Web Template Extraction Based on Hyperlink Analysis

Julián Alarte David Insa Josep Silva
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València, Valencia, Spain
jualal@doctor.upv.es dinsa@dsic.upv.es jsilva@dsic.upv.es

Salvador Tamarit
Babel Research Group
Universidad Politécnica de Madrid, Madrid, Spain
stamarit@babel.ls.fi.upm.es

Web templates are one of the main development resources for website engineers. Templates allow them to increase productivity by plugin content into already formatted and prepared pagelets. For the final user templates are also useful, because they provide uniformity and a common look and feel for all webpages. However, from the point of view of crawlers and indexers, templates are an important problem, because templates usually contain irrelevant information such as advertisements, menus, and banners. Processing and storing this information is likely to lead to a waste of resources (storage space, bandwidth, etc.). It has been measured that templates represent between 40% and 50% of data on the Web. Therefore, identifying templates is essential for indexing tasks. In this work we propose a novel method for automatic template extraction that is based on similarity analysis between the DOM trees of a collection of webpages that are detected using menus information. Our implementation and experiments demonstrate the usefulness of the technique.

1 Introduction

A web template (in the following just template) is a prepared HTML page where formatting is already implemented and visual components are ready so that we can insert content into them. Templates are used as a basis for composing new webpages that share a common look and feel. This is good for web development because many tasks can be automated thanks to the reuse of components. In fact, many websites are maintained automatically by code generators that generate webpages using templates. Templates are also good for users, which can benefit from intuitive and uniform designs with a common vocabulary of colored and formatted visual elements.

Templates are also important for crawlers and indexers, because they usually judge the relevance of a webpage according to the frequency and distribution of terms and hyperlinks. Since templates contain a considerable number of common terms and hyperlinks that are replicated in a large number of webpages, relevance may turn out to be inaccurate, leading to incorrect results (see, e.g., [1, 16, 18]). Moreover, in general, templates do not contain relevant content, they usually contain one or more pagelets [5, 1] (i.e., self-contained logical regions with a well defined topic or functionality) where the main content must be inserted. Therefore, detecting templates can allow indexers to identify the main content of the webpage.

Modern crawlers and indexers do not treat all terms in a webpage in the same way. Webpages are preprocessed to identify the template because template extraction allows them to identify those pagelets that only contain noisy information such as advertisements and banners. This content should not be indexed in the same way as the relevant content. Indexing the non-content part of templates not only affects accuracy, it also affects performance and can lead to a waste of storage space, bandwidth, and time.

S. Escobar (Ed.): XIV Jornadas sobre Programación
y Lenguajes, PROLE 2014, Revised Selected Papers
EPTCS 173, 2015, pp. 16–26, doi:10.4204/EPTCS.173.2

Template extraction helps indexers to isolate the main content. This allows us to enhance indexers by assigning higher weights to the really relevant terms. Once templates have been extracted, they are processed for indexing—they can be analyzed only once for all webpages using the same template—. Moreover, links in templates allow indexers to discover the topology of a website (e.g., through navigational content such as menus), thus identifying the main webpages. They are also essential to compute pageranks.

Gibson et al. [8] determined that templates represent between 40% and 50% of data on the Web and that around 30% of the visible terms and hyperlinks appear in templates. This justifies the importance of template removal [18, 16] for web mining and search.

Our approach to template extraction is based on the DOM [6] structures that represent webpages. Roughly, given a webpage in a website, we first identify a set of webpages that are likely to share a template with it, and then, we analyze these webpages to identify the part of their DOM trees that is common with the original webpage. This slice of the DOM tree is returned as the template.

Our technique introduces a new idea to automatically find a set of webpages that potentially share a template. Roughly, we detect the template's menu and analyze its links to identify a set of mutually linked webpages. One of the main functions of a template is in aiding navigation, thus almost all templates provide a large number of links, shared by all webpages implementing the template. Locating the menu allows us to identify in the topology of the website the main webpages of each category or section. These webpages very likely share the same template. This idea is simple but powerful and, contrarily to other approaches, it allows the technique to only analyze a reduced set of webpages to identify the template.

