Cluster-based Page Segmentation – a fast and precise method for web page pre-processing

Jan Zeleny
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
izeleny@fit.vutbr.cz

Radek Burget
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
burgetr@fit.vutbr.cz

ABSTRACT
Segmenting a web page may be one of initial steps of information retrieval or content classification performed on that page. While there has been an extensive research in this area, the approaches usually focus either on performance or quality of the results. Vision based segmentation is one of the quality focused methods, which are considerably slow. This paper proposes an approach for boosting the performance of vision based algorithms. Our approach is based on concepts of modern web and a very common scenario in which an entire web site is processed at once. In this scenario, a great amount of performance boost can be gained by isomorphic mapping of previous results gathered from pages within the site to other pages on the same site. We provide the results of experiments performed on VIPS, the most common algorithm for page segmentation.

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

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 for web-focused information retrieval grows with it. When performing the information retrieval, a big problem arises and that is the fact that one web page can contain multiple areas with very different information content[19]. Hence the page has to be split into parts and these parts examined separately for the results to be more precise. Web page segmentation and related methods have been developed for a number of years to address this task. Also, other tasks can be addressed by page segmentation, for example from area of content classification.

Page segmentation is a process of splitting up a web page into hierarchically organized smaller blocks which are 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[9, 7, 10, 13]. 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[1][2, 11, 3, 1]. One of the most significant families of algorithms using the latter approach is vision-based page segmentation. This family of algorithms will be described more closely is this paper.

Template detection is an approach which is close to page segmentation but the main aspect is different. The result of page segmentation is a set or tree of page areas, each containing single type of content (text, advertisements, navigation, . . .) whereas the result of template detection is a web page split into two regions – template and content. In this perspective, template detection can be observed as a very specific and very limited page segmentation. Although the result might be perceived as similar to page segmentation, the way how to get to the result is completely different. Template detection algorithms work on a principle of comparing more web pages and identifying repetitive patterns in them. This approach is much faster compared to page segmentation at the expense of very rough granularity which is acceptable for template detection.

In this paper we propose a method combining vision-based page segmentation with clustering algorithms, utilizing template detection algorithms as measurement methods. This approach is faster than the vision-based segmentation while keeping high accuracy. The method, called Cluster-based Page Segmentation, builds on existing algorithms but it is not strictly bound to any particular one of them. It is not even strictly bound to vision-based segmentation although it has the greatest potential when used with it. From this perspective, the Cluster-base Page Segmentation is a complementary algorithm. It is based on templates, a principle on which modern web pages greatly depend.

When we consider the principle of templates, it is possible to group pages from the same site in several clusters.

[1]Literature usually defines accuracy as a level of coherence between results of segmentation and human perception.

Then we can create cluster-bound structures containing information common to all pages in their respective clusters. These structures are called Cluster Representatives. Each of these representatives contains information about DOM tree of its cluster, meaning it can be segmented. From that point the segmentation of all other pages is unnecessary because each Cluster Representative corresponds to every page in its cluster. In this paper we take the most known segmentation algorithm, called VIPS[2], enhance it with our method and demonstrate that Cluster-based Page Segmentation improves performance of the segmentation process significantly, when applied to a set of pages at once. This paper summarizes what structures and what features must each Cluster Representative have so it can be used for further site processing as easily as possible. Besides that this paper offers algorithms for utilizing these structures.

The rest of this paper is organized as follows: section 2 explains state of the art in web page preprocessing, including reason why is it performed. Page preprocessing includes page segmentation and template detection. Section 3 gives an overview of Cluster-based Page Segmentation. Sections 4 and 5 closely describe data structures and algorithms used in Cluster-based Page Segmentation. Finally, section 6 shows result of our experimental evaluation. It also describes some detail of the implementation that are deviated from previous text. Finally, section 7 summarizes results of this paper and concludes the potential of this work.

2. WEB PAGE PREPROCESSING

Because web pages are semi-structured (or weakly structured) documents containing text mixed with HTML tags, they need some preprocessing before one can use data mining techniques on their content. The simplest option is to strip all HTML tags and related data from the document thus leaving just bare text. The problem is that results of this approach are not entirely accurate, because a lot of data on a modern web pages is just a noise [4] which can be misleading when it comes to information retrieval or content classification. This noise can consist of elements like advertisements, navigation, links to similar pages, user comments and various other data. Thus, to gather relevant data from the page, we need to do the above mentioned preprocessing. The goal of this preprocessing can vary from cleaning up those noisy parts [18, 12] to identifying semantically distinguished blocks that the page contains[3]. The goal is usually depending on circumstances. The former of the two examples can be used for information retrieval where the author just wants to have a clear-text representation of data on a web page as a source for the information retrieval. The latter one is more convenient for content classification if we want to replace a web page by an information block as an atomic unit of web content.

