Master's Thesis

Julián Alarte Aleixandre

September 12, 2014

Abstract

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. Templates are also useful for the final users, 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. This work proposes 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.

Resumen

En el desarrollo Web, el uso de plantillas es uno de los recursos más importantes para los ingenieros. Las plantillas les permiten incrementar la productividad mediante la inserción de contenido dentro de páginas previamente formateadas con un diseño definido. Las plantillas también son de gran utilidad para el usuario final, porque proporcionan uniformidad y un diseño común al sitio web. Sin embargo, desde el punto de vista de los buscadores e indexadores, las plantillas pueden suponer un importante problema, porque normalmente suelen contener información irrelevante como publicidad, etc. El procesamiento y almacenamiento de toda esa información incide en un mal aprovechamiento de recursos tales como espacio de almacenamiento, ancho de banda, etc. Algunos estudios indican que entre el 40 % y el 50 % de los datos totales de las páginas web forman parte de las plantillas. Por lo tanto, la identificación de las plantillas es fundamental para llevar a cabo tareas como la indexación. Este trabajo propone un nuevo método para la extracción automática de plantillas basado en el análisis de la similitud de árboles DOM extraídos de una colección de páginas web seleccionadas mediante la información de sus menús.

Contents

Chapter 1

Introduction

1.1 Template detection

A web template (in the following just template) is a prepared HTML page where formatting is already implemented and visual components are ready to insert 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, which can benefit from intuitive and uniform designs with a common vocabulary of colored and formatted visual elements. This fact improves the user experience and the usability of the website.

Contrarily, templates suppose 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 (see, e.g., [2, 21, 23]). Moreover, in general, templates do not contain relevant content, they usually contain one or more pagelets [6, 2] (i.e., self-contained logical regions with a well defined topic or functionality) where the main content must be inserted. Therefore, detecting templates can help indexers to identify the main content of the webpage.

Figure 1.1 shows two webpages belonging to the UPV website. Both webpages share a header where we can distinguish the university logo and an accessibility menu. Below the logo we can find the main menu and all its options. In both webpages, the main content, which obviously does not belong to the template, is found inside the dotted square. The footer is shared by both webpages and it can be found below the main content. In

this case, the template could be the union of the header and the footer.

Figure 1.1: UPV webpage template

Modern crawlers and indexers do not treat all terms in a webpage in the same way. First of all, the template is identified by preprocessing the webpages. Template detection 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. The processing and treatment of this amount of irrelevant data can lead to a waste of storage space, bandwidth and time.

Template detection enhance indexers by isolating the main content and assigning higher weights to the really relevant terms. As the main content is usually complementary to the template, the result of removing the template will probably isolate the main content. 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. [9] 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 [23, 21] for web mining and search.

This approach to template detection is based on the DOM [7] structures that represent webpages. Roughly, given a webpage in a website, the first step is to identify a set of webpages that are likely to share a template with it. Then these webpages have to be analyzed to identify the part of their DOM trees that they share with the original webpage. This slice of the DOM tree is returned as the template.

This technique exploits a new idea to automatically find a set of webpages that potentially share a template. The process detects the template's

menu and analyzes the links of the menu to identify a set of mutually linked webpages. One of the main functions of a template is to aid navigation, thus almost all templates provide a large number of links, shared by all webpages implementing the template. Locating the menu allows to identify the main webpages of each category or section in the topology of the website. 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. Therefore, it can extract the template with good accuracy in a reduced time.

1.2 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.

Content Extraction is a discipline very close to template detection. 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., [11, 22, 5, 12]), and all of them are directly related to template detection. Many works have been presented in the CleanEval competition [3], which periodically proposes a collection of examples to be analyzed with a gold standard. The examples proposed are especially thought for boilerplate removal and content extraction.

Template detection 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.

In the area of template detection, there are three main different ways to solve the problem:

- Using the textual information of the webpage (i.e., the HTML code)
- Using the rendered image of the webpage in the browser
- 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 [8]. 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 [22] used to discriminate the main content. Other approaches exploit densitometric features based on the observation that some specific terms are more common in templates [16, 14]. 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 [4]. This approach has been less studied because rendering webpages for classification is a computational expensive operation [15].

The third approach is where our technique falls. While some works try to identify pagelets analyzing the DOM tree with heuristics [2], others try to find common subtrees in the DOM trees of a collection of webpages in the website [23, 21]. Our technique is similar to these last two works.

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 [17], take as input a set of webpages that are given by the programmer. The same happens in the methodology for template detection proposed in [13].

Even though [23] uses a method for template detection, 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 [21], the approach 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* [18]. 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 [19], 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.

1.3 Preliminary definitions

The DOM 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 successor that represents the internal table (see Figure 1.2).

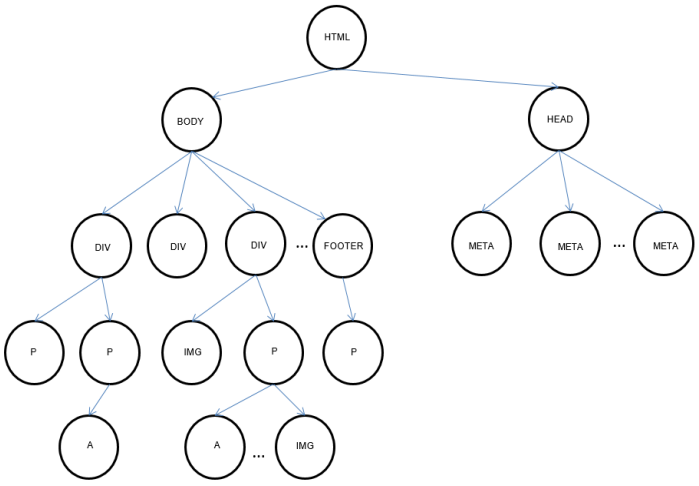

Figure 1.2: Webpage represented with a DOM tree

In a DOM tree $T = (N, E)$ representing a webpage, where N is a finite set of nodes and E is a set of edges between nodes in N, $root(T)$ 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 $path(n) = n_0 n_1 \dots n_m$ such that $\forall i, 0 \leqslant i < m.\ n_i = parent(n_{i+1})$.

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 1.3.1 (mapping) *(based on Kuo's definition of mapping [20])* *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 $n'_1 = n'_2$.*

Definition 1.3.2 (top-down mapping) *A* mapping *M between a tree T_1 and a tree T_2 is said to be top-down only if for every pair $(i_1, i_2) \in M$ there is also a pair $(parent(i_1), parent(i_2)) \in M$, where i_1 and i_2 are non-root nodes of T_1 and T_2 respectively*

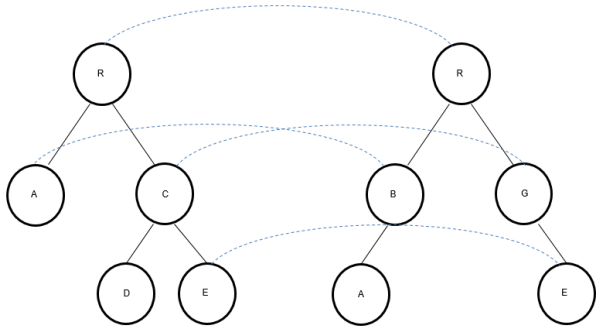

Figure 1.3: Top-down mapping between DOM trees

Definition 1.3.3 (restricted top-down mapping) *([18]) A* top-down mapping *M between a tree T_1 and a tree T_2 is said to be restricted top-down only if for every pair $(i_1, i_2) \in M$, such that $t_1[i_1] \neq t_2[i_2]$, there is no descendent of i_1 or i_2 in M, where i_1 and i_2 are non-root nodes of T_1 and T_2 respectively.*

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

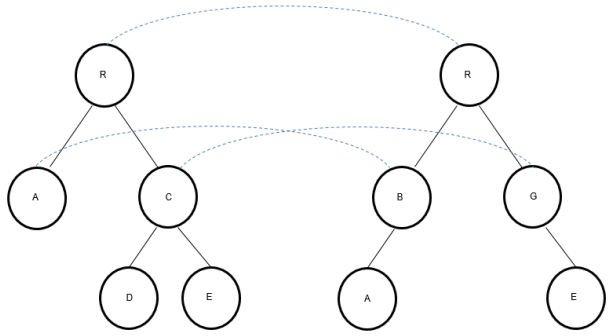

Figure 1.4: Restricted top-down mapping between DOM trees

Definition 1.3.4 (exact top-down mapping) *Given an equality function* $\triangleq$ *between tree nodes, a mapping* M *between two trees* T *and* T' *is said to be* exact top-down *if and only if*

- *exact: for every pair* $(n, n') \in M$, $n \triangleq n'$.
- *top-down*

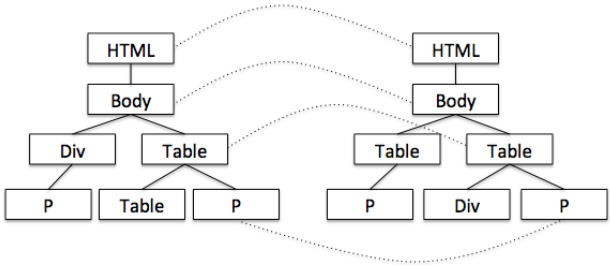

Figure 1.5: Exact top-down mapping between DOM trees

Note that this definition is parametric with respect to equality relation $\triangleq$. 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 CSS classes, its HTML identifier, 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 [18]. 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 1.4 presents an ETDM example where two nodes with different label can be mapped, but not their descendants. Figure 1.5 shows an example of an ETDM using: $n \triangleq n'$ if and only if n and n' have the same label.

Figure 1.3 shows a comparison between exact top-down mapping and restricted top-down mapping. The main difference is that ETDM only allows mapping pairs of nodes with the same label and RTDM allows mapping pairs of nodes with different labels. When RTDM maps two nodes with different labels, their descendants cannot be mapped.

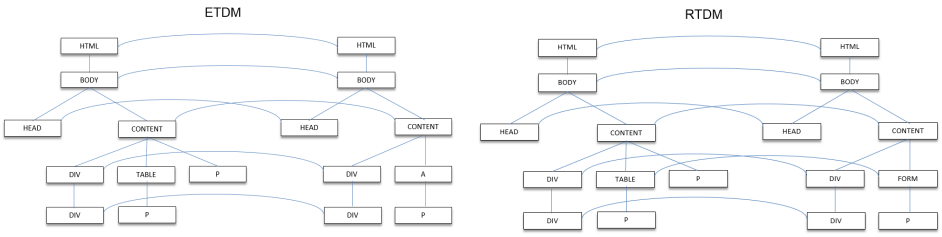

Figure 1.6: ETDM vs RTDM

We can now give a definition of template using ETDM.

Definition 1.3.5 *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 $T = \{T_1 \dots T_n\}$. A* template *of p_0 with respect to P is a tree (N, E) where*

- *nodes: $N = \{n \in N_0 \mid \forall t \in T \,.\, (n, \_) \in M_{T_0,t}\}$ where $M_{T_0,t}$ is an exact top-down mapping between trees T_0 and t.*
- *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.

Figure 1.7 shows a mapping example of 3 webpages using the ETDM algorithm. The middle column is the initial webpage from which we want to extract its template. The webpages on both sides are webpages used to compare their nodes with the initial webpage nodes. The comparison is made using the ETDM algorithm with the equality relation $\triangleq$. If we focus on the middle column, we can distinguish nodes in grey and nodes in white: Nodes in grey are equal[1] in the three webpages, so they belong to the template. However nodes in white do not belong to the template because they do not appear in the three webpages.

[1] Here, we assume that nodes are equal (according to some equality relation $\triangleq$) if they have the same label.

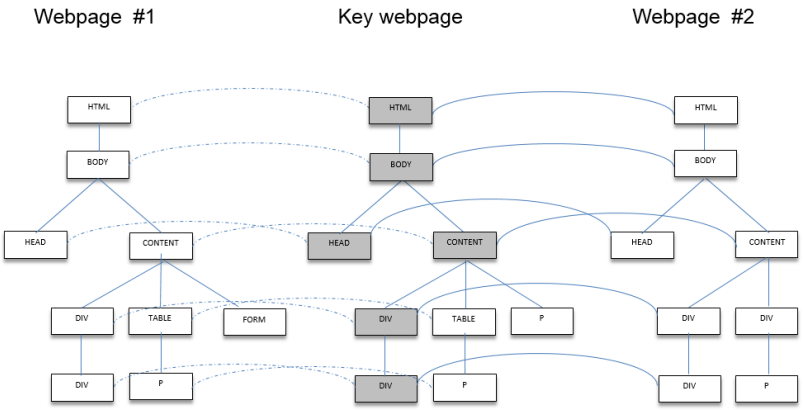

Figure 1.7: ETDM mapping example between the key page and 2 webpages

Chapter 2

Identifying webpages sharing a template

2.1 Templates

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.

