Main Content Extraction from Heterogeneous Webpages[⋆]

Julian Alarte, David Insa, Josep Silva, and Salvador Tamarit

Universitat Politècnica de València
Camí de Vera s/n, E-46022 València, Spain
{jalarte,dinsa,jsilva,stamarit}@dsic.upv.es

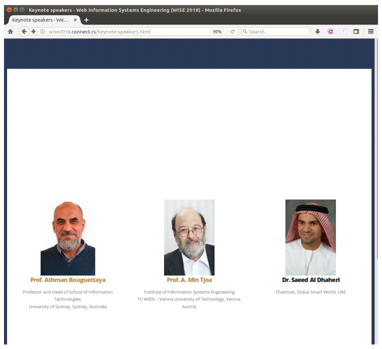
Abstract. Besides the main content, webpages often contain other complementary and noisy data such as advertisements, navigational information, copyright notices, and other template-related elements. The detection and extraction of main content can have many applications, such as web summarization, indexing, data mining, content adaptation to mobile devices, web content printing, etc. We introduce a novel site-level technique for content extraction based on the DOM representation of webpages. This technique analyzes some selected pages in any given website to identify those nodes in the DOM tree that do not belong to the webpage template. Then, an algorithm explores these nodes in order to select the main content nodes. To properly evaluate the technique, we have built a suite of benchmarks by downloading several heterogeneous real websites and manually marking the main content nodes. This suite of benchmarks can be used to evaluate and compare different content extraction techniques.

Keywords: Information Retrieval, Content Extraction, Template Extraction, Block Detection

1 Introduction

Extracting information from webpages is useful for humans and for many different systems. The most important information in a webpage is the main content. However, the main content is almost always next to other noisy elements such as banners, footers, main menus, advertisements, etc. The task of extracting the main content from a webpage consists in isolating the useful information from the noise, removing the elements that do not contain useful knowledge for the user (see, e.g., Figure 1).

[⋆] This work has been partially supported by the EU (FEDER) and the *Spanish Ministerio de Ciencia, Innovación y Universidades*/AEI under grant TIN2016-76843-C4-1-R and by the *Generalitat Valenciana* under grant PROMETEO-II/2015/013 (SmartLogic). Salvador Tamarit was partially supported by the *Conselleria de Educación, Investigación, Cultura y Deporte de la Generalitat Valenciana* under the grant APOSTD/2016/036.



Keynote speakers - Web Information Systems Engineering (WISE 2018) - Mozilla Firefox

Keynote speakers - We...

wise2018.connect.rs/keynote-speakers.html

Prof. Athman Bouguettaya

Professor and Head of School of Information Technologies,
University of Sydney, Sydney, Australia

Prof. A. Min Tjoa

Institute of Information Systems Engineering
TU WIEN – Vienna University of Technology, Vienna, Austria

Dr. Saeed Al Dhaheri

Chairman, Dubai Smart World, UAE

Fig. 1. Main content of WISE 2018's 'keynote-speakers' webpage (extracted with our web content extraction tool).

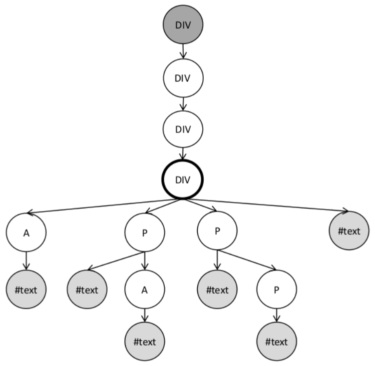


Fig. 2. Main Content DOM nodes example

Our method inputs an arbitrary webpage (the key page) and it outputs a set of DOM nodes representing its main content. Our approach is site-level, so it loads and analyzes several other webpages to detect the main content. This allows it to increase accuracy, because it can detect template (repeated) content.

This approach is divided into three phases:

1. An algorithm selects a set of webpages that belong to the same website of the key page.
2. For each webpage in the set, an algorithm maps its DOM nodes with the DOM nodes of the key page. If it finds that a node of the key page is repeated in another webpage, the algorithm updates a counter, so that it can know how many times each node appears in other webpages.
3. The set of DOM nodes in the key page that are not repeated in any other page are added to a set of *candidate nodes*. These nodes are analyzed in the following way:
   – An algorithm selects only those DOM nodes of the set that do not have ancestors in the set. They form the *reduced set of candidate nodes*.
   – If there is only one node in the *reduced set of candidate nodes*, that node and all its descendants correspond to the main content. However, if there are several candidate nodes in the set:
     • Each candidate node is analyzed to detect the branch of the DOM tree that more likely contains the main content.
     • Finally, the algorithm selects the candidate nodes (DOM nodes that only appear in the key page) that belong to the main content branch.

2 Main content extraction

This section explains the three phases followed in our method to extract the main content of a webpage.

