Template extraction based on menu information

Julian Alarte  David Insa  Josep Silva  Salvador Tamarit
Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación
Camino de Vera s/n, E-46022, Valencia, Spain.
jalarte@dsic.upv.es  dinsa@dsic.upv.es  jsilva@dsic.upv.es  stamarit@dsic.upv.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 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 is a prepared HTML page where formatting is already implemented and visual components are ready to insert content. Web 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, which generate webpages using templates. Web templates are also good for users, which can benefit from intuitive and uniform designs with a common vocabulary of colored and formatted visual elements.

Contrarily, web templates are an important problem for crawlers and indexers, because they 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 [1,15,17]. 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. The main content of a webpage is often complementary to its template. Therefore, detecting templates can allow indexers to identify the main content that is usually inside a specific pagelet of the template.

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. Indexing the non-content part of templates not only affects accuracy, it also affects performance and is, in general, a waste of storage space, bandwidth and time.

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 should
be processed for indexing. 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. [7] 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 [17, 15] for web mining and search.

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

The technique exploits a new idea to automatically find a set of webpages that potentially share a web template. Roughly, we detect the template’s menu and analyze the links of the menu to identify a set of mutually linked webpages. One of the main functions of a template is in aiding navigation, thus almost all web 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 web 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 have been presented in the CleanEval competition [2], which periodically proposes a collection of examples to be analyzed with a gold standard. The examples proposed are especially though for boilerplate removal, and content extraction.

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, 16, 4]), and all of them are directly related to template extraction.

In the area of template extraction, there are three main 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 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 [6]. This have been measured directly on the HTML code by counting the number of characters inside text, and characters inside labels. This measure produce a ratio called CETR [16] used to discriminate the main content. Other approaches exploit densitometric features based on the observation that some specific terms are more common in templates [12, 10].

The second approach assumes that the main content of a webpage use to be 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 [11].

The third approach is where our technique falls. While some works try to identify pagelets analyzing the DOM tree with heuristics [11], others try to find common subtrees in the DOM trees of a collection of webpages in the website [17, 15]. Our technique is similar to these last two works.

Even though [17] 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.

In [15], 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 [13]. 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. In their experiments, they approximated this threshold as a few dozen 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 highly likely contain the template. We only need to explore a few webpages to identify them. Moreover, contrarily to us, they assume that all webpages in the website share the same template, and this is a strong restriction for many websites.

3 Preliminaries

In this paper webpages are represented with a DOM tree \( T = (N, E) \) where nodes are labelled with their corresponding HTML tags (see Figure 1). \( |T| \) denotes the number of nodes in \( T \). \( T[i] \) denotes the \( i \) node of \( T \) whose position in a preorder traversal of \( T \) is \( i \). \( \text{root}(T) \) denotes the root node of \( T \). Given a node \( T[i] \in N \), \( \text{parent}(i) \) represents node \( T[j] \) such that \( (T[j], T[i]) \in E \). \( \text{subtree}(T[i]) \) denotes the subtree of \( T \) whose root is \( T[i] \).

![Figure 1: Top-down restricted mapping between DOM trees](image)

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 [14] A mapping from a tree $T$ to a tree $T'$ is any set $M$ of pairs of integers satisfying:

1. For any pair $(i, j)$ in $M$,
   
   (a) $1 \leq i \leq |T|, 1 \leq j \leq |T'|$.

2. For any two pairs $(i_1, j_1)$ and $(i_2, j_2)$ in $M$,
   
   (a) $i_1 = i_2$ iff $j_1 = j_2$;
   (b) $i_1 < i_2$ iff $j_1 < j_2$;
   (c) $T[i_1]$ is an ancestor (descendant) of $T[i_2]$ iff $T'[j_1]$ is an ancestor (descendant) of $T'[j_2]$.

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

Definition 3.2 A mapping $M$ between two trees $T_1$ and $T_2$ is said to be exact top-down if and only if

- exact: for every pair $(i, j) \in M$, $T_1[i] = T_2[j]$.
- top-down: for every pair $(i, j) \in M$, with $T_1[i] \neq \text{root}(T_1)$ and $T_2[j] \neq \text{root}(T_2)$, there is also a pair $(\text{parent}(i), \text{parent}(j)) \in M$.

This mapping is even more restrictive than other mappings such as, e.g., the restricted top-down mapping (RTDM) introduced in [13]. While RTDM permits the mapping of different nodes (e.g., a node labelled with \textit{table} with a node labelled with \textit{div}), ETDM only allows the mapping of nodes that are equal. Figure 1 shows an example of a ETDM. We can now give a definition of template using ETDM.

