# Accelerating the process of web page segmentation via template clustering


## Jan Zeleny

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic E-mail: izeleny@fit.vutbr.cz

## Radek Burget

Faculty of Information Technology
Brno University of Technology
IT4Innovations Centre of Excellence
Brno, Czech Republic E-mail: burgetr@fit.vutbr.cz



**Abstract:** Segmenting a web page is often one of the initial steps when performing some data mining on that page. We acknowledge that there is a lot of research in the area of segmentation based on visual perception of the web page. In this paper we propose a method how to improve the efficiency of virtually all vision-based segmentation algorithms. Our method, called Cluster-based Page Segmentation, takes the widely spread concept of web templates and utilizes it to improve the efficiency of vision-based page segmentation by clustering web pages and performing the segmentation on the cluster instead of on each page in that cluster. To prove the efficiency of our algorithm we offer experimental results gathered using three different vision-based segmentation algorithms.



**Keywords:** VIPS, vision-based page segmentation, clustering, template, template detection



**Biographical notes:** Jan Zeleny is a PhD student at the Brno University of Technology, Czech Republic. His reserach interests are in the area of Web Data Mining and Information Retrieval in general. He is also interested in the area of electronic privacy in context of these two topics.

Radek Burget received his PhD in Information Technology in 2004 from the Brno University of Technology. He is an assistant professor at the Faculty of Information Technology, Brno University of Technology. His research interests include data mining methods, semi-structured data modeling, knowledge engineering and the semantic web.


## 1 Introduction

In recent years, the World Wide Web has become perhaps the most important source of information in the world. A family of algorithms to process that information grows with it. There are several tasks that we might want to perform on the data on the Web. The largest

group of tasks falls into the area of data mining. These are tasks like information retrieval, content extraction and classification and others. Another big group of tasks targets web page restructuring for mobile devices.

All algorithms working with the data on the web require one common step – initial preprocessing in a form of web page segmentation. The segmentation step works with a simple premise – that the web page is not one coherent block of information. Instead, it contains multiple such blocks, each one containing different type of content [1]. The goal of segmentation algorithms is to identify these blocks so they can be processed separately. In case of the data mining tasks, the segmentation is often complemented by classification with the intended result being identification of blocks that are either relevant in the context of the web page or that are important for the subsequent algorithm. In case of restructuring for mobile devices, the subsequent goal is to rearrange the web page, maybe even remove some blocks that are not considered important.

There is a variety of ways how to perform the segmentation. They differ in all possible aspects, starting with the requirements on the input side and ending with granularity of the output.

Template detection (further referred to as TD) methods are a good example of that. In the literature it is considered being just remotely related but we see some common parts. For example TD identifies page segments as well, even though it just distinguishes two – useful content and the rest. The results of segmentation methods are more generic. It's a set of various blocks, each block being internally consistent in some way, most often either visually or logically.

In our work we focus on unsupervised page segmentation. Although this area has been extensively researched, there are just two usual ways how to approach the task. The first one usually utilizes DOM tree of a web page or its textual content and focuses on performance with the result being potentially inaccurate [2, 3, 4, 5]. The source of this inaccuracy is that large portions of information from the web page, such as computed CSS styles, are dropped. The second one on the other hand strongly prefers accurate results even at the expense of the result being produced more slowly [6, 7, 8, 9]. The most distinct feature of these algorithms is that they work on every web page independently. This feature is both their benefit (no need to have more pages on the input) and problem (scaling).

The motivation of our research is to remove the negative aspect of the independent processing of single web pages by eliminating the need to perform the segmentation on every inspected web page. This significantly boosts performance and scaling of vision-based page segmentation. To achieve the goal, we use clustering algorithm that is based on methods used in template detection. This combination of template detection and vision-based page segmentation addresses the greatest issue of vision-based page segmentation (time complexity) while keeping its level of accuracy. It also keeps the positive aspect of the independent page processing (only a single page is required for the segmentation itself). The method, called *Cluster-based Page Segmentation* (further referred to as CPS), is in fact a complementary algorithm. Its potential usage is not limited to any particular algorithm, it is not even bound to the area of vision-based page segmentation if the complemented algorithm in question and its implementation keep some basic rules.

In this paper we describe structures and algorithms that are comprised in CPS. We then use the theoretical and practical description and demonstrate the efficiency of our algorithm by selecting three vision-based segmentation methods and running them with and without the CPS.

## 2 Related work

Our research belongs into the extended area of web page segmentation. It is extended because we are not strictly focused on page segmentation, as explained above. While the core of our research is in the wider area because we examine possible utilization of template detection, the state of the art is more important in the area of web page segmentation itself.

To understand the area of web page segmentation, it is first important to properly classify possible motivations. Reorganizing a web page for devices with small display is a motivation that is only marginally relevant to our research, therefore it won't be analyzed further.

The most important motivation to segment web pages is preprocessing for data mining techniques. This needs to be done because web pages are semi-structured documents that contain other elements among the useful content, for example HTML tags, various descriptors and other metadata. The most simple option how to perform the preprocessing would be to strip all the metadata. However that approach is not entirely accurate because not all the metadata is easily detectable. A significant portion of what in fact belongs to metadata or is strictly speaking just a noise is presented as a data to user who looks at the page [10]. Back to the motivation, the most obvious is to clean up the metadata and the noise. That can be done by virtually every segmentation method but the template detection methods focus solely on this goal [11, 12]. Built on top of that, the next motivation is to identify semantically distinguished blocks that the page contains [8].

The goal of page segmentation is to find blocks that are internally consistent more than the page itself. The consistency can be either logical or visual, based on input parameters and used segmentation method. Both types of consistency often overlap.

The logical consistency is targeted by DOM-based and text-based segmentation methods [2, 3, 4, 5]. These analyze only textual representation of a web page, either in a form of source code of the page or a DOM tree which is just a model of this text form. The distinctive feature of these methods is that they don't need any complex transformations to perform their task, they completely depend on heuristics applied on some form of the textual representation of the analyzed page. The set of heuristics is also the only factor defining the quality of segmentation results. The array of heuristics can vary from pure text evaluation [2] to complex algorithms taking a wide variety of properties into account [5]. However because these methods don't perform any complex transformation of the textual representation of the page, they always fail to take one very important aspect into account and that is the real layout of the inspected page. Even the text-based DOM model doesn't accurately describe the real relations of individual blocks in consideration of their visual appearance [6]. When we consider CSS rules in all their complexity, the DOM tree and the corresponding tree representing the visual appearance can be very different. As an example, figure 1 demonstrates how individual parts of a web page can be moved or transformed by CSS. The example is very simple and yet the CSS changed the appearance of the page completely.