The rest of the paper has been structured as follows: In Section 2 we discuss the state of the art and show some problems of current techniques that can be solved with our approach. In Section 3 we provide some preliminary definitions and useful notation. Then, in Section 4, we present our technique with examples and explain the algorithms used. In Section 5 we give some details about the implementation and show the results obtained from a collection of benchmarks. Finally, Section 6 concludes.

2 Related Work

Template detection and extraction are hot topics due to their direct application to web mining, searching, indexing, and web development. For this reason, there are many approaches that try to face this problem. Some of them are especially thought for boilerplate removal and content extraction; and they have been presented in the CleanEval competition [2], which proposes a collection of examples to be analyzed with a gold standard.

Content Extraction is a discipline very close to template extraction. Content extraction tries to isolate the pagelet with 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., [9, 17, 4, 10]), and all of them are directly related to template extraction.

In the area of template extraction, there are three different ways to solve the problem, namely, (i) using the textual information of the webpage (i.e., the HTML code), (ii) using the rendered image of the webpage in the browser, and (iii) using the DOM tree of the webpage.

The first approach analyzes the plain HTML code, and it is based on the idea that the main content of the webpage has more density of text, with less labels. For instance, the main content can be identified selecting the largest contiguous text area with the least amount of HTML tags [7]. This has been measured directly on the HTML code by counting the number of characters inside text, and characters inside labels. This measure produces a ratio called CETR [17] used to discriminate the main content.

Other approaches exploit densitometric features based on the observation that some specific terms are more common in templates [13, 11]. The distribution of the code between the lines of a webpage is not necessarily the one expected by the user. The format of the HTML code can be completely unbalanced (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 distributed in only a few lines without any legible structure. In this kind of webpages CETR is useless.

The second approach assumes that the main content of a webpage is often located in the central part and (at least partially) visible without scrolling [3]. This approach has been less studied because rendering webpages for classification is a computational expensive operation [12].

The third approach is where our technique falls. While some works try to identify pagelets analyzing the DOM tree with heuristics [1], others try to find common subtrees in the DOM trees of a collection of webpages in the website [18, 16]. Our technique is similar to these last two works.

Even though [18] uses a method for template extraction, its main goal is to remove redundant parts of a website. For this, they use the Site Style Tree (SST), a data structure that is constructed by analyzing a set of DOM trees and recording every node found, so that repeated nodes are identified by using counters in the SST nodes. Hence, an SST summarizes a set of DOM trees. After the SST is built, they have information about the repetition of nodes. The most repeated nodes are more likely to belong to a noisy part that is removed from the webpages. Unfortunately, this approach does not use any method to detect the webpages that share the same template. They are randomly selected, and this can negatively influence the performance and the precision of the technique.

In [16], the approach is based on discovering optimal mappings between DOM trees. This mapping relates nodes that are considered redundant. Their technique uses the RTDM-TD algorithm to compute a special kind of mapping called *restricted top-down mapping* [14]. Their objective, as ours, is template extraction, but there are two important differences. First, we compute another kind of mapping to identify redundant nodes. Our mapping is more restrictive because it forces all nodes that form pairs in the mapping to be equal. Second, in order to select the webpages of the website that should be mapped to identify the template, they pick random webpages until a threshold is reached. In their experiments, they approximated this threshold as a few dozens of webpages. In our technique, we do not select the webpages randomly, we use a method to identify the webpages linked by the main menu of the website because they very likely contain the template. We only need to explore a few webpages to identify the webpages that implement the template. Moreover, contrarily to us, they assume that all webpages in the website share the same template, and this is a strong limitation for many websites.

3 Preliminaries

The Document Object Model (DOM) [6] 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 CSS styles) hierarchically. This means that a table that contains another table is represented with a node with a child that represents the internal table.

In the following, webpages are represented with a DOM tree $T = (N, E)$ where N is a finite set of

nodes and E is a set of edges between nodes in N (see Figure 1). $root(T)$ denotes the root node of T. Given a node $n \in N$, $link(n)$ denotes the hyperlink of n when n is a node that represents a hyperlink (HTML label `<a>`). $parent(n)$ represents node $n' \in N$ such that $(n', n) \in E$. Similarly, $children(n)$ represents the set $\{n' \in N \mid (n, n') \in E\}$. $subtree(n)$ denotes the subtree of T whose root is $n \in N$. $path(n)$ is a non-empty sequence of nodes that represents a *DOM path*; it can be defined as $path(n) = n_0 n_1 \ldots n_m$ such that $\forall i, 0 \leq i < m.\ n_i = parent(n_{i+1})$.