Example 2.1.1 *Consider two webpages that belong to the ZME website in Figure 2.1. 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.*

Example 2.1.2 *Figure 2.2 shows two webpages that belong to the BBC website. The one on the left is the main UK news webpage, and the one on the right is the US and Canada main news webpage. At the top of both websites there are two menus, one on the top with the main sections of the BBC website and another one with the subsections of each section. These both websites are subsections of the main section "News".*
Both webpages share their header which includes the logo, the main menu and the submenu. They also share the footer which does not appear in the pictures. The central part of the websites is divided into two pagelets. The pagelet on the left, is the main content of the website. The pagelet on the right contains sections like videos, the most popular news, ads, etc.

Figure 2.1: Webpages of ZMEscience sharing a template

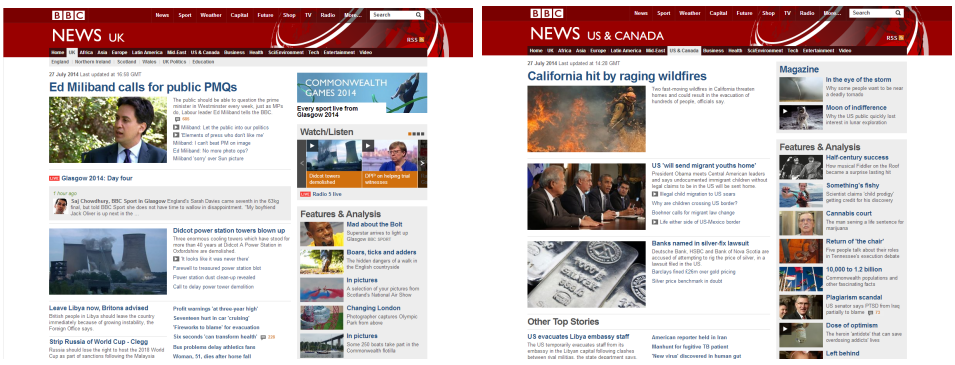

Figure 2.2: Webpages of BBC sharing a template

Our approach is very simple yet powerful:

1. Starting from the key page, it identifies a complete subdigraph (CS) 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.

2.2 Finding template candidates in a website topology

Analyzing the hyperlinks of a website is good way of identifying its topology. We observed that webpages linked by the items in a menu are usually

mutually linked, thus, they form a complete subdigraph. Identifying the CS 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 2.2.1 *Consider the ZMEscience website. Two of its webpages are shown in Figure 2.1. 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 2.3.*

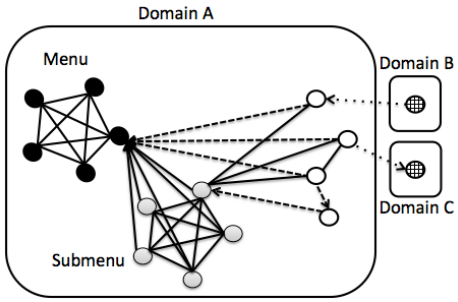

Figure 2.3: ZMEscience Website topology

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.

Not all the webpages in a website implement the same template, some of them only implement a subset of a template. For this reason, one of the main problems of template detection is deciding what webpages should be analyzed. Minimizing the number of webpages analyzed is essential to reduce the task. 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 2.3. 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 by 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 (but not necessarily by the black nodes).

Example 2.2.2 *Consider again the graph in Figure 2.3. Figures 2.4, 2.5 and 2.6 provide examples to illustrate the differences between each kind of node. Firstly, Figure 2.4 represents a white node because it is a webpage pointed by a submenu option, that submenu option is pointed by the "news" main menu option. This webpage shares several components with the grey nodes also pointed by the "news" main menu option, like the webpages in Figure 2.5. They all share their header, they both are formed by two columns, a wider one on the left and another thinner on the right. They also share the footer element and the boxes style and distribution. Figure 2.6 webpages represent black node pages, they are links from the main menu. Comparing this pair of webpages with the webpage in Figure 2.4 shows that they have only a few blocks in common, like the footer and the main menu, even the header is now different between them, so they share only a small part of the template.*

Figure 2.4: White node belonging to bbc.co.uk

2.2.1 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

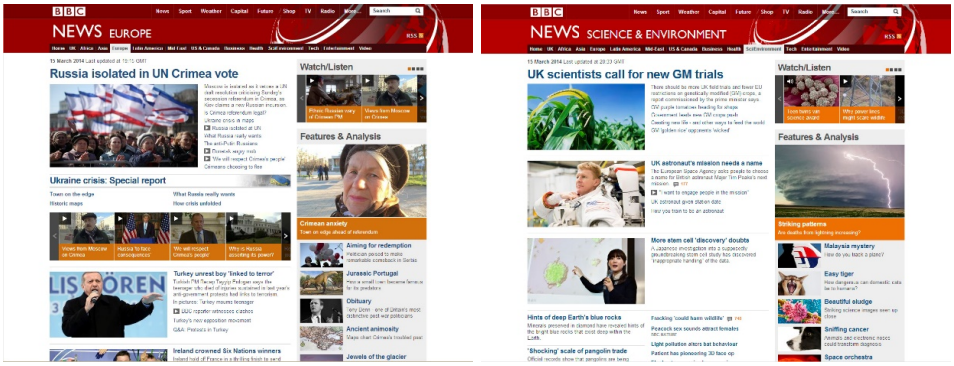

Figure 2.5: Gray nodes belonging to bbc.co.uk

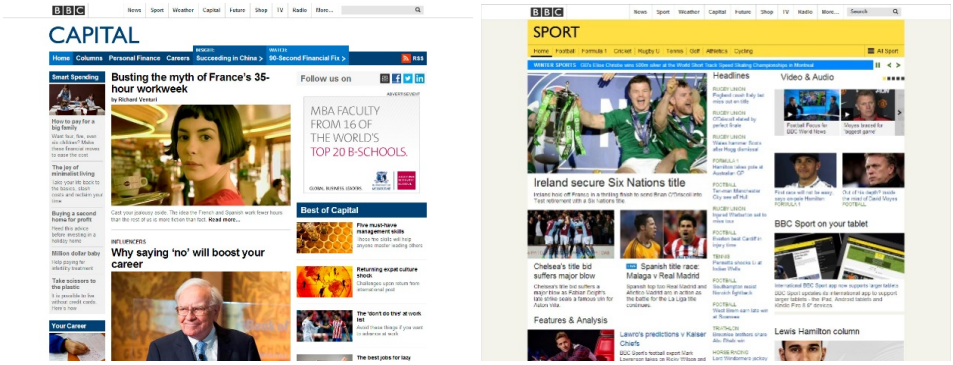

Figure 2.6: Black nodes belonging to bbc.co.uk

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 2.2.3 *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 folder 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) of a folder 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 in a subfolder of the current folder. Probably, this webpage is semantically related to the key page, and it contains specialized information (it possibly extends the template with additional information).*

2.2.2 Hyperlink distance

Analyzing the links in the key page, we can establish an order of relevance. In order to formally define a partial order that can be used by our algorithms to sorts the links, we need first to provide a notion of link and distance between links.

Definition 2.2.4 (hyperlink) *A* hyperlink *(or just* link*) is a sequence of words separated by slash: $h = (dir/)^+$. The length of a link h is represented with $|h|$ and it denotes the number of words in the sequence:*

$$|h| = \begin{cases} 1 & \text{if } h = dir/ \\ 1 + |h'| & \text{if } h = dir/h' \text{ where } h' \text{ is a link} \end{cases} \tag{2.1}$$

Note that this definition 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/*. In the following we use $head(h)$ to refer to the first word in a link. E.g., $head(dir1/dir2/dir3/) = dir1$. We now provide a notion of distance between two URLs.

Definition 2.2.5 (hyperlink distance) *Given two hyperlinks h, h', the distance from h to h' is defined as:*

$$hDistance(h, h') = \begin{cases} 0 & \text{if } h = h' \\ +|h_1| & \text{if } h' = hh_1 \\ -|h_1| & \text{if } h = h'h_1 \\ -|h_1| & \text{if } h = h_0h_1 \text{ and } h' = h_0h_2 \text{ and } head(h_1) \neq head(h_2) \\ -|h| & \text{if } head(h) \neq head(h') \end{cases} \tag{2.2}$$

Observe that the distance is defined from the first link to the second link. This is illustrated in Figure 2.7 that 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

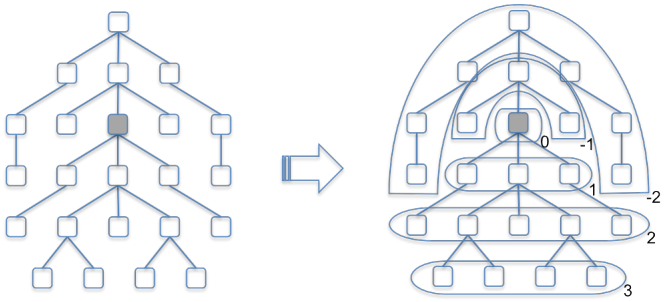

Figure 2.7: Hyperlink distance

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 inside 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 in the key page. Those links with distance 0 are preferred. Then those with a positive distance, and finally those with a negative distance.

Example 2.2.6 *Consider the BBC website in Figure 2.8. Some examples of distances between its hyperlinks are the following:*

Figure 2.8: BBC news webpage

1. *Distance between the news webpage in Figure 2.8 and the tech news category webpage:*

The tech news webpage is referenced by the news webpage. The tech

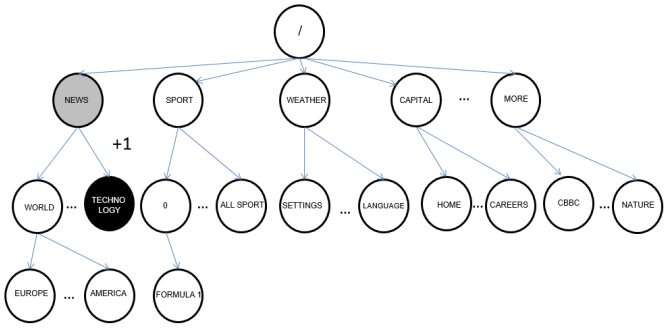

Figure 2.9: Hyperlink distance in a website hierarchy

news webpage is stored in a subdirectory of the news directory, so the distance between them is a positive distance with value 1.
hDistance(www.bbc.com/news/technology, www.bbc.com/news) = +1

2. *Distance between the news webpage in Figure 2.8 and the main webpage:*
   The root directory contains the main webpage and the news directory

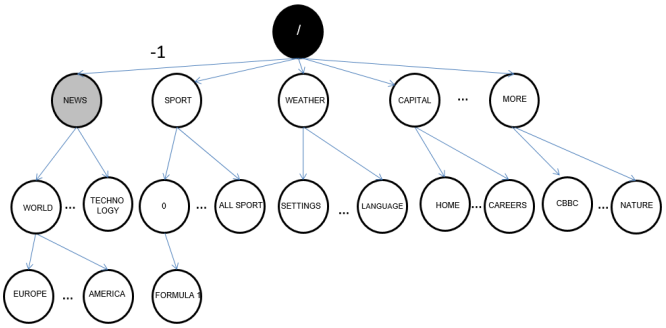

Figure 2.10: Hyperlink distance in a website hierarchy

is a subdirectory of the root directory, so in this example the hyperlink distance is negative with value 1.
hDistance (www.bbc.com/news, www.bbc.com) = -1

3. *Distance between the news webpage in Figure 2.8 and the European news webpage:*
   The europe directory is a subdirectory of world, which is a subdirectory of news. That is, there is a positive distance with value 2 between the pages inside the news directory and the europe directory.

