Insa Cabrera, D.; Silva Galiana, JF.; Tamarit, S. (2013). Using the words/leafs ratio in the DOM tree for content extraction. Journal of Logic and Algebraic Programming. 82(8):311-325. doi:10.1016/j.jlap.2013.01.002.

Using the Words/Leafs Ratio in the DOM Tree for Content Extraction[☆]

David Insa, Josep Silva*, Salvador Tamarit

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
E-46022 Valencia, Spain.

Abstract

The main content in a webpage is usually centered and visible without the need to scroll. It is often rounded by the navigation menus of the website and it can include advertisements, panels, banners, and other not necessarily related information. The process to automatically extract the main content of a webpage is called *content extraction*. Content extraction is an area of research of widely interest due to its many applications. Concretely, it is useful not only for the final human user, but it is also frequently used as a preprocessing stage of different systems (i.e., robots, indexers, crawlers, etc.) that need to extract the main content of a web document to avoid the treatment and processing of other useless information. In this work we present a new technique for content extraction that is based on the information contained in the DOM tree. The technique analyzes the hierarchical relations of the elements in the webpage and the distribution of textual information in order to identify the main block of content. Thanks to the hierarchy imposed by the DOM tree the technique achieves a considerable recall and precision. Using the DOM structure for content extraction gives us the benefits of other approaches based on the syntax of the webpage (such as characters, words and tags), but it also gives us a very precise information regarding the related components in a block (not necessarily textual such as images or videos), thus, producing very cohesive blocks.

1. Introduction

Block Detection is a discipline that tries to isolate every information block in a webpage. For instance, the webpage in Figure 1 has been divided in different

[☆]This work has been partially supported by the Spanish *Ministerio de Economía y Competitividad (Secretaría de Estado de Investigación, Desarrollo e Innovación)* under grant TIN2008-06622-C03-02 and by the *Generalitat Valenciana* under grant PROMETEO/2011/052. Salvador Tamarit was partially supported by the Spanish MICINN under FPI grant BES-2009-015019. David Insa was partially supported by the Spanish Ministerio de Eduación under FPU grant AP2010-4415.

*Corresponding author

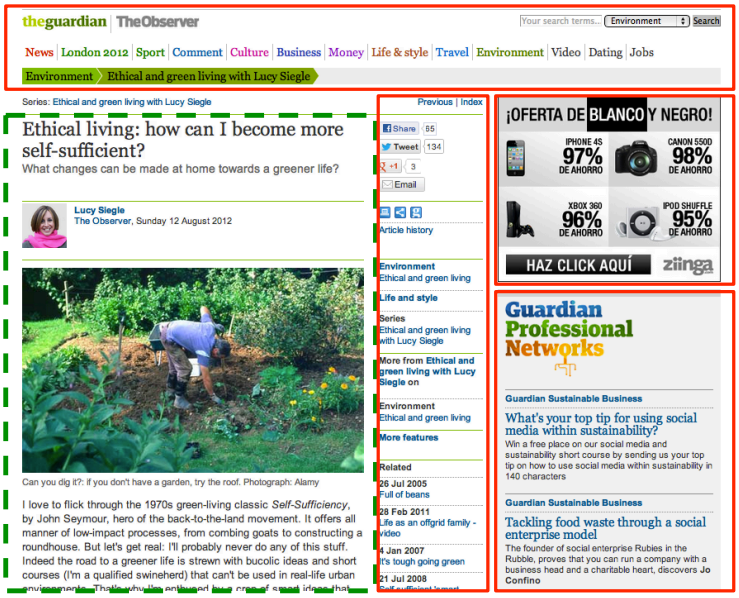


Figure 1: Blocks of a webpage from The Guardian's website

areas that correspond to blocks (e.g., menu, related links panel, advertisement, banner, etc.). The block that contains the main content of the webpage has been squared with a dashed line. Finding this main block is very important for many applications because it allows us to concentrate on the main information avoiding the processing of superfluous content such as menus, status bars, advertisements, sponsored information, etc. The process of isolating the main block in a webpage is known as *content extraction*. Finding the main block is not easy at all because the main block is not necessarily continuous in the webpage. Many times it is inside other block, or it contains other blocks that should be discarded. Thus the main information is mixed with other noisy information.

It has been measured that almost 40-50% of the content in a webpage can be considered irrelevant [1]. Therefore, determining the main block of a webpage is very useful for indexers and text analyzers to increase their performance by only processing relevant information. Other interesting applications are the extraction of the main content of a webpage to be suitably displayed in a small device such as a PDA or a mobile phone; and the extraction of the relevant content to make the webpage more accessible for visually impaired or blind.

Our technique combines ideas from other works such as [2, 3, 4], and it introduces new ideas based on the use of explicit information contained in the Document Object Model (DOM) tree of webpages. This combination of ideas allows the technique to produce very accurate results.

In summary, the main advantages of our technique are the following:

- It does make no assumptions about the particular structure of webpages.

- It only needs to process a single webpage (no templates, neither other webpages of the same website are needed).
- No preprocessing stages are needed. The technique can work online.
- It is fully language independent (it can work with pages written in English, Greek, Japanese, etc.).
- The particular text formatting and indentation of the webpage does not influence the performance of the technique.

The rest of the paper has been structured as follows: In Section 2 we discuss the related work and show some problems of current techniques that can be solved with our approach. In Section 3 we recall the DOM model and provide some useful notation. Then, in Section 4, we present and formalize our content extraction technique and explain it with examples. In Section 5 we give some details about the implementation and show the results obtained with a collection of benchmarks. Section 6 presents a case of study using our tool. Finally, Section 7 concludes.

2. Related Work

There exist many different techniques devoted to solve the problem of content extraction. Practically all of them are based on the textual information of the webpage being analyzed. This is due to the fact that the main content usually contains the biggest concentration of text in the webpage. Some of the approaches are based on the assumption that the webpage has a particular structure (e.g., based on table markup-tags) [5], that the main content text is continuous [6], that the system knows a priori the format of the webpage [5], or even that the whole website to which the webpage belongs is based on the use of some template that is repeated [7]. This allows the system to analyze several webpages and try to deduce the template of the webpage in order to discard menus and other repeated blocks.

The main problem of these approaches is a big loss of generality. In general, they require to previously know or parse the webpages, or they require the webpage to have a particular structure. This is very inconvenient because modern webpages are mainly based on `div` tags that do not require to be hierarchically organized (as in the table-based design). Moreover, nowadays, many webpages are automatically and dynamically generated and thus it is often impossible to analyze the webpages a priori.

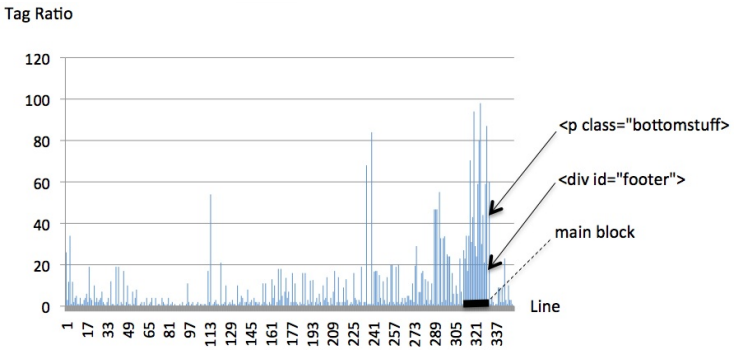
There are, however, other approaches that are able to work with any webpage and without the need to preprocess them or know their structure in advance. One of these approaches is the technique presented in [3]. This technique uses a *content code vector* (CCV) that represents all characters in a document determining whether they are content or code. With the CCV, they compute a *content code ratio* to identify the amount of code and content that surrounds the elements of the CCV. Finally, with this information, they try to determine what parts of the document contain the main content. Another powerful approach

also based on the labeling of the characters of a document has been presented in [4]. This work is based on the use of *tag ratios* (TR). Given a webpage, the TR is computed for each line with the number of non-HTML-tag characters divided by the number of HTML-tags. The main problem of the approaches based on characters or lines such as these two, or words such as [8], is that they completely ignore the structure of the webpage. Using characters or words as independent information units and ignoring their interrelations produce an important loss of information that is present and explicit in the webpage, and that makes the system to fail in many situations.