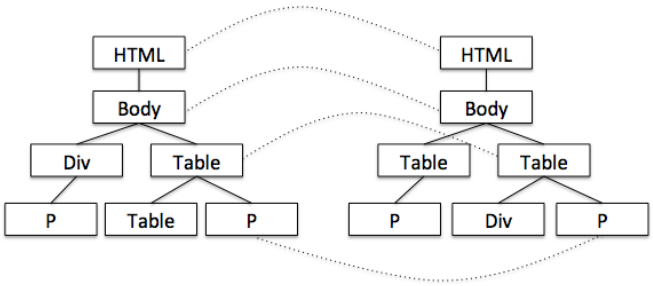

Figure 1: Equal top-down mapping between DOM trees

In order to identify the part of the DOM tree that is common in a set of webpages, our technique uses an algorithm that is based on the notion of mapping. A mapping establishes a correspondence between the nodes of two trees.

Definition 3.1 *(based on Kuo's definition of mapping [15]) A* mapping *from a tree $T = (N, E)$ to a tree $T' = (N', E')$ is any set M of pairs of nodes $(n, n') \in M$, $n \in N, n' \in N'$ such that, for any two pairs (n_1, n'_1) and $(n_2, n'_2) \in M$, $n_1 = n_2$ iff $n'_1 = n'_2$.*

In order to identify templates, we are interested in a very specific kind of mapping that we call *equal top-down mapping* (ETDM).

Definition 3.2 *Given an equality relation $\triangleq$ between tree nodes, a mapping M between two trees T and T' is said to be* equal top-down *if and only if*

- *equal: for every pair $(n, n') \in M$, $n \triangleq n'$.*
- *top-down: for every pair $(n, n') \in M$, with $n \neq root(T)$ and $n' \neq root(T')$, there is also a pair $(parent(n), parent(n')) \in M$.*

Note that this definition is parametric with respect to the equality function $\triangleq : N \times N' \to [0..1]$ where N and N' are sets of nodes of two (often different) DOM trees. We could simply use the standard equality (=), but we left this relation open, to be general enough as to cover any possible implementation. In particular, other techniques consider that two nodes n_1 and n_2 are equal if they have the same label. However, in our implementation we use a notion of node equality much more complex that uses the label of the node, its classname, its HTML attributes, its children, its position in the DOM tree, etc.

This definition of mapping allows us to be more restrictive than other mappings such as, e.g., the *restricted top-down mapping* (RTDM) introduced in [14]. While RTDM permits the mapping of different nodes (e.g., a node labelled with *table* with a node labelled with *div*), ETDM can force all pairwise mapped nodes to have the same label. Figure 1 shows an example of an ETDM using: $n \triangleq n'$ if and only if n and n' have the same label. We can now give a definition of template using ETDM.

Figure 2: Webpages of BBC sharing a template

Definition 3.3 *Let p_0 be a webpage whose associated DOM tree is $T_0 = (N_0, E_0)$, and let $P = \{p_1 \dots p_n\}$ be a collection of webpages with associated DOM trees $\{T_1 \dots T_n\}$. A* template *of p_0 with respect to P is a tree (N, E) where*

- *nodes: $N = \{n \in N_0 \mid \forall i, 1 \leq i \leq n \;.\; (n, \_) \in M_{T_0,T_i}\}$ where M_{T_0,T_i} is an equal top-down mapping between trees T_0 and T_i.*
- *edges: $E = \{(m,m') \in E_0 \mid m,m' \in N\}$.*

Hence, the template of a webpage is computed with respect to a set of webpages (usually webpages in the same website). We formalize the template as a new webpage computed with an ETDM between the initial webpage and all the other webpages.

4 Template extraction

Templates are often composed of a set of pagelets. Two of the most important pagelets in a webpage are the menu and the main content. For instance, in Figure 2 we see two webpages that belong to the "News" portal of BBC. At the top of the webpages we find the main menu containing links to all BBC portals. We can also see a submenu under the big word "News". The left webpage belongs to the "Technology" section, while the right webpage belongs to the "Science & Environment" section. Both share the same menu, submenu, and general structure. In both pages the news are inside the pagelet in the dashed square. Note that this pagelet contains the main content and, thus, it should be indexed with a special treatment. In addition to the main content, there is a common pagelet called "Top Stories" with the most relevant news, and another one called "Features and Analysis".