2.1 Set of webpages selection

This section proposes an algorithm to identify a set of webpages (in the following n-SET) from the same website of the key page that very likely share the same template of the key page. The process of selecting the n-SET is:

1. The algorithm analyzes the key page and extracts a set containing all the hyperlinks that point to webpages in the same domain.
2. Then, the algorithm sorts the hyperlinks of the set.
3. Finally, the algorithm selects the first n hyperlinks of the sorted set and collects the set of webpages pointed by these hyperlinks.

The first step is trivial. The DOM tree of the key page is traversed and those hyperlink nodes (with the HTML A tag) that point to webpages in the same domain as the key page, and that do not point to the key page are collected.

Once the set of valid hyperlinks of the key page is created, it is necessary to establish an order of relevance. Thereby, the most related hyperlinks to the key page will be positioned in the first places. For this, we use a combination of the two metrics proposed in [3]: the *hyperlink distance* and the *DOM distance*.

The final set of webpages (the n-SET) returned in this phase contains those hyperlinks with a *hyperlink distance* as closer as possible to zero. Webpages with the same *hyperlink distance* are sorted according to their *DOM distance*, selecting hyperlinks as far as possible (among all of them) in the DOM tree (see [2] for details).

2.2 Webpages mapping

In order to identify the nodes of the key page repeated in other webpages of the n-SET, we use a tree mapping algorithm called *equal top-down mapping* (ETDM) [3]. This mapping analyzes two DOM trees and establishes a correspondence between their nodes.

After we have built the *set of webpages* (n-SET), we compute an ETDM between the key page and each webpage in the n-SET. We have implemented an algorithm that performs each comparison by traversing the DOM trees top-down. Specifically, starting at the root, the algorithm tries to map each node of the key page with the nodes in the other webpage that are at the same depth. If two nodes can be mapped, it means that both nodes are equal (represented with $n_1 \triangleq n_2$), and the algorithm updates an attribute (it works as a counter) called *occurrences* on the node in the key page. This attribute indicates the number of times a node is repeated in the webpages of the n-SET. Then, the algorithm recursively continues with the children of both mapped nodes. When one node cannot be mapped, the algorithm stops exploring the descendants of this node.

Algorithm 1 inputs a key page and a set of webpages (n-SET), and it outputs the same key page including a counter for each node. This counter specifies the number of times a node of the key page appears in the webpages of the n-SET. This counter is used to identify template nodes of the key page: those where the counter is greater than zero (i.e., they are repeated in other webpages of the

Algorithm 1 Compute the number of occurrences of each node in the key page

```
Input: A key page $p_k = (N_1, E_1)$ and a set of $n$ webpages $P$.
Output: The key page $p_k$ equipped with a variable occurrences for each node.
Initialization: $\forall n \in N_1 \; . \; n.occurrences = 0$.

begin
    $r_1 = root(p_k)$;
    foreach ($p = (N_2, E_2)$ in $P$)
        $r_2 = root(p)$;
        if ($r_1 \triangleq r_2$)
            $r_1.occurrences = r_1.occurrences + 1$;
            $assignOccurrences(r_1, r_2)$;
    return $p_k$;
end

procedure $assignOccurrences$(node $r_1 \in N_1$, node $r_2 \in N_2$)
    foreach ($n_1 \in N_1, n_2 \in N_2 \; . \; n_1 \triangleq n_2, (r_1, n_1) \in E_1$ and $(r_2, n_2) \in E_2$)
        $n_1.occurrences = n_1.occurrences + 1$;
        $assignOccurrences(n_1, n_2)$;
end procedure
```

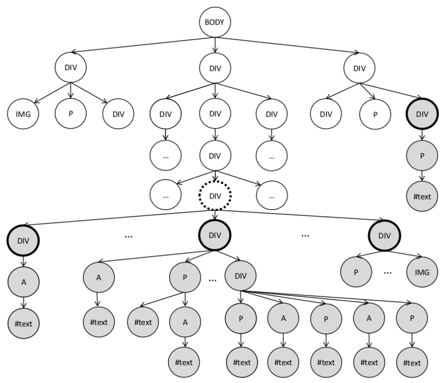


Fig. 3. Set of candidate nodes (excerpt from Figure 1)

website). The main content of the webpage will be probably formed by nodes with $occurrences = 0$.

The output of this phase is a set of *candidate nodes*. For instance, Figure 3 shows an example of a key page where grey nodes represent the candidate nodes. Candidate nodes do not appear in other webpages of the n-SET and, thus, some of them can be used to represent the main content. This is explained in the following section.

2.3 Candidate set reduction

The set of candidate nodes can be simplified by representing subtrees of candidate nodes with their roots. For instance, in Figure 3, all grey nodes are represented with the four bold-line nodes. This new set is called the *reduced set of candidate nodes*, and it can be computed by exploiting the following theorem.