Example 2.1. *Consider the following portion of a source code extracted from an IEEE's webpage:*

```
<body>
     (...)
     <div id="maincontent">
          <a name="Abstract"><h2>Abstract</h2></a>
          <p>Most HTML documents (...)
          (...)
          (...)  delivers the best results.</p>
     </div>
     <div id="footer">
          <p class="bottomstuff">Copyright IEEE. All rights are (...)  </p>
          <p class="links">
          <a href="IEEE_Xplore.html" target="blank">Help</a> |
          <a href="/xpl/techform.jsp">Contact Us</a> | (...)
     </div>
     (...)
</body>
```

The tag ratios associated to this webpage are shown bellow.



Observe that the initial part of the footer, which is not part of the main content, is classified as part of the main content because it starts with a high tag ratio. Unfortunately, this method does not take into account the information provided by tags, and thus, it fails to infer that the footer text belongs to a different `div` *tag than the other text classified as relevant.*

The distribution of the code between the lines of a webpage is not necessarily the one expected by the user. HTML code can be completely unindented (i.e., without tabulations, spaces or even carriage returns), specially when it is

generated by a non-human directed system. As a common example, the reader can see the source code of the main Google's webpage. At the time of writing these lines, all the code of the webpage is completely unindented and without carriage returns. In this kind of webpages tag ratios are useless.

Some of the authors of this work proposed a first idea to solve the problems of tag ratios [2]. This idea is to use a new ratio called *chars-nodes ratio* (CNR) that is similar to the tag ratio but based on the DOM structure of the webpage. Concretely, given a node n in the DOM tree, the CNR of n is computed with the number of characters contained in the subtree of n divided by the number of nodes in this subtree. Once all CNRs have been computed, they defined an algorithm to select one node of the DOM tree using the CNRs of the nodes.

The approach based on CNRs is in the good direction, because it keeps the good properties of tag ratios, and at the same time, it also solves some of their problems. In particular, because the DOM tree is independent of the distribution of the code between the lines of the HTML webpage, the technique is able to work with any webpage independently of how the webpage was generated or indented. Moreover, thanks to the DOM hierarchy, the technique can identify precisely the HTML containers and avoid mixing different sections (such as the footer and the other `div` tag in Example 2.1).

Even though our technique has been designed over the CNRs, the method proposed here to select a node from the DOM tree is essentially different to the one used with CNR. Moreover, the ratio itself has been changed. The new ratio is called *words-leaves ratio* (WLR). First, it counts words instead of characters. This allows the technique to better measure the relevance of a text. Using characters was inherited from TR, and it is a bad idea because it unnecessarily prizes long words (e.g., 'medicine' counts double than 'work' without any justified reason). Second, after intensive testing we discovered another weakness of CNR, namely, counting all nodes in the subtree of a node. In contrast, we only consider the leaves because they are the only nodes that contain textual information.

Although essentially different to our work, there exist other techniques that make use of the DOM structure, and thus, they could exploit the same information than our technique. The most similar approach is the one presented in [9]. This approach presents a proxy server that implements a set of filters for HTML documents. These filters include HTML cleaning (e.g., removing images, scripts, etc.), HTML refactoring (e.g., removing empty tables), and deleting advertisements (e.g., with a blacklist of URLs that allow them to remove external publicity content). Some of these transformations are used by our technique, but the objective is different, we do not want to clean, improve or transform the original webpage; our goal is to detect the main content and remove all the other components of the webpage. Also the implementation is different, our tool is not based on a proxy server; it is implemented in the client side, and thus it is independent of any external resource.

There are some approaches specialized for a particular content such as tables that are somehow related to our work. They do not focus on block detection but in content extraction from tables [10], or in wrappers induction [11, 12]. Other

related approaches are based on statistical models [13, 14] and machine learning techniques [15, 16] and they use densitometric features such as link density and text features such as number of words starting with an uppercase letter [17].

3. The DOM tree

The Document Object Model [18] is an API that provides programmers with a standard set of objects for the representation of HTML and XML documents. Our technique is based on the use of DOM as the model for representing webpages. Given a webpage, it is completely automatic to produce its associated DOM structure and vice-versa. In fact, current browsers automatically produce the DOM structure of all loaded webpages before they are processed.

The DOM structure of a given webpage is a tree where all the elements of the webpage are represented (included scripts and styles) hierarchically. For example, bold text is represented with a node representing the bold format and a descendant of this node that represents the text. In the current DOM model there exist 12 node types: 11 types for the different HTML tags and the type *text* which is the only one that contains text. Each node contains all the information associated with the tag it represents (e.g., a image node contains all its attributes including *alt*, *src*, etc.). An important property of DOM trees is that text nodes are always leaves. We exploit this property in our technique.

Definition 3.1 (DOM Tree). *Given an HTML document D, the DOM tree of D 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 tag* `body`*. E is a finite set of edges such that $(n \to n') \in E$, with $n, n' \in N$ if and only if the tag or text associated with n' is inside the tag associated with n in D, and not exists an unclosed tag between them. We represent with E^* the reflexive and transitive closure of E.*

For the purpose of this work, it does not matter how the DOM tree is built. In practice, the DOM API provides mechanisms to add nodes and attributes, and provides methods to explore and extract information from the tree.

Example 3.2. *As an example of DOM representation of a webpage observe in Figure 2 a portion of the source code extracted from the entry "Gerard Salton" at DBLP. The webpage generated by this code is shown in Figure 3, and a portion of its associated DOM tree is depicted in Figure 4. For the time being the reader can ignore the different colors and borders of nodes.*

The formalization of our technique has been done using the own DOM representation. This allows us to directly map our algorithms to the implementation. In particular, each node in the DOM tree has a number of attributes depending on the tag they represent. In our formalization, we add new attributes to the nodes in order to represent information that will be later used to extract the main content.

```
<body>
     <div class="llogo">
          <a href="../../../index.html">
               <img alt="dblp.uni-trier.de" src="../../../Logo.gif"
                    border="0" height="60" width="170"/>
          </a>
     </div>
     (...)
     <hr/>
     (...)
     <h1 key="homepages/s/GerardSalton">Gerard Salton
          <a href="http://www.cs.virginia.edu/ clv2m/salton.txt">(...)</a>
     </h1>
     (...)
     <p>
          <table border="1">
               <tr>
                    <th>(...)</th>
                    <th colspan="2" bgcolor="#FFFFCC">1997</th>
               </tr>
               <tr>
                    <td align="right" valign="top" class="J" bgcolor="#FFCCCC">138</td>
                    <td valign="top">(...)</td>
                    <td>Gerard Salton,
                        <a href="../s/Singhal:Amit.html">Amit Singhal</a> (...)
                        Automatic Text Structuring and Summarization.
                        <a href="../../../../db/journals/ipm/ipm33.html#SaltonSMB97">
                        Inf.  Process.  Manage.  33</a>(2):  193-207 (1997)
                    </td>
               </tr>
               (...)
          </table>
     </p>
     <h2>
          <a name="coauthors" href="http://dblp.uni-trier.de/rec/pers/s/Salton:Gerard/xc">
               Coauthor Index
          </a>
     </h2>
     <p>
          <table border="1">
               (...)
          </table>
     </p>
     <script type="text/javascript" src="/ ley/www.dblp.org/completesearch.js"></script>
</body>
```

