Automatic Detection of Webpage Candidates for Site-Level Web Template Extraction

Julián Alarte David Insa Josep Silva
Universitat Politècnica de València
Valencia, Spain
{jalarte,dinsa,jsilva}@dsic.upv.es

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

Template extraction is the process of isolating the template of a given webpage. It is widely used in several disciplines, including webpages development, content extraction, block detection, and webpages indexing. One of the main goals of template extraction is identifying a set of webpages with the same template without having to load and analyze too many webpages prior to identifying the template. This work introduces a new technique to automatically discover a reduced set of webpages in a website that implement the template. This set is computed with an hyperlink analysis that computes a very small set with a high level of confidence.

1 Introduction

Internet is full of web templates (in the following just template). Web developers normally use templates to equip their webpages with a common vocabulary of colors, panels and menus. Templates are prepared HTML pages where formatting is already implemented and visual components are ready to insert content. Thus, they speed up the development process and they are also useful for the automatic generation of webpages that only need to be fed with content.

Templates allow developers to compose their webpages with independent blocks that can be reused. This is good for web development because many tasks can be automated and webpage sections can be maintained separately. In fact, many webpage development environments and code generators offer collections of templates that already include Javascript, CSS, Flash, etc. Templates are also good for users, who can benefit from intuitive and uniform designs with a common look and feel and colored and formatted visual elements that increase the navigability and usability of the webpage.

The importance of templates also affects 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, 13, 15]). Moreover, in general, templates do not contain relevant content, they usually contain one or more pagelets [3, 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.

Submitted to:
WWV 2014

Template extraction enhance indexers by isolating the main content and 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. [5] 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 [15, 13] for web mining and search.

Given a webpage w, conceptually, template extraction is made in two steps:[1]

1. Detect a set S of webpages that implement the same template than w.
2. Analyze all webpages in $S \cup \{w\}$ to identify the template.

Our technique solves the first step through an analysis of the hyperlinks in the webpages. We introduce a new idea to automatically find a set of webpages that potentially share a template. Roughly, we detect the 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.

In practice, not all webpages in a website implement the whole template. They often implement a part of the template. As a consequence, our hyperlink analysis has two apparently contradictory objectives: (i) finding webpages as similar as possible to the target webpage, so that it is easy to identify the template, and (ii) finding webpages as different as possible between them, so that we have an heterogeneous sample that implements different parts of the template.

The rest of the paper has been structured as follows: In Section 2 we discuss the state of the art and compare other approaches. Then, in Section 3, we present our technique with examples and explain the algorithms used. In Section 4 we give some details about the implementation. Finally, Section 5 concludes.

2 Related Work

Template extraction techniques are often classified into two groups: page-level and site-level. In both cases, the objective is the same, detecting the template of a given webpage; but they use different information. While page-level techniques only use the information contained in the target webpage, site-level techniques also use the information contained in other webpages, typically of the same website.

Site-level techniques usually work in two (not necessarily independent) phases. First, they collect a set of webpages that (hopefully) implement the same template as the target webpage. Then, they extract the template by comparing the target webpage with the collected webpages. Our technique automates the first step. Despite there are many site-level template extraction techniques in the literature, there are very few approaches that describe a methodology to detect webpage candidates, and very often, this process is done manually.

[1] In practice, these two steps are not necessarily sequential. In fact, they are often interlaced.

There exist three main different approaches to template extraction, namely, (i) using the textual information of the webpages (i.e., the HTML code), (ii) using the rendered images of the webpages in the browser, and (iii) using the DOM trees of the webpages.

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 [4]. This has 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 [14] used to discriminate the main content. Other approaches exploit densitometric features based on the observation that some specific terms are more common in templates [9, 7].

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 [2]. This approach has been less studied because rendering webpages for classification is a computational expensive operation [8].

The third approach analyzes the attributes and relative positions of DOM nodes. 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 [15, 13].

With independence of the approach followed, the most extended way of selecting the webpage candidates is manually. For instance, the content extractor algorithm and its improved version, the fast content extractor algorithm [10], take as input a set of webpages that are given by the programmer. The same happens in the methodology for template extraction proposed in [6].