Algorithm 2 Candidate set reduction

```
Input: A set of DOM nodes $candidatesSet$
Output: A reduced set of DOM nodes $reducedSet$.
Initialization: $reducedSet = \{\}$.

begin
    foreach ($node$ in $candidatesSet$)
        $cand = node$;
        while ($parent(cand) \in candidatesSet$)
            $cand = parent(cand)$;
            $candidatesSet = candidatesSet \setminus subtree(cand)$;
        $reducedSet = reducedSet \cup cand$;
    return $reducedSet$;
end
```

Algorithm 3 Main content branch detection

```
Input: A key page $p_k$, and a set $reducedSet$
       of DOM nodes in $p_k$
Output: A DOM node $branch$.

begin
    $count = 0$;
    foreach ($n$ in $reducedSet$)
        $node = parent(n)$;
        if $|subtree(node)| > count$
            $branch = node$;
            $count = |subtree(node)|$;
    return $branch$;
end
```

Theorem 1 (parent-child relation of candidate nodes). *Let* $P = (N, E)$ *be a webpage and let* candidates $\subseteq N$ *be the set of all candidate nodes of* P*. Then,* $n \in$ candidates $\implies \forall n', (n, n') \in E^* \ . \ n' \in$ candidates.

Proof. First, if $|descendants(n)| = 0$ the claim follows trivially. We prove the case when $|descendants(n)| \geq 0$ by contradiction. We assume that $n \in candidates$ $\wedge \ \exists n', (n, n') \in E^* \ . \ n' \notin candidates$. Because $n' \notin candidates$, there must exist a webpage P' with an ETDM mapping M and $(n', n'') \in M$ (for some n''). Moreover, according to the top-down property of ETDM (see [3]), all ancestors of n' also belong to the ETDM mapping M, and therefore, all ancestors of n' (including n) are not candidates: $n \notin candidates$. But this is a contradiction with the premise $n \in candidates$.

Theorem 1 states an important property that is used by Algorithm 2 to compute the *reduced set of candidate nodes*. This algorithm inputs the set of candidate nodes and, for each node, it explores its ancestors. The ancestor with lower depth on the DOM tree that belongs to the set of candidate nodes, is added to a new set of nodes. Finally, the new set of nodes is returned as the *reduced set of candidates*.

2.4 Main content branch detection

If the reduced set of candidates only contains one node, then this node corresponds to the main content of the webpage. Therefore, it is not necessary to execute the following phases and the algorithm returns that node as the main content of the webpage. If, on the contrary, it contains more than one node, then we need to further analyze the DOM tree to remove those nodes that are not main content.

For each parent of a node in the *reduced set of candidates*, the algorithm counts its number of descendants and stores the value. The node with more descendants represents the root node of the main content branch. Considering

Algorithm 4 Candidate set reduction

```
Input: A set of DOM nodes $reducedSet$ and the branch node $branch$
Output: A set of DOM nodes $finalReducedSet$ only including nodes that belong to the main content branch

begin
    foreach ($node$ in $reducedSet$)
        if ($branch \notin ancestors(node)$)
            $reducedSet = reducedSet \setminus \{node\}$
    return $reducedSet$;
end
```

Algorithm 5 Main content selection

```
Input: A set of DOM nodes $reducedSet$
Output: A set of DOM nodes $mainCont$.

begin
    $mainCont = reducedSet$;
    foreach ($n_1, n_2$ in $mainCont$ with $parent(n_1) == parent(n_2)$)
        $mainCont = (mainCont \setminus \{n_1, n_2\}) \cup \{parent(n_1)\}$
    return $mainCont$;
end
```

the number of nodes in a DOM tree, a draw is very difficult, but possible. If it happens, the node selected is the first one in a deep first traversal because it is more likely to appear in the webpage without scrolling. The reason of selecting the parent of each DOM node in the *reduced set of candidates* is that those parent nodes appear in other webpages, so they likely belong to the template of the website or they are probably located in the border between the template and the main content (according to Theorem 1 all the descendants of these parent nodes are candidates and, therefore, they do not appear in other webpages).

Algorithm 3 selects the main content branch node of a webpage.

The main content branch is a node used to select a branch of the DOM tree that contains the main content. All nodes outside this branch are discarded (see Section 2.5). Of course, it is possible that the selected branch contains several candidates, and thus they should be further processed to extract the final set of main content nodes (this is explained in Section 2.6).

For instance, Figure 3 shows that the node selected as the root node of the main content branch is the dotted-line "DIV". This "DIV" node is the parent node of 3 nodes that belong to the reduced set of candidates. Therefore, it is the root node of the main content branch.

2.5 Discarding candidates

Once the branch that contains the main content is selected, the nodes that do not belong to that branch can be removed from the *reduced set of candidate nodes*. Therefore, for each node in the *reduced set of candidate nodes*, an algorithm checks whether the node belongs to the main content branch. If the node does not belong to that branch, it is removed. In practice, this means that whenever we find different separate groups of candidate nodes in the DOM tree, we try to join as many groups as possible by selecting the *branch* node with Algorithm 3, and then we discard the other groups.