Contrary to the DOM processing algorithms, vision-based page segmentation methods are based on a simple concept with quite large computing demands. They address the issue of real layout by calculating real values of attributes defining the layout. In its complexity this calculation corresponds to real rendering of a web page, thus it will be referred to as such. The process of rendering is very complex due to complexity of both HTML and CSS specifications. That implies quite high demands both for computational power and time to process one page and that's even before the segmentation itself. Rendered page is often segmented in several iterations [6] which is also very demanding. The most commonly

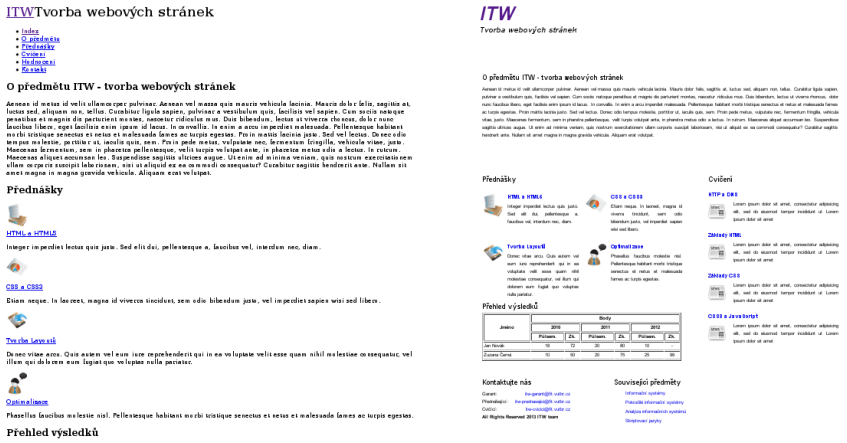


**Figure 1** Influence of CSS: a) is a slice of raw page, b) is the same page with CSS applied

used algorithm in the area of vision base segmentation is VIPS [7] and algorithms using it as a black box and improving its results [8, 9]. Another approach, partially derived from the original VIPS specification, has been offered by Burget [6]. In this paper we will demonstrate results of Cluster-based Page Segmentation working on top of several known implementations of vision-based page segmentation, including VIPS as an industry standard.

Template detection can be perceived as a special type of page segmentation. Template detection methods [11, 12, 13, 14, 15] are used to identify the noise on the processed web page so it can be removed afterwards. Algorithms in the template detection group are based on the concept of page templates [16]. This concept comes from the area of modern web design where a relatively small number of templates is used to dynamically build all the content on a web site. Each template defines core structure of a large set of pages within that site. Physically, each template is a pre-defined code which creates a frame with marked spaces in it. Based on user input, a data set is fetched from the underlying data source and a web page is created by filling these fetched data into the frame. While being a great help to web designers and content authors, templates pose a problem for information retrieval algorithms. A substantial part of every page contains information unrelated to the core topic of the page – for example navigation, advertisement and other noise. In the context of templates and template detection, all this unrelated content is considered to be a part of the template. The job of template detection methods is to correctly separate the aforementioned blank spaces from the pre-defined code.

The focus of template detection methods implies one of the disadvantages these methods have. They can be only used to distinguish between the template and the content, no finer granularity is available. Because of their design, they also usually lack the ability to perform inspection on a single web page. On the other hand they are usually designed to be as fast as possible and they scale very well.

One subset of the template detection methods makes them closely related to our research. Its a subset containing methods that are based on DOM tree comparison [11, 13, 15, 17]. Their core functionality can be used to determine a degree of similarity between two web

pages. This can be represented for example by a simple probability-based equality indicator. If the value of such indicator goes over a pre-defined threshold, the two pages are considered to be significantly similar. In the context of template detection, the significant similarity translates to both pages being based on the same template.

## 3 Cluster-based Segmentation

In our work we consider the following scenario where page segmentation is used as the key part: we have a crawling algorithm that wants to index as many documents as possible. For the initial description of the use case we can define a constraint that only a specific web site will be scanned. Every web page on that site is supposed to be segmented and the result further processed. At this point it doesn't matter what is the subsequent processing going to be, it might be anything starting from information retrieval and ending with semantic classification.

With the standard segmentation approach, every scanned web page is going to be segmented. For some large servers like world-wide news servers this means performing the segmentation task hundreds of thousands of times. Obviously this doesn't scale at all and the time required to process mid- to large-size web site is unacceptably long.

In our research we propose a new way how to deal with scaling and performance problems when processing large sets of web pages. One of the advantages is that the bigger the set of web pages is, the greater optimization this approach achieves. Another advantage is that our algorithm doesn't require all the pages to be processed at once, partial data set can be retrieved, the process interrupted and continued any time later. Our algorithm can be summarized in the following proposition.

**Proposition 1:** *When processing more pages within the same site, it is possible to increase the performance of a segmentation algorithm by performing the actual segmentation only for a limited number of pages and transform these pages to represent their respective template-based clusters. When a page can be matched to an existing cluster, an isomorphic mapping between the page and the structure representing the corresponding cluster can be used to get the results of page segmentation without performing it.*

Most often, the clustering is performed on a complete set of values. However in our case it is not convenient to store all pages and perform the clustering on the entire set of pages. One of the reasons is that it is not possible to estimate upfront how many pages will be in the set. Also there is the issue of continuous content generation. Big web sites like news servers keep publishing new content, therefore the set of web pages will never be completed. All these reasons lead to the conclusion that some form of stream clustering algorithm is required.

This aligns with another preferred feature – to store as few data as possible in order to optimize memory and disk space consumption in practical application. To achieve this goal, we don't store processed web pages at all, we just store minimal structures representing the individual cluster. These structures will be further called Cluster Representatives. Note that while we aim for the structure to be minimal, the basic requirement is to store all the data for situations where the Cluster Representative is used. These will be analyzed further in the following sections.