Even though [15] 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. Their technique inputs a collection of webpages to construct the SST. They do not have a methodology to select the webpages, and they do not propose a number of webpages needed. In their experiments, they randomly sample 500 webpages, and the time taken to build a SST is always below 20 seconds.

The approach in [13] is based on discovering optimal mappings between DOM trees. This mapping relates nodes that appear in more than one webpage, and thus they are considered redundant. Their technique uses the RTDM-TD algorithm to compute a special kind of mapping called *restricted top-down mapping* [11]. 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 dozen of webpages. They need 25 webpages to reach a 0.95 F1 measure. Contrarily, in our technique, we do not select the webpages randomly, we use a method to identify the webpages by analyzing their hyperlinks. We only need to explore a few webpages to identify the candidates 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.

In [12], authors exploit the idea that those pages stored in the same directory contain the same template. In particular, they use as webpage candidates those webpages stored in the same directory as the target webpage. Somehow, we also exploit this idea, but we do not restrict ourselves to one directory. We define an order of relevance using the tree of directories according to a definition of distance between directories.

Figure 1: Webpages of ZMEscience sharing a template

3 Identifying webpages that implement the same template

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 1 we see two webpages that belong to the ZMEscience website. At the top of the webpages we see the main menu containing links to all ZMEscience principal topics. In the left webpage we can also see an example of a submenu showing the subsections of topic "Research". The left webpage belongs to "Robotics" subsection of topic "Science", while the right webpage belongs to "Animals" section of topic "Environment". Both share the same menu, their respective submenus, and general structure. In both webpages the main content, i.e., the news, is inside the pagelet in the dashed square. In addition to the main content, there is a common pagelet called "Popular this week" with the most relevant news, and another one for subscription and social networks. Additionally, a set of related news (different for each webpage) is shown between the menu and the main content.

Our technique inputs a webpage (called key page) and it outputs a set of webpages that implement (a part of) the same template. To discover these webpages, it identifies a complete subdigraph in the website topology.

3.1 Complete subdigraphs

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 1 *Consider the ZMEscience website. Two of its webpages are shown in Figure 1. 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 ZMEscience website may be represented with the topology shown in Figure 2.*

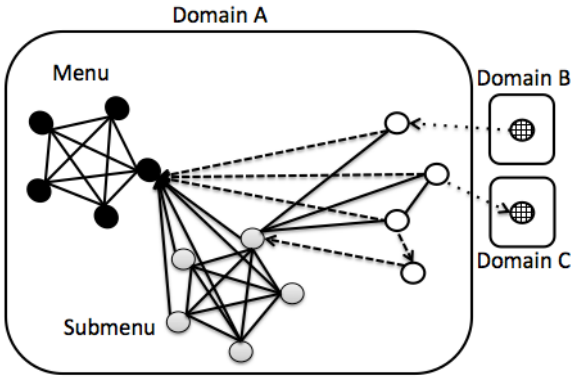

Figure 2: ZMEscience Website topology

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 a submenu, thus, all grey nodes together also form a complete graph. White nodes are webpages inside one of the sections 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 ensure high precision, we search for a CS that contains enough webpages that implement the template. It is important to remark that a webpage can contain several menus and submenus; and not all of them produce equally good CSs. For instance, consider again the topology shown in Figure 2. If we assume that the key page is one of the white nodes, then, a CS formed with the grey nodes (the submenu) will be probably better than a CS formed with the black nodes (the main menu). This happens because the white nodes belong to one of the items in the submenu, and thus, they (probably) are more related semantically, and they (probably) share more syntax components. Note that at least the submenu is a common substructure shared by all grey and white nodes.

3.2 Hyperlink analysis

By analyzing the links in the key page, it is possible to select those links that most likely produce the best CS. This is essential to avoid analyzing all links and thus significantly increasing the performance. Our

strategy to identify the links that should be analyzed is based on the structure of the website. We obtain information about the structure of the website from the URLs of the links.