Example 1. In Figure 3, once we have computed the branch node (the dotted-line node), Algorithm 4 discards the three grey nodes at the top-right of the tree (because they are not descendants of the branch node).

2.6 Main content selection

Once the candidates that do not belong to the main content branch have been discarded, all the remaining nodes in the *reduced set of candidates* are considered main content. However, sometimes, these nodes can be grouped, e.g., when two of them are sibling nodes (as in Figure 3).

Therefore, the main content is formed by all nodes in the *reduced set of candidates* except for sibling nodes, which are recursively replaced by their parent. This is computed with Algorithm 5.

Example 2. In Figure 5, after having removed the three grey nodes at the top right side, only three nodes remain in the *reduced set of candidates*: the three bold-line sibling nodes. According to Algorithm 5, the final main content of the webpage is the dotted-line node, because it is replaced by its three children.

3 Empirical evaluation

We implemented this technique and integrated all the algorithms proposed as a Firefox's addon, publicly available at: `http://www.dsic.upv.es/~jsilva/retrieval/Web-ConEx/`. For the evaluation, we used the template detection and content extraction benchmark suite (TECO)[1]. Traditionally, most authors of content extraction techniques have measured the recall, precision, and F1 (computed as $(2 * P * R)/(P + R)$, being P the precision and R the recall) of the retrieved words. This implicitly means that they measure the quality of their techniques considering that the main content is text. One important advantage of our technique is that it not only measures the recall, precision, and F1 of the retrieved words, but also of the retrieved DOM nodes. Therefore, we do not only consider the main content as text, it can also contain images, video, and other types of content.

3.1 Precision, recall, and F1 evaluation

To evaluate the precision and recall of our technique, we produced a version of our Firefox addon that automatically executes our content extraction algorithm with all the webpages of the benchmark suite. It displays for each benchmark the recall, precision, and F1 of the retrieved words and the retrieved DOM nodes, and the total execution time.

The only parameter we needed to determine was the size of the set of webpages (the n value of the n-SET) needed by Algorithm 1. In order to develop our technique and determine the optimum size of the n-SET, we measured the recall, precision and F1 of the retrieved text words and DOM nodes for different n-SET sizes.

Table 1 summarizes the results of the performed evaluation experiments, with a n-SET size from 2 to 8, and with a training set of 15 benchmarks. Each row

[1] `http://users.dsic.upv.es/~jsilva/retrieval/teco/`

	DOM nodes			Words			
Size	Recall	Precision	F1	Recall	Precision	F1	Runtime
2	76,97 %	68,15 %	71,63 %	78,88 %	72,52 %	74,54 %	16,32 s.
3	86,25 %	85,80 %	83,53 %	89,24 %	87,33 %	86,62 %	23,59 s.
4	92,92 %	92,59 %	90,26 %	95,90 %	94,76 %	93,71 %	33,14 s.
5	85,21 %	97,91 %	86,30 %	88,50 %	99,59 %	91,61%	41,31 s.
6	85,21 %	98,22 %	86,44 %	88,50 %	99,91 %	91,76 %	50,41 s.
7	85,21 %	98,22 %	86,44 %	88,50 %	99,91 %	91,76 %	59,36 s.
8	84,80 %	98,50 %	86,37 %	88,50 %	99,91 %	91,76 %	68,15 s.

Table 1. Determining the optimal size of the n-SET

is the average of repeating all the experiments in the evaluation subset of 15 benchmarks with a different value for n in the n-SET. Column `Size` represents the size of the n-SET. In addition, the table shows, for the retrieved DOM nodes and the retrieved text words, the average `Recall`, `Precision`, and `F1`.

We determined that a set of webpages of size 4 (4-SET) is the best option because it keeps the best F1 value, both in retrieved DOM nodes (90,26%) and in retrieved words (93,71%). Table 1 reveals that sets of 2 webpages (2-SET) obtain low F1 values (around 70%). On the other hand, sets of webpages with 5 or more webpages obtain similar values of F1. Note that sets of 5 or more webpages do increase the precision up to almost 100%, but the recall decreases and their F1 values are lower than the F1 value of the 4-SET. It is also important to highlight that the size of the set directly affects the performance, because as the size is increased, more webpages must be loaded, and more ETDM mappings must be calculated. This is another good reason to select the 4-SET.

In order to evaluate our main content extraction technique, we selected 30 benchmarks from our benchmark suite as the evaluation subset. For the 30 benchmarks, we computed the `Recall`, `Precision`, and `F1` of the retrieved DOM nodes and the retrieved words. In addition, we computed the `Runtime` in seconds. The results (computed with a 4-SET) are shown in Table 2.