Definition 3.3 Let $P = \{p_1 \ldots p_n\}$ be a collection of webpages. A template of $P$ is a tree $T = (N, E)$ where

- nodes: $N = \{p_j[m] \mid \forall p_j, p_k, 1 \leq j \neq k \leq n . (m, m') \in M_{j,k} \}$ where $M_{j,k}$ is a exact top-down mapping between $p_j$ and $p_k$.
- edges: $E = \{(a, b) \in \bigcap_{1 \leq i \leq n} E_i \mid a, b \in N \land p_i = (N_i, E_i)\}$.

We formalize a template of a set of webpages as a new webpage extracted from their intersection (computed by means of a ETDM between all the webpages).

4 Template extraction

Web 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 see 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 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”.

In our technique we input a webpage (called key page) and we output its template. To infer the template, we analyze some webpages from the (usually huge) universe of related webpages. Therefore, we need to decide what related webpages should be analyzed. Our approach is very simple yet powerful:
1. Starting from the key page, we identify a complete subdigraph in the website topology and then
2. we extract the template by comparing the DOM tree of the key page with the DOM trees of the
   nodes in the complete subdigraph. The template is the intersection between the key page and all
   DOM trees in the subdigraph. This intersection is computed with an ETDM between the DOM
trees.

4.1 Finding a complete subdigraph in a website topology

Given a website topology, a complete subdigraph (CS) is a collection of webpages that are pairwise
mutually linked. It is a n-complete subdigraph (n-CS) if it is formed by n nodes. The interest in complete
subdigraphs comes from the fact that the webpages linked by the items in a menu usually form a CS.
This idea is original and is a way of identifying the webpages that contain the menu and, at the same
time, they 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 a website where 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. This is the
case of, e.g., the BBC webpages shown in Figure 2. The site map of this website may be represented with
the topology shown in Figure 3.

In this figure, each node represents a webpage and each edge represents a link between two webpages
(we only draw some of the edges for clarity). 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. 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 the 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

---

1In our implementation, we restrict our search to webpages in the same domain as the key page. This is not necessary, but
significantly increases the performance with a small (rarely appreciable) cost in the precision.
deciding what webpages should be analyzed. Minimizing the number of webpages analyzed is essential to reduce the load of web crawlers. In our technique, we assume that the template contains at least one menu, and we try to identify the webpages pointed out by the menu. Therefore, 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 (hopefully) implement the template. This CS can be identified with Algorithm 1. The algorithm uses three trivial functions: loadPage(link) that loads the webpage pointed by the input link, getLinks(webpage) that returns the collection of links in the input webpage, and function size that computes set cardinality. Observe that the main loop iteratively explores the links of the initialPage (i.e., the key page) until it finds a n-CS. Note also that it only loads those pages needed to find the n-CS, and it stops when the n-CS has been found. We want to highlight the following mathematical expression:

\[
CS = \{ls \in \mathcal{P}(\text{links}) \mid \text{link} \in ls \land \forall l', l' \in ls. (l \rightarrow l'), (l' \rightarrow l) \in \text{connections}
\]

used to find the set of all CS that can be constructed with the current link.

**Algorithm 1** Extract a n-CS from a website

Input: An initialLink that points to a webpage and an integer n
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
initialPage = loadPage(initialLink);
reachableLinks = getLinks(initialPage);
links = {initialLink};
connections = {};
BCS = {};
foreach (link in reachableLinks)
  page = loadPage(link);
  existingLinks = getLinks(page) \cap reachableLinks;
  links = links \cup \{link\};
  connections = connections \cup \{(initialLink \rightarrow link)\};
  connections = connections \cup \{(link \rightarrow \text{existingLink}) \mid \text{existingLink} \in \text{existingLinks}\};
CS = \{ls \in \mathcal{P}(\text{links}) \mid \text{link} \in ls \land \forall l', l' \in ls. (l \rightarrow l'), (l' \rightarrow l) \in \text{connections}\};
MCS = \{cs \in CS \mid \text{size}(cs) \geq \text{size}(cs')\};
if size(MCS) = n then return MCS;
if size(MCS) > size(BCS) then BCS = MCS;
return BCS;
end
4.2 Template extraction from a complete subdigraph

After we have found the set of webpages linked by the menu of the site (the complete subdigraph), we identify a ETDM between the key page and all webpages in the set. For this, initially, the template is considered to be the first webpage in the set. Then, we compute a ETDM between the template and the second webpage in the set. The result is the new refined template, that is further refined with another ETDM with the next 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.