Example 2 *Consider a key page P whose URL is:*
www.upv.es/research/maths/index.html
Consider that P contains four links with the following URLs:

- *URL 1 = www.tesco.es/*
- *URL 2 = www.upv.es/research/maths/pi.html*
- *URL 3 = www.upv.es/sport/*
- *URL 4 = www.upv.es/research/maths/news/computers.html*

URL 1 *points to a webpage in another domain. Therefore, the template of this webpage is probably completely different from the template of the key page.*

URL 2 *points to a webpage in the same directory as the key page. Hence, very likely, they both belong to the same section in the hierarchy of the website, and thus their structure is probably similar.*

URL 3 *points to a webpage (index.html) inside a directory that is two levels above the current directory. It probably points to another section (e.g., to another section in the main menu called* sport*). Therefore, the structures of the key page and the webpage pointed by URL 3 are possibly different. Probably, they will only share a small part of their templates.*

URL 4 *points to a webpage located inside a subdirectory of the reference directory. Probably, this webpage is semantically related to the key page, and it contains specialized information (it possibly extends the template with additional information).*

Therefore, by analyzing the links in the key page, we can establish an order of relevance. To formally define a partial order we need first to provide a notion of link and distance between links.

Definition 1 (path, hyperlink) *A* path *is a sequence of words joined by juxtaposition, $s = w_1w_2w_3...w_n$, the length of the sequence is represented with $|s|$ and it denotes the number of words in the sequence. A* hyperlink *(or just* link*) is a path where each word finishes with slash: $h = (dir/)^+$.*

Note that this definition of hyperlink deliberately ignores the name of the resource pointed by the URL; it only focusses on the structure (directories or domains). It is general enough as to include URLs such as *www.upv.es/*, *research/* and *research/maths/*. We use function *head* to select the first word (i.e., directory) of a hyperlink: $head(dir1/dir2/dir3/) = dir1$. We now provide a notion of distance between two URLs.

Definition 2 (hyperlink distance) *Given two hyperlinks h, h', the distance from h to h' is defined as:*

$$hDistance(h,h') = \begin{cases} 0 & \text{if } h = h' \\ +|h_1| & \text{if } h' = hh_1 \\ -|h_1| & \text{if } h = h'h_1 \\ -|h_1| & \text{if } h = h_0h_1 \text{ and } h' = h_0h_2 \text{ and } head(h_1) \neq head(h_2) \\ -|h| & \text{if } head(h) \neq head(h') \end{cases} \quad (1)$$

Note that the distance is defined from the first link to the second link. This can be observed in Figure 3, which represents a tree of directories that contain webpages. There, we can see the distance of all webpages to the webpage in the gray directory. Some particular examples follow:

hDistance(research/maths/, research/maths/) = 0
hDistance(research/maths/, research/maths/geometry/) = +1
hDistance(research/maths/, research/) = -1
hDistance(research/maths/, research/physics/dynamics) = -1
hDistance(research/maths/, www.upv.es/research/) = -2

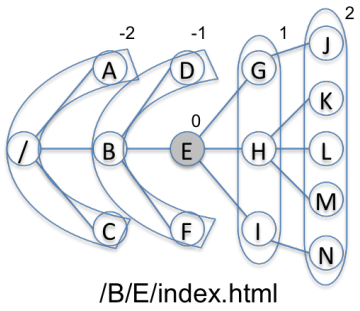

Figure 3: Hyperlink distance

Intuitively, given two links $h1$ and $h2$, a distance of 0 means that both links point to the same directory. A positive distance from $h1$ to $h2$, means that $h2$ points to a subdirectory of the directory pointed by $h1$. A negative distance from $h1$ to $h2$, means that $h2$ points to a directory outside the directory pointed by $h1$. We use the url of the key page as a reference link to compute distances. And we compare the distances of the links of the key page. Those links with distance 0 are preferred. Then, those with a positive distance. And finally, those with a negative distance.

In case of a draw[2], we use another information to determine what link is better. Concretely, we analyze their position in the DOM tree. Often, pagelets agglutinate semantically related information. Thus, different pagelets contain different information. Therefore, two links that belong to different pagelets usually point to webpages whose content is semantically different. This is very useful, because we are interested in locating webpages that share the same template, and that contain as more differences as possible so that we can precisely identify the template.