The experiments reveal an average F1 around 88% for retrieved DOM nodes, and an average F1 over 91% for retrieved words. To the best of our knowledge, these values are the highest F1 in the state of the art for benchmarks formed by heterogeneous websites. On the one hand, similar techniques as ours that also use heterogeneous websites, produce the following results: Insa et al. obtain an F1 of 74% [11], Gottron et al. 77% [9], and Shanchan et al. 82% [23]. On the other hand, there are techniques based on evaluating prepared datasets such as Cleaneval [5], BIG5, MYRIAD40, MSS, etc. Other techniques evaluate RSS feeds, or prepared websites (collections of automatically generated webpages that share the same template). These techniques usually obtain high F1 values: Adam et al. obtain an F1 value of 93% [1], Zhao et al. 88% [15], Pasternack et al. 95% [16], and Qureshi et al. 94% [17]. Obviously, the comparison of different techniques should not be done using different datasets, thus, we have compared the techniques against the same benchmarks suite. Results are shown in Table 3, and explained in Section 4.

	DOM nodes			Words			
Benchmark	Rec.	Prec.	F1	Rec.	Prec.	F1	Runtime
wise2018.connect.rs	100,00 %	95,63 %	97,76 %	100,00 %	98,69 %	99,33 %	33,57 s.
www.javiercelaya.es	100,00 %	84,59 %	91,65 %	100,00 %	100,00 %	100,00 %	27,01 s.
www.trendencias.com	99,91 %	71,18 %	83,13 %	100,00 %	82,29 %	90,28 %	189,32 s.
www.turfparadise.com	98,59 %	92,51 %	95,45 %	100,00 %	100,00 %	100,00 %	36,89 s.
www.u-tokyo.ac.jp	100,00 %	92,38 %	96,04 %	100,00 %	100,00 %	100,00 %	12,61 s.
www.savethechildren.net	16,67 %	100,00 %	28,57 %	21,90 %	100,00 %	35,93 %	38,90 s.
college.harvard.edu	100,00 %	84,29 %	91,47 %	100,00 %	88,21 %	93,74 %	89,64 s.
www.raspberrypi.org	100,00 %	82,94 %	90,67 %	100,00 %	86,36 %	92,68 %	11,82 s.
www.annmalaspina.com	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	10,04 s.
dublin.ie	100,00 %	94,59 %	97,22 %	100,00 %	100,00 %	100,00 %	29,20 s.
www.amateurgourmet.com	100,00 %	90,09 %	94,79 %	100,00 %	97,09 %	98,52 %	708,39 s.
www.museodelprado.es	98,43 %	97,67 %	98,05 %	99,32 %	100,00 %	99,66 %	18,89 s.
www.rfet.es	99,90 %	96,97 %	98,41 %	99,73 %	97,64 %	98,68 %	128,88 s.
www.centralparknyc.org	71,23 %	72,22 %	71,72 %	100,00 %	58,59 %	73,89 %	121,43 s.
manytools.org	100,00 %	66,19 %	79,65 %	100,00 %	91,49 %	95,56 %	32,49 s.
clotheshor.se	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	7,47 s.
www.unicef.org	100,00 %	99,48 %	99,74 %	100,00 %	97,56 %	98,77 %	61,82 s.
www.news-medical.net	99,59 %	77,71 %	87,30 %	100,00 %	88,85 %	94,09 %	104,72 s.
teachreal.wordpress.com	99,29 %	67,57 %	80,41 %	100,00 %	90,70 %	95,12 %	37,90 s.
www.grandcentralterminal.com	100,00 %	97,95 %	98,96 %	100,00 %	100,00 %	100,00 %	77,00 s.
www.cleanclothes.org	100,00 %	88,12 %	93,69 %	100,00 %	91,94 %	95,80 %	58,37 s.
riotimesonline.com	99,74 %	63,46 %	77,57 %	100,00 %	65,90 %	79,45 %	140,09 s.
www.ox.ac.uk	96,93 %	100,00 %	98,44 %	100,00 %	100,00 %	100,00 %	80,31 s.
www.filmaffinity.com	99,80 %	100,00 %	99,90 %	100,00 %	100,00 %	100,00 %	72,31 s.
www.coiicv.org	97,70 %	100,00 %	98,84 %	99,47 %	100,00 %	99,73 %	43,38 s.
www.thelawyer.com	91,31 %	82,66 %	86,77 %	94,47 %	86,92 %	90,53 %	281,72 s.
www.toureiffel.paris	95,37 %	100,00 %	97,63 %	99,71 %	100,00 %	99,86 %	38,55 s.
institute-events.mit.edu	88,76 %	96,34 %	92,40 %	94,35 %	100,00 %	97,09 %	83,40 s.
www.w3schools.com	99,76 %	100,00 %	99,88 %	100,00 %	100,00 %	100,00 %	734,78 s.
stackoverflow.com	5,97 %	98,05 %	11,26 %	2,76 %	94,92 %	5,37 %	1237,59 s.
Average - using metrics	92,08 %	89,53 %	87,91 %	93,72 %	93,90 %	91,14 %	151,62 s.
Average - random pages	83,80 %	75,70 %	73,54 %	84,18 %	81,41 %	79,47 %	113,68 s.