Figure 2: HTML code of a DBLP webpage showing Gerard Salton's papers

Definition 3.3 (node attributes). *Every node n in a DOM tree contains the attributes specified in the DOM model [18] including:*

Name: *We refer to the name of a DOM node by using name(n) and it corresponds to the DOM attribute nodeName.*

Additionally, n contains the following new attributes:

Visible: *Once the tree is computed, it is possible to know whether a node is visible or not. It is done checking whether the attribute 'visibility' is set to 'hidden' or 'collapse', or the attribute 'display' is set to 'none'. This boolean attribute is represented with v(n).*

Words: *It represents the number of words included in the content of this node.*

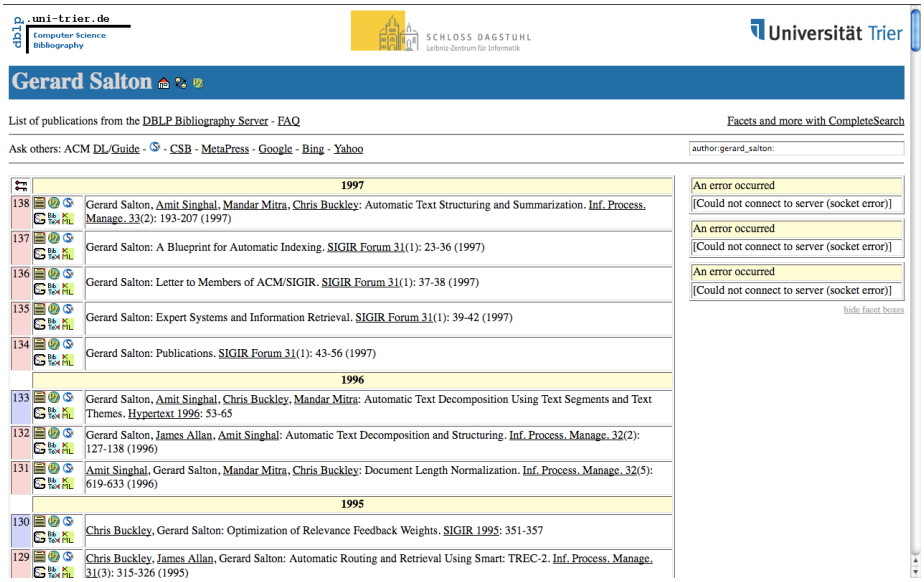


Figure 3: DBLP webpage associated with the code in Figure 2

This number can be directly inferred from the attribute 'content' already present in the DOM model. We represent this attribute with $w(n)$. If the attribute 'content' is not defined for n then $w(n) = 0$.

4. Content extraction using DOM trees

In this section we formalize our technique for content extraction. The technique is based on the notion of *words-leaves ratio* (WLR), which shows for each node in the DOM tree the relation between the amount of words and leaves in its subtree.

First of all we need to discard those nodes that are not useful for the analysis. Concretely, we have three different scenarios where nodes can be removed. First, all the subtrees whose root is a not content node such as `meta`, `title`, `script`, `#comment`, etc. These nodes can be removed because they do not contribute to the visible content of the webpage. Similarly, we can remove all the subtrees whose root is not visible. Finally, we remove all the leaves of the tree that do not contain any word. This is computed using a fix-point bottom-up process because deleting one leaf could produce a new removable leaf.

The remaining nodes after removing these three sets is a new set that we call *content nodes set* and it contains the only nodes that contribute to the textual content of the webpage, its hierarchical structure and its format. This set of nodes form a tree equivalent to the original one where some branches have been removed. The nodes of this tree are the only ones considered in the rest of the analysis. The following definition formalizes all these concepts.

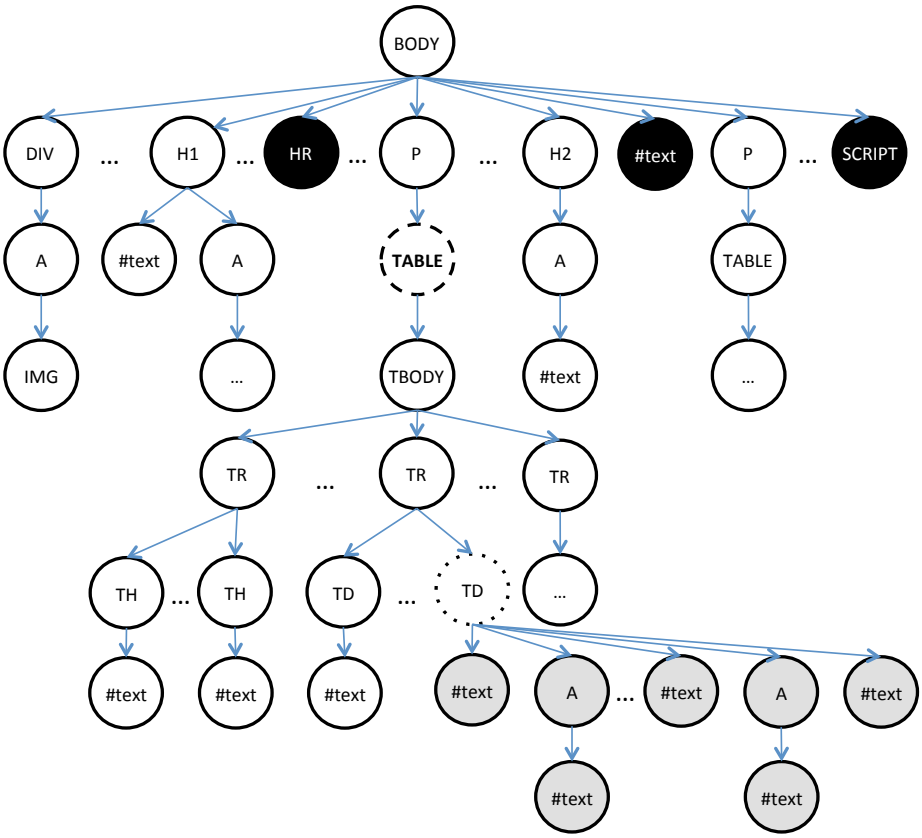


Figure 4: DOM representation associated with the code in Figure 2

Definition 4.1 (content nodes set). *Given a DOM tree* (N, E)*, the set of* notContent *nodes is the union of the following sets:*

- $notValid = \{n \in N \mid \exists n' \in N \text{ such that } (n' \to n) \in E^* \wedge name(n') \in \{\texttt{meta}, \texttt{title}, \texttt{head}, \texttt{link}, \texttt{style}, \texttt{script}, \texttt{select}, \texttt{noscript}, \#\texttt{comment}\}\ \}$
- $notVisible = \{n \in N \mid \exists n' \in N \text{ such that } (n' \to n) \in E^* \wedge \neg v(n')\}$
- $notWords$ *is the result of the following fix-point:*
  - $notWords_0 = \emptyset$
  - $notWords_{i+1} = notWords_i \cup \{n \in N \backslash notWords_i \mid w(n) = 0 \wedge (\nexists n' \in N \backslash notWords_i \ : (n \to n') \in E)\}$

Thus, $notContent = notValid \cup notVisible \cup notWords$*. The set of* content nodes *is represented with* $\mathcal{C}$ *and defined as:* $\mathcal{C} = N \backslash nonContent$*.*

We need additional attributes for each node in $\mathcal{C}$ to be able to properly define the notion of WLR. Concretely, for each node in $\mathcal{C}$ we add one attribute that is used to uniquely identify each node, we add another attribute to know whether the position is static, and we add two attributes that respectively represent the total amount of words and leaves in its subtree.