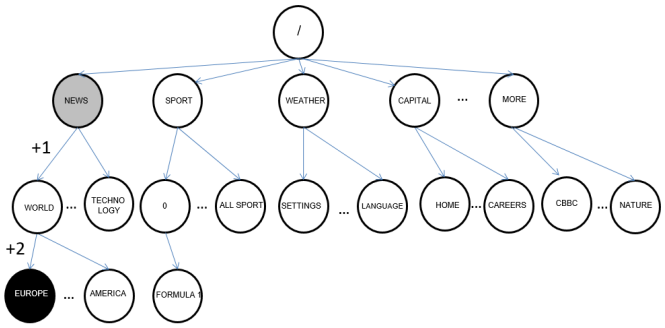

Figure 2.11: Hyperlink distance in a website hierarchy

hDistance (www.bbc.com/news, www.bbc.com/news/world/europe/) = +2

4. *Distance between the news webpage in Figure 2.8 and the weather main webpage:*
   The weather directory, as the news directory, is a subdirectory of the

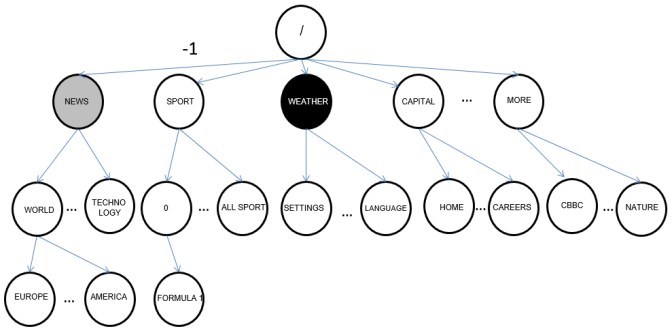

Figure 2.12: Hyperlink distance in a website hierarchy

root directory. Therefore, it is necessary to go back only one level to reach a path to the weather directory. Thus, the hyperlink distance is a negative distance with value 1.
hDistance (www.bbc.com/news, www.bbc.com/weather) = -1

5. *Distance between the news webpage in Figure 2.8 and a sports article webpage:*
   The path to reach the formula 1 directory is completely different to the path started by the news directory. Therefore, it will be necessary to go back one level to reach the path to the formula 1 directory, so the distance between them will be negative with value 1.

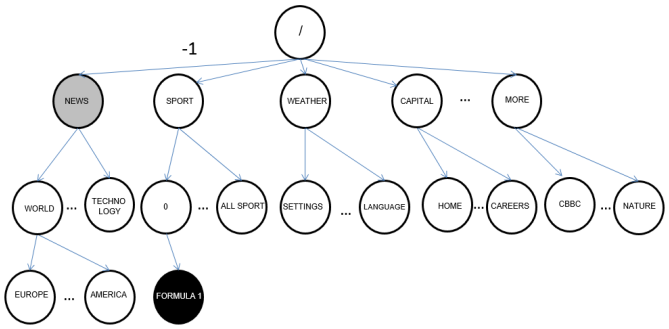

Figure 2.13: Hyperlink distance in a website hierarchy

hDistance (www.bbc.com/news, www.bbc.com/sport/0/formula1/26599556) = -1

2.2.3 DOM distance

In case of a draw[1], we use another information to determine what link is better. For this, 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 2.2.7 *Consider a set of links in 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. The similar information shared by the products can be confused as part of the template because it appears repeated in many webpages. Hence, template detection 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) different semantic information. In summary, observe that we obtain webpages that share the

[1]Note that, according to Definition 2.2.5, the hyperlink distance defines a partial order, and thus, two different links can have the same distance to a third link.

same template (using the hyperlink distance) but being as different as possible (using their position in the DOM tree). To compare the position in the DOM tree of two links we measure the length of their paths.

Definition 2.2.8 (length of a DOM path) *Given a node n in a DOM tree, the length of its path $path(n)$ is represented with $|path(n)|$, and it denotes the number of nodes in the sequence:*

$$|path(n)| = \begin{cases} 1 & \text{if } path(n) = n_0 \\ 1 + |path(n')| & \text{if } path(n) = n_0 path(n') \text{ where } path(n') \text{ is a path} \end{cases} \tag{2.3}$$

Example 2.2.9 *Consider the DOM Tree in Figure 2.14, the DOM path of the black <SPAN> node on the lower left side is:*
<BODY><CONTENT><DIV><P><SPAN>
The length of the path corresponds to the number of nodes in the sequence, so its length value is 5.

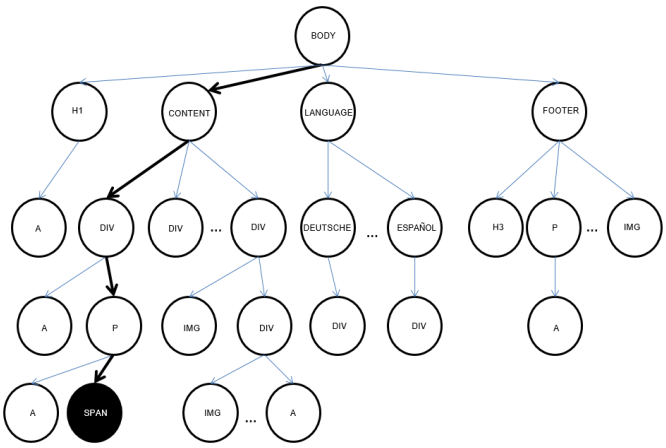

Figure 2.14: DOM path of a node

In the following definition we use function *head* to select the first node in a path: $head(n_0 n_1 \dots n_m) = n_0$.

Definition 2.2.10 (DOM distance) *Given a DOM tree $T = (N, E)$, two nodes $n, n' \in N$, and their DOM paths $path(n), path(n')$, where $head(path(n)) = head(path(n')) = root(T)$; the* DOM distance *from n to n' is defined as:*

$$dDistance(n, n') = \begin{cases} 0 & \text{if } path(n) = path(n') \\ |path(n_1)| + |path(n_2)| & \text{if } path(n) = path(n_0)path(n_1) \\ & \text{and } path(n') = path(n_0)path(n_2) \\ & \text{and } head(path(n_1)) \neq head(path(n_2)) \end{cases} \tag{2.4}$$

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.

Example 2.2.11 *Consider the DOM Tree in Figure 2.15:*

1. *The DOM path of the black <SPAN> node on the lower left side, followed by black arrows, is:*
   <BODY><CONTENT><DIV><P><SPAN>

2. *The DOM path of the gray <IMG> node on the lower side, followed by gray arrows, is:*
   <BODY><CONTENT><DIV><DIV><IMG>

The DOM distance between both nodes is calculated as the sum of the nodes that follow a different path. Both sequences share their first two nodes, then the third node is different, while in the first sequence it is a <DIV>, in the second it is another <DIV>. Therefore, the DOM distance between these two nodes is calculated as the sum of their nodes starting from the third. Both sequences have 3 nodes from the third to the last one, so the DOM distance between them is 6.

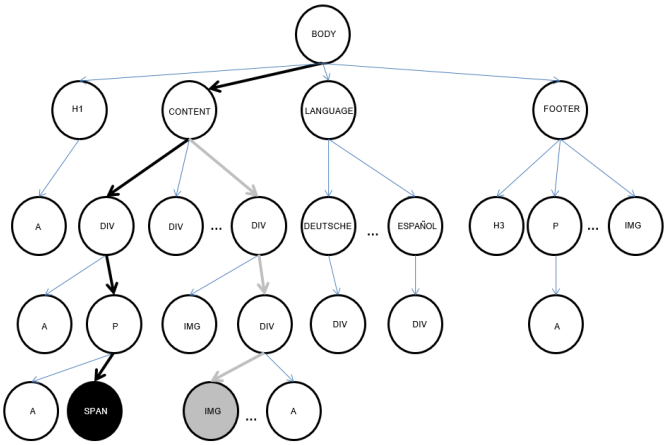

Figure 2.15: DOM distance of two nodes

2.2.4 Hyperlinks order

We can now define an order of 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 $\leqslant^h_{link}$, which uses the link distance, and the DOM relevance order $\leqslant^N_{DOM}$, which uses the DOM distance.

Definition 2.2.12 (link relevance) *Given any set of hyperlink nodes N and a reference hyperlink h, N is equipped with the preorder $\leqslant^h_{link}$ called* link relevance *and defined as follows. For any $n_1, n_2 \in N$ with*

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

we have:

$$\underline{\textit{Link Relevance:}}$$
$$n_1 =^h_{link} n_2 \textit{ iff } hd_1 = hd_2$$
$$n_1 <^h_{link} n_2 \textit{ iff } \begin{cases} 0 \leqslant hd_1 < hd_2 \ \vee \\ hd_2 < hd_1 \leqslant 0 \ \vee \\ hd_2 < 0 \leqslant hd_1 \end{cases}$$

Example 2.2.13 *Consider the following hyperlinks belonging to the BBC website:*

$link(A^1) = www.bbc.com/news/$
$link(A^2) = www.bbc.com/news/world/europe/$
$link(A^3) = www.bbc.com$
$link(A^4) = www.bbc.com/news/uk/$
$link(A^5) = www.bbc.com/sport/0/football/28497920/$
$link(A^6) = www.bbc.com/news/also\_in\_the\_news/$

If we take the first hyperlink as the reference hyperlink, first of all we need to calculate the hDistance between the reference hyperlink and the others:

$hDistance(A^1, A^2) = +2$
$hDistance(A^1, A^3) = -1$
$hDistance(A^1, A^4) = +1$
$hDistance(A^1, A^5) = -1$
$hDistance(A^1, A^6) = +1$

Knowing the distance between the reference hyperlink and the others, we could order them depending on the link relevance function:

$www.bbc.com/news/uk/ =^h_{link} www.bbc.com/news/also\_in\_the\_news/ \leqslant^h_{link}$
$www.bbc.com/news/world/europe/ \leqslant^h_{link} www.bbc.com =^h_{link}$
$www.bbc.com/sport/0/football/28497920/$

Definition 2.2.14 (DOM relevance) *Given any set of hyperlink nodes N and a reference set of hyperlink nodes N', N is equipped with the preorder $\leqslant^{N'}_{DOM}$ called* DOM relevance *and defined as follows. For any $n_1, n_2 \in N$ with*

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

we have:

$$\underline{\textit{DOM Relevance:}}$$
$$n_1 =^{N'}_{DOM} n_2 \textit{ iff } dn'_1 = dn'_2$$
$$n_1 <^{N'}_{DOM} n_2 \textit{ iff } dn'_1 > dn'_2$$

We strictly follow the 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 $\leqslant^{h}_{link}$ and $\leqslant^{N}_{DOM}$.

begin
  $sortedLinks = []$;
  while $(links \neq \varnothing)$
    $links = links \backslash links'$;
    $sortedLinks' = []$;
    while $(links' \neq \varnothing)$
      $link = l \in links' \mid \nexists l' \in links' \wedge l' <^{sortedLinks'}_{DOM} l$;
      $links' = links' \backslash \{link\}$;
      $sortedLinks' = sortedLinks' +\!\!+ [link]$;
    $sortedLinks = sortedLinks +\!\!+ sortedLinks'$;
  return $sortedLinks$;
end

Algorithm 1 sorts the links in a webpage combining the link relevance and the 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.

Example 2.2.15 *Consider the DOM tree in Figure 2.16. It represents a partial DOM tree of the webpage www.bbc.com/news/. The gray nodes are links to other pages:*

$link(A^1) = www.bbc.com/news/world/europe/$
$link(A^2) = www.bbc.com$
$link(A^3) = www.bbc.com/news/uk/$
$link(A^4) = www.bbc.com/sport/0/football/28497920/$
$link(A^5) = www.bbc.com/news/also\_in\_the\_news/$

As in example 2.2.13, the hyperlink relevance order is:

$www.bbc.com/news/uk/ =^{h}_{link} www.bbc.com/news/also\_in\_the\_news/ \leqslant^{h}_{link}$
$www.bbc.com/news/world/europe/ \leqslant^{h}_{link} www.bbc.com =^{h}_{link}$
$www.bbc.com/sport/0/football/28497920/$

Once the set of hyperlinks is ordered by the link relevance algorithm, it has to be sorted again using the DOM relevance, following Algorithm 1.