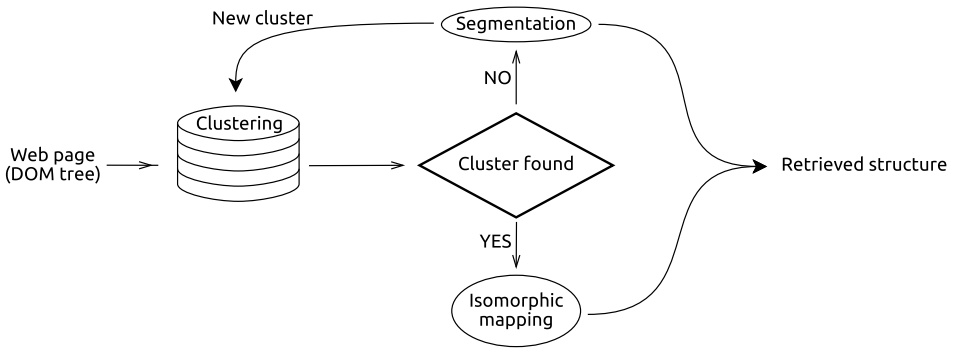


**Figure 2** Block schema of the entire approach

The high-level overview of Cluster-based Page Segmentation is outlined on figure 2. Our design starts at the *clustering* step where we try to identify which cluster the page belongs to. If such cluster is identified, an isomorphic mapping between the Cluster Representative and the page is performed and, by extension of that action, all interesting parts of the web page identified without performing the segmentation. If there is no cluster corresponding to a page on the input, we have to create a new cluster and use the input web page as an initial representative of this cluster. Before we create the representative, we need to segment the page first, as the segmentation result is an important part of each Cluster Representative. The *segmentation* process is considered to be a black box taking DOM tree as an input and returning a set or hierarchy of visual areas on the output. Any algorithm acting in conformity with these requirements can be used for the segmentation. Section 7 covers experiment with three different segmentation algorithms.

## 4 Segmentation methods

Before going more deeply into the structures and algorithms of Cluster-based Page Segmentation, we need to take a closer look at the segmentation methods this algorithm is primarily meant for. This section will be divided into two parts. The first part will cover some currently used segmentation methods that are described in the literature. The second part will be oriented towards an alternative segmentation method that we have designed. In both cases we will outline how these methods work and we will especially focus on their output format. That will be important for the subsequent design of data structures we use in the Cluster-based Page Segmentation.

### *4.1 Hierarchical segmentation methods*

There are two methods that will be described here as representatives of hierarchical segmentation methods. Their common feature is the output format. While the specifics are different, the main characteristic is the same for both – the output format is structured as a tree.

The first representative of this algorithm group is *VIPS* [7]. It segments the page in three steps. The first step is a top-down analysis of the DOM tree where the algorithm identifies visual blocks on the web page using various heuristics. For each identified block a decision is

made whether or not will the block be recursively split further – that creates a tree of blocks. In the second step the algorithm identifies separators between previously identified blocks. The approach is top-down again – at first the entire web page is considered a separator and it is recursively split to smaller ones. The last step of VIPS is content structure construction. In involves merging some boxes to achieve the right granularity.

After being processed by VIPS, the web page is represented by a set of blocks $O$, a set of separators $\Phi$ and a relation between blocks $\delta$(two blocks are in relation if they are adjacent). The most important feature of blocks is that they are not overlapping. Each block in the set is recursively segmented and then represented by another set of blocks, separators and relation. Formally, it is designated as follows [7]: $\Omega = (O, \Phi, \delta)$ where $O = \Omega_1, \Omega_2, \ldots \Omega_n$ and every $\Omega_i$ is defined in the same way as $\Omega$. This definition implies the tree structure of the returned result.

For each block the information about its position and size is absolutely essential, as well as its internal coherence. Also the alignment with its parent is used [9]. For separators it is important to store their visual impact, which can be in form of width or visibility.

Compared to VIPS, Burget takes the opposite approach in his work [6]. First of all his algorithm doesn't work on a DOM tree itself. The web page is rendered first and the algorithm then processes another structure that is called *render tree*. This tree basically represents a hierarchy of visual boxes as they are really placed on the web page.

After the render tree is created, the analysis goes bottom-up. In the next step a new tree is created – a tree of visual areas. Each visual area corresponds to exactly one box in the render tree. Only those boxes that are visually distinct from adjacent boxes are considered. After the initial tree of visual areas is created, it is modified by merging similar visual areas (for example adjacent paragraphs of a text). Finally the significant areas are identified using some slightly modified techniques used by VIPS.

The result of Burget's algorithms is different from the result of VIPS in several ways. First of all, the tree produced by Burget's algorithm contains two node types: *visual areas* and content nodes called *boxes*. All visual areas contain information about the position and dimensions of the area but the system is completely different. Burget assigns a special topographical grid to each non-leaf visual area and puts all child areas on the grid. A position of each area is represented by the grid cell in which the top-left corner of the area is. Area dimensions are represented by the number of rows and columns the area takes. This grid representation provides the possibility to disregard real distances between boxes while keeping the information about their mutual position. Every non-leaf visual area in the tree can contain only other visual areas. Each leaf visual area contains just a list of contained boxes representing either *images* or *text*. Each of these types contains different attributes describing its appearance.

### *4.2 Box Clustering Segmentation*

The Box Clustering Segmentation is included in this paper for two reasons: 1) it is designed specifically to remove the hierarchy from the result of page segmentation and 2) it is much faster than both of the methods above, therefore by including it in the result set we demonstrate that the variety of algorithms that can be improved by Cluster-based Page Segmentation is very wide.

First it is important to outline how the algorithm works and then what its results look like. Unlike the previous two, our algorithm is based on building everything from basic elements on the web page – elementary boxes. To get these boxes, the first step is to render

the web page, similarly to Burget's original approach [6]. After we have the tree of boxes, it is filtered so only the smallest visible boxes remain. These are often leaf nodes in the render tree output.

On this set of boxes we create a graph structure $G = (B, E, sim)$ where $B$ is a set of boxes on the web page, $E$ is a set of edges and $sim$ is a similarity function. To get more performance from the segmentation algorithm we optimize the graph creation by creating edges only between boxes that share at least one pixel column or row on the rendered page.