The following definition formalizes the new attributes where we represent with $\mathit{Children}(n)$ the set of children of a node $n \in \mathcal{C}$, formally, $\mathit{Children}(n) = \{n' \in \mathcal{C} \mid (n \to n') \in E\}$.

Definition 4.2 ($\mathcal{C}$ attributes). *Every node $n \in \mathcal{C}$ contains the following attributes:*

Identifier: *A number that uniquely identifies each node in the tree. Identifiers are assigned in pre-order and incrementally, being 0 the identifier of the root node. The identifier of n is represented with $id(n)$.*

Static position: *A boolean that indicates whether the position of the HTML element associated to this node is static. The position of all nodes is static except* `div` *nodes whose value in their attribute 'position' is 'absolute' or '$fixed$'. It is represented with $sp(n)$.*

Total words: *The number of words in the subtree of a node n is computed using the following function:*

$$tw(n) = \begin{cases} w(n) & \text{if } \mathit{Children}(n) = \emptyset \\ \sum_{n_c \in \mathit{Children}(n)} tw(n_c) & \text{otherwise} \end{cases}$$

Leaves: *The number of leaves in the subtree of n is computed using this function:*

$$l(n) = \begin{cases} 1 & \text{if } \mathit{Children}(n) = \emptyset \\ \mathit{leaves}(n) & \text{otherwise} \end{cases}$$

where function leaves is defined in Figure 5.

Identifiers play an important role in the technique because they allow us to compare the position of HTML elements in the final webpage displayed. This happens thanks to an important property of DOM, namely, elements are displayed in the screen following the order of their identifiers. It is important to also remark the way in which the number of leaves in a subtree is computed. Observe that, according to function *leaves*, the number of leaves in a subtree depends not only on its structure, but also on the specific HTML tags distributed among its nodes. In particular, those HTML tags used to format the text appear together in the DOM model as consecutive siblings. Some of them apply a special format such as italics or bold, and others apply special properties such as paragraphs, or links. In any case, the effect of these tags is that an unformatted text that would appear as a single leaf in the DOM tree, is represented as a subtree with many leaves with different text formats. Our analysis detects this situation and considers this formatted text as a single leaf in the DOM tree.

$leaves(n) =$
$children = Children(n)$
$leaves = 0$
$joining = false$
while $(children \neq \emptyset)$
$child = n' \in children$. $\nexists n'' \in children$ such that $id(n'') < id(n')$
$children = children \backslash \{child\}$
if $(name(child) \in \{\#\texttt{text}, \texttt{p}, \texttt{a}, \texttt{u}, \texttt{b}, \texttt{i}, \texttt{em}, \texttt{span}, \texttt{sub}, \texttt{sup}, \texttt{strong}, \texttt{div}\}$
$\wedge$ sp(child) $\wedge$ l(child)==1)
$joining = true$
else
$leaves = leaves + l(child)$
if $(joining)$
$leaves = leaves + 1$
$joining = false$
if $(joining)$
$leaves = leaves + 1$
return $leaves$

Figure 5: Function *leaves*

The HTML tags affected by this process are `#text`, `p`, `a`, `u`, `b`, `i`, `em`, `span`, `sub`, `sup`, `strong` and `div`. This is done to avoid penalizing the ratio between words and leaves of those texts that need several leaves in order to give special format to some parts of the text. We illustrate the necessity of this special treatment for leaves in Example 4.3.

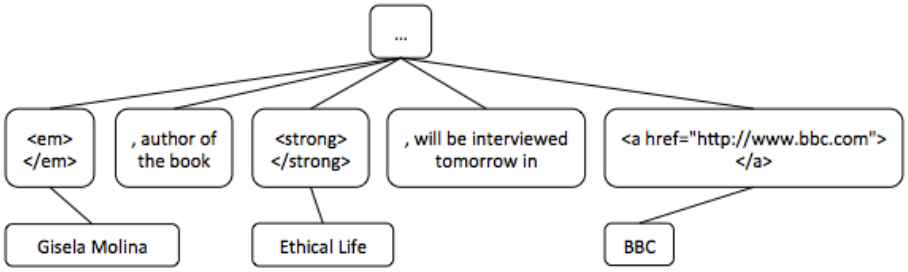
Example 4.3. *The following HTML code:*

```
<em>Gisela Molina</em>, author of the book <strong>Ethical Life</strong>,
will be interviewed tomorrow in <a href="http://www.bbc.com">BBC</a>.
```

is displayed by any browser as:

Gisela Molina, author of the book **Ethical Life**, will be interviewed tomorrow in BBC.

and its associated DOM tree is the following:



Note that the whole tree represents a single sentence, and thus the intermediate nodes are only used to format the text (i.e., if the text was plain text, a single node would be used). Therefore, we should consider this text as a single leaf, and this is exactly what function leaves implements.

We are now in a position to define the WLR measure that allows the analysis to identify those nodes associated with the main textual content of a webpage.

Definition 4.4 (words–leaves ratio). *Given a node $n \in \mathcal{C}$, its* words–leaves ratio *is represented with $WLR(n)$ and defined as:*

$$WLR(n) = tw(n)/l(n)$$

Using WLR we extract a subset of $\mathcal{C}$ that we call *initial nodes*. This set is the starting point of our analysis and it contains those nodes whose WLR is closer to the maximum WLR of the whole tree than to the average WLR of the tree.[1] Hence, we need to find a value for variable x that satisfies the following equation:

$$\frac{max_{WLR}}{x} = \frac{x}{WLR(root)}$$

Therefore,

$$x = \sqrt{max_{WLR} \times WLR(root)}$$

This number is an acceptance threshold that our analysis uses to select and to discriminate nodes. Now we can introduce the formal definition of the initial nodes set.

Definition 4.5 (initial nodes set). *Given the set $\mathcal{C}$, the initial nodes set $\mathcal{I}$ is defined as:*

$$\mathcal{I} = \{n \in \mathcal{C} \mid WLR(n) \geq \sqrt{max_{WLR} \times WLR(root)}\}$$

The initial nodes set contains those nodes in the DOM tree with a higher density of text. Therefore, it is highly probable that some of these nodes together form the main content (and, thus, the other nodes in $\mathcal{I}$ should be discarded). Hence, at this point, we need a mechanism to suitably combine the information of one or more nodes in $\mathcal{I}$. For this purpose, we can use the DOM hierarchy, so that two nodes can be naturally combined with a common ancestor. Moreover, the combination mechanism must take into account another factor that has been ignored until now. This factor is the relative position of the content in the webpage and can be computed with the identifier of the nodes. In order to combine several nodes in $\mathcal{I}$ taking into account both measures (WLR and position in the DOM tree) we introduce the relevance of a node, whose formalization follows.

[1] A moment of thought will convince the reader that, according to Definition 4.4, the average WLR value in a DOM tree always corresponds to the WLR value of the root node.