The first two elements in the hyperlink relevance order, $link(A^3)$ and $link(A^5)$ have the same relevance, so they have to be ordered by the DOM relevance algorithm:

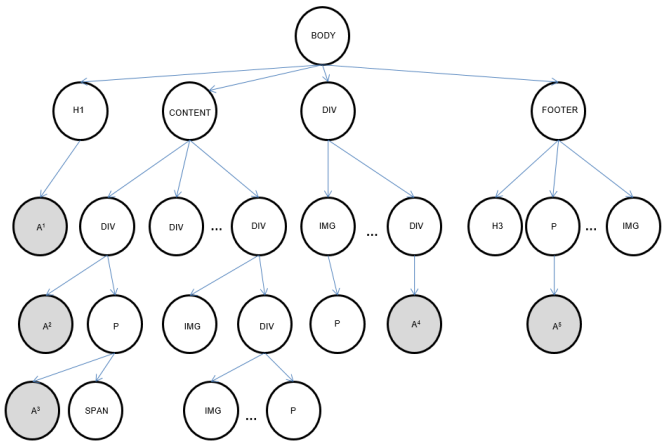

Figure 2.16: Partial DOM tree of www.bbc.com/news/

$dDistance(A^3,A^1) = 6$
$\boldsymbol{dDistance(A^3,A^2)} = 3$
$dDistance(A^3,A^4) = 7$
$dDistance(A^3,A^5) = 7$

$\boldsymbol{dDistance(A^5,A^1)} = 5$
$dDistance(A^5,A^2) = 6$
$dDistance(A^5,A^3) = 7$
$dDistance(A^5,A^4) = 6$

The minimum values of the distances above, following the DOM relevance algorithm, indicate that link(A^5) is prior to link(A^3), so the correct order of the first two links is:

{www.bbc.com/news/also_in_the_news/, www.bbc.com/news/uk/}

Then, we can add the following hyperlink, link(A^4), in the link relevance order:

{www.bbc.com/news/also_in_the_news/,
www.bbc.com/news/uk/,
www.bbc.com/news/world/europe/}

The following two hyperlinks link(A^2) and link(A^4) in the link relevance order are drawn, so we need the DOM relevance algorithm:

$dDistance(A^2,A^1) = 5$
$\boldsymbol{dDistance(A^2,A^3)} = 3$

$dDistance(A^2,A^4) = 6$
$dDistance(A^2,A^5) = 6$

$\mathbf{dDistance(}A^4,A^1\mathbf{)} = 5$
$dDistance(A^4,A^2) = 6$
$dDistance(A^4,A^3) = 7$
$dDistance(A^4,A^5) = 6$

The minimum values of the distances above, following the DOM relevance algorithm, indicate that link(A^4) is prior to link(A^2), therefore the set of links is ordered as follows:

{www.bbc.com/news/also_in_the_news/,
www.bbc.com/news/uk/,
www.bbc.com/news/world/europe/,
www.bbc.com/sport/0/football/28497920/,
www.bbc.com}

2.2.5 Extracting the n-CS

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

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

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[2] 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

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

where $\mathcal{P}(X)$ 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.
Figure 2.17 shows an IEEE webpage on the left, and 4 hyperlinks forming a 4-CS obtained from that IEEE webpage on the right.

[2]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.

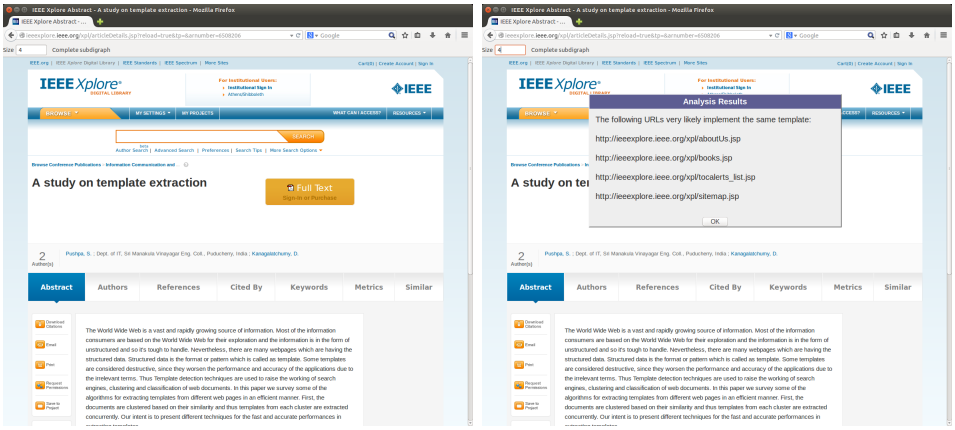

Figure 2.17: 4-CS in the IEEE website

Chapter 3

Template detection

3.1 Parameters

Once we have found a set of webpages linked by a menu of the website (the CS), we need to decide what nodes in the DOM tree of the key page belong to the template. Of course, those nodes that belong to the DOM trees of all webpages in the CS should be considered part of the template. But the question is: what should we do with those nodes that appear in only some of the webpages? Probably, a node that appears in all webpages but one belongs to the template. It is just that for some reason this webpage has skipped this part of the template. What should be the general criterion? To how many CS webpages should belong a node to mark it as part of the template? What is the best CS size?

We have answered these questions with empirical evaluation. Given a n-CS, we call t to the threshold used to decide in how many webpages must be a node to be considered part of the template. The process to obtain the optimal n and t values is explained in Section 5.2.2.

Here we present the formalization of our algorithm that is parametric and can be used with any value for n and t. Given a concrete values for n and t, we first obtain a n-CS with Algorithm 2, and then we use the ETDM formalism to determine how many webpages contain a given node of the key page. In particular, all those nodes that belong to an ETDM between the key page and t webpages of the n-CS are considered part of the template.

3.2 Attributes

We need to enhance the nodes with new attributes that we will use in the rest of the section to infer whether a node belong to a template. The new node attributes are described in the following definition.

Definition 3.2.1 (node attributes) *Every node n in a DOM tree $T = (N, E)$ contains the attributes specified in the DOM model [7] including:*

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

Id: *We refer to the id of a DOM node by using $id(n)$ and it corresponds to the DOM attribute id.*

Classes: *We refer to the set of classes assigned to a DOM node by using $classes(n)$ and it corresponds to the DOM attribute $className$.*

Attributes: *We refer to the set of attribute names defined in a DOM node by using $attributes(n)$ and it corresponds to the DOM attribute attributes where we remove the attributes named class and id.*

Child Index: *We refer to the child index of a DOM node using $childIndex(n)$ and it is derived from the DOM attributes parentNode and childNodes.*

With these node attributes, we can find out whether two nodes are equal. This information is crucial to check the template membership of a node. If a node is equal to a node in the key page, then we can relate both webpages and start to infer the template. Our notion of equality cannot be defined with an exact formula, because DOM trees belonging to different webpages rarely share exactly the same nodes. Instead of that, we use a notion based on probabilities. We use each attribute of the nodes to infer individuals probabilities that we join in the last step to give a probability for two nodes to be equal. When we join these individual probabilities, we can give different weights to each of them. We chose to let this information as a parameter, allowing us to test with empirical evaluation (described in Section 4) the best values for each one. Apart form these parameters, we need another ones to define the probability of two nodes when one of the attributes is not present in both nodes, so we defined a parameter for missed attributes for each individual probability. Our notion of equality is formalized in the following definition.

Definition 3.2.2 (equality probability) *The probability $P_=$ for two nodes n_1 and n_2 to be equal is calculated as follows.*

$$P_=(n_1,n_2) = \begin{cases} 0 & \text{if } tag(n_1) \neq tag(n_2) \\ 1 & \text{if } tag(n_1) = tag(n_2) \wedge id(n_1) = id(n_2) \\ w_c \times P_c(n_1,n_2,P_{nc}) \ + & \\ w_a \times P_a(n_1,n_2,P_{na}) \ + & \\ w_{ch} \times P_{ch}(n_1,n_2,P_{nch}) \ + & \\ w_p \times P_p(n_1,n_2) & \text{otherwise} \end{cases}$$

where:

- *P_c is the probability to be equal according to the nodes classes. It is defined as follows:*

$$P_c(n_1,n_2,P_{nc}) = \begin{cases} P_{nc} & \text{if } classes(n_1) = \varnothing \wedge classes(n_2) = \varnothing \\ \frac{|classes(n_1) \bigcap classes(n_2)|}{|classes(n_1) \bigcup classes(n_2)|} & \text{otherwise} \end{cases}$$

- *P_a is the probability to be equal according to the nodes attributes. It is defined as follows:*

$$P_a(n_1, n_2, P_{na}) = \begin{cases} P_{na} & \text{if } attributes(n_1) = \varnothing \wedge attributes(n_2) = \varnothing \\ \frac{|attributes(n_1) \bigcap attributes(n_2)|}{|attributes(n_1) \bigcup attributes(n_2)|} & \text{otherwise} \end{cases}$$

- *P_{ch} is the probability to be equal according to the number of children. It is defined as follows:*

$$P_{ch}(n_1, n_2, P_{nch}) = \begin{cases} P_{nch} & \text{if } |children(n_1)| = 0 \wedge |children(n_2)| = 0 \\ \frac{min(|children(n_1)|, |children(n_2)|)}{max(|children(n_1)|, |children(n_2)|)} & \text{otherwise} \end{cases}$$

- *P_p is the probability to be equal according to the nodes position. It is defined as follows:*

  Given two trees $T = (N, E)$ and $T' = (N', E')$, four nodes $p \in T$, $p' \in T'$, $n \in T$ such that $(p \rightarrow n) \in E$, $n' \in T'$ such that $(p' \rightarrow n') \in E'$, c and c' the number of children of p and p' respectively, i and i' the child positions starting from the left of n and n' respectively, j and j' the child position starting from the right of n and n' respectively, and $c = min(c, c')$. P_p is the probability of n to be n' (and n' to be n) according to the nodes position and is calculated as follows:*

  - *If $c' = c$ then 1 – ($|i - i'| \ / \ c*$)*
  - *If $c' > c$ then 1 – ($max(0, i - i', j - j') \ / \ c*$)*
  - *If $c' < c$ then 1 – ($max(0, i' - i, j' - j) \ / \ c*$)*

P_p shows how to assign a position probability to each pair of nodes of different trees. Roughly speaking, the probability is higher as more similar are both positions in their trees. Once their positions are less similar, their probability descends as well. We show two examples showing this idea.

Example 3.2.3 *Figure 3.1 shows that the fourth node starting from the left in the tree A has a probability of 100% of being the fourth node starting from the left in the tree B. In addition, it is the first node starting from the right in the tree A, so the first node starting from the right in the tree B will also have a 100% probability.*

Example 3.2.4 *Figure 3.2 shows that the fourth node starting from the left in the tree A has a probability of 100% of being the fourth node starting from the left in the tree B. In addition, it is the second node starting from the right in the tree A, so the second node starting from the right in the tree B will also have a 100% probability.*

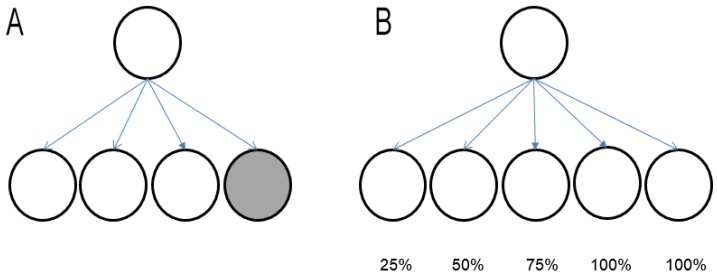

Figure 3.1: Position probability example

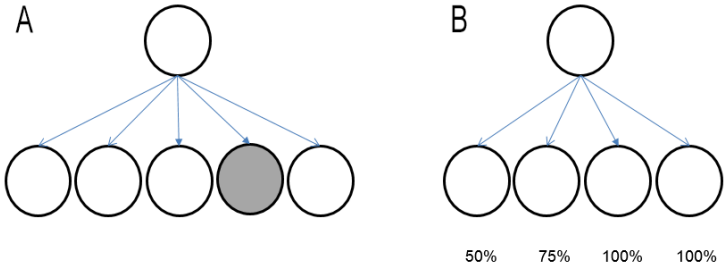

Figure 3.2: Position probability example

Example 3.2.5 *Consider the two DOM trees in Figure 3.3.*

The gray node of the DOM tree on the left has the following attributes:

1. *ClassName = "Author"*
2. *Attributes = {float:left, color: black, background-color:silver}*
3. *Children = 2*
4. *Position = 11*

The gray node of the DOM tree on the right has the following attributes:

1. *ClassName = "Author"*
2. *Attributes = {float:left, color: gray, background-color:silver}*
3. *Children = 3*
4. *Position = 9*

We assume we obtained the following values with empirical evaluation: $w_c = 0.4$, $w_a = 0.1$, $w_{ch} = 0.1$ *and* $w_p = 0.4$. *Now, we can calculate the probability for the two nodes to be equal applying the definition 3.2.5:*

The tag of both nodes is DIV but the id is not present, so the probability cannot be 0 or 1 and we have to infer it from the attributes.