This graph structure is then used for the next step – clustering. Each cluster is started and then grown by merging boxes with other boxes or clusters. The process it iterative – in every iteration we take the most similar entities in the graph and if nothing blocks it, we merge them. The merging step includes detection of possible overlaps of the new cluster with other boxes and their subsequent inclusion in the cluster. The clustering stops when a threshold value of similarity is reached. This threshold must be set upfront and its optimal value differs for every processed page.

The mutual similarity if given by our specially designed similarity two-fold model. For simple boxes, the similarity is calculated based on similarity of their shape, size and color and by their mutual position. For clusters we then remember the mean similarity between the boxes in each cluster and use a set of special formulas to calculate the resulting cluster similarity.

There are two entities playing role in the output of Box Clustering Segmentation: boxes and clusters. Clusters represent the identified visual areas after the algorithm run is complete. For every cluster we disregard the boxes it contain. However some boxes might remain unclustered after the algorithm finishes. Those are somewhat important but they can be safely dropped in the Cluster-based Page Segmentation process because they most likely don't contain any content that might be worth storing for future retrieval.

Taking that into account, only the clusters remain useful for the Cluster-based Page Segmentation. Because we disregard the boxes they contain and there is not tree structure involved in the result of the segmentation process, they are just simple boxes represented by their position and dimensions, both measured in pixels. This demonstrates the simplicity of the result compared to the methods described in section 4.1.

## 5 Cluster set data structures

We will describe the maximally optimized cluster set as introduced in section 3. That means each cluster consists only of the Cluster Representative. There are three parts we have to consider:

- templates represented by DOM trees
- trees or sets of visual areas
- mapping between the previous two

Before going deeper, let's formally define the cluster set in general:

**Definition 1:** *Let the cluster set be defined as a set of Cluster Representatives* $S = \{C_i | 0 < i < n\}$ *where* $n$ *is the number of elements in the set and each Cluster Representative is defined as an n-tuple* $C = (D, V, M_{VD})$. $V$ *represents a result of segmentation performed*

*on the Cluster Representative, $D$ represents its DOM tree and $M_{VD}$ represents mapping between $V$ and $D$.*

Each of these parts will be explained in detail in the following parts of the paper.

### *5.1 DOM tree*

In our work we use a pruned DOM tree to represent the web page. The following definitions therefore define the pruned version of the DOM tree, as used in each Cluster Representative.

**Definition 2:** *Let a DOM tree be defined as a three-tuple $D = (V_D, v_r, P_D)$ where $V_D$ is a set of vertices, $v_r$ is a root node and $P_D$ is a set of paths. Each vertex in the set $V_D$ represents a node of the DOM tree, i.e. DOM node. A structure of the tree is encapsulated within these DOM nodes.*

Some general features of the DOM tree which should be considered when storing it follow [18, 19]. There are four basic data types in the DOM tree: string, timestamp represented by an integer number, user data blob and object. The last one represents a reference to any other DOM node. Each DOM node can have, depending on its type, $0..N$ child nodes. Similarly, a DOM node can have $0..N$ attributes. These attributes can be represented either by one of basic data types as element properties (deprecated) or by child nodes of `Attr` type.

**Definition 3:** *Let a DOM node be defined as an n-tuple $v = (p_v, C_v, A_v)$, where $p_v \in V_D$ is a parent of the node, $C_v$ represents an ordered set of its child nodes and $A_v$ is a set of its attributes.*

As it was stated above, tags in HTML follow one another in a specific order and this order usually matters for rendering. That means the same order has to be preserved in the DOM tree. The following definition specifies the relation on top of elements of $C_v$ which makes the set ordered.

**Definition 4:** *Let $C_v$ be an ordered set of child nodes of node $v$: $C_v = \{u_1, u_2, \ldots, u_n\}; \forall i : u_i \in V_D$. The element order preservation in $C_v$ can be expressed as: let $u_i, u_j \in C_v; i \neq j$, then the following condition must be obliged $i < j \leftrightarrow u_i \, precedes \, u_j \, in \, the \, HTML \, code$.*

The unordered set of attributes $A_v$ can contain virtually any HTML attribute as well as style definition. All these attributes can be later used for both more accurate matching of DOM trees and more accurate mapping of DOM nodes. In this paper only one attribute is important and that is the `id` attribute. All other attributes can be dropped from the set $A_v$. The formal definition of set $A_v$ follows.

**Definition 5:** *Let the set of node attributes of node $v$ be defined as $A_v = \{(k, v)\}$, that is a set of key-value pairs, where $k$ designates a name of the attribute and $v$ its value.*

The algorithm for matching DOM trees works with path sets. It is highly inconvenient to retrieve paths by traversing the tree every time we need to match a Cluster Representative

to new page. Thus the best option is to create the set and store it as a part of the DOM tree itself. The path set is designated $P_D$ and is defined as follows.

**Definition 6:** *Let a path in the tree be defined as an n-tuple* $p = (v_r, v_1, \ldots, v_{n-2}, v_l)$ *with the following conditions met:*
$v_1 \in C_{v_r}; v_l \in C_{v_{n-2}}; \forall 0 < i < n-2 : v_{i+1} \in C_{v_i}, C_{v_n} = \emptyset$.
*Path set is an unordered set of paths* $P_D = \{p\}$.

Note that the path set is not a multiset. The DOM tree itself can contain multiple identical paths that lead to different nodes of the tree. However the intended application of the path sets doesn't require this property to be preserved so only one instance of each element is kept in the set.

Each piece of text in the web page is always considered to be a special node in the DOM tree. Text on a web page is split into these pieces by any occurrence of an HTML element. As it will be explained in section 6, DOM nodes representing text are excluded from paths in the path set, however we still need them for mapping DOM nodes between the Cluster Representative and inspected page. Therefore we keep them in the DOM tree but just as bare DOM nodes, the content itself won't be included. The same applies for images – only the DOM node will be included, not their content. In this context it's important to note that some properties of both the image and the text might be stored in our representation, as they might be important for some steps following the Cluster-base Page Segmentation. These will be then stored in the set of attributes $A_v$.

### *5.2 Visual areas*

Section 4 demonstrates that the representation of segmentation results varies significantly and each method has some specifics. However we need just a single and generic enough representation that will unify the results.