Table 2. Evaluation of the precision, recall, F1, and runtime

We also wanted to evaluate the usefulness of the analyses performed in Section 2.1. If we select the pages in the n-SET randomly (instead of using the *hyperlink distance* and *DOM distance* metrics), then our technique decreases its F1 value more than 10% (see the last two rows of Table 2). "using metrics" is the average when using these metrics, while "random pages" do not use the metrics and it selects the pages randomly. Experiments reveal that, in retrieved nodes, the F1 decreases more than about 15%; being about 12% in retrieved words. Regarding the algorithm runtime, the random selection of pages is 38 seconds faster (on average).

3.2 Performance evaluation

Column `Runtime` in Table 2 and Figure 4 show the time needed to extract the main content from different webpages. 50% of the benchmarks took less than 60 seconds but, as we expected, usually, the larger (in terms of DOM nodes) the

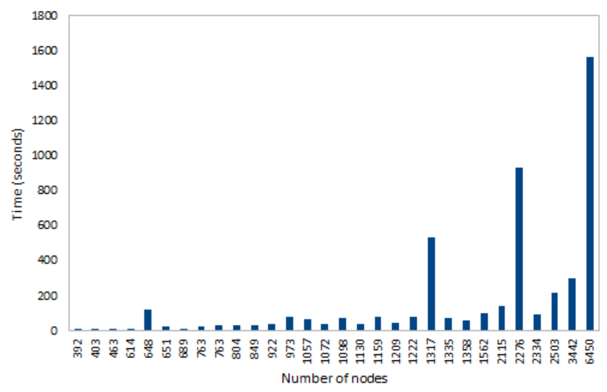


Fig. 4. Relation between webpage size and runtime

webpages are, the more time the algorithm needs to process them. For larger webpages (those with thousands of DOM nodes) the algorithm takes minutes. We observed that the runtime not only depends on the number of DOM nodes of the website, but it also depends on their structure and features, such as their number of children, their relative position in the DOM tree, their HTML attributes, etc. For that reason, the chart in Figure 4 is irregular, and the runtime does not strictly grow with the number of DOM nodes.

4 Comparison with other algorithms

We wanted to compare our technique with other well-known algorithms ([24, 20, 21, 19, 3]). Obviously, it is not possible to compare different content extractors by just comparing the results reported in the bibliography, because each report uses a different measure (e.g., words, characters, DOM nodes...) and also because their evaluations were done with different benchmarks. Unfortunately, some implementations are proprietary or not available, so we decided to reimplement them from scratch. All of them are now available as open-source at: `http://users.dsic.upv.es/~jsilva/retrieval/Web-TemEx/`

We compared the performance and accuracy of the following techniques:

SST (2003) [24]: This algorithm introduced a new data structure called *Site Style Tree* (SST) to represent a collection of webpages. The SST stores information about the DOM nodes in the selected set of webpages. The most repeated nodes or groups of nodes are more likely to belong to the template of the key page.

RTDM-TD (2006) [20]: This algorithm inputs a set of webpages and compares their DOM trees using a top-down variant of the tree edit distance (TED) algorithm. The intersection of the DOM trees (the nodes repeated in all webpages) is considered as the template.

IWPTD (2008) [21]: This algorithm divides the DOM trees of the webpages into a set of subtrees whose root nodes are associated with concrete HTML tags (i.e., TABLE, DIV, UL, etc.). Then, it compares the text segments inside the subtrees of all the webpages. The text segments that appear in 5 or more webpages are considered template segments. Finally, it computes a ratio to decide whether a subtree belongs to the template or not.

RBMTD (2009) [19]: This algorithm is similar to RTDM-TD but, in this case, it uses a bottom-up variant of the tree edit distance (TED) algorithm. While comparing the DOM trees, it introduces a restriction to classify a common subtree as template: those subtrees that appear in all webpages must be exactly in the same position.

TemEx (2015) [3]: This algorithm selects a set of webpages from the website and uses a mapping between their DOM trees to determine the number of times a node is repeated across the webpages in the set. A node is considered to belong to the template if it is repeated in *t* webpages, being *t* a threshold determined empirically.

We extracted the main content from the test set of 30 benchmarks using all algorithms (see Table 3). Regarding the `recall`, `precision`, and `F1`, the experiments reveal that our algorithm obtains the best F1 values for both, retrieved DOM nodes and retrieved words. The second best `F1` values for both retrieved DOM nodes and words is achieved by TemEx. RBMTD is a very conservative algorithm. It achieved 100% recall in all experiments. So, it is a good choice if retrieving the whole main content is critical. Another interesting observation is that, in all cases, the F1 values are better if they are measured in retrieved words instead of retrieved DOM nodes. This means that these algorithms are more oriented to retrieve text, but they sometimes miss some image or container (e.g., a `DIV` or a `TABLE`) that do belong to the main content. This fact can be observed because we have used both measures to compare the benchmarks. In the case of RBMTD this difference is higher than 15%.