1. P_c = *1 because the ClassName of both nodes is equal.*
2. P_a = *0.66 because they share two of their three attributes.*
3. P_{ch} = *0.66 because the first node only has two children and the other has three.*
4. P_p = *0.95 because of their position.*

If we sum all the partial probabilities we get: $P_{=}(n_1, n_2) = w_c \times P_c + w_a \times P_a + w_{ch} \times P_{ch} + w_p \times P_p = 0.4 \times 1 + 0.1 \times 0.66 + 0.1 \times 0.66 + 0.4 \times 0.95 = 0.912$

In consequence, their probability of being the same node is 91.20%. If there is not another pair of nodes (n_1, n_2) *with higher probability, they will be considered the same node.*

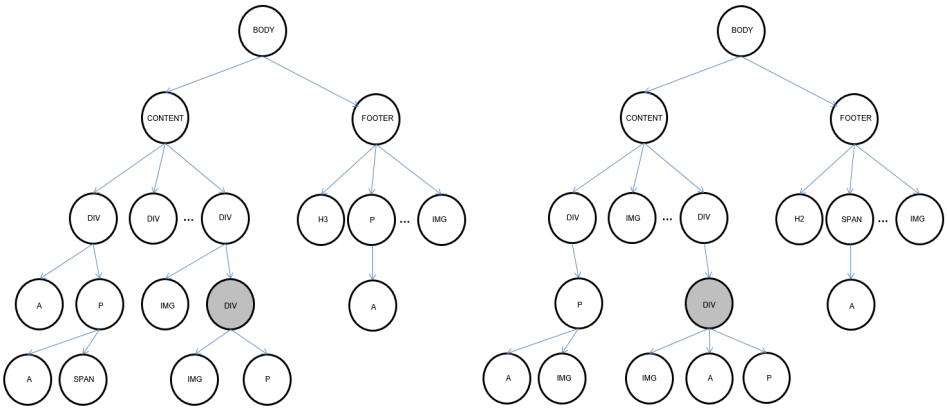

Figure 3.3: DOM trees

3.3 Exact top-down mapping

Before introducing our algorithm to extract the template, we need to define a mapping between two sets of nodes. This mapping is essential to relate the nodes of two DOM trees because it allows us to discard those parts that are not equal. The mapping that we use here is an exact top-down mapping according to Definition 1.3.4 and it is mainly based on the equality probability of Definition 3.2.2. The following definition describes and formalizes our mapping.

Definition 3.3.1 (exact top-down mapping based on equality probability) *The exact top-down mapping between two sets of nodes N_1 and N_2 according to the equality probability is defined as follows.*

$$\begin{aligned} mapping(N_1, N_2) = \; & (n_1, n_2) \bigcup \\ & mapping(before(n_1, N_1), before(n_2, N_2) \bigcup \\ & mapping(after(n_1, N_1), after(n_2, N_2)) \end{aligned}$$

where:

- $n_1 \in N_1, n_2 \in N_2$ *such that* $\nexists (n_1', n_2') \in N_1 \times N_2 \; . \; P_{=}(n_1', n_2') > P_{=}(n_1, n_2) \wedge P_{=}(n_1, n_2) > P_t$, *being* P_t *an arbitrary constant*
- $before(n, N) = \{n' \in N \mid childIndex(n') < childIndex(n)\}$
- $after(n, N) = \{n' \in N \mid childIndex(n') > childIndex(n)\}$

The key idea in the mapping calculation is that given two sets of nodes we select the best matching nodes among all the possible combinations. After this step, we divide the rest of nodes into two groups (nodes before and after the selected nodes), then, we continue calculating the mapping between these sets separately. The constant P_t is a threshold used to decide whether two nodes n_1 and n_2 are the same based on a given probability $P_{=}(n_1, n_2)$.

3.4 Extraction algorithm

We are now in a position to describe our algorithm. After we have found a set of webpages 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 achieve this, we use Algorithm 3 that is mainly based on the function *ETDM*. This function computes the biggest ETDM between a tree and a set of trees. It iterates between the children of the root of the given tree T. For each child n_1, it first looks for nodes that match n_1 among the set of trees P. This matching is checked with an exact top-down mapping based on equality probability, as defined in Definition 3.3.1. With the results of this search a new set of trees P' is built. This set includes the subtrees of the matching nodes. Note that, at this point, some of the trees that form P can be skipped. With the set P' and the subtree of n_1, a recursive call is made with the ETDM of the subtrees. This ETDM is joined to the ETDM computed for the other children of the root of T. The function includes a third parameter, a number t that represents the minimum number of coincidences (i.e., threshold) that have to share a node in tree T with nodes in set P to include it in the ETDM. The nodes that do not fulfill this condition will be discarded by means of the first condition of the function.

Algorithm 3 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 $n_1 \in N_1$, $n_2 \in N_2$. $n_1 \triangleq n_2, (r_1, n_1) \in E_1$ 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 $(nodes, edges)$;

Chapter 4

Implementation

4.1 Implementation

The technique presented, 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.

Firefox extensions add more functionality to the Firefox browser. Firefox has several kinds of extensions. Ours is an *overlay extension*, which is a traditional extension that uses a XUL overlay. Overlay extensions always use:

- XUL overlays to specify the interface.
- APIs available to privileged code such as tabbrowser and Javascript modules to interact with the application and content.

Figure 4.1 shows our Firefox plugin architecture. A user browsing a webpage uses the toolbar to get the web template. The toolbar obtains the web template using the API, that extracts the DOM nodes that form the template from the webpage represented as a DOM tree.

The source code of the plugin is public (but the use of this code needs permission from the authors). It contains the following files:

- chrome.manifest: Identifies the plugin (within Firefox) and specifies its internal organization.
- install.rdf: Contains installation information (version, developers, mimium requirements...)
- ff-overlay.xul: The interface specified in XUL files.

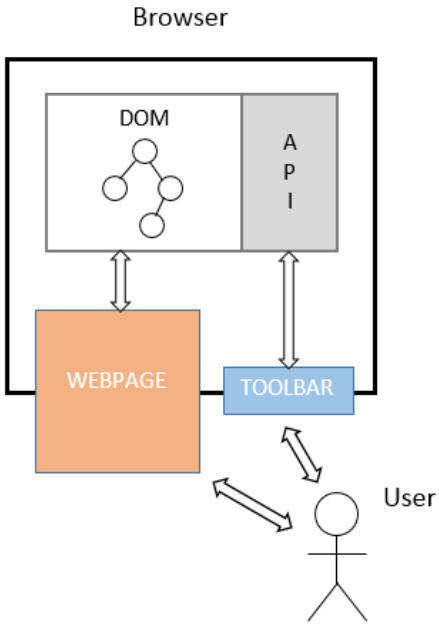

Figure 4.1: Firefox plugin architecture

The core algorithm of the technique is implemented with Javascript. It is formed by the following files:

- TemplateExtractor.js:
- DomainGraph.js:
- DOMNodesMap.js:
- WebNode.js:
- HierarchyLinks.js:

The main options of the toolbar are *extract web template* and *toggle view*. *Extract web template* gets the template from the current webpage loaded by the active window in the browser. Once the template is obtained and the browser shows only the template nodes, *Toggle view* is used to switch between the template and the original webpage.

Figure 4.2 shows the main modules from the core algorithm. Once the template detection process is started, the algorithm starts computing a CS. Functions *load current page*, *load page* and *load next page* load the pages the algorithm needs to analyze in order to build the CS. Function *process node* basically loads and processes the links in a webpage. The *load graph* function calls the *obtain CG* function which computes the CS and then, it calls the *filter template* function which uses the CS to compute the ETDM.

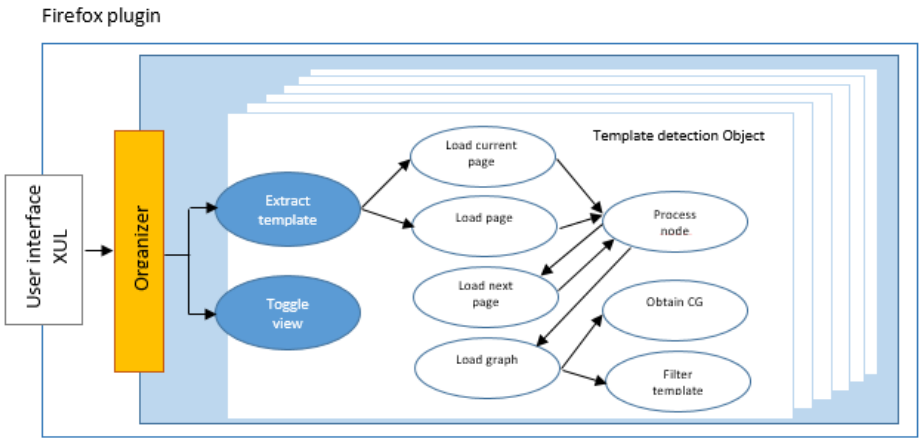

Figure 4.2: Firefox plugin architecture

In this tool, the user can browse on the Internet as usual. Then, when she wants to extract the template of a webpage, 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. The tool also has a button to toggle the view between the full webpage and the template.

4.2 The firefox toolbar

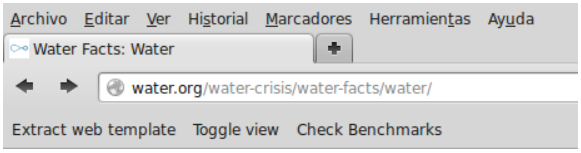

Figure 4.3: Firefox toolbar

The Firefox plugin toolbar has three buttons:

- *Extract web template* is used to extract the template from the webpage the user is browsing in that moment. The plugin reads the webpage links, then it orders the links and loads the appropriate webpages to build a CS. Finally, it analyzes the CS and extracts the template.
  Note that the plugin only selects the webpages in the domain[1] from which it extracts the template.

[1]In 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.

- *Toggle view* changes the browser view when the user has already extracted the template. It is possible to change the view between the template view and the opposite to the template.

- *Check benchmarks* executes a prepared benchmark on some webpages previously prepared for that process. The plugin needs to be configured with a list of webpages and the source location where it can find them. However, these websites need to be prepared by marking the nodes that do not belong to the template. When executing the benchmarks, the plugin extracts the webpages' template and compares that template with the template marked by the user. That is the way the plugin can find out the precision, recall and F1 measures.

4.3 Examples

The following examples show the result of applying the plugin to extract the template on some websites.

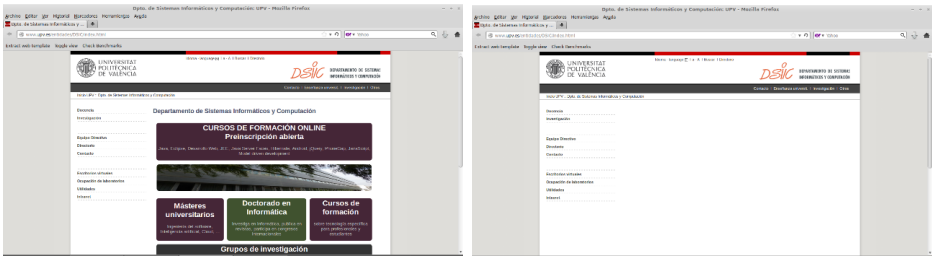

Figure 4.4: DSIC website: Before and after extracting the template

Figure 4.4 shows the template detection result on the www.dsic.upv.es main webpage. There is a header at the top of the page and two columns below it. There is also a footer at the bottom of the webpage. The template detection process keeps as template the header and the left column which contains the main menu. It also keeps the footer. The right column disappears because it contains the main content of the webpage and does not belong to the template.

Figure 4.5 shows the template detection result on the www.eclipse.org main webpage. At the top of the webpage we can find a header which includes an advertisement, the main menu and a secondary menu. The main content block is below the header, and there is also a footer at the bottom of the page. The plugin extracts the template from the webpage and keeps as template the header except the advertisement. It also keeps the footer. The main content block disappears because it is not part of the template.