We need to start the definitions from the building blocks – visual areas:

**Definition 7:** *Let the visual node* $v \in V_V$ *be defined as* $v = (A_v, C_v, D_v)$*, where* $A_v = \{(k, v)\}$ *is an implementation specific set of area attributes, defined as key-value pairs.* $C_v$ *is a set of child nodes and* $D_v$ *is a set of corresponding DOM nodes.*

This representation of visual area allows both the tree structure of Burget's algorithm and VIPS and also for the flat structure of Box Clustering Segmentation. In the last case, the set of child nodes will be empty for all visual areas.

The set of attributes can contain attributes like position, visual features, size and others that might be required in further processing. What is important about the attribute $C_v$ is that unlike in case of DOM tree, this time the order of children doesn't matter. The same situation applies to $D_v$, therefore both these attributes are plain sets.

The attribute set $A_v$ is a solution of the requirement for the design to be generic for any type of tree of visual areas, as it can be simply ignored in generic implementation. Attributes $C_v$ and $D_v$ reflect common properties of outputs of all the segmentation algorithms described in section 4.

Similarly to the visual area, we need to define generic enough structure that will hold all the visual areas. We call this structure *tree of visual areas*, even though the tree might have just one level of nodes in some cases.

**Definition 8:** *Let the tree of visual areas be defined as a two-tuple $V = (V_V, v_r)$ where $V_V$ is a set of visual vertices and $v_r$ is a root node of the tree. Each vertex in the set $V_V$ represents a node in the tree of visual areas, i.e. a visual area. A structure of the tree is encapsulated within these visual areas.*

When using the tree to contain the results of Box Clustering Segmentation, root node $v_r$ will be set to $nil$ value and it won't be further used. The tree of visual areas is derived from the DOM tree, the main difference between the two is just in their nodes.

### *5.3 Tree mapping*

The description in sections 5.1 and 5.2 leads to a conclusion that the cluster set can be viewed as a forest of $D$ and $V$. If references from nodes of $V$ to nodes in $D$ are not omitted, it can be also viewed as one big tree rooted at node $v_r \in V$.

References from $V$ to $D$ are important and only their basic version has been described. Because these connections are utilized in some algorithms working on top of vision-based page segmentation [8, 9], the mapping between both trees should be well defined:

**Definition 9:** *Let the relationship $A \supset B, A \in V_V, B \in V_D$ be defined as $A$ visually contains entire $B$. Let the mapping $M_{VD}$ between $V$ and $D$ be defined as a set of two-tuples: $M_{VD} = \{(v, d) | v \in V_V, d \in V_D, v \supset d, \nexists v_1 (v_1 \supset d, v_1 \in C_v)\}$.*

In order to be consistent with previous sections, we amend the definition of a visual area by the following definition:

**Definition 10:** *Let $D_v$ be a set of DOM nodes corresponding to a visual area $v$. This set is defined as $D_v = \{d | (v, d) \in M_{VD}\}$.*

## 6 CPS Algorithms

Now that all structures related to cluster set and Cluster Representatives have been described, algorithms working on top of them can be defined. Algorithm 1 displays the Cluster-based Page Segmentation algorithm, utilizing structures defined in section 5. The Algorithm 1 is more formal expression of figure 2. All the following algorithms are written in pseudo-code based on Python syntax.

The algorithm is fully automatic, no human intervention is needed. Also no learning phase is necessary, it "learns" new templates while processing the web site. Based on the algorithm outline, it is possible to identify three distinct non-trivial parts. Their description follows.

### *Creating a Cluster Representative*

This task consists of series of small transformations of DOM tree of the original page and the corresponding set or tree of visual areas with the result being the initial form of the Cluster Representative. This transformation leads to the Cluster Representative's n-tuple $C = (V, D, M_{VD})$.

Before the structure $V$ is built, the simplified representation of DOM tree $D$ has to be created by cleaning the original DOM tree of redundant nodes, according to definition

**Algorithm 1** Segmentation using the CPS algorithm

```
def segment_cps(page, cluster_set):
  dom = parse_page(page)
  representatives = cluster_set.get_all()
  for representative in representatives:
    if dom.matches(representative):
      return dom.visual_tree()
  visual = segment_page(dom)
  cluster_set.store(dom, visual)
  return visual
```

in section 5.1. This step is straightforward – a simple recursive, post-order tree traversing algorithm can be used. Creating the graph structure $D$ is trivial, it can be done as a part of the traversing algorithm.

Since $V$ is very similar to the output returned by segmenting algorithm, the only thing remaining to build $V$ is to ensure that each node has a set of DOM nodes it visually contains. This is done during the creation of $M_{VD}$. For purpose of this paper, the assumption is that this is handled by the segmenting algorithm in use, since the algorithm is the only component that has this information. With this assumption in consideration, we have $D_v$ for every node $v$ at the moment $V$ is created and we just have to extract all the information into $M_{VD}$ and then verify that $M_{VD}$ is correct by checking its conformity with definitions 9 and 10. After this step, we have a valid Cluster Representative which can be added into the cluster set.

### *Matching DOM tree to the cluster set*

After everything is stored, the trivial approach would be to segment another page. But with the Cluster-based Page Segmentation we can utilize having the cluster set and try to find a cluster which the new page belongs to. A comparison with all loaded Cluster Representatives for the site (more specifically with their $D$ elements) has to be performed first. As it was outlined in section 3, a simple iteration over the set $D$ containing all clusters and matching one by one can be performed. Because there is only a small number of clusters for each site, performing a simple iteration is sufficient to gain considerable performance boost against the plain page segmentation. The only condition that has to be met is for the DOM-to-DOM matching algorithm to be fast.

For this DOM-to-DOM matching we use modification of Common Paths Distance measuring algorithm [13] as it has been proven significantly faster than tree-edit-distance algorithms while still being precise enough to match the DOM tree to the correct template. However in our practical evaluation we needed to adjust the original algorithm for better precision. Our modifications are as follows:

- Filter out all nodes that are not representing particular HTML elements (e.g. attribute nodes, text nodes, etc.)
- If any element has an id, don't use the plain element name in the path, use it in combination with the id

These simple modifications improved results of the matching algorithm significantly and enabled higher level of result granularity. That means more clusters are created, thus less false-positives for cluster matches are encountered.

When a web page is matched to a $D$ belonging to some Cluster Representative, it is possible to use the corresponding structure $V$ containing visual areas that associated with the DOM tree. If no matching Cluster Representative is found, we consider the page to be based on a template that the algorithm hasn't encountered yet. In such case the segmentation process has to be performed on it and the result has to be used to create a new Cluster Representative.