Algorithm 2 Extract a template from a set of webpages

\begin{verbatim}
Input: A set of n webpages \( P = \{p_1 \ldots p_n\} \)
Output: A template

begin
  template = p_1;
  i = 2;
  while (i <= n)
    template = ETDM(template, p_i);
    i++;
  return template;
end

function ETDM(tree \( T_1 = (N_1,E_1) \), tree \( T_2 = (N_2,E_2) \))
  \( r_1 = \text{root}(T_1) \);
  \( r_2 = \text{root}(T_2) \);
  if \( r_1 == r_2 \) then
    nodes = \{r_1\};
    edges = \{\}\;
    for each \( n_1 \in N_1, n_2 \in N_2, n_1 == n_2, (r_1,n_1) \in E_1 \) and \( (r_2,n_2) \in E_2 \) do
      edges = edges \cup\{(r_1,n_1)\};
      (nodes_{st},edges_{st}) = ETDM(subtree(n_1),subtree(n_2));
      nodes = nodes \cup nodes_{st};
      edges = edges \cup edges_{st};
    return (nodes,edges);
  else
    return (\{\},\{\});
end
\end{verbatim}

5 Implementation

We implemented the technique presented in this paper as a toolbar Firefox’s plugin. We can browse on the Internet as usual. Then, when we want to extract the template of a webpage, we only need to press the “Extract Template” button and the tool automatically (internally) loads the associated webpages to extract the template. The template is then displayed in the browser as any other webpage.

We conducted several experiments with real, online webpages to provide a measure of the average performance regarding recall, precision and the F1 measure (see, e.g., [8] for a discussion on these metrics). For the experiments, we selected 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 websites, news articles, forums, etc.). Then, we randomly selected the final evaluation set. We determined the template of each webpage (the gold standard) by:

1. Downloading the complete website of each benchmark (the key page together with all reachable webpages from it in the same domain).
2. Four different engineers did the following independently:
   (a) Manually exploring the key page and the webpages accessible from it to decide what part of the webpage is the template.
   (b) Printing the key page in paper and marking the template.
3. The four engineers met and together decided what was the template, and they marked it in a printed version of the key page.
4. Each element marked in the printed page was mapped to the DOM tree of the key page. All elements in the DOM tree that did not belong to the template were included in an HTML class non-template (i.e., we enriched the HTML code of the key page with a new class). This class was later used by an algorithm that we programmed to evaluate the results obtained by our tool.