Figure 4.6 shows the template detection result on the water.org/water-crisis/water-facts/water/ webpage. There is a header block at the top of

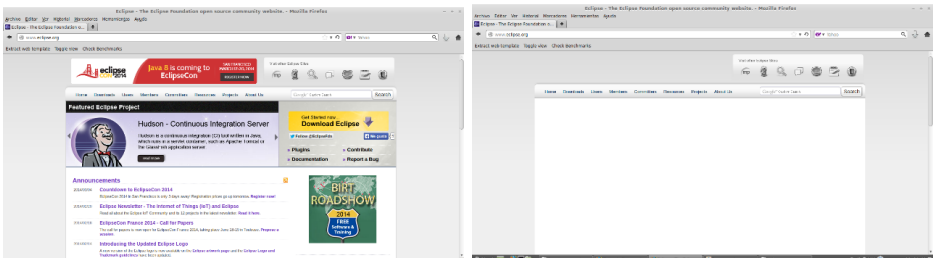

Figure 4.5: Eclipse.org website: Before and after extracting the template

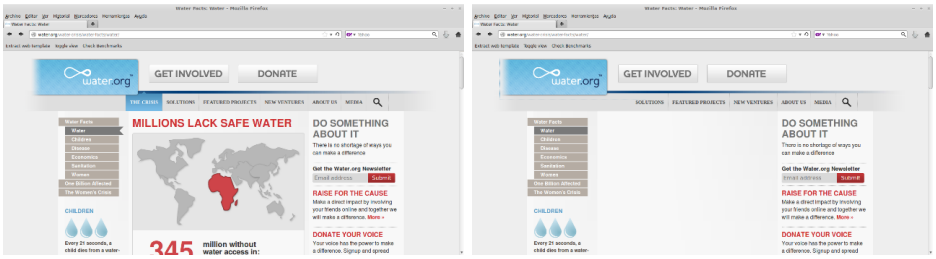

Figure 4.6: Water.org website: Before and after extracting the template

the webpage. It contains a logo, a main menu and two buttons. There are three columns below the header. The left column contains a secondary menu, the central column contains the main content and the right column contains other secondary blocks. We can also find a footer at the bottom of the webpage. The template detection process keeps as template the entire header, the left and the right columns, and finally the footer. It does not mark as template the central column, which contains the main content of the webpage.

Chapter 5

Empirical evaluation

5.1 Experiments

Several experiments were conducted with real, online webpages to provide a measure of the average performance regarding recall, precision and the F1 measure (see, e.g., [10] for a discussion on these metrics). Initially, we wanted to use a public standard collection of benchmarks, but we did not find any public dataset for template detection. In particular, we could not use the standard CleanEval suite [3] of content extraction benchmarks, because it contains a gold standard prepared for content extraction (each part of the webpages is labelled as *main-content* or *non-content*), but it is not prepared for template detection. Then, we tried to use the same benchmark set as the authors of other template detection papers. However, due to privacy restrictions, copyright, or unavailability[1] of the benchmarks we could not use a previous dataset. Therefore, we decided to produce a new suite of benchmarks. We have produced a new publicly accessible dataset, with an automatizable gold standard. It is one of the main contributions of our work. This benchmark suite is explained in detail in Chapter 6.

The dataset is composed of a collection of web domains with different layouts and page structures. This allows us to study the performance of the techniques in different contexts (e.g., company websites, news articles, forums, etc.). To measure our technique, we randomly selected an evaluation subset.

Table 5.1 summarizes the results of the performed experiments. The first column contains the URLs of the evaluated website domains (the URL of the key page). For each benchmark, column `DOM nodes` shows the number of nodes of the key page's DOM tree; 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

[1] Some authors answered that their benchmarks were not stored for future use, or that they did not save the gold standard.

the number of correctly retrieved nodes divided by the number of nodes in the gold standard; column `Precision` shows the number of correctly retrieved nodes divided by the number of retrieved nodes; finally, column `F1` shows the F1 metric that is computed as $(2 * P * R)/(P + R)$ being P the precision and R the recall.

Benchmark	DOM nodes	Template nodes	Total retreived	Template retreived	Recall %	Precision %	F1 %
water.org	948	711	665	665	93,53 %	100 %	96,66 %
www.jdi.org.za	626	305	305	305	100 %	100 %	100 %
stackoverflow.com	6450	447	459	447	100 %	97,39 %	98,68 %
www.eclipse.org	256	156	160	152	97,44 %	95,00 %	96,20 %
www.history.com	1246	669	579	576	86,10 %	99,48 %	92,31 %
www.landcoalition.org	1247	393	433	387	98,47 %	89,38 %	93,70 %
es.fifa.com	1324	276	239	234	84,78 %	97,91 %	90,87 %
cordis.europa.eu/fp7/ict/fire/	959	335	327	326	97,01 %	99,39 %	98,19 %
clotheshor.se	459	231	225	225	97,40 %	100 %	98,68 %
www.emmaclothes.com	1080	374	360	360	96,26 %	100 %	98,09 %
www.cleanclothes.org	1335	266	286	264	99,25 %	92,31 %	95,65 %
www.mediamarkt.es	805	337	329	329	97,63 %	100 %	98,80 %
www.ikea.com/gb/en/	1545	407	564	406	99,75 %	71,99 %	83,63 %
www.swimmingpool.com	607	499	349	349	69,94 %	100 %	82,31 %
www.skipallars.cat/en/	1466	842	828	828	98,34 %	100 %	99,16 %
www.tennis.com	1300	624	542	542	86,86 %	100 %	92,97 %
www.tennischannel.com	661	303	227	227	74,92 %	100 %	85,66 %
www.turfparadise.com	1057	726	815	724	99,72 %	88,83 %	93,96 %
riotimesonline.com	2063	879	861	861	97,95 %	100 %	98,97 %
www.beaches.com	1928	1306	1171	1171	89,66 %	100 %	94,55 %
users.dsic.upv.es/~jsilva/ /wwv2013/index2.html	197	163	163	163	100 %	100 %	100 %
users.dsic.upv.es/~dinsa/en/	241	74	89	74	100 %	83,15 %	90,80 %
www.engadget.com	1818	768	445	439	57,16 %	98,65 %	72,38 %
www.bbc.co.uk/news/	2991	364	353	353	96,98 %	100 %	98,47 %
www.vidaextra.com	2331	1137	992	992	87,25 %	100 %	93,19 %
www.ox.ac.uk/staff/	948	525	533	522	99,43 %	97,94 %	98,68 %
clinicaltrials.gov	543	389	392	378	97,17 %	96,43 %	96,80 %
en.citizendium.org	1083	414	447	414	100 %	92,62 %	96,17 %
www.filmaffinity.com/es/	1333	351	355	351	100 %	98,87 %	99,43 %
edition.cnn.com	3934	192	174	174	90,63 %	100 %	95,08 %
www.lashorasperdidas.com	1822	553	536	536	96,93 %	100 %	98,44 %
labakeryshop.com	1368	218	169	159	72,94 %	94,08 %	82,17 %
www.felicity.co.uk	300	232	232	232	100 %	100 %	100 %
www.thelawyer.com	3349	1293	1443	1213	93,81 %	84,06 %	88,67 %
www.us-nails.com	249	171	227	171	100 %	75,33 %	85,93 %
www.informatik.uni-trier.de	3085	64	56	56	87,50 %	100 %	93,33 %
www.wayfair.co.uk	2671	660	665	640	96,97 %	97,71 %	97,34 %
catalog.atsfurniture.com	340	301	304	301	100 %	99,01 %	99,50 %
www.glassesusa.com	1794	1506	1498	1498	99,47 %	100 %	99,73 %
www.mysmokingshop.co.uk	575	407	421	407	100 %	96,67 %	98,31 %
Average	1458	497	480	461	93,53 %	96,15 %	94,34 %

Table 5.1: Results of the experimental evaluation

Experiments reveal a very high average precision and recall: more than 93% in both cases. This is the highest recall and precision that we have seen in a tool for template detection. We have observed that other techniques obtain good values of F1 in certain webpages, but their average precision,

recall and F1 are significantly lower than ours. For instance, in [2] the average precision is always lower than 91% in template removal. In [21], the authors get an F1 higher than 95% in some cases, but their average F1 is significantly lower than ours. Other techniques related to content extraction like [10] and [22] get F1 values between 80% and 94% in certain cases

Observe that some benchmarks produced a recall of about 70%. These benchmarks are particularly difficult ones that produce the same problem in previous techniques such as [21]. The problem in these benchmarks is that some webpages pointed by the key page's links do not use exactly the same template than the key page (i.e., some webpages feature the menu, or the necessary links in some form, but they do not implement the whole template). Therefore, the intersection with these webpages produces one with less items, that is, a webpage with a lower recall. The interested reader is referred to the code of these webpages and their gold standard that are publicly available in the benchmarks repository.

5.2 Optimizations and tunning

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; and the voting system of Algorithm 3 needs to know how many votes are needed to consider a DOM node as part of the template. In this section we discuss how to know the impact of these parameters, and we give concrete values to our algorithms based on empirical evaluation.

5.2.1 Optimization experiments

In order to determine the best parameters, we created a set of 20 training benchmarks. Then, we prepared a set of experiments that helped us to determine all the variables we wanted to test.

At the end we did 478720 experiments repeating the set of 20 training benchmarks with different parameter combinations. Each experiment took 4.5 seconds on average. Therefore, the execution time for all the experiments was about 600 hours.

The execution of all the experiments produced the following average values:

1. Average precission = 63,39
2. Average recall = 94.91
3. Average F1 = 67,15

But, the optimal combination of values obtained is:

1. Optimal precission = 93,53
2. Optimal recall = 96,15
3. Optimal F1 = 94,34

The following sections present the optimal values found for each parameter.

5.2.2 Determining n and t parameters

Algorithm 2 computes a n-CS in a website. As we previously explained, there are several combinations of webpages that form a CS. One could think that the more links the key page has, 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.

As we introduced in section 3.1, given a n-CS, we call t to the threshold used to decide in how many webpages must be a node to be considered part of the template. We evaluated a collection of webpages with different sizes for the CS, from size 1 to size 8, and with all possible values of t. We determined that the best value for n is 8 and the best value for t is 4. But we have considered that the best values are not the optimal values because the time needed to compute an 8-CS with $t = 4$ is excessively high. Therefore, we determined that the optimal value for n is 3 and the optimal value for t is 2. There are better combinations for n and t, but they increase the time to compute the ETDM. In certain cases, increasing the n parameter in one unit almost doubles the time needed to compute the ETDM. Therefore we consider $n = 3$ and $t = 2$ optimal values because the computing time with these values is reasonably acceptable. As a result, all our experiments have been done using these values for n and t.

t \ n	1	2	2	4	5	6	7	8
1	75,90 %	89,22 %	88,97 %	86,82 %	86,84 %	86,28 %	86,08 %	86,06%
2		88,16 %	92,35 %	92,11 %	92,08 %	93,82 %	93,31 %	93,15%
3			88,26 %	92,81 %	92,81 %	92,54 %	92,24 %	94,11%
4				87,04 %	92,84 %	92,15 %	92,52 %	94,63%
5					86,60 %	91,87 %	92,28 %	92,68%
6						86,46 %	90,86 %	91,73%
7							83,58 %	90,49%
8								83,52%
BEST	75,90%	89,22%	92,35%	92,81%	92,84%	93,82%	93,31%	94,63%

Table 5.2: Determining values n and t

Figure 5.2 shows the F1 values obtained with different combinations of values for n and t. We have chosen $n = 3$ and $t = 2$ as the best combination

considering F1 and computation time; but there are several better combinations like $n = 4$ and $t = 3$, $n = 5$ and $t = 4$, etc. if we only consider F1. Our choice produces a value for F1 very similar to other combinations but a significantly better computation time. We observed that increasing n and t makes the time to compute the ETDM sensibly higher. Therefore, we repeated the experiments to measure the time needed to compute all the possible combinations.

t \ n	1	2	3	4	5	6	7	8
1	3 s.	6,1 s.	10,7 s.	13,1 s.	15,91 s.	21,17 s.	23,08 s.	22,65 s.
2		4,9 s.	10,4 s.	12,7 s.	15,44 s.	20,76 s.	22,4 s.	22,65 s.
3			6,91 s.	12,7 s.	15,45 s.	20,68 s.	22,42 s.	22,71 s.
4				8,8 s.	15,41 s.	20,72 s.	22,51 s.	22,77 s.
5					10,62 s.	20,7 s.	22,39 s.	22,76 s.
6						13,34 s.	22,49 s.	22,8 s.
7							15,03 s.	22,74 s.
8								15,97 s.