Our technique inputs a webpage (called key page) and it outputs its template. To infer the template, it analyzes some webpages from the (usually huge) universe of directly or indirectly linked webpages. Therefore, we need to decide what concrete webpages should be analyzed. Our approach is very simple yet powerful:

1. Starting from the key page, it identifies a complete subdigraph in the website topology, and then
2. it extracts the template by calculating an ETDM between the DOM tree of the key page and some of the DOM trees of the webpages in the complete subdigraph.

Both processes are explained in the following sections.

4.1 Finding a complete subdigraph in a website topology

Given a website topology, a complete subdigraph (CS) represents a collection of webpages that are pairwise mutually linked. A n-complete subdigraph (n-CS) is formed by n nodes. Our interest in complete subdigraphs comes from the observation that the webpages linked by the items in a menu usually form a CS. This is a new way of identifying the webpages that contain the menu. At the same time, these webpages are the roots of the sections linked by the menu. The following example illustrates why menus provide very useful information about the interconnection of webpages in a given website.

Example 4.1 *Consider the BBC website. Two of its webpages are shown in Figure 2. In this website all webpages share the same template, and this template has a main menu that is present in all webpages, and a submenu for each item in the main menu. The site map of the BBC website may be represented with the topology shown in Figure 3.*

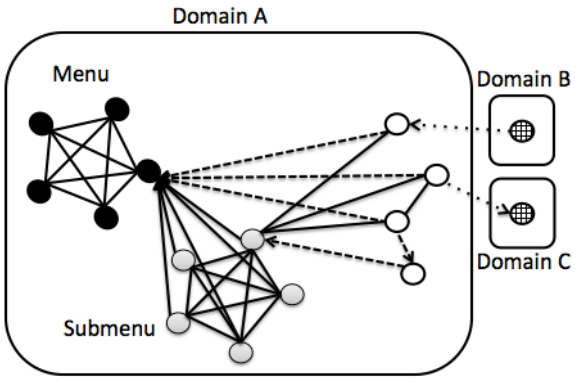

Figure 3: BBC Website topology

In this figure, each node represents a webpage and each edge represents a link between two webpages. Solid edges are bidirectional, and dashed and dotted edges are directed. Black nodes are the webpages pointed by the main menu. Because the main menu is present in all webpages, then all nodes are connected to all black nodes (we only draw some of the edges for clarity). Therefore all black nodes together form a complete graph (i.e., there is an edge between each pair of nodes). Grey nodes are the webpages pointed by a submenu, thus, all grey nodes together also form a complete graph. White nodes are webpages inside one of the categories of the submenu, therefore, all of them have a link to all black and all grey nodes.

Of course, not all webpages in a website implement the same template, and some of them only implement a subset of a template. For this reason, one of the main problems of template extraction is deciding what webpages should be analyzed. Minimizing the number of webpages analyzed is essential to reduce the web crawlers work. In our technique we introduce a new idea to select the webpages that must be analyzed: we identify a menu in the key page and we analyze the webpages pointed out by this menu. Observe that we only need to investigate the webpages linked by the key page, because they will for sure contain a CS that represents the menu.

In order to increase precision, we search for a CS that contains enough webpages that implement the template. This CS can be identified with Algorithm 1.

This algorithm inputs a webpage and the size n of the CS to be computed. We have empirically approximated the optimal value for n, which is 4. The algorithm uses two trivial functions:

Algorithm 1 Extract a n-CS from a website

Input: An *initialLink* that points to a webpage and the expected size n of the CS.
Output: A set of links to webpages that together form a n-CS.
If a n-CS cannot be formed, then they form the biggest m-CS with m < n.