Definition 4.6 (relevance). *Given a node $n \in \mathcal{C}$, its* weight *is represented with $W(n)$ and defined as:*

$$W(n) = \begin{cases} r_{pos}(n) \times r_{WLR}(n) & \text{if } n \in \mathcal{I} \\ 0 & \text{if } n \notin \mathcal{I} \end{cases}$$

where

$$r_{pos}(n) = 1 - \frac{id(n) - min_{id}}{max_{id} - min_{id}}$$

$$r_{WLR}(n) = \frac{WLR(n) - min_{WLR}}{max_{WLR} - min_{WLR}}$$

and min_{id}, max_{id} are the minimum and maximum values of identifiers in $\mathcal{I}$, whereas min_{WLR}, max_{WLR} are the minimum and maximum WLR values in $\mathcal{C}$. Given a node n from $\mathcal{C}$, its relevance *is represented with $R(n)$ and defined as:*

$$R(n) = WLR(n) \times max(W(n), \sum_{n' \in Children(n)} R(n') \ )$$

The relevance is computed by only taking into account the nodes with a higher density of text (those in $\mathcal{I}$). The other nodes are discarded by assigning them a weight of 0. Contrarily, the nodes in $\mathcal{I}$ are assigned a weight that combines their WLR and their relative position in the webpage. With the weight, we compute the relevance. The relevance of a node is the WLR of this node multiplied by the maximum between its weight and the sum of the relevances of its children. Hence, the relevance is composed by two expressions: $WLR(n)$, and the expression:

$$M(n) = max(W(n), \sum_{n' \in Children(n)} R(n') \ )$$

The expression $M(n)$ increases when it combines nodes in $\mathcal{I}$. Therefore, $M(n)$ increases bottom-up in the tree. In particular, given two nodes $n, n' \in \mathcal{C}$, if $(n \to n') \in E^*$, then $M(n) \geq M(n')$. Hence, our mechanism to combine nodes in $\mathcal{I}$ needs a criterion to stop combining (otherwise, we would always select the root of the tree). Concretely, the criterion must combine nodes with a high density of text and stop combining when the new nodes combined produce a decrease in that density. For this objective, we have the other component of the relevance, namely, $WLR(n)$. This component decreases when the amount of text associated with a node is low. In summary, the relevance uses $M(n)$ to go up in the tree combining high textual density nodes, and it uses $WLR(n)$ to stop going up when the ancestor of a node introduces subtrees with a low density of text. The relevance is used to select the final node that represents the main content of the webpage. For this purpose, we order all nodes in the content nodes set according to their relevance to obtain the best node. In the case that two nodes have the same relevance, we select the one with a lower identifier.

Definition 4.7 (best node). *Given the set $\mathcal{C}$, the best node n_{best} is defined as:*

$$n_{best} = n \in \mathcal{C} \text{ such that } \nexists n' \in \mathcal{C} : R(n') > R(n) \vee (R(n') = R(n) \wedge id(n') < id(n))$$

Algorithm 1 Content extraction algorithm

Input: A DOM tree $T = (N, E)$
Output: A DOM tree $T' = (N', E')$ with $N' \subseteq N$ and $E' \subseteq E$
Inicialization: $T' = T$

begin
1) Obtain the content nodes $\mathcal{C}$ by filtering nodes without textual content:
  $\mathcal{C} = N \backslash notContent$ [See Definition 4.1]
2) Compute WLR for all nodes in $\mathcal{C}$:
  $Processed = \emptyset$
  $max_{WLR} = 0$
  while $Processed \neq \mathcal{C}$
    take $n \in \mathcal{C} \backslash Processed$ such that $Children(n) \subseteq Processed$
    $WLR(n) = tw(n)/l(n)$ [See Definition 4.2]
    if $(WLR(n) > max_{WLR})$ **then** $max_{WLR} = WLR(n)$
    $Processed = Processed \cup \{n\}$
3) Obtain the initial nodes, the relevance and the best node:
  $Processed = \emptyset$
  n_{best} is a fresh node reference
  $R(n_{best}) = -1$
  while $Processed \neq \mathcal{C}$
    take $n \in \mathcal{C} \backslash Processed$ such that $Children(n) \subseteq Processed$
    3.1) Obtain the initial nodes $\mathcal{I}$ with those nodes with a higher WLR:
    if $WLR(n) \geq \sqrt{max_{WLR} \times WLR(root)}$
    then $W(n) = r_{pos}(n) \times r_{WLR}(n)$ [See Definition 4.6]
    else $W(n) = 0$
    3.2) Compute the *relevance* of all nodes in $\mathcal{C}$:
    $R(n) = WLR(n) \times max(W(n), \sum_{n' \in Children(n)} R(n'))$
    3.3) Select the best node, i.e., the node with the highest *relevance*:
    if $(R(n) > R(n_{best})) \vee (R(n) = R(n_{best}) \wedge id(n) < id(n_{best}))$
    then $n_{best} = n$
    $Processed = Processed \cup \{n\}$
4) Extract the subtree of T whose root is n_{best}:
  $N' = \{n \in N \mid (n_{best} \rightarrow n) \in E^*\}$
  $E' = \{(n \rightarrow n') \in E \mid n, n' \in N'\}$
end
return T'

The technique is summarized in Algorithm 1. Essentially, this algorithm performs the following steps:

1. Obtain the content nodes $\mathcal{C}$ by filtering nodes without textual content.
2. Compute WLR for all nodes in $\mathcal{C}$.
3. Obtain the initial nodes $\mathcal{I}$ with those nodes with a higher WLR.
4. Compute the *relevance* of all nodes in $\mathcal{C}$.
5. Select the *best node*, i.e., the node with the highest *relevance*.
6. Extract the subtree whose root is the *best node*.

The cost of the whole process is $\mathcal{O}(N)$ being N the number of nodes in the DOM tree. The first two steps can be computed with a single pass because *notContent* nodes can be discarded at the same time that WLR is computed. In the algorithm, for clarity, the computation of $\mathcal{C}$ has not been detailed, but it should be performed inside the first **while** loop. Similarly, steps 3, 4, 5 and 6 can be computed with a single traversal (here again, step 6 is not computed inside the second **while** loop to simplify the reading, but it is trivial). Therefore, the total cost is $2N$. Note that the computation of $\mathcal{I}$ requires to know in advance max_{WLR}. For this reason, max_{WLR} has been computed inside the first **while** loop. Similarly, r_{pos} and r_{WLR} need to know max_{WLR}, min_{WLR}, max_{id} and min_{id}. Of course, the computation of all these values can also be done at the same time, but it has been omitted for the sake of clarity.

Example 4.8. *Consider again the HTML code in Figure 2 and its associated DOM tree shown in Figure 4. The black nodes are the nodes removed by the first step of the process, i.e., the rest of the nodes belong to the set $\mathcal{C}$. Note that the tags* `script` *or* `hr` *are removed because they do not contribute to the content of the webpage. The* `#text` *node between nodes labeled with* `h2` *and* `p` *is also removed because it does not contain text (as is can be observed in the HTML code).*[2]

When $\mathcal{C}$ has been calculated, the next step is to compute WLR for each node by counting words and leaves. The only complicated part here is, maybe, to compute the total amount of leaves for each node. According to our definitions, in Figure 4, for instance, those nodes with a grey background are considered as a single leaf. This means that the value for the attribute leaves of the node with a dotted shape is one.

With WLR we can determine which nodes belong to the initial set $\mathcal{I}$. The only node in the figure that belongs to $\mathcal{I}$ is the node with a dotted shape (labelled with `td`*). When $\mathcal{I}$ has been computed and all nodes have been assigned a relevance, we can determine the best node. In this example, the best node is the one with a dashed shape (labelled with* `table`*). It corresponds to the table which forms the main content of the webpage (the central table with the Gerard Salton's list of papers classified by year shown in Figure 3).*

5. Experiments

We have implemented the technique presented in this paper and made it publicly available, including the source code. It was implemented as a plugin that can be installed in the Firefox browser as a toolbar. The tool is able to automatically analyze any webpage online and filter out all the irrelevant content. Additionally, when required, it can show the DOM tree and produce information about the WLR of each node. This allows us and other researchers

[2]Nodes of type text without text are normal and frequent in DOM. DOM places a node of type text in every place where text could appear. When no text exists, these nodes are useless. Our technique removes all these nodes in the first step.

to study the performance of WLR and to compare our technique with future approaches.