Table 1 summarizes the results of the performed experiments, which were computed with a 4-CS. The first column contains the URLs of the evaluated webpages. For each benchmark, column DOM nodes shows the number of nodes of the whole DOM tree associated with this benchmark; column Template shows the number of nodes of the gold standard template; column Retrieved shows the number of nodes that were identified by the tool as the template; column Recall shows the number of template nodes correctly retrieved divided by the total number of actual template nodes; column Precision shows the number of template nodes correctly retrieved divided by the total number of retrieved nodes; finally, column F1 shows the F1 metric that is computed as \((2 \times \text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})\) being P the precision and R the recall.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>DOM nodes</th>
<th>Template</th>
<th>Retrieved</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.felicity.co.uk">www.felicity.co.uk</a></td>
<td>300 nodes</td>
<td>232 nodes</td>
<td>223 nodes</td>
<td>100 %</td>
<td>98,72 %</td>
<td>99,36 %</td>
</tr>
<tr>
<td><a href="http://www.dsic.upv.es/~dinsa">www.dsic.upv.es/~dinsa</a></td>
<td>241 nodes</td>
<td>74 nodes</td>
<td>74 nodes</td>
<td>100 %</td>
<td>90,24 %</td>
<td>94,87 %</td>
</tr>
<tr>
<td><a href="http://www.engadget.com">www.engadget.com</a></td>
<td>1818 nodes</td>
<td>768 nodes</td>
<td>763 nodes</td>
<td>99,35 %</td>
<td>99,22 %</td>
<td>99,28 %</td>
</tr>
<tr>
<td><a href="http://www.bbc.co.uk/news/">www.bbc.co.uk/news/</a></td>
<td>2991 nodes</td>
<td>604 nodes</td>
<td>552 nodes</td>
<td>91,39 %</td>
<td>67,73 %</td>
<td>77,80 %</td>
</tr>
<tr>
<td><a href="http://www.videextra.com">www.videextra.com</a></td>
<td>2351 nodes</td>
<td>1137 nodes</td>
<td>18 nodes</td>
<td>1,58 %</td>
<td>100 %</td>
<td>3,12 %</td>
</tr>
<tr>
<td><a href="http://www.oz.ac.uk/staff/">www.oz.ac.uk/staff/</a></td>
<td>948 nodes</td>
<td>538 nodes</td>
<td>104 nodes</td>
<td>19,33 %</td>
<td>92,86 %</td>
<td>32,00 %</td>
</tr>
<tr>
<td>clinicaltrials.gov</td>
<td>543 nodes</td>
<td>389 nodes</td>
<td>378 nodes</td>
<td>97,17 %</td>
<td>96,92 %</td>
<td>97,05 %</td>
</tr>
<tr>
<td>en.citizendium.org</td>
<td>992 nodes</td>
<td>399 nodes</td>
<td>318 nodes</td>
<td>79,70 %</td>
<td>91,64 %</td>
<td>85,25 %</td>
</tr>
<tr>
<td><a href="http://www.filmaffinity.com">www.filmaffinity.com</a></td>
<td>1316 nodes</td>
<td>340 nodes</td>
<td>340 nodes</td>
<td>100 %</td>
<td>98,84 %</td>
<td>99,42 %</td>
</tr>
<tr>
<td><a href="http://www.cnn.com">www.cnn.com</a></td>
<td>3860 nodes</td>
<td>192 nodes</td>
<td>148 nodes</td>
<td>77,08 %</td>
<td>98,67 %</td>
<td>86,55 %</td>
</tr>
<tr>
<td><a href="http://www.lashorasperdidas.com">www.lashorasperdidas.com</a></td>
<td>1822 nodes</td>
<td>553 nodes</td>
<td>252 nodes</td>
<td>45,57 %</td>
<td>100 %</td>
<td>62,61 %</td>
</tr>
<tr>
<td>labakeryshop.com</td>
<td>1368 nodes</td>
<td>403 nodes</td>
<td>175 nodes</td>
<td>43,42 %</td>
<td>96,15 %</td>
<td>59,83 %</td>
</tr>
<tr>
<td><a href="http://www.dsic.upv.es/~jsilva/wwv2013">www.dsic.upv.es/~jsilva/wwv2013</a></td>
<td>197 nodes</td>
<td>163 nodes</td>
<td>163 nodes</td>
<td>100 %</td>
<td>96,45 %</td>
<td>98,19 %</td>
</tr>
<tr>
<td><a href="http://www.thelawyer.com">www.thelawyer.com</a></td>
<td>2708 nodes</td>
<td>949 nodes</td>
<td>742 nodes</td>
<td>78,19 %</td>
<td>76,50 %</td>
<td>77,33 %</td>
</tr>
<tr>
<td><a href="http://www.us-nails.com">www.us-nails.com</a></td>
<td>250 nodes</td>
<td>184 nodes</td>
<td>184 nodes</td>
<td>100 %</td>
<td>83,64 %</td>
<td>91,09 %</td>
</tr>
<tr>
<td><a href="http://www.informatik.uni-trier.de">www.informatik.uni-trier.de</a></td>
<td>3083 nodes</td>
<td>117 nodes</td>
<td>8 nodes</td>
<td>6,84 %</td>
<td>100 %</td>
<td>12,8 %</td>
</tr>
<tr>
<td><a href="http://www.wayfair.co.uk">www.wayfair.co.uk</a></td>
<td>1950 nodes</td>
<td>1507 nodes</td>
<td>697 nodes</td>
<td>46,25 %</td>
<td>99,57 %</td>
<td>63,16 %</td>
</tr>
<tr>
<td>catalog.atsfurniture.com</td>
<td>340 nodes</td>
<td>301 nodes</td>
<td>301 nodes</td>
<td>100 %</td>
<td>99,01 %</td>
<td>99,50 %</td>
</tr>
<tr>
<td><a href="http://www.glassesusa.com">www.glassesusa.com</a></td>
<td>1952 nodes</td>
<td>1708 nodes</td>
<td>1659 nodes</td>
<td>97,13 %</td>
<td>99,70 %</td>
<td>98,40 %</td>
</tr>
<tr>
<td><a href="http://www.mysmokingshop.co.uk">www.mysmokingshop.co.uk</a></td>
<td>575 nodes</td>
<td>407 nodes</td>
<td>407 nodes</td>
<td>100 %</td>
<td>98,31 %</td>
<td>99,15 %</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>1479 nodes</td>
<td>548 nodes</td>
<td>376 nodes</td>
<td>74,15 %</td>
<td>94,21 %</td>
<td>76,84 %</td>
</tr>
</tbody>
</table>

Table 1: Benchmark results

Experiments reveal an average precision of more than 94%, and an average recall of almost 75% even though two experiments produced a recall under 7%. These experiments are really difficult ones that produce the same problem in previous techniques such as [15]. The problem in these benchmarks is that some webpages pointed by the main menu do not use the template (i.e., some webpages feature the menu, or the necessary links in some form, but they do not use the template). Therefore, the intersection with these webpages produce an almost empty page, and this is the cause of the low recall.
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 menus, 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 webpages. These blocks together form the template. To the best of our knowledge, the idea of using the menus to locate the webpages that share 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. Our implementation and experiments have shown the usefulness of the technique.

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 of the template 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 Spanish Ministerio de Ciencia e Innovación (Secretaría de Estado de Investigación) under grant TIN2008-06622-C03-02 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 the Spanish MICINN under FPI grant BES-2009-015019.

References


Template extraction based on menu information