Table 5.3: Determining values n and t

Figure 5.3 presents the average time needed to compute the ETDM in seconds for different n and t values. Our selected combination ($n = 3$ and $t = 2$) has a computation time of 10.4 seconds, and the best combination ($n = 8$ and $t = 4$) has a computation time of 22.77 seconds. However, the best combination only gets an F1 value 2.3% higher than our selected combination. This is why we choose $n = 3$ and $t = 2$ in our implementation: it is almost as good as the best combination, but it needs less than half of the computation time.

5.2.3 Making precise the comparison of DOM nodes

Algorithm 3 needs to compare DOM nodes to define a ETDM. As we introduced in section 3.2, deciding whether two DOM nodes of two different webpages are the same node is an important problem. From the implementation point of view, it is a difficult problem because some parts of the DOM tree can be missing, or changed. There can exist new DOM nodes that change the relative position between the original nodes, etc.

In section 3.2 we explained that in order to compare two DOM nodes we consider the following information:

- The HTML tag of the node
- Node identifier
- Class name of the node
- Position of the node in the DOM tree

- Attributes of the node
- Children of the node in the DOM tree

We also assigned a weighting system based on the following DOM node components: Classes, Position, Attributes and Children. At first, the weighting was 40% for classes, 40% for position, 10% for attributes and 10% for children. We experimented changing the weights and finally we found that the best results happen when the weights of the position and attributes were 20%. The weight of the children is only 10% relevant. As a result, we observed that the classes is the most important aspect when comparing two nodes, 50%. This optimization increased the F1 value in a 2%.

The node comparison added another need to the implementation. When comparing the classes of two nodes, we found the need to determine what happened when the nodes had no classes. When the classes attribute was not defined, the node could not be reliably compared with other nodes without classes attribute. We needed to add a penalization when comparing two nodes with no classes, because we cannot be sure if two nodes without classes are the same or not. This penalization is defined as a constant called *notClassesProbability*. Our experiments demonstrate that a value higher than 0,75 and lower than 1 produced the best results. Likewise, we had to define a similar constant to determine what happened when two nodes had no attributes, which is called *notAttributesProbability*, and another one to determine when two nodes had no children, called *notChildrenProbability*. Experiments revealed that the best values for these constants are 0,25 for *notAttributesProbability* and 1 for *notChildrenProbability*.

We also defined a boolean variable called *activatePosition*. When this variable is true, it checks the order of the nodes when they match with other nodes, e.g., if the algorithm resolves that two nodes A and B are the same, another node prior to A cannot match to a node after B. This is used to guarantee the order of the nodes when comparing nodes to find the template, if the order of the nodes is not correct, we will be unable to find the correct template. Experiments revealed that this optimization provides an improvement about 1% to the F1 value.

The addition of a constant called *similarityThreshold* has been an important optimization. This constant establishes the nodes mapping threshold value that was introduced in Definition **??**. Two nodes cannot be the same if their calculated probability is lower than the probability threshold. If this constant is not used, the mapping between two nodes will be always produced, even if their are not probably the same node because their probability is low. Therefore, this constant prevents the mapping of two nodes with low probability. Our experiments reveal that with the introduction of this constant, the F1 value has been improved about a 2,5%.

5.2.4 Domain boundaries

Another important optimization 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, the CS represents a set of intra-domain linked webpages (usually by a common menu).

5.2.5 Implementation and experiments download

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

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

Chapter 6

Benchmark suite

6.1 Introduction

In the last decade, there have been important advances that produced several techniques for both template detection and content extraction. Hybrid methods that exploit the strong points of several techniques have been defined too. In order to test, compare and tune these techniques, researchers need:

- collections of benchmarks that are heterogeneous (to ensure generality of the techniques) and
- a gold standard (to ensure the same evaluation criteria).

A benchmark suite is essential to measure the performance of these techniques, and to compare them with previous approaches. Benchmark suites are used in the testing phase and in the evaluation phase. The testing phase allows developers to optimize the techniques by adjusting parameters. Once the technique has been tuned, the evaluation phase allows us to know its performance with objective measures. It is obvious that the set of benchmarks used in the testing phase cannot be used in the evaluation phase, thus, they need disjoint sets of webpages.

This chapter presents a benchmark suite together with a gold standard that can be used for template detection and for content extraction. All benchmarks have been labelled so that every HTML element of the webpages indicates whether it should be classified as main content or not, and whether it should be classified as template or not. The suite also incorporates scripts to automatize the benchmarking process.

This suite has been developed as the result of a research project. We developed a new technique for content extraction [12] that was later adapted for template detection [1]. In the evaluation phase, our initial intention was to use a public benchmark suite. We first tried to use the CleanEval [3] suite

of content extraction benchmarks, because it has been widely used in the literature. Unfortunately, it is not prepared for template detection. Then, we contacted the authors of other techniques that had already evaluated their techniques. However, we could not use these benchmarks due to privacy (they belong to a company or project whose results were not shared), copyright (they were not publicly available) or unavailability (they had been lost). Finally, we decided to build or own benchmark suite and make it free and publicly available.

6.2 The TECO Benchmark Suite

TECO (TEmplate detection and COntent extraction benchmarks suite) was created as a benchmark suite specifically designed for template detection and content extraction. It can be used for testing and evaluation of these techniques, and it is formed from 40 real websites downloaded from Internet. We selected heterogenous websites such as blogs, companies, forums, personal websites, sports websites, newspapers, etc. Some of the websites are well known, like the BBC website or the FIFA website, and others are less known like personal blogs or small companies websites. The downloading of the webpages was done in some cases using the OS X software SiteSucker, and in other cases using the Linux command `wget`.

It is important to know how the websites were downloaded and stored, so that other researchers can increase the suite if it is needed. The following command downloads a website from the Linux terminal using the `wget` command:

```
$ wget -convert-links -no-clobber -random-wait -r 3 -p -E -e
robots=off -U mozilla http://www.example.org
```

The meaning of the flags used is:

- `-convert-links`: Converts links so they can work locally.
- `-no-clobber`: Do not overwrite any existing file.
- `-random-wait`: Random waits between downloads.
- `-r 3`: Recursive downloading up to 3 levels of links.
- `-p`: Downloads everything.
- `-e robots=off`: Act as not being a robot.
- `-E`: Get the right file extension.
- `-U mozilla`: Identify as a Mozilla browser.

Each benchmark is composed of:

- A principal webpage, called *key page*. It is the target webpage from which the techniques should extract the main content or the template—note that it is not necessarily the main webpage of the website (e.g., index.html)—.
- A set of webpages that belong to the same website as the key page. This set contains all those webpages that are linked by the key page, and also the webpages linked by them.

6.2.1 Producing the gold standard

The suite comes with a gold standard that can be used as a reference to compare different techniques. The gold standard specifies for each key page what parts form the template. This is indicated in the own webpage by using HTML classes that indicate what elements are classified as *notTemplate*. It has been produced manually by careful inspection of the websites and mixing the opinion of several people.

In particular, once all the websites were downloaded (the key page and two levels of linked webpages in the same domain), four different engineers did the following independently:

- They manually explored the key page and the webpages accessible from it to decide what part of the webpage is the template and what part is the main content.
- They printed the template and the main content of the webpage.

Then, the four engineers met and performed again these two actions but now all together sharing their individual opinions. Using the results of this agreement, each website was prepared for both, template detection and content extraction. On the one hand, all elements from the key page not belonging to the template were included in a HTML class called *notTemplate*. This way, a template detection tool can automatically compare its output with the nodes not belonging to the *notTemplate* class. On the other hand, all elements belonging to the main content were included in an HTML class called *mainContent*. Therefore, a content extraction tool can easily compare its output with the nodes belonging to that class.

6.2.2 Benchmark details

A classification of the benchmarks is important and useful depending on the application and technique that is being fed with them. We provide different classifications according to the purpose and properties of the benchmarks. First, all benchmarks have been classified into five groups:

Companies / Shops, Forums / Social, Personal websites / Blogs, Media / Communication, Institutions / Associations.

Table 6.1 shows this classification together with the URLs from which we extracted the benchmarks.

Website type	**Original URL of the webpage**
Companies / Shops	clotheshor.se
	www.emmaclothes.com
	www.mediamarkt.es
	www.ikea.com/gb/en.html
	www.swimmingpool.com
	www.skipallars.cat/en.html
	www.turfparadise.com
	www.beaches.com
	www.felicity.co.uk
	www.us-nails.com
	www.wayfair.co.uk
	catalog.atsfurniture.com
	www.glassesusa.com
	www.mysmokingshop.co.uk
Forums / Social	stackoverflow.com
	www.filmaffinity.com/es/main.html
Personal / Blogs	users.dsic.upv.es/∼jsilva/wwv2013/index2.html
	users.dsic.upv.es/∼dinsa/en/index.html
	labakeryshop.com
Media / Communication	www.history.com
	www.tennis.com
	www.tennischannel.com
	riotimesonline.com
	www.engadget.com
	www.bbc.co.uk/news
	www.vidaextra.com
	en.citizendium.org
	edition.cnn.com
	www.lashorasperdidas.com
	www.thelawyer.com
Institutions / Associations	water.org
	www.jdi.org.za
	www.eclipse.org
	www.landcoalition.org
	es.fifa.com
	cordis.europa.eu/fp7/ict/fire.html
	www.cleanclothes.org
	www.ox.ac.uk/staff/index.html
	clinicaltrials.gov/ct2/search/index/index.html
	www.informatik.uni-trier.de/∼ley/pers/hd/s/Silva_Josep.html

Table 6.1: Sources of the benchmarks

Table 6.2 shows some properties of the benchmarks. Here, column **Nodes**

indicates the total number of DOM nodes in the key page, column **Tem. Nodes** shows the number of DOM nodes that belong to the template and column **M.C. Nodes** refers to the number of DOM nodes that belong to the main content.

Id	Benchmark	Nodes	Tem. Nodes	M.C. Nodes
1	water.org/index.html	948	711	237
2	www.jdi.org.za/index.html	626	305	225
3	stackoverflow.com/index.html	6450	447	6003
4	www.eclipse.org/index.html	256	156	100
5	www.history.com/index.html	1246	669	260
6	www.landcoalition.org/index.html	1247	393	588
7	es.fifa.com/index.html	1324	276	737
8	cordis.europa.eu/fp7/ict/fire.html	959	335	179
9	clotheshor.se/index.html	459	231	228
10	www.emmaclothes.com/index.html	1080	374	706
11	www.cleanclothes.org/index.html	1335	266	1069
12	www.mediamarkt.es/index.html	805	337	40
13	www.ikea.com/gb/en.html	1545	407	1138
14	www.swimmingpool.com/index.html	607	499	176
15	www.skipallars.cat/en.html	1466	842	573
16	www.tennis.com/index.html	1300	463	676
17	www.tennischannel.com/index.html	661	303	148
18	www.turfparadise.com/index.html	1057	726	322
19	riotimesonline.com/index.html	2063	879	969
20	www.beaches.com/index.html	1928	1306	149
21	users.dsic.upv.es/∼jsilva/wwv2013/index2.html	197	163	34
22	users.dsic.upv.es/∼dinsa/en/index.html	241	74	167
23	www.engadget.com/index.html	1818	768	1050
24	www.bbc.co.uk/news/index.html	2991	364	1360
25	www.vidaextra.com/index.html	2331	1137	1194
26	www.ox.ac.uk/staff/index.html	948	525	410
27	clinicaltrials.gov/ct2/search/index/index.html	543	389	120
28	en.citizendium.org/index.html	1083	414	667
29	www.filmaffinity.com/es/main.html	1333	351	976
30	edition.cnn.com/index.html	3934	192	3742
31	www.lashorasperdidas.com/index.html	1822	553	722
32	labakeryshop.com/index.html	1368	218	962
33	www.felicity.co.uk/index.html	300	232	68
34	www.thelawyer.com/index.html	3349	1293	1580
35	www.us-nails.com	250	184	35
36	www.informatik.uni-trier.de	3085	64	3021
37	www.wayfair.co.uk/index.html	1950	702	437
38	catalog.atsfurniture.com/index.html	340	301	39
39	www.glassesusa.com/index.html	1952	1708	244
40	www.mysmokingshop.co.uk/index2.html	575	407	168

Table 6.2: Benchmark properties

The benchmarks were also classified according to the number of webpages

that implement the template. Table 6.3 shows this information. Here, the identifier of the benchmarks (**Id**) comes from Table 6.2. For each benchmark, column **VL** indicates the number of hyperlinks in the main menu, column **TT** shows the number of webpages accessible from the main menu that implement entirely the template, column **PT** indicates the number of webpages accessible from the main menu that implement partially the template, column **DT** shows the number of pages accessible from the main menu that do not implement the template at all, and finally, column **Notes** explains, when applicable, why not all webpages implement the template.