As we argued in the related work section, WLR should be better than CNR because counting words makes more sense than counting characters, and because counting only the leaves is more accurate than counting all nodes. We wanted to empirically prove this in order to measure how much better is WLR than CNR. And, moreover, we also wanted to study the other possible combinations: chars–leaves ratio (CLR) and words–nodes ratio (WNR). Therefore, we conducted several experiments with real and online webpages to provide a measure of the average performance regarding recall, precision and the F1 measure (see, e.g., [19] for a discussion on these metrics) for all four ratios.

For the experiments, we selected from the top-most 500 visited webpages (see `http://www.alexa.com/topsites/`) a collection of domains with different layouts and page structures in order to study the performance of the technique in different contexts (e.g., company's websites, news articles, forums, etc.). Then, we randomly selected the final evaluation set. We determined the actual content of each webpage by downloading it and manually selecting the main content text. The DOM tree of the selected text was then produced and used for comparison evaluation later.

Tables 1, 2, 3 and 4 summarize the results of the performed experiments, and Table 5 compares the average results of these four tables. The first column contains the URLs of the evaluated webpages. For each benchmark, column `DOM words` shows the total number of words in the webpage; column `Main block` shows the number of words contained in the main block; column `Recall` shows the number of relevant words retrieved divided by the total number of relevant words; column `Precision` shows the number of relevant words retrieved divided by the total number of retrieved words; finally, column `F1` shows the F1 metric that is computed as $(2 * P * R)/(P + R)$ being P the precision and R the recall.

The results of all four tables have been measured with the same unit: words. This is very convenient to be able to compare the results. Thanks to the DOM structure, we can straightforwardly obtain the words retrieved even in the case when the technique uses characters to retrieve information (i.e., in CNR and CLR). This is due to the fact that, at the end, the technique retrieves a set of nodes in the DOM tree, and every (textual) node contains a discrete number of words. Therefore, we only have to count the number of words contained in the final set of retrieved nodes.

Experiments reveal that using leaves instead of nodes increases the recall in around 40% while the precision remains similar. This happens because CNR and WNR often select a node in the tree which is in the middle of the main content, while CLR and WLR select a node which is an ancestor of this node thus increasing the recall. The comparison of CNR and WNR reveals that using words instead of characters produces an average increment of 6.48% in the recall and 3.38% in the precision. This increment is not observable with CLR and WLR because using leaves already performs this increment with respect to nodes and thus using either characters or words have no influence. In fact, CLR and WLR have exactly the same empirical results.

Benchmark	DOM words	Main block	Recall	Precision	F1
en.citizendium.org	4817 words	4568 words	1.8 %	100 %	3.54 %
googleblog.blogspot.com	801 words	737 words	6.65 %	100 %	12.47 %
us.gizmodo.com	901 words	518 words	13.9 %	100 %	24.41 %
www.applesfera.com	1626 words	1087 words	5.06 %	100 %	9.63 %
www.bbc.co.uk	1531 words	802 words	100 %	96.74 %	98.34 %
www.blogdecine.com	17647 words	1059 words	0.0 %	0.0 %	0.0 %
www.cbc.ca	1636 words	924 words	97.73 %	100 %	98.85 %
www.cbssports.com	3683 words	1836 words	5.77 %	100 %	10.91 %
www.cinemablend.com	827 words	263 words	18.63 %	100 %	31.41 %
www.cnn.com	1555 words	1065 words	97.65 %	81.25 %	88.7 %
www.elmundo.es	2365 words	1353 words	0.0 %	0.0 %	0.0 %
www.engadget.com	5150 words	3783 words	3.7 %	100 %	7.14 %
www.gizmologia.com	1264 words	202 words	0.0 %	0.0 %	0.0 %
www.healthopedia.com	765 words	205 words	45.85 %	100 %	62.87 %
www.microsiervos.com	1171 words	744 words	9.14 %	100 %	16.75 %
www.nationalfootballpost.com	822 words	205 words	20.49 %	100 %	34.01 %
www.news.cnet.com	5051 words	2396 words	98.12 %	100 %	99.05 %
www.nlm.nih.gov	956 words	642 words	12.31 %	100 %	21.92 %
www.people.com	1183 words	308 words	17.86 %	100 %	30.31 %
www.philly.com	865 words	477 words	32.29 %	100 %	48.82 %
www.sportingnews.com	1496 words	772 words	95.6 %	100 %	97.75 %
www.thefreedictionary.com	2605 words	1267 words	10.58 %	100 %	19.14 %
www.umm.edu	433 words	79 words	0.0 %	0.0 %	0.0 %
www.usatoday.com	1080 words	676 words	8.28 %	100 %	15.29 %
www.vidaextra.com	2381 words	257 words	0.0 %	0.0 %	0.0 %
www.wikipedia.org	3835 words	3657 words	1.59 %	100 %	3.13 %
www.wordreference.com	332 words	95 words	0.0 %	0.0 %	0.0 %
Average	2460 words	1109 words	26.04 %	76.96 %	30.91 %

Table 1: Benchmark results obtained with CNR

Benchmark	DOM words	Main block	Recall	Precision	F1
en.citizendium.org	4817 words	4568 words	1.8 %	100 %	3.54 %
googleblog.blogspot.com	801 words	737 words	6.65 %	100 %	12.47 %
us.gizmodo.com	901 words	518 words	13.9 %	100 %	24.41 %
www.applesfera.com	1626 words	1087 words	5.06 %	100 %	9.63 %
www.bbc.co.uk	1531 words	802 words	100 %	96.74 %	98.34 %
www.blogdecine.com	17647 words	1059 words	0.0 %	0.0 %	0.0 %
www.cbc.ca	1636 words	924 words	97.73 %	100 %	98.85 %
www.cbssports.com	3683 words	1836 words	5.77 %	100 %	10.91 %
www.cinemablend.com	827 words	263 words	18.63 %	100 %	31.41 %
www.cnn.com	1555 words	1065 words	97.65 %	81.25 %	88.7 %
www.elmundo.es	2365 words	1353 words	100 %	100 %	100 %
www.engadget.com	5150 words	3783 words	3.7 %	100 %	7.14 %
www.gizmologia.com	1264 words	202 words	0.0 %	0.0 %	0.0 %
www.healthopedia.com	765 words	205 words	45.85 %	100 %	62.87 %
www.microsiervos.com	1171 words	744 words	9.14 %	100 %	16.75 %
www.nationalfootballpost.com	822 words	205 words	20.49 %	100 %	34.01 %
www.news.cnet.com	5051 words	2396 words	98.12 %	100 %	99.05 %
www.nlm.nih.gov	956 words	642 words	12.31 %	100 %	21.92 %
www.people.com	1183 words	308 words	92.86 %	91.08 %	91.96 %
www.philly.com	865 words	477 words	32.29 %	100 %	48.82 %
www.sportingnews.com	1496 words	772 words	95.6 %	100 %	97.75 %
www.thefreedictionary.com	2605 words	1267 words	10.58 %	100 %	19.14 %
www.umm.edu	433 words	79 words	0.0 %	0.0 %	0.0 %
www.usatoday.com	1080 words	676 words	8.28 %	100 %	15.29 %
www.vidaextra.com	2381 words	257 words	0.0 %	0.0 %	0.0 %
www.wikipedia.org	3835 words	3657 words	1.59 %	100 %	3.13 %
www.wordreference.com	332 words	95 words	0.0 %	0.0 %	0.0 %
Average	2460 words	1109 words	32.52 %	80.34 %	36.89 %