begin
$keyPage = loadWebPage(initialLink)$;
$reachableLinks = getLinks(keyPage)$;
$processedLinks = \emptyset$;
$connections = \emptyset$;
$bestCS = \emptyset$;
foreach $link$ **in** $reachableLinks$
$webPage = loadWebPage(link)$;
$existingLinks = getLinks(webPage) \cap reachableLinks$;
$processedLinks = processedLinks \cup \{link\}$;
$connections = connections \cup \{(link \rightarrow existingLink) \mid existingLink \in existingLinks\}$;
$CS = \{ls \in \mathscr{P}(processedLinks) \mid link \in ls \wedge \forall l, l' \in ls \,.\, (l \rightarrow l'), (l' \rightarrow l) \in connections\}$;
$maximalCS = cs \in CS$ such that $\forall cs' \in CS \,.\, |cs| \geq |cs'|$;
if $|maximalCS| = n$ **then return** $maximalCS$;
if $|maximalCS| > |bestCS|$ **then** $bestCS = maximalCS$;
return $bestCS$;
end

$loadWebPage(link)$, which loads and returns the webpage pointed by the input link, and $getLinks(webpage)$, which returns the collection of (non-repeated) links[1] in the input webpage (ignoring self-links). Observe that the main loop iteratively explores the links of the webpage pointed by the *initialLink* (i.e., the key page) until it founds a n-CS. Note also that it only loads those webpages needed to find the n-CS, and it stops when the n-CS has been found. We want to highlight the mathematical expression

$$CS = \{ls \in \mathscr{P}(processedLinks) \mid link \in ls \wedge \forall l, l' \in ls \,.\, (l \rightarrow l'), (l' \rightarrow l) \in connections\},$$

where $\mathscr{P}(X)$ returns all possible partitions of set X.

It is used to find the set of all CS that can be constructed with the current $link$. Here, *processedLinks* contains the set of links that have been already explored by the algorithm. And *connections* is the set of all links between the webpages pointed by *processedLinks*. *connections* is used to identify the CS. Therefore, the set *CS* is composed of the subset of *processedLinks* that form a CS using *connections*.

Observe that the current link must be part of the CS ($link \in ls$) to ensure that we make progress (not repeating the same search of the previous iteration). Moreover, because the CS is constructed incrementally, the statement

if $|maximalCS| = n$ **then return** $maximalCS$

ensures that whenever a n-CS can be formed, it is returned.

4.2 Template extraction from a complete subdigraph

After we have found a set of webpages mutually linked and linked by the menu of the site (the complete subdigraph), we identify an ETDM between the key page and all webpages in the set. For this, initially, the template is considered to be the key page. Then, we compute an ETDM between the template and one webpage in the set. The result is the new refined template, that is further refined with another ETDM with another webpage, and so on until all webpages have been processed. This process is formalized in Algorithm 2, that uses function *ETDM* to compute the biggest ETDM between two trees.

[1] In our implementation, this function removes those links that point to other domains because they are very unlikely to contain the same template.

Algorithm 2 Extract a template from a set of webpages

Input: A key page $p_k = (N_1, E_1)$ and a set of n webpages P.
Output: A template for p_k with respect to P.

begin
  $template = p_k$;
  foreach (p **in** P)
    if $root(p_k) \triangleq root(p)$
      $template = ETDM(template, p)$;
  return $template$;
end

function $ETDM$(tree $T_1 = (N_1, E_1)$, tree $T_2 = (N_2, E_2)$)
  $r_1 = root(T_1)$;
  $r_2 = root(T_2)$;
  nodes = $\{r_1\}$;
  edges = $\emptyset$;
  foreach $n_1 \in N_1 \backslash nodes, n_2 \in N_2 \backslash nodes \;.\; n_1 \underset{max}{\triangleq} n_2, (r_1, n_1) \in E_1$ and $(r_2, n_2) \in E_2$
    $(nodes\_st, edges\_st) = ETDM(subtree(n_1), subtree(n_2))$;
    $nodes = nodes \cup nodes\_st$;
    $edges = edges \cup edges\_st \cup \{(r_1, n_1)\}$;
  return $(nodes, edges)$;

As in Definition 3.2, we left the algorithm parametric with respect to the equality function $\triangleq$. This is done on purpose, because this relation is the only parameter that is subjective and thus, it is a good design decision to leave it open. For instance, a researcher can decide that two DOM nodes are equal if they have the same label and attributes. Another researcher can relax this restriction ignoring some attributes (i.e, the template can be the same, even if there are differences in colors, sizes, or even positions of elements. It usually depends on the particular use of the extracted template). Clearly, $\triangleq$ has a direct influence on the precision and recall of the technique. The more restrictive, the more precision (and less recall). Note also that the algorithm uses $n_1 \underset{max}{\triangleq} n_2$ to indicate that n_1 and n_2 maximize function $\triangleq$.