Given these results, those systems that need a high `Recall` should use RBMTD or RTDM-TD, those systems that need a high `Precision` should use our algorithm or TemEx, and those systems that need high performance should use IWPTD.

	DOM nodes			Words			
Algorithm	Rec.	Prec.	F1	Rec.	Prec.	F1	Runtime
SST	66,40 %	42,28 %	48,08 %	70,25 %	50,22 %	54,25 %	554,26 s.
RTDM-TD	99,81 %	40,74 %	53,64 %	100,00 %	55,25 %	68,72 %	92,55 s.
IWPTD	68,33 %	62,53 %	61,24 %	81,19 %	68,87 %	72,55 %	21,03 s.
RBMTD	100,00 %	44,20 %	56,46 %	100,00 %	58,94 %	71,55 %	151,26 s.
TemEx	97,17 %	76,19 %	82,52 %	98,59 %	82,17 %	87,97 %	63,37 s.
Our algorithm	92,08 %	89,53 %	87,91 %	93,72 %	93,90 %	91,14 %	151,62 s.

Table 3. Empirical evaluation of six web content extraction algorithms

With respect to the computation time, it is significantly different for each algorithm. IWPTD is the quickest one, it takes an average of about 21 seconds per benchmark. In contrast, TemEx takes an average of about one minute per benchmark. Our algorithm is slow compared to the others, it takes more than 2,5 minutes per benchmark. This is due to the fact that a 4-SET was used in the configuration (it had to find 4 webpages in the website and compare the key page with them). Reducing this number to 3 would speed up the algorithm potentially reducing the runtime around 30% at the cost of 7% F1 (see Table 1). The slowest algorithm is SST, whose computation time is extremely high compared to the others, above 9 minutes per benchmark on average.

5 Related work

Besides the techniques explained in the previous section, there are other interesting techniques related to our work. We overview some of them in this section. Content extraction, template extraction, menu detection, etc. are interesting topics due to their relation to web mining, searching, indexing, and web development. There are many different approaches that try to face these problems (see, e.g., [23, 10, 22, 7, 11]). Some of these techniques were presented in the CleanEval competition [5], which proposed a collection of examples to be analyzed with a gold standard. This collection of examples was prepared for boilerplate removal and content extraction.

Content Extraction and *template extraction* are very close disciplines. While content extraction tries to isolate the main content pagelets[2] of the webpage, template extraction tries to isolate the template of the webpage. These disciplines are considered an instance of a more general discipline called *Block Detection*, which tries to detect all pagelets that exist in a webpage. In the area of block detection, researchers use three main different approaches to solve the problem:
(1) Using the textual information of the webpage (i.e., the HTML code). The main idea is that the main content on a webpage has more density of text with less labels. For instance, Ferraresi et al. [8] proposed the main content as the largest contiguous text area with the least amount of HTML tags. In the same way, Weninger et al. [22] defined the Content Extraction via Tag Ratios (CETR). This method analyzes the HTML code and computes a ratio (CETR) by counting the number of characters and labels inside each label. Kohlschütter et al. [14, 12] proposed the exploitation of densitometric features based on the observation of the more common terms in webpage templates.
(2) Using a rendered image of the webpage on the browser. Burget et al. [6] proposed an approach based on the idea that the main content of a webpage is often located in the central part and it is often visible without scrolling (see, e.g., Figure 1). Kohlschütter et al. [13] concluded that this kind of techniques are not so widespread as others because rendering webpages for classification is a computational expensive operation.

[2] Bar-Youssef et al. [4] defined a pagelet as a self-contained logical region with a well defined topic of functionality. Accordingly, webpages are composed of pagelets.

(3) Using the representation of the webpage as a DOM tree. Bar-Yossef et al. [4] proposed a method for template detection based on the analysis of the DOM tree. This approach counts the frequent pagelet item sets. Vieira et al. [20], Yi et al. [24] and Alarte et al. [3] proposed techniques based on finding common subtrees in the DOM trees of a set of webpages from the same website. These common subtrees between webpages of the same website are defined as noisy information or template. Yi et al. [24] introduced a data structure called Site Style Tree (SST). The SST summarizes a set of DOM trees of different webpages of the same website. Every DOM node in the webpages is represented in a single tree (the SST), and the repeated nodes (those appearing in different webpages) are identified by using counters in the SST nodes. The most repeated nodes in the SST (those with highest counter values) are more likely to belong to the template of the website.

Vieira et al. [20] proposed the use of optimal mappings between DOM trees. This mapping (RTDM-TD) finds duplicated nodes across webpages from the same website. RTDM-TD (*restricted top-down mapping*) algorithm was proposed by Reis et al.[18].