Example 3 *Consider a set of links in the menu of a shopping webpage. The links can point to similar webpages where each webpage contains information about one particular product. Often, these webpages share the same template that is filled with similar information about the products. Some of the information shared by the products can be confused as part of the template because it appears repeated in many webpages. Hence, template extraction algorithms must avoid comparing only these webpages because they do not provide sufficient information to isolate the template.*

As a consequence, in case of a draw, we prefer those links that are as separated as possible from the other already selected links in the DOM tree. In this way, we give preference to links with (probably)

[2]Note that, according to Definition 2, the hyperlink distance defines a partial order, and thus, two different links can have the same distance to a third link.

different semantic information. In summary, observe that we obtain webpages that share the same template (using the hyperlink distance) but being as different as possible (using their position in the DOM tree).

In the following, webpages are represented with a DOM tree $T = (N, E)$ (see Figure 4). $root(T)$ 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>).

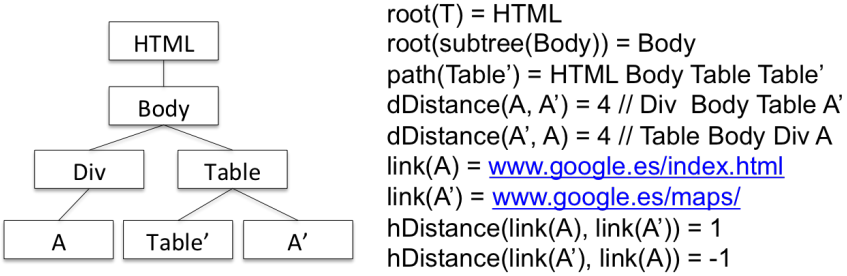

Figure 4: A DOM tree T

To compare the position in the DOM tree of two links we measure the length of their DOM paths. The DOM path of a node n, $path(n)$ is the path from the root to that node.

Definition 3 (DOM distance) *Given a DOM tree $T = (N, E)$ and two nodes $n, n' \in N$, the* DOM distance *from n to n' is defined as:*

$$dDistance(n,n') = \begin{cases} 0 & \text{if } path(n) = path(n') \\ j+k & \text{if } path(n) = n_0 \ldots n_i m_1 \ldots m_j \text{ and } path(n') = n_0 \ldots n_i m'_1 \ldots m'_k \text{ and } m_1 \neq m'_1 \end{cases} \quad (2)$$

Note that two links have zero DOM distance if and only if they are exactly the same link. Contrarily, two different links (even if they have the same URL, and thus the same hyperlink distance) necessarily have a positive DOM distance.

We can now define an order for the links in a webpage. It allows us to decide what links should be explored to extract the template. This order combines the link relevance order $\leq^h_{link}$, which uses the link distance, and the DOM relevance order $\leq^N_{DOM}$, which uses the DOM distance. Their formal definition follows.

Definition 4 (link relevance) *Given any set of hyperlink nodes N of a DOM tree and a reference hyperlink h, N is equipped with the preorder $\leq^h_{link}$ called* link relevance *and defined as follows. For any $n_1, n_2 \in N$ we have:*

Link Relevance:

$$n_1 =^h_{link} n_2 \quad iff \quad hd_1 = hd_2$$

$$n_1 <^h_{link} n_2 \quad iff \quad \begin{cases} 0 \leq hd_1 < hd_2 \vee \\ hd_2 < hd_1 \leq 0 \vee \\ hd_2 < 0 \leq hd_1 \end{cases}$$

where

$$hd_1 = hDistance(h, link(n_1)) \qquad hd_2 = hDistance(h, link(n_2))$$

Definition 5 (DOM relevance) *Given any set of hyperlink nodes N of a DOM tree T and a reference set of hyperlink nodes N' in T, N is equipped with the preorder $\leq^{N'}_{DOM}$ called* DOM relevance *and defined as follows. For any $n_1, n_2 \in N$ we have:*

$$\underline{\textit{DOM Relevance:}}$$

$$n_1 =^{N'}_{DOM} n_2 \quad \textit{iff} \quad \begin{cases} N' = \emptyset \vee \\ dn'_1 = dn'_2 \end{cases}$$

$$n_1 <^{N'}_{DOM} n_2 \quad \textit{iff} \quad dn'_1 > dn'_2$$

where

$$n'_1 = \min_{n \in N'} dDistance(n, n_1) \qquad dn'_1 = dDistance(n'_1, n_1)$$
$$n'_2 = \min_{n \in N'} dDistance(n, n_2) \qquad dn'_2 = dDistance(n'_2, n_2)$$