In our implementation, function $\triangleq$ is defined with a ponderation that compares two nodes considering their classname, their number of children, their relative position in the DOM tree, and their HTML attributes. We refer the interested reader to our open and free implementation (`http://www.dsic.upv.es/~jsilva/retrieval/templates/`) where function $\triangleq$ is specified.

5 Implementation

The technique presented in this paper, including all the algorithms, has been implemented as a Firefox's plugin. In this tool, the user can browse on the Internet as usual. Then, when he/she wants to extract the template of a webpage, he/she only needs to press the "Extract Template" button and the tool automatically loads the appropriate linked webpages to form a CS, analyzes them, and extracts the template. The template is then displayed in the browser as any other webpage. For instance, the template extracted for the webpages in Figure 2 contains the whole webpage except for the part inside the dashed box.

Our implementation and all the experimentation is public. All the information of the experiments, the source code of the benchmarks, the source code of the tool, and other material can be found at:

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

6 Conclusions

Web templates are an important tool for website developers. By automatically inserting content into web templates, website developers, and content providers of large web portals achieve high levels of productivity, and they produce webpages that are more usable thanks to their uniformity.

This work presents a new technique for template extraction. The technique is useful for website developers because they can automatically extract a clean HTML template of any webpage. This is particularly interesting to reuse components of other webpages. Moreover, the technique can be used by other systems and tools such as indexers or wrappers as a preliminary stage. Extracting the template allows them to identify the structure of the webpage and the topology of the website by analyzing the navigational information of the template. In addition, the template is useful to identify pagelets, repeated advertisement panels, and what is particularly important, the main content.

Our technique uses the menus of a website to identify a set of webpages that share the same template with a high probability. Then, it uses the DOM structure of the webpages to identify the blocks that are common to all of them. These blocks together form the template. To the best of our knowledge, the idea of using the menus to locate the template is new, and it allows us to quickly find a set of webpages from which we can extract the template. This is especially interesting for performance, because loading webpages to be analyzed is expensive, and this part of the process is minimized in our technique. As an average, our technique only loads 7 pages to extract the template.

This technique could be also used for content extraction. Detecting the template of a webpage is very helpful to detect the main content. Firstly, the main content must be formed by DOM nodes that do not belong to the template. Secondly, the main content is usually inside one of the pagelets that are more centered and visible, and with a higher concentration of text.

For future work, we plan to investigate a strategy to further reduce the amount of webpages loaded with our technique. The idea is to directly identify the menu in the key page by measuring the density of links in its DOM tree. The menu has probably one of the higher densities of links in a webpage. Therefore, our technique could benefit from measuring the links–DOM nodes ratio to directly find the menu in the key page, and thus, a complete subdigraph in the website topology.

7 Acknowledgements

This work has been partially supported by the EU (FEDER) and the Spanish *Ministerio de Economía y Competitividad (Secretaría de Estado de Investigación, Desarrollo e Innovación)* under Grant TIN2013-44742-C4-1-R and by the *Generalitat Valenciana* under Grant PROMETEO/2011/052. David Insa was partially supported by the Spanish Ministerio de Educación under FPU Grant AP2010-4415. Salvador Tamarit was partially supported by research project POLCA, Programming Large Scale Heterogeneous Infrastructures (610686), funded by the European Union, STREP FP7.

References

[1] Ziv Bar-Yossef & Sridhar Rajagopalan (2002): *Template detection via data mining and its applications*. In: *Proceedings of the 11th International Conference on World Wide Web (WWW'02)*, ACM, New York, NY, USA, pp. 580–591, doi:10.1145/511446.511522.

[2] Marco Baroni, Francis Chantree, Adam Kilgarriff & Serge Sharoff (2008): *Cleaneval: a Competition for Cleaning Web Pages*. In: *Proceedings of the International Conference on Language Resources and*

Evaluation (LREC'08), European Language Resources Association, pp. 638–643. Available at `http://www.lrec-conf.org/proceedings/lrec2008/summaries/162.html`.