Id	VL	TT	PT	DT	Notes (peculiarities)
1	9	0	9	0	All pages add a block in the footer that does not belong to the key page.
2	10	10	0	0	
3	4	4	0	0	
4	8	8	0	0	
5	12	5	6	1	The website uses two different footers (hence, two templates). Therefore, some pages only implement partially the template of the key page.
6	26	7	19	0	All pages share the same header and footer but there are pages with a different layout.
7	8	0	8	0	Some pages use two columns while other use three. All of them are different to the key page.
8	24	5	19	0	Some pages use two columns while other use three. All of them are different to the key page.
9	6	6	0	0	
10	6	6	0	0	
11	6	6	0	0	
12	16	16	0	0	
13	10	0	10	0	The submenu appears inside the main content.
14	16	16	0	0	The main content of the key page uses a layout that is different to the other pages.
15	42	42	0	0	
16	13	13	0	0	
17	29	0	29	0	All pages use more blocks than the key page. For instance, advertisement blocks.
18	74	74	0	0	
19	23	23	0	0	
20	77	77	0	0	
21	12	12	0	0	
22	5	5	0	0	
23	65	65	0	0	
24	5	0	5	0	There are several different templates (but they are very similar).
25	7	7	0	0	
26	7	7	0	0	All pages share the same template. There is a breadcrumb inside the main content.
27	36	36	0	0	
28	32	32	0	0	
29	32	32	0	0	
30	13	13	0	0	All pages share the same template, but the header is a bit different between the key page and the other pages.
31	11	11	0	0	All pages share the same template, but the header is a bit different between the key page and the other pages.
32	l4	4	0	0	All pages share the same template. There is a big amount of javascript.
33	6	6	0	0	
34	69	69	0	0	
35	10	10	0	0	
36	6	5	0	1	One page linked from the main menu uses a different template.
37	377	377	0	0	
38	6	6	0	0	
39	86	86	0	0	
40	35	35	0	0	

Table 6.3: Template data of the benchmarks

6.2.3 Guidelines for using the suite

Downloading and configuring the suite

TECO is freely distributed and can be downloaded from the URL:

http://www.dsic.upv.es/∼jsilva/retrieval/teco

After downloading the suite, a directory that contains 40 folders, one for each website, is created. Table 6.4 shows the path to the key page of each benchmark.

Rules for using the suite and report

All researchers and developers that use TECO must follow two basic principles:

1. They must publish their results so that they are publicly available.
2. They must provide enough information so that anyone can easily duplicate their experiments.

Id	Path to the key page
1	pages/water.org/index.html
2	pages/www.jdi.org.za/index.html
3	pages/stackoverflow.com/index.html
4	pages/www.eclipse.org/index.html
5	pages/www.history.com/index.html
6	pages/www.landcoalition.org/index.html
7	pages/es.fifa.com/index.html
8	pages/cordis.europa.eu/fp7/ict/fire.html
9	pages/clotheshor.se/index.html
10	pages/www.emmaclothes.com/index.html
11	pages/www.cleanclothes.org/index.html
12	pages/www.mediamarkt.es/index.html
13	pages/www.ikea.com/gb/en.html
14	pages/www.swimmingpool.com/index.html
15	pages/www.skipallars.cat/en.html
16	pages/www.tennis.com/index.html
17	pages/www.tennischannel.com/index.html
18	pages/www.turfparadise.com/index.html
19	pages/riotimesonline.com/index.html
20	pages/www.beaches.com/index.html
21	pages/users.dsic.upv.es/∼jsilva/wwv2013/index2.html
22	pages/users.dsic.upv.es/∼dinsa/en/index.html
23	pages/www.engadget.com/index.html
24	pages/www.bbc.co.uk/news/index.html
25	pages/www.vidaextra.com/index.html
26	pages/www.ox.ac.uk/staff/index.html
27	pages/clinicaltrials.gov/ct2/search/index/index.html
28	pages/en.citizendium.org/index.html
29	pages/www.filmaffinity.com/es/main.html
30	pages/edition.cnn.com/index.html
31	pages/www.lashorasperdidas.com/index.html
32	pages/labakeryshop.com/index.html
33	pages/www.felicity.co.uk/index.html
34	pages/www.thelawyer.com/index.html
35	pages/www.us-nails.com/Unternehmen/Ueber_Uns/ueber_uns_l2-dat=5860687c.php.html
36	pages/www.informatik.uni-trier.de/∼ley/pers/hd/s/Silva_Josep.html
37	pages/www.wayfair.co.uk/index.html
38	pages/catalog.atsfurniture.com/index.html
39	pages/www.glassesusa.com/index.html
40	pages/www.mysmokingshop.co.uk/index2.html

Table 6.4: Path to the key page of each benchmark

Chapter 7

Conclusions

Web templates are a very useful tool for website developers. By automatically inserting content into templates, website developers and content providers of large web portals achieve high levels of productivity. Not only is the productivity improved, but also the usability. Website developers using web templates produce webpages that are more usable thanks to their uniformity.

This work presents a new technique for template detection. The technique provides a great benefit for website developers because they can automatically extract a clean HTML template of any webpage. This feature is particularly interesting in order 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, pagelets, repeated advertisement panels, and what is particularly important, the main content.

The technique is also beneficial for indexers and crawlers in another way because they can easily detach the content from the template. As a result, they can save resources and optimize the storage space by analyzing the web templates only once.

Our technique uses the initial webpage hyperlinks for identifying a set of webpages that share the same template with a high probability. Then, it uses the DOM structure of the webpages in that set to identify the blocks that are common to the major part of them. These blocks together form the template. To the best of our knowledge, the idea of using the webpage hyperlinks 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. Our implementation and experiments have shown the usefulness of the technique.

This approach could be also used for content extraction. Detecting and removing the template of a webpage is very helpful in order to detect the main content. Firstly, the main content must be formed by DOM nodes that do not belong to the template, so removing the template of a website may result in the main content or the main content plus any other information. Secondly, the main content is usually inside one of the pagelets that are more centered and visible, and with a higher concentration of text.

To sum up, this template detection technique is not only helpful to extract the template of a website, it can also provide significant improvements to content extraction algorithms by removing the template.

Chapter 8

Future work

- **Reduction of the number of webpages loaded:** 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.

- **Integration with content extraction:** The template detection algorithm can be integrated with a content extraction algorithm, like [12], in order to improve its performance. Using a content extraction algorithm before detecting the template makes the template detection process more efficient. The process has to use the content extraction algorithm to remove the main content, thus webpages will have less nodes and the process of computing the ETDM will be more efficient because most of those nodes will belong to the template with a high probability.

  Our technique can also provide a benefit to content extraction algorithms like [12]. Removing the template reduces the amount of DOM information that a content extraction algorithm has to analyze. This could result in more efficient content extraction algorithms because they should analyze less amount of data.

- **Improved DOM node comparison:** This technique needs to compare DOM nodes in order to define a ETDM. As we introduced in section 3.2, the main problem is to decide if two nodes from two different webpages are the same or not. This problem appears because two different webpages surely have two different DOM trees, even if they have share the same template. There can also be new DOM nodes that change the relative position between the original nodes, etc. We plan

to improve the algorithm with the comparison of DOM subtrees. If a template is present in two or more webpages, their DOM trees will share similar parts. Those parts of their DOM trees could be in different positions, but if they really share the template, they will share some DOM subtrees.

- **Atypical template detection:** Sometimes one of the webpages included in the n-CS does not follow the template at all. In this case, the algorithm needs a mechanism to detect that webpages with atypical template, so they can be replaced by another webpage. This feature could be implemented in the algorithm that computes the ETDM. If there are several nodes that match with high probability in all webpages but one (always the same), this webpage will probably implement a different template.

Chapter 9

Contributions

During the thesis writing process, the following articles have been published:

9.1 International publications

1. Julian Alarte, David Insa, Josep Silva, and Salvador Tamarit. Template Extraction Based on Menu Information. In Josep Silva and Antonio Ravara, editors, *Proceedings of the 9th International Workshop on Automated Specification and Verification of Web Systems (WWV 13)*, page Article 5, 2013.

   This paper provides an approach to the template etraction process based on the information provided by the menu of the webpage. The links analyzed to buid the complete subdigraph are in the main menu, so all the webpages that belong to the complete subdigraph share the same menu. Then, an ETDM is obtained with the pages of the CS. Once the ETDM is obtained, the nodes that belong to it are the template nodes.

2. Julian Alarte, David Insa, Josep Silva, and Salvador Tamarit. Automatic Detection of Webpages Candidates for Site-Level Web Template Extraction. In Josep Silva and Antonio Ravara, editors, *Invited Conference of the 10th International Workshop on Automated Specification and Verification of Web Systems (WWV 14)*, page Article 13, 2014.

   This publication is an approach to improve the method of the building a complete subdigraph from a website. It is based on the hyperlink analysis of a webpage.

3. Julián Alarte, David Insa, Josep Silva, and Salvador Tamarit. Automatic Detection of Webpages that Share the Same web Template. In Maurice H. ter Beek and António Ravara, editors, Proceedings 10th International Workshop on *Automated Specification and Verification*

of Web Systems, Vienna, Austria, July 18, 2014, volume 163 of *Electronic Proceedings in Theoretical Computer Science*, pages 2–15. Open Publishing Association, 2014.

The article presents a new way of computing a complete subdigraph of a website based on the analysis of the hyperlinks in the key webpage. The algorithm uses concepts like *hyperlink distance*, *DOM distance*, *link relevance* and *DOM relevance* to ensure that the CS really represents the website template.

9.2 National publications

1. Julián Alarte, David Insa, Josep Silva, and Salvador Tamarit. Site-Level Template Extraction Based on Hyperlink Analysis. *Proceedings of the 14th Spanish Workshop on Programming Languages (PROLE'14)*, sep 2014. to appear in proceedings of PROLE 2014.

   This publication uses two of the main ideas presented in this thesis, the building of a complete subdigraph based on hyperlink analysis, and then, the compute of an ETDM using the CS. However, the ETDM in an simpler way.

Bibliography

[1] Julian Alarte, David Insa, Josep Silva, and Salvador Tamarit. Template Extraction Based on Menu Information. In Josep Silva and Antonio Ravara, editors, *Proceedings of the 9th International Workshop on Automated Specification and Verification of Web Systems (WWV 13)*, page Article 5, 2013.

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

[3] Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. Cleaneval: a Competition for Cleaning Web Pages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC'08)*, pages 638–643. European Language Resources Association, may 2008.

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

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

[6] Soumen Chakrabarti. 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)*, pages 211–220, New York, NY, USA, 2001. ACM.

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

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

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

[10] Thomas Gottron. Evaluating content extraction on HTML documents. In Vic Grout, Denise Oram, and Rich Picking, editors, *Proceedings of the 2nd International Conference on Internet Technologies and Applications (ITA'07)*, pages 123–132. National Assembly for Wales, sep 2007.

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

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

[13] Vidya Kadam and Prakash R. Devale. A methodology for template extraction from heterogeneous web pages. *Indian Journal of Computer Science and Engineering (IJCSE)*, 3(3), jun-jul 2012.

[14] Christian Kohlschütter. A densitometric analysis of web template content. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 1165–1166. ACM, apr 2009.

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

[16] Christian Kohlschütter and Wolfgang Nejdl. 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, and Abdur Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*, pages 1173–1182. ACM, oct 2008.

[17] Dat Quoc Nguyen, Dai Quoc Nguyen, Son Bao Pham, and The Duy Bui. 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, pages 232–236. IEEE Computer Society, 2009.

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

[19] Tom Rowlands, Paul Thomas, and Stephen Wan. Web indexing on a diet: Template removal with the sandwich algorithm. In *Proceedings of the 14th Australasian Document Computing Symposium*, 2009.

[20] Kuo Chung Tai. The Tree-to-Tree Correction Problem. *Journal of the ACM*, 26(3):422–433, jul 1979.

[21] Karane Vieira, Altigran S. da Silva, Nick Pinto, Edleno S. de Moura, João M. B. Cavalcanti, and Juliana Freire. A fast and robust method for web page template detection and removal. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM'06)*, pages 258–267, New York, NY, USA, 2006. ACM.

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

[23] Lan Yi, Bing Liu, and Xiaoli Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD'03)*, pages 296–305, New York, NY, USA, 2003. ACM.