3.3 Finding a webpage candidates in a website

We use a combination of link relevance and DOM relevance to select the links that should be explored first to find a CS in the website topology. For this, we use Algorithm 1.

Algorithm 1 Sort links

Input: A set of hyperlink nodes *links* and a reference hyperlink h.
Output: A sorted list of *links* with respect to the preorders $\leq^{h}_{link}$ and $\leq^{N}_{DOM}$.

```
begin
  sortedLinks = [];
  while (links ≠ ∅)
    links' = {l ∈ links | ∄l' ∈ links ∧ l' <^h_link l};
    links = links \ links';
    sortedLinks' = [];
    while (links' ≠ ∅)
      link = l ∈ links' | ∄l' ∈ links' ∧ l' <^sortedLinks'_DOM l;
      links' = links' \ {link};
      sortedLinks' = sortedLinks' ++ [link];
    sortedLinks = sortedLinks ++ sortedLinks';
  return sortedLinks;
end
```

Algorithm 1 sorts the links in a webpage combining link relevance and DOM relevance. First, it takes each set of hyperlink nodes in the order provided by the link relevance. Then, it sorts each of these sets using the order provided by the DOM relevance. Finally, the concatenation of each sorted set is the order that we use to explore the links of a webpage.

Now we are in a position to describe our algorithm that identifies a CS in a website. This algorithm is Algorithm 2.

The algorithm uses two trivial functions: $loadWebPage(link)$, which loads and returns the webpage pointed by the input link, and $getLinks(webpage)$, which returns the collection of (non-repeated) links[3] in the input webpage (ignoring self-links). Function *sortLinks* corresponds to Algorithm 1. 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

[3] In our implementation, this function removes those links that point to other domains because they are very unlikely to contain the same template. Here, we do not impose this restriction to keep the algorithm general.

Algorithm 2 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
$keyPage = loadWebPage(initialLink)$;
$reachableLinks = getLinks(keyPage)$;
$processedLinks = \emptyset$;
$connections = \emptyset$;
$bestCS = \emptyset$;
$sortedLinks = sortLinks(reachableLinks, initialLink)$;
foreach ($link$ **in** $sortedLinks$)
  $webPage = loadWebPage(link)$;
  $existingLinks = getLinks(webPage) \cap reachableLinks$;
  $processedLinks = processedLinks \cup \{link\}$;
  $connections = connections \cup \{(link \rightarrow existingLink) \mid existingLink \in existingLinks\}$;
  $CS = \{ls \in \mathscr{P}(processedLinks) \mid link \in ls \wedge \forall l, l' \in ls \,.\, (l \rightarrow l'), (l' \rightarrow l) \in connections\}$;
  $maximalCS = cs \in CS$ such that $\forall cs' \in CS \,.\, |cs| \geq |cs'|$;
  if $|maximalCS| = n$ **then return** $maximalCS$;
  if $|maximalCS| > |bestCS|$ **then** $bestCS = maximalCS$;
return $bestCS$;
end

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

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

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

$$\textbf{if } |maximalCS| = n \textbf{ then return } maximalCS$$

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

4 Implementation

The technique presented in this paper including all algorithms described has been implemented as a Firefox's toolbar. We selected Firefox because it is one of the most powerful and widely used browsers, and it is free and open source. Firefox toolbars are implemented using XUL, an XML based language used to implement the interface; and Javascript, which implements the behavior and actions of the toolbar. In total, it contains 2577 LOC.

With this tool, the user can browse the Internet as usual. Then, when she wants to extract the template of a webpage, she only needs to press a button and the tool automatically loads the appropriate linked webpages to form a CS. The links to all webpages are then displayed in the browser.