### *Mapping nodes of both DOM trees*

Mapping nodes of the processed page to those stored in the matched Cluster Representative is the last step for the Cluster-based Page Segmentation to return useful results. We need this step so the actual content of visual areas can be retrieved, as that is the next process that is likely to be performed on the result of the Cluster-based Page Segmentation. The mapping procedure is trivial for Cluster Representatives themselves, as the mapping is already a part of its stored structure.

However in case of any other page in the cluster the process is more difficult. Again, we have a set of visual blocks of the Cluster Representative in which we are interested and the corresponding set of Cluster Representative's DOM nodes. Now we need to find the corresponding DOM node of the input page for every DOM node of the Cluster Representative. If we designate the DOM tree on input $D_I$, we are looking for a *Tree mapping between $D_I$ and $D$*. The tree-mapping problem for two DOM trees is quite complex in general, as many examples demonstrate [11, 17]. However, as we explain in our previous work [20], our scenario is very specific so we can afford some simplifications.

We are not looking for a mapping of each node, we are looking for a subtree rooted at corresponding node. Therefore the assumption is that once we find a root, all descendant nodes will correspond in both trees. For finding the root we use a *distinguished path* to a node. In its simplest version, this path is defined below. We acknowledge that this representation is rather crude but it serves our purpose well. Possible improvements are covered in our previous work [20].

**Definition 11:** *Let the distinguished path $p_N$ from root of given DOM tree to a node in that DOM tree $N$ be defined as n-tuple of two-tuples:*
$p_N = ((p_1, c_1), (p_2, c_2), \ldots, (p_k, c_k))$ *where $p_i$ is a position of a node withing its siblings and $c_i$ is a total count of siblings including that node. $i$ is an index designating a level of DOM tree from its root, i.e. how many nodes deep in the structure is the selected node and its siblings.*

This indexing approach is based on the premise stated in section 5.1 that order of DOM nodes within DOM tree has to be preserved to preserve the content layout on the web page. Therefore if a certain node in Cluster Representative was on position $3/5$ within its siblings, its corresponding DOM nodes from other pages in that cluster will again be positioned as $3/5$. This condition will be always true when traversing the part of DOM tree that corresponds to template. Once out of the template scope, thus in a particular data region, the condition might not be true but the assumption is that interesting visual areas don't have root outside of the template scope. To add at least some level of error detection

in case DOM nodes of Cluster Representative and inspected page don't correspond, there is the $c_i$ parameter which is used as a simple failsafe mechanism – if $c_i$ differs for Cluster Representative and inspected web page, two things could have caused this. Either the page has been incorrectly matched to the Cluster Representative or the node identified on level $i$ is already outside the template and within one of its data regions. In any case, the algorithm stops at that moment, as the result would be wrong anyway.

With the indexing approach described above, the algorithm using it will need to store these distinguished paths somewhere or they have to be constructed on the fly. If they are stored as part of DOM nodes or visual areas of Cluster Representatives, the algorithm for finding the DOM node within inspected page can be used directly.

**Algorithm 2** Finding a node within given DOM tree

```
def find_dom_node(distinguished_path, root_node):
  node = root_node
  for (position, count) in distinguished_path:
    if node == None or count != len(node.C):
      return None
    node = node.C[position]
  return node
```

If distinguished paths are not stored as a part of Cluster Representatives, we need to use algorithm 3 for their construction first. After construction of each path, algorithm 2 can be be used again for finding corresponding DOM node within inspected web page using the path.

**Algorithm 3** Construction of the path from root to the given node

```
def get_path(node):
  if node.p == None:
    path = ()      # empty tuple
  else:
    path = get_path(node.p)
    sibling_count = len(node.p.C)
    node_pos = 0
    for n in node.p.C:
      if n == node:
        break
      node_pos += 1
    path.prepend((node_pos, sibling_count))
  return path
```

## 7 Experimental evaluation

An experimental implementation has been designed and realized as a proof of concept. We used our Java implementation of all three algorithms in question. The Box Clustering segmentation and Burget's segmentaion algorithm are both original. Our VIPS implementation gives comparable results as the original with slightly worse performance. We use this implementation so the all three algorithms are based on the same rendering core and are thus comparable.

### *Implemented scenario*

The following scenario is considered: we have a simple crawler program, which is given a web site to process in a form of starting URL. It is processing the site page by page and extracting an interesting content from that page. If the page contains multiple areas with the interesting content, the algorithm extracts all of them.

For each algorithm we selected a different content type to be designated as interesting. The selection was based on capabilities of the algorithm – because the content classification is not our primary concern, we selected what was easiest for the algorithm to detect. For Box Clustering segmentation it was a body of an article - basically all the areas significantly bigger than the mean size of areas on the page. Within results of VIPS and Burget's algorithm we detect headlines – areas with font size significantly bigger than then average size on the page. When common web page design is considered, there is usually one heading for one article body, thus the number of selected areas should be similar for all the algorithms.

The crawling algorithm takes a very simple breadth-first-search approach. We need a global list of all links which are planned to be inspected. The crawler always takes the first URL in this list that has not yet been visited and loads the page on this URL for further processing. The second list contains visited links so we can quickly filter out already visited pages and not visit them again. Not performing this would cause deviations in our measurement. The last list contains links that lead out of the site(usually recognized by the same second-level domain) that is currently processed.

If a link leads to different site than the one that is currently inspected, instead of processing it is just stored for later usage. The crawling mechanism doesn't start parsing new site before the entire site it is currently parsing is processed. At that point, we stop our inspection but the program has capability to continue to the other site by clearing the *Extracted list*, adding the first link from *Outgoing list* into it and continue crawling on the new site.