[3] Radek Burget & Ivana Rudolfova (2009): *Web Page Element Classification Based on Visual Features*. In: *Proceedings of the 1st Asian Conference on Intelligent Information and Database Systems (ACIIDS'09)*, IEEE Computer Society, Washington, DC, USA, pp. 67–72, doi:10.1109/ACIIDS.2009.71.

[4] Eduardo Cardoso, Iam Jabour, Eduardo Laber, Rogério Rodrigues & Pedro Cardoso (2011): *An efficient language-independent method to extract content from news webpages*. In: *Proceedings of the 11th ACM symposium on Document Engineering (DocEng'11)*, ACM, New York, NY, USA, pp. 121–128, doi:10.1145/2034691.2034720.

[5] Soumen Chakrabarti (2001): *Integrating the Document Object Model with hyperlinks for enhanced topic distillation and information extraction*. In: *Proceedings of the 10th International Conference on World Wide Web (WWW'01)*, ACM, New York, NY, USA, pp. 211–220, doi:10.1145/371920.372054.

[6] W3C Consortium (1997): *Document Object Model (DOM)*. Available from URL: `http://www.w3.org/{DOM}/`.

[7] Adriano Ferraresi, Eros Zanchetta, Marco Baroni & Silvia Bernardini (2008): *Introducing and evaluating ukWaC, a very large web-derived corpus of english*. In: *Proceedings of the 4th Web as Corpus Workshop (WAC-4)*, pp. 47–54.

[8] David Gibson, Kunal Punera & Andrew Tomkins (2005): *The volume and evolution of web page templates*. In Allan Ellis & Tatsuya Hagino, editors: *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*, ACM, pp. 830–839, doi:10.1145/1062745.1062763.

[9] Thomas Gottron (2008): *Content Code Blurring: A New Approach to Content Extraction*. In A. Min Tjoa & Roland R. Wagner, editors: *Proceedings of the 19th International Workshop on Database and Expert Systems Applications (DEXA'08)*, IEEE Computer Society, pp. 29–33, doi:10.1109/DEXA.2008.43.

[10] David Insa, Josep Silva & Salvador Tamarit (2013): *Using the words/leafs ratio in the DOM tree for content extraction*. *The Journal of Logic and Algebraic Programming* 82(8), pp. 311–325, doi:10.1016/j.jlap.2013.01.002.

[11] Christian Kohlschütter (2009): *A densitometric analysis of web template content*. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek & Wolfgang Nejdl, editors: *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, ACM, pp. 1165–1166, doi:10.1145/1526709.1526909.

[12] Christian Kohlschütter, Peter Fankhauser & Wolfgang Nejdl (2010): *Boilerplate detection using shallow text features*. In Brian D. Davison, Torsten Suel, Nick Craswell & Bing Liu, editors: *Proceedings of the 3th International Conference on Web Search and Web Data Mining (WSDM'10)*, ACM, pp. 441–450, doi:10.1145/1718487.1718542.

[13] Christian Kohlschütter & Wolfgang Nejdl (2008): *A densitometric approach to web page segmentation*. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi & Abdur Chowdhury, editors: *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*, ACM, pp. 1173–1182, doi:10.1145/1458082.1458237.

[14] Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares Silva & Alberto Henrique Frade Laender (2004): *Automatic web news extraction using tree edit distance*. In: *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*, ACM, New York, NY, USA, pp. 502–511, doi:10.1145/988672.988740.

[15] Kuo Chung Tai (1979): *The Tree-to-Tree Correction Problem*. *Journal of the ACM* 26(3), pp. 422–433, doi:10.1145/322139.322143.

[16] Karane Vieira, Altigran S. da Silva, Nick Pinto, Edleno S. de Moura, João M. B. Cavalcanti & Juliana Freire (2006): *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)*, ACM, New York, NY, USA, pp. 258–267, doi:10.1145/1183614.1183654.

[17] Tim Weninger, William Henry Hsu & Jiawei Han (2010): *CETR: Content Extraction via Tag Ratios*. In Michael Rappa, Paul Jones, Juliana Freire & Soumen Chakrabarti, editors: *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*, ACM, pp. 971–980, doi:10.1145/1772690.1772789.

[18] Lan Yi, Bing Liu & Xiaoli Li (2003): *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)*, ACM, New York, NY, USA, pp. 296–305, doi:10.1145/956750.956785.