Example 4 *Consider the key page in Figure 5 left. We can introduce in the top bar the number of webpages that implement the same template than the key page, or we can left the default value 3, which produces the best balance between precision and performance according to our experiments. In this case, 4 webpages were required, thus, if we press the button in the top bar, then the webpage at the right is automatically generated. These links point to webpages that together form a 4-CS. All of them are very likely to implement the same template than the key page.*

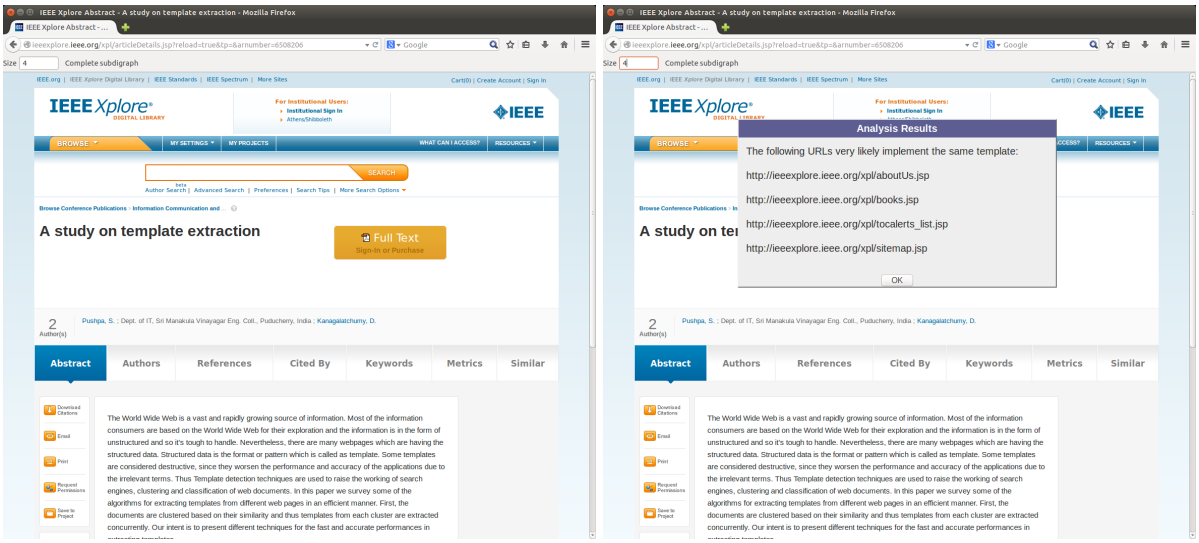

Figure 5: Keypage (left) and a set of webpages (right) automatically identified with the tool.

Empirical evaluation

In our theoretical formalization we presented our technique in an abstract way. Some definitions such as hyperlink distance, DOM distance, and relevance, reveal features of a website that must be considered when extracting templates. Other features, however, have been left as parameters of our algorithms. For instance, Algorithm 2 can compute a CS of any specified size. In this section we discuss how to determine the size of the CS based on empirical evaluation.

One important design decision is related to the domain boundaries of the websites analyzed. It is possible that several webpages of different domains are mutually linked. Sometimes this is even usual between the main webpages of different companies in an alliance. They all point to the others, e.g., with a set of logos. Nevertheless, the templates of the companies are often different. In fact, in our experiments, we did not find a shared template between different domains. Therefore, for efficiency reasons, external domains are omitted when computing the CS. In our implementation, we restrict our search to webpages in the same domain as the key page.

Determining the size of the complete subdigraph

Algorithm 2 computes a n-CS in a website. As previously explained, there are several combinations of webpages that form a CS. One could think that the bigger the CS is, the better; so we could even think in calculating the maximal CS. Nevertheless, this is not a good idea. Firstly, because computing the maximal CS has an exponential cost. And secondly, because experiments reveal that increasing the size of the CS does not necessarily imply a better precision or recall.