Table 2: Benchmark results obtained with WNR

Benchmark	DOM words	Main block	Recall	Precision	F1
en.citizendium.org	4817 words	4568 words	99.98 %	98.09 %	99.03 %
googleblog.blogspot.com	801 words	737 words	100 %	99.59 %	99.79 %
us.gizmodo.com	901 words	518 words	98.46 %	100 %	99.22 %
www.applesfera.com	1626 words	1087 words	8.65 %	100 %	15.92 %
www.bbc.co.uk	1531 words	802 words	100 %	96.74 %	98.34 %
www.blogdecine.com	17647 words	1059 words	0.0 %	0.0 %	0.0 %
www.cbc.ca	1636 words	924 words	97.73 %	100 %	98.85 %
www.cbssports.com	3683 words	1836 words	100 %	100 %	100 %
www.cinemablend.com	827 words	263 words	100 %	68.49 %	81.3 %
www.cnn.com	1555 words	1065 words	97.65 %	81.25 %	88.7 %
www.elmundo.es	2365 words	1353 words	0.0 %	0.0 %	0.0 %
www.engadget.com	5150 words	3783 words	100 %	93.18 %	96.47 %
www.gizmologia.com	1264 words	202 words	91.58 %	100 %	95.6 %
www.healthopedia.com	765 words	205 words	45.85 %	100 %	62.87 %
www.microsiervos.com	1171 words	744 words	19.35 %	92.9 %	32.03 %
www.nationalfootballpost.com	822 words	205 words	99.51 %	100 %	99.75 %
www.news.cnet.com	5051 words	2396 words	98.12 %	100 %	99.05 %
www.nlm.nih.gov	956 words	642 words	12.31 %	100 %	21.92 %
www.people.com	1183 words	308 words	92.86 %	91.08 %	91.96 %
www.philly.com	865 words	477 words	32.91 %	100 %	49.52 %
www.sportingnews.com	1496 words	772 words	100 %	99.61 %	99.8 %
www.thefreedictionary.com	2605 words	1267 words	100 %	72.4 %	83.99 %
www.umm.edu	433 words	79 words	0.0 %	0.0 %	0.0 %
www.usatoday.com	1080 words	676 words	100 %	79.81 %	88.77 %
www.vidaextra.com	2381 words	257 words	90.17 %	90.56 %	90.36 %
www.wikipedia.org	3835 words	3657 words	99.95 %	100 %	99.97 %
www.wordreference.com	332 words	95 words	100 %	100 %	100 %
Average	2460 words	1109 words	73.52 %	83.84 %	73.82 %

Table 3: Benchmark results obtained with CLR

Benchmark	DOM words	Main block	Recall	Precision	F1
en.citizendium.org	4817 words	4568 words	99.98 %	98.09 %	99.03 %
googleblog.blogspot.com	801 words	737 words	100 %	99.59 %	99.79 %
us.gizmodo.com	901 words	518 words	98.46 %	100 %	99.22 %
www.applesfera.com	1626 words	1087 words	8.65 %	100 %	15.92 %
www.bbc.co.uk	1531 words	802 words	100 %	96.74 %	98.34 %
www.blogdecine.com	17647 words	1059 words	0.0 %	0.0 %	0.0 %
www.cbc.ca	1636 words	924 words	97.73 %	100 %	98.85 %
www.cbssports.com	3683 words	1836 words	100 %	100 %	100 %
www.cinemablend.com	827 words	263 words	100 %	68.49 %	81.3 %
www.cnn.com	1555 words	1065 words	97.65 %	81.25 %	88.7 %
www.elmundo.es	2365 words	1353 words	100 %	100 %	100 %
www.engadget.com	5150 words	3783 words	100 %	93.18 %	96.47 %
www.gizmologia.com	1264 words	202 words	91.58 %	100 %	95.6 %
www.healthopedia.com	765 words	205 words	45.85 %	100 %	62.87 %
www.microsiervos.com	1171 words	744 words	19.35 %	92.9 %	32.03 %
www.nationalfootballpost.com	822 words	205 words	99.51 %	100 %	99.75 %
www.news.cnet.com	5051 words	2396 words	98.12 %	100 %	99.05 %
www.nlm.nih.gov	956 words	642 words	12.31 %	100 %	21.92 %
www.people.com	1183 words	308 words	92.86 %	91.08 %	91.96 %
www.philly.com	865 words	477 words	32.91 %	100 %	49.52 %
www.sportingnews.com	1496 words	772 words	100 %	99.61 %	99.8 %
www.thefreedictionary.com	2605 words	1267 words	100 %	72.4 %	83.99 %
www.umm.edu	433 words	79 words	0.0 %	0.0 %	0.0 %
www.usatoday.com	1080 words	676 words	100 %	79.81 %	88.77 %
www.vidaextra.com	2381 words	257 words	90.17 %	90.56 %	90.36 %
www.wikipedia.org	3835 words	3657 words	99.95 %	100 %	99.97 %
www.wordreference.com	332 words	95 words	0.0 %	0.0 %	0.0 %
Average	2460 words	1109 words	73.52 %	83.84 %	73.82 %

Table 4: Benchmark results obtained with WLR

Ratio	Recall	Precision	F1
Average CNR	26.04 %	76.96 %	30.91 %
Average WNR	32.52 %	80.34 %	36.89 %
Average CLR	73.52 %	83.84 %	73.82 %
Average WLR	73.52 %	83.84 %	73.82 %

Table 5: Summary of results in the experiments

As shown in the experiments, the technique presents a high precision. However, sometimes the technique recovers some extra content. This extra content is usually around the main content and sometimes is a layer that the designer of the webpage inserted in the middle of the main content. It often shows a picture or video related to the main content, but sometimes it is just an advertisement. This is clearly a bad design policy used by the programmer that avoids to correctly reuse the code. However, we believe that it is done on purpose to ensure that those webpages that reuse the main content will force the user to see the advertisements. This is for instance the case of the benchmark www.philly.com. In this benchmark the main content is divided into two parts with a banner in the middle. This banner has produced that the tool only recognizes as main content one of the two parts, and for this reason F1 is 49.52%. In other benchmarks the tool failed with F1=0%. This problem was caused in all cases because there is some part of the webpage with a very high concentration of text that is not main content (e.g., the footer, a comment in a new, etc.), but was classified as the only main content.

All the information related to the experiments, the source code of the benchmarks, the source code of the tool and other material can be found at `http://users.dsic.upv.es/~jsilva/retrieval/`

6. Case of study

This section provides a real example of usage of our tool. Once installed in Firefox, the plugin adds to Firefox a new toolbar. This toolbar contains a button to extract the main content of a webpage. The technique works totally automatic because the user only have to press this button once a webpage has been loaded. Then, the technique uses the DOM tree internally loaded by the browser to select the best node. For instance, observe in Figure 6 the result produced after the main content has been extracted in the main webpage of CBS Sports. It has been perfectly filtered and both the horizontal menu and the vertical panel with advertisements have been removed, thus, the content extracted is exactly the main content.

