectly retrieved the menu of the webpage, and we obtained an average precision of 98.21% and an average F1 metric of 94.46%.

References

1. Z. Bar-Yossef and S. Rajagopalan. Template Detection via data mining and its applications. In *Proceedings of the 11th International Conference on World Wide Web (WWW'02)*, pages 580–591, New York, NY, USA, 2002. ACM.
2. R. Burget and I. Rudolfova. Web page element classification based on visual features. In *Proceedings of the 1st Asian Conference on Intelligent Information and Database Systems (ACIIDS'09)*, pages 67–72, Washington, DC, USA, 2009. IEEE Computer Society.
3. E. Cardoso, I. Jabour, E. Laber, R. Rodrigues, and P. Cardoso. An efficient language-independent method to extract content from news webpages. In *Proceedings of the 11th ACM symposium on Document Engineering (DocEng'11)*, pages 121–128, New York, NY, USA, 2011. ACM.
4. W. Consortium. Document Object Model (DOM). Available at URL: `http://www.w3.org/DOM/`, 1997.
5. T. Gottron. Content code blurring: A new approach to Content Extraction. In A. M. Tjoa and R. R. Wagner, editors, *Proceedings of the 19th International Workshop on Database and Expert Systems Applications (DEXA'08)*, pages 29–33. IEEE Computer Society, sep 2008.
6. D. Insa, J. Silva, and S. Tamarit. Using the words/leafs ratio in the DOM tree for Content Extraction. *The Journal of Logic and Algebraic Programming*, 82(8):311–325, 2013.
7. C. Kohlschütter. A densitometric analysis of web template content. In J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 1165–1166. ACM, apr 2009.
8. C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text teatures. In B. D. Davison, T. Suel, N. Craswell, and B. Liu, editors, *Proceedings of the 3rd International Conference on Web Search and Web Data Mining (WSDM'10)*, pages 441–450. ACM, feb 2010.
9. C. Kohlschütter and W. Nejdl. A densitometric approach to web page segmentation. In J. G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D. A. Evans, A. Kolcz, K.-S. Choi, and A. Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*, pages 1173–1182. ACM, oct 2008.
10. H. Sano, S. Shiramatsu, T. Ozono, and T. Shintani. A web page segmentation method based on page layouts and title blocks. *IJCSNS International Journal of Computer Science and Network Security*, 11(10):84–90, oct 2011.
11. K. Vieira, A. S. da Silva, N. Pinto, E. S. de Moura, J. a. M. B. Cavalcanti, and J. Freire. A fast and robust method for web page template detection and removal. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM'06)*, pages 258–267, New York, NY, USA, 2006. ACM.
12. T. Weninger, W. Henry Hsu, and J. Han. CETR: Content Extraction via tag ratios. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*, pages 971–980. ACM, apr 2010.
13. L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD'03)*, pages 296–305, New York, NY, USA, 2003. ACM.