In order to prove this, we performed several experiments with a template extractor that implemented our technique to identify the set of webpages used to identify the template. Then, we repeated all the experiments with different sizes (1,2,3,4,5,6,7 and 8) for the CS in order to determine the best value. We have made public the details of the experiments including the suit of benchmarks used at:

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

Size	Recall	Precision	F1	Loads
1	88,56 %	94,89 %	88,69 %	2
2	96,34 %	90,32 %	91,93 %	5,6
3	95,44 %	96,35 %	95,61 %	10,13
4	94,61 %	96,88 %	95,27 %	16,52
5	94,69 %	96,96 %	95,40 %	18,68
6	95,21 %	96,82 %	95,69 %	23,68
7	95,46 %	96,31 %	95,57 %	30
8	95,14 %	96,57 %	95,54 %	32,08

Table 1: Determining the size of the complete subdigraph

The results are presented in Table 1.

This table summarizes several experiments. Each row in the table is the average of 40 template extractions from 40 different webpages. Each row is the result of repeating the experiments with a different value for n in the n-CS searched by Algorithm 2. In particular, each column has the following meaning:

`Size:` represents the size of the CS that the algorithm tried to find in the websites. In the case that there did not exist a CS of the searched size, then the algorithm used the biggest CS with a size under the specified size (see Algorithm 2).

`Recall:` shows the number of correctly retrieved nodes divided by the number of nodes in the gold standard.

`Precision:` shows the number of correctly retrieved nodes divided by the number of retrieved nodes.

`F1:` shows the F1 metric that is computed as $(2*P*R)/(P+R)$ being P the precision and R the recall.

`Loads:` represents the average number of webpages loaded to construct the n-CS.

Observe that F1 is stabilized in 95% with a CS of size 3. Increasing the size of the CS does not significantly increase F1, but it increases the number of pages loaded to construct the (bigger) CS. Therefore, we determined that a subdigraph of size 3 is the best option because it keeps almost the best F1 value and it is quite efficient, the algorithm has to load significantly less webpages than in a size bigger than 3. As a direct consequence, our implementation, by default, stops when a subdigraph of size 3 has been found (but it can be configured to search for a subdigraph of any size (4,5,6, etc.)).

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

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

5 Conclusions

Templates are useful for website developers, for crawlers and for final users. This work presents a new technique for template extraction. Given a webpage, the technique automatically detects a set of (linked) webpages that very likely implement the same template. This is done by analyzing the links in the menus of the website. To the best of our knowledge, the idea of using the menus to locate the template is new, and it allows us to find a set of webpages from which we can extract the template with a reduced amount of pages loaded. 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.

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.

6 Acknowledgements

This work has been partially supported by 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 Eduación 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. Available at `http://doi.acm.org/10.1145/511446.511522`.

[2] 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. Available at `http://dx.doi.org/10.1109/ACIIDS.2009.71`.

[3] 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. Available at `http://doi.acm.org/10.1145/371920.372054`.

[4] 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.

[5] 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.

[6] Vidya Kadam & Prakash R. Devale (2012): *A Methodology for Template Extraction from Heterogeneous Web Pages*. *Indian Journal of Computer Science and Engineering (IJCSE)* 3(3).

[7] 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.

[8] 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.

[9] 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.

[10] Dat Quoc Nguyen, Dai Quoc Nguyen, Son Bao Pham & The Duy Bui (2009): *A Fast Template-Based Approach to Automatically Identify Primary Text Content of a Web Page*. In: *Proceedings of the 2009 International Conference on Knowledge and Systems Engineering, KSE 2009* , IEEE Computer Society, pp. 232–236.

[11] 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. Available at `http://doi.acm.org/10.1145/988672.988740`.

[12] Tom Rowlands, Paul Thomas & Stephen Wan (2009): *Web indexing on a diet: Template removal with the sandwich algorithm*. In: *Proceedings of the 14th Australasian Document Computing Symposium*. Available at `http://es.csiro.au/adcs2009/proceedings/poster-presentation/06-rowlands.pdf`.

[13] 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. Available at `http://doi.acm.org/10.1145/1183614.1183654`.

[14] 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.

[15] 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. Available at `http://doi.acm.org/10.1145/956750.956785`.