6 Conclusions

Our work presents a new technique for content extraction from heterogeneous websites. In contrast to other content extraction techniques, our approach is based on DOM nodes, and it allows for extracting not only text as main content, but also images, videos, animations, and other types of content. As it is a site-level technique, it uses the information of several webpages to extract the main content. In particular, it analyzes the key page and extracts its hyperlinks. Once extracted, the hyperlinks are sorted in order to select the webpages that can provide more information about the main content of the key page. We measured that a set of 4 webpages (4-SET) obtains the best values of F1 for both, retrieved DOM nodes and retrieved words. To select the webpages that should be analyzed we propose the use of two metrics (*hyperlink distance* and *dom distance*). The use of these metrics produce (as an average) an increase of 10% F1. To compare the DOM nodes of the webpages we use an ETDM mapping. Once the webpages are compared, we have information about the number of times each DOM node appears in them. In our algorithms, we consider that a DOM node that only appears in the key page more likely belongs to its main content. This could be relaxed. However, we repeated our experiments with other numbers (i.e., considering that the key page's nodes could be repeated 2, 3, or 4 times in other webpages) and the results were worst in all cases.

The idea of mapping the DOM nodes in order to infer the branch of the DOM tree that probably contains the main content is simple but effective. Usually, the main content of a webpage is not repeated on other webpages of the same website, so identifying the non-repeated nodes can lead us to find the main content. Moreover, the main content usually concentrates a large amount of information that is structurally close. The identification of the main content branch follows this idea.

References

1. G. Adam, C. Bouras, and V. Poulopoulos. Cuter: An efficient useful text extraction mechanism. In *2009 International Conference on Advanced Information Networking and Applications Workshops*, pages 703–708, May 2009.
2. J. Alarte, D. Insa, J. Silva, and S. Tamarit. Automatic Detection of Webpages that Share the Same Web Template. In M. H. ter Beek and A. Ravara, editors, *Proceedings of the 10th International Workshop on Automated Specification and Verification of Web Systems (WWV 14)*, volume 163 of *Electronic Proceedings in Theoretical Computer Science*, pages 2–15. Open Publishing Association, July 2014.
3. J. Alarte, D. Insa, J. Silva, and S. Tamarit. Site-level web template extraction based on DOM analysis. In M. Mazzara and A. Voronkov, editors, *Perspectives of System Informatics - 10th International Andrei Ershov Informatics Conference, PSI 2015, in Memory of Helmut Veith, Kazan and Innopolis, Russia, August 24-27, 2015, Revised Selected Papers*, volume 9609 of *Lecture Notes in Computer Science*, pages 36–49. Springer, 2015.
4. Z. Bar-Yossef and S. 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.
5. M. Baroni, F. Chantree, A. Kilgarriff, and S. 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.
6. R. Burget and I. 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.
7. E. Cardoso, I. Jabour, E. Laber, R. Rodrigues, and P. 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.
8. A. Ferraresi, E. Zanchetta, M. Baroni, and S. 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. T. Gottron. Content code blurring: A new approach to content extraction. In *Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, DEXA '08, pages 29–33, Washington, DC, USA, 2008. IEEE Computer Society.
10. T. Gottron. Content Code Blurring: A New Approach to Content Extraction. In A. M. Tjoa and R. 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.
11. D. Insa, J. Silva, and S. 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.

12. C. Kohlschütter. A densitometric analysis of web template content. In J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 1165–1166. ACM, apr 2009.
13. C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In B. D. Davison, T. Suel, N. Craswell, and B. Liu, editors, *Proceedings of the 3th International Conference on Web Search and Web Data Mining (WSDM'10)*, pages 441–450. ACM, feb 2010.
14. C. Kohlschütter and W. Nejdl. A densitometric approach to web page segmentation. In J. G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D. A. Evans, A. Kolcz, K.-S. Choi, and A. Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*, pages 1173–1182. ACM, oct 2008.
15. Z. Li, W. K. Ng, and A. Sun. Web data extraction based on structural similarity. *Knowledge and Information Systems*, 8(4):438–461, Nov 2005.
16. J. Pasternack and D. Roth. Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 971–980, New York, NY, USA, 2009. ACM.
17. P. A. R. Qureshi and N. Memon. Hybrid model of content extraction. *J. Comput. Syst. Sci.*, 78(4):1248–1257, July 2012.
18. D. d. C. Reis, P. B. Golgher, A. S. Silva, and A. H. F. 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. K. Vieira, A. L. da Costa Carvalho, K. Berlt, E. S. de Moura, A. S. da Silva, and J. Freire. On finding templates on web collections. *World Wide Web*, 12(2):171–211, 2009.
20. K. Vieira, A. S. da Silva, N. Pinto, E. S. de Moura, J. a. M. B. Cavalcanti, and J. 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.
21. Y. Wang, B. Fang, X. Cheng, L. Guo, and H. Xu. Incremental web page template detection. In *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*, pages 1247–1248, New York, NY, USA, 2008. ACM.
22. T. Weninger, W. Henry Hsu, and J. Han. CETR: Content Extraction via Tag Ratios. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*, pages 971–980. ACM, apr 2010.
23. S. Wu, J. Liu, and J. Fan. Automatic web content extraction by combination of learning and grouping. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1264–1274, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.
24. L. Yi, B. Liu, and X. 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.