The reason for the program to process one site at a time is simple: there is only a very limited number of templates for each site. Our measurements show us that the number of templates on a single site is quite low (see table 1 for details). Since the number of Cluster Representatives is equal to the number of templates used on the web site, having this number small makes it simple to store all the Cluster Representatives in memory while inspecting the web site they belong to. Such approach is highly convenient because we want fast access to these Cluster Representatives when processing the site. However storing them only in memory would mean that all Cluster Representatives of the site are dropped once inspection of that site finishes. That's not something we want because it would mean longer processing of the site the next time. Rather than that we store all Cluster Representatives in persistent storage. Once a site is selected for processing, all its Cluster Representatives are loaded from database and after the processing finished, the database is updated if necessary.

| site | clusters | hit ratio |
|---|---|---|
| iDnes.cz | 42 | 91.6% |
| e15.cz | 27 | 94.6% |
| telegraph.co.uk | 32 | 93.6% |
| slashdot.org | 18 | 96.4% |
| businessinsider.com | 16 | 96.8% |
| gizmodo.com | 11 | 97.8% |

**Table 1** Cluster counts for different sites

### *Implementation details*

We used in-memory database in form of a simple list of Cluster Representatives. This list is retrieved from persistent storage managed by OrientDB object-oriented database engine before processing any pages on the site. When matching DOM trees, we made one modification in the implementation for the sake of simplicity – section 6 suggests that the algorithm is working on the tree structure $D$, however the implementation works with string representation of all paths. A set of paths of a DOM tree is constructed when the DOM tree is prepared for matching and in case the DOM tree is used for a Cluster Representative, this set of paths is used along with it. The algorithm just iterates over all Cluster Representatives from the web site and compares the path set of the input DOM tree with their respective path sets one-by-one. After the DOM tree is matched, its segmented counterpart is fetched and desired content is extracted.

### *Testing and measurements*

The implementation has been tested on several Czech and world-wide web sites. Three of them are quite extensive news sites (iDnes.cz, novinky.cz, telegraph.co.uk) and two of them are highly focused CMS-based portals. Each testing set contained 500 pages recursively fetched from index page of that site. Test were performed on following hardware: Intel Core2Duo P8700 2.53GHz, 4GB RAM, HDD 5400RPM.

Table 1 illustrates how many clusters were detected per site. The column *clusters* gives the actual number of clusters on the site, while *hit ratio* illustrates how many percent of pages were matched when 500 pages were inspected. Note that matched page doesn't have to be segmented therefore the higher the hit ratio is the more time saving is achieved. One important observation has been made during the testing and that is that during the first inspection of a site, the majority of clusters is usually detected early in the process. If the number of total processed web pages has been doubled, the number of detected templates has risen for at most 33%. This demonstrates logarithmic growth of cluster set size, leading to confirmation that the number of clusters is low compared to number of the web pages on the site. Figure 3 demonstrates graphically that the number of templates on a single site converges fast to a relatively small number.

Tables 2, 3 and 4 demonstrate time difference between standard segmentation methods and Cluster-based Page Segmentation for each web site. The *plain* column tells the time necessary to segment inspected 500 pages within the site. The time includes only segmentation and retrieval of all desired data. The *CPS* column contains the time necessary to retrieve the same data with Cluster-based Page Segmentation. The time includes segmentation of pages when creating a new cluster, comparison of incoming pages against

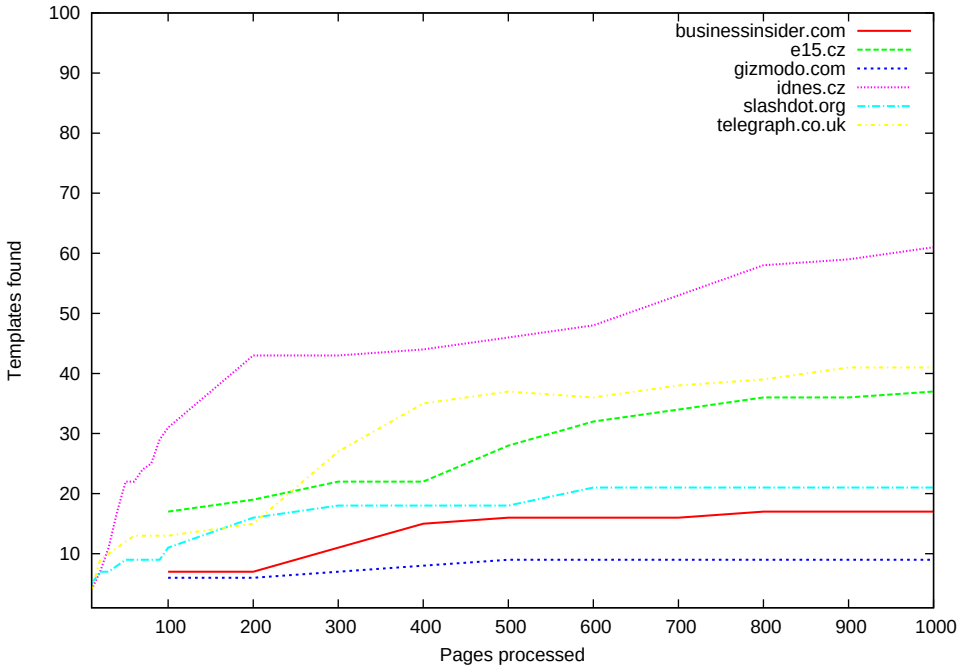


**Figure 3** Dependency of cluster count on page count

existing Cluster Representatives and retrieval of desired content from the page through node mapping. The time represents the worst case scenario for CPS, i.e. the first processing of the site. This scenario is the worst case because if the site is already processed and the corresponding Cluster Representatives are retrieved from database, the time to do this is orders of magnitude better than the time necessary to create Cluster Representatives by segmentation, even if the cluster set to create is very small. The *time saved* intuitively demonstrates how many percent of the time necessary for each segmentation algorithm to process the page set is saved when using Cluster-based Page segmentation. Both plain and CPS times are measured as a sum of times necessary to retrieve the data from all 500 processed pages.

| **site** | **plain** | **CPS** | **time saved** |
|---|---|---|---|
| iDnes.cz | 1 158 s | 145 s | 87.5% |
| e15.cz | 661 s | 55 s | 91.7% |
| telegraph.co.uk | 2 719 s | 841 s | 69.1% |
| slashdot.org | 1 925 s | 89 s | 95.4% |
| businessinsider.com | 766 s | 38 s | 95.0% |
| gizmodo.com | 560 s | 39 s | 93.0% |

**Table 2** Performance measurements of the VIPS

| site | plain | CPS | time saved |
|---|---|---|---|
| iDnes.cz | 4 019 s | 282 s | 93.0% |
| e15.cz | 420 s | 34 s | 91.9% |
| telegraph.co.uk | 1 569 s | 97 s | 93.8% |
| slashdot.org | 946 s | 30 s | 96.8% |
| businessinsider.com | 587 s | 29 s | 95.1% |
| gizmodo.com | 832 s | 38 s | 95.4% |