In order to display the main content, the tool transforms the webpage to only display the relevant content identified in the DOM tree. The natural approach would be to construct a new webpage directly from the nodes that form the main content ignoring the other nodes (i.e., making the node selected by the technique the only child of `body` and removing the other nodes in the tree). However, this would produce a webpage where the original structure has been lost. This would happen because the space that was used by those elements removed would be

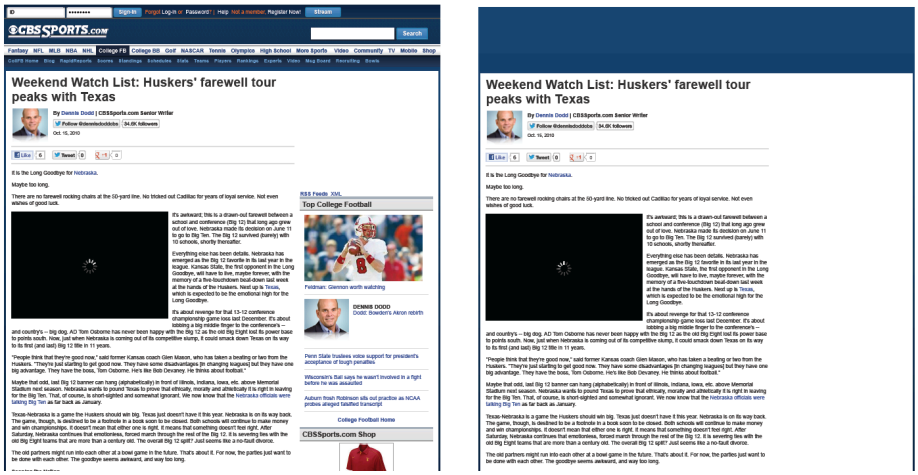


Figure 6: Main webpage of CBS Sports (left) and its filtered version (right)

reassigned and redistributed between the remaining elements. Tabular designs and designs based on divs would be destroyed.

Therefore, we cannot remove the filtered nodes. We only have to hide them. Hence, once the nodes of the main content have been identified, the tool traverses the DOM tree and, for each root of a subtree that does not belong to the main content, it sets the attribute *visibility* to *hidden*. With this property, the HTML elements still exist in the webpage and they fill their original space even though they are invisible.

7. Conclusions

Content extraction is useful not only for the final user, but also for many systems and tools such as indexers as a preliminary stage. It extracts the relevant part of a webpage allowing us to ignore the rest of content that can become useless, irrelevant, or even worst, noisy. In this work, we have presented a new technique for content extraction that uses the DOM structure of the webpage to identify the blocks that group those nodes with a higher proportion of text.

The DOM structure allows us to improve the detection of blocks, but it also allows us to discard parts of the webpage that have a big amount of textual information but belong to other HTML containers (i.e., they belong to a different DOM subtree). Our implementation and experiments have shown the usefulness of the technique presenting a high recall and precision.

The technique could be used not only for content extraction, but also for blocks detection. It could detect all blocks in a webpage by applying the presented method iteratively to detect one block after the other. In this way, we could detect the most relevant block; then, remove from the DOM tree all its nodes, and detect the next relevant block in the remaining DOM tree. This process would identify all blocks in relevant order. Another interesting open line of

research is using the technique to detect the menus of a webpage. A preliminary study showed that instead of using a ratio based on words, we could use a ratio based on hyperlinks to discover big concentrations of links in the DOM tree. If we collect those concentrations of links where the links contain one or two words, we will find the menus of the webpage.

References

[1] D. Gibson, K. Punera, A. Tomkins, The volume and evolution of web page templates, in: A. Ellis, T. Hagino (Eds.), Proceedings of the 14th International Conference on World Wide Web (WWW'05), ACM, 2005, pp. 830–839.

[2] S. Lopez, J. Silva, D. Insa, Using the DOM Tree for Content Extraction, in: J. Silva, F. Tiezzi (Eds.), Proceedings of the 8th International Workshop on Automated Specification and Verification of Web Systems (WWV 12), volume 98 of *EPTCS*, pp. 46–59.

[3] T. Gottron, Content Code Blurring: A New Approach to Content Extraction, in: A. M. Tjoa, R. R. Wagner (Eds.), Proceedings of the 19th International Workshop on Database and Expert Systems Applications (DEXA'08), IEEE Computer Society, 2008, pp. 29–33.

[4] T. Weninger, W. Henry Hsu, J. Han, CETR: Content Extraction via Tag Ratios, in: M. Rappa, P. Jones, J. Freire, S. Chakrabarti (Eds.), Proceedings of the 19th International Conference on World Wide Web (WWW'10), ACM, 2010, pp. 971–980.

[5] X. Li, B. Liu, Learning to Classify Texts Using Positive and Unlabeled Data, in: G. Gottlob, T. Walsh (Eds.), Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Morgan Kaufmann, 2003, pp. 587–594.

[6] J. Arias, K. Deschacht, M.-F. Moens, Language independent content extraction from web pages, in: R. Aly, C. Hauff, I. Hamer den, D. Hiemstra, T. Huibers, F. Jong de (Eds.), Proceedings of the 9th Dutch-Belgian Information Retrieval Workshop (DIR 09), number 09-01 in Workshop Proceedings Series, Centre for Telematics and Information Technology, Enschede, 2009, pp. 50–55.

[7] B. Krüpl, M. Herzog, W. Gatterbauer, Using visual cues for extraction of tabular data from arbitrary HTML documents, in: A. Ellis, T. Hagino (Eds.), Proceedings of the 14th International Conference on World Wide Web (WWW'05), ACM, 2005, pp. 1000–1001.

[8] A. Finn, N. Kushmerick, B. Smyth, Fact or Fiction: Content Classification for Digital Libraries, in: DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries, pp. 1–6.

[9] S. Gupta, G. E. Kaiser, D. Neistadt, P. Grimm, DOM-based content extraction of HTML documents, in: Proceedings of the 12th International Conference on World Wide Web (WWW'03), ACM, 2003, pp. 207–214.

[10] B. B. Dalvi, W. Cohen, J. Callan, WebSets: Extracting sets of entities from the web using unsupervised information extraction, in: E. Adar, J. Teevan, E. Agichtein, Y. Maarek (Eds.), Proceedings of the 5th International Conference on Web Search and Web Data Mining (WSDM 12), ACM, 2012, pp. 243–252.

[11] N. Kushmerick, D. S. Weld, R. B. Doorenbos, Wrapper Induction for Information Extraction, in: M. E. Pollack (Ed.), Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97), Morgan Kaufmann, 1997, pp. 729–737.

[12] W. Cohen, M. Hurst, L. Jensen, A flexible learning system for wrapping tables and lists in HTML documents, in: Proceedings of the 11th International Conference on World Wide Web (WWW'02), ACM, 2002, pp. 232–241.

[13] C. Kohlschütter, 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, A. Chowdhury (Eds.), Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08), ACM, 2008, pp. 1173–1182.

[14] C. Kohlschütter, A densitometric analysis of web template content, in: J. Quemada, G. León, Y. S. Maarek, W. Nejdl (Eds.), Proceedings of the 18th International Conference on World Wide Web (WWW'09), ACM, 2009, pp. 1165–1166.

[15] S. Baluja, Browsing on small screens: Recasting web-page segmentation into an efficient machine learning framework, in: L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, M. Dahlin (Eds.), Proceedings of the 15th International Conference on World Wide Web (WWW 06), ACM, 2006, pp. 33–42.

[16] J. Gibson, B. Wellner, S. Lubar, Adaptive web-page content identification, in: I. Fundulaki, N. Polyzotis (Eds.), Proceedings of the 9th ACM International Workshop on Web Information and Data Management (WIDM'07), ACM, 2007, pp. 105–112.

[17] C. Kohlschütter, P. Fankhauser, W. Nejdl, Boilerplate detection using shallow text features, in: B. D. Davison, T. Suel, N. Craswell, B. Liu (Eds.), Proceedings of the 3th International Conference on Web Search and Web Data Mining (WSDM'10), ACM, 2010, pp. 441–450.

[18] W. Consortium, Document Object Model (DOM), Available from URL: `http://www.w3.org/{DOM}/`, 1997.

[19] T. Gottron, Evaluating content extraction on HTML documents, in: V. Grout, D. Oram, R. Picking (Eds.), Proceedings of the 2nd International Conference on Internet Technologies and Applications (ITA'07), National Assembly for Wales, 2007, pp. 123–132.