**Table 3** Performance measurements of the Burget's algorithm

| site | plain | CPS | time saved |
|---|---|---|---|
| iDnes.cz | 1 423 s | 195 s | 86.3% |
| e15.cz | 411 s | 42 s | 89.7% |
| telegraph.co.uk | 1 521 s | 603 s | 60.3% |
| slashdot.org | 597 s | 50 s | 91.6% |
| businessinsider.com | 530 s | 34 s | 93.6% |
| gizmodo.com | 771 s | 49 s | 93.6% |

**Table 4** Performance measurements of the BCS

Our results clearly prove that Cluster-based Page Segmentation offers high performance boost. This is confirmed by another result not visible in table 3. The time necessary for retrieving data from page belonging to existing cluster is lower by one to three orders of magnitude compared to the time necessary for retrieving the data from page not belonging to any cluster. The accuracy of the Cluster-based Page Segmentation is the same as accuracy of the used algorithm because that is the element performing the segmentation itself, no further modifications of returned results are performed. Moreover it is not purpose of this paper to evaluate accuracy, as it is depending solely on the used segmentation algorithm and selection of the algorithm is virtually not limited.

## 8 Conclusion

In this paper we presented a new way how to deal with performance shortcomings of vision-base page segmentation algorithms. Templates, one of fundamental concepts of modern web, have been used for significant performance boost of these algorithms. By combining precision of vision-based segmentation algorithms with performance superiority of template detection algorithms, it is possible to create an algorithm both precise and fast while keeping its universality.

We showed that Cluster-based Page Segmentation significantly improves performance of vision-based page segmentation when used on large sites and it therefore compensates for the greatest disadvantage of plain vision-based page segmentation algorithms.

We have also shown an alternative segmentation method called Box Clustering Segmentation and its possible usage in the Cluster-based Page Segmentation. When combined to perform the segmentation on entire web sites, these two algorithms vastly outperform any known segmentation method.

This research laid down solid base for future research. That might include improved adaptation of the Cluster-based Page Segmentation to potential post-processing algorithms and further integration with the Box Clustering Segmentation.

## Acknowledgement

This paper is a revised and expanded version of a paper entitled Cluster-based Page Segmentation - a fast and precise method for web page pre-processing presented at WIMS '13, June 12-14 2013, Madrid, Spain. This work was supported by the BUT FIT grant FIT-S-11-2 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

## References

[1] Shipeng Yu, Deng Cai, Ji-Rong Wen, and Wei-Ying Ma. Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 11–18, New York, NY, USA, 2003. ACM.

[2] Eduardo Sany Laber, Críston Pereira de Souza, Iam Vita Jabour, Evelin Carvalho Freire de Amorim, Eduardo Teixeira Cardoso, Raúl Pierre Rentería, Lúcio Cunha Tinoco, and Caio Dias Valentim. A fast and simple method for extracting relevant content from news webpages. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 1685–1688, New York, NY, USA, 2009. ACM.

[3] Jer Lang Hong, Eu-Gene Siew, and Simon Egerton. Information extraction for search engines using fast heuristic techniques. *Data Knowl. Eng.*, 69(2):169–196, February 2010.

[4] Bing Liu, Robert Grossman, and Yanhong Zhai. Mining data records in web pages. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 601–606, New York, NY, USA, 2003. ACM.

[5] Miroslav Spousta, Michal Marek, and Pavel Pecina. Victor: the Web-Page Cleaning Tool. In *Proceedings of the 4th Web as Corpus Workshop, LREC*, 2008.

[6] R. Burget. Layout based information extraction from HTML documents. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, pages 624–628, Washington, DC, USA, 2007. IEEE Computer Society.

[7] Deng Cai, Shipeng Yu, Ji rong Wen, and Wei ying Ma. VIPS: a vision-based page segmentation algorithm. Microsoft technical report MSR-TR-2003-79, November 2003.

[8] Petasis G., P. Fragkou, A. Theodorakos, V. Karkaletsis, and C. D. Spyropoulos. Segmenting HTML pages using visual and semantic information. In *Proceedings of the 4th Web as a Corpus Workshop, 6th Language Resources and Evaluation Conference.*, LREC 2008, pages 18–25, June 2008.

[9] Wei Liu, Xiaofeng Meng, and Weiyi Meng. ViDE: A vision-based approach for deep web data extraction. *IEEE Trans. on Knowl. and Data Eng.*, 22(3):447–460, March 2010.

[10] David Gibson, Kunal Punera, and Andrew Tomkins. The volume and evolution of web page templates. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 830–839, New York, NY, USA, 2005. ACM.

[11] D. C. Reis, P. B. Golgher, A. S. Silva, and A. 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.

[12] Lan Yi, Bing Liu, and Xiaoli Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 296–305, New York, NY, USA, 2003. ACM.

[13] Thomas Gottron. Bridging the gap: from multi document template detection to single document content extraction. In *Proceedings of the IASTED International Conference on Internet and Multimedia Systems and Applications*, EuroIMSA '08, pages 66–71, Anaheim, CA, USA, 2008. ACTA Press.

[14] Karane Vieira, André Luiz Costa Carvalho, Klessius Berlt, Edleno S. Moura, Altigran S. Silva, and Juliana Freire. On finding templates on web collections. *World Wide Web*, 12(2):171–211, June 2009.

[15] 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.

[16] 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.

[17] Gabriel Valiente. An efficient bottom-up distance between trees. In *Proceedings of the 8th International Symposium of String Processing and Information Retrieval*, pages 212–219. Press, 2001.

[18] Arnaud Le Hors, Philippe Le Hegaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document object model (DOM) level 3 document object model core. W3C Recommendation, April 2004.

[19] Johnny Stenback, Philippe Le Hegaret, and Arnaud Le Hors. Document object model (DOM) level 2 document object model html. W3C Recommendation, January 2003.

[20] Jan Zeleny and Radek Burget. Isomorphic mapping of dom trees for cluster-based page segmentation. In *Proceedings of the Twelfth International Conference on Informatics INFORMATICS'2013*, INFORMATICS '13, 2013.