2.1 Page segmentation

All these preprocessing methods often have a common element and that is page segmentation. Its task is to divide given page to smaller blocks which are consistent either logically or visually, based on input parameters and used algorithm.

Basic segmentation methods can be split into two groups: DOM-based (text-based) and vision-based. Methods in the former group are based on analyzing web page without any need for rendering it. That means selected approach is either based on inspecting HTML code directly or (more often) traversing DOM tree corresponding to the HTML code and evaluating information gathered from it. Quality and speed of these methods is usually completely dependent on used heuristics. The array of heuristics can vary from pure text evaluation[9] to complex algorithms taking a wide variety of properties into account[13]. However these methods always fail to take one very important aspect into account and that is layout of the page. As Burget discussed in [1], the DOM based model isn't accurately describing real relation of individual blocks in terms of their visual appearance. If the complexity of CSS is taken into account, any node of DOM tree can be situated at a completely different part of a page when compared with the position in the DOM tree. It can be even invisible, thus virtually nonexistent.

DOM-based methods in the literature are in general much faster than vision-based methods and caching their results most likely wouldn't get much performance gain. As already mentioned, the reason for their speed is that they don't compute all the information contained in CSS about true layout of the inspected page. Therefore in further text we will be interested only in vision-based segmentation methods. This family of methods is based on an approach with a simple concept but quite large computing demands. The concept is to identify blocks on a web page as any user would perceive them if he was looking at the rendered page in his browser. This implies an advantage of these methods over DOM-based of not being strictly limited to web page processing but also being applicable (with minor changes) to PDF and other document formats. Vision-based segmentation algorithms have to simulate user view of given web page, which means a page has to be rendered either to an actual picture or at least to a corresponding internal representation of the visual information contained on that page. This process of rendering is very complex due to complexity of both HTML and CSS specifications. That means demands both for computational power and time to process one page are quite high, which is problematic. After being rendered, the page has to be segmented in several iterations which is also very demanding. The most commonly used algorithm in the area of vision base segmentation is VIPS[2] and algorithms using it as a black box and improving its results[11, 3]. Another approach, partially derived from the original VIPS specification, has been offered by Burget[1]. In this paper we will demonstrate results of Cluster-based Page Segmentation working on top on VIPS algorithm, as it is currently considered to be industry standard.

2.2 Templates and template detection

As it is outlined in section 1, the principle of templates defines that there is a number of templates for a site, each one defining core structure of a 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 underlying data source and a web page is created by filling these fetched data into the frame. An example of template is displayed on figure 1. The data areas are untouched, whereas template areas are darkened.

While being a great help to web designers and content au-

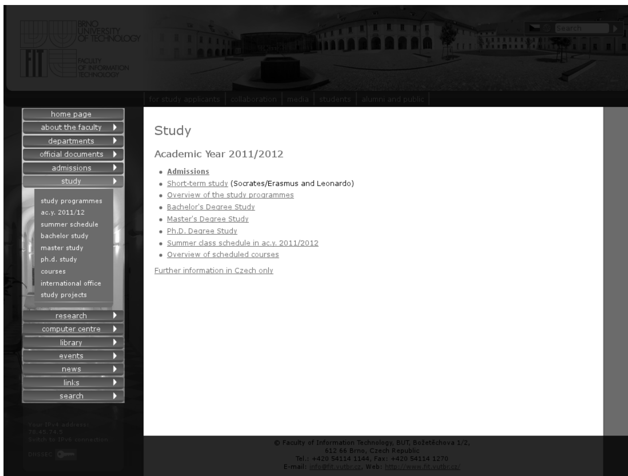

Figure 1: An example of a template

thors, templates pose a problem for information retrieval algorithms. As section 2.1 already covered, a great extent of every page consists of information unrelated to the core topic of the page, navigation, advertisements and other noise. In context of templates and template detection, all this unrelated content is considered to be a part of template, as figure 1 illustrates.

Template detection (further designated also as TD) can be perceived as a special type of page segmentation. Template detection methods[17, 5, 16, 18] are used to identify and subsequently filter out noise which is part of template of given page, leaving only the useful content from that page. That is however the one disadvantage these methods have. They can be only used to detect and filter out the template, no finer granularity is usually available.

TD algorithms are important for this paper because their modifications can be used as a distance measurement methods when we want to perform clustering on a set of web pages to split them into groups based on pages' templates. Examples of such algorithms can be found in the family of tree edit distance algorithms[17, 15]. The important thing about these algorithms is that they are usually designed to be as fast as possible. That will play a key role in one of the steps of Cluster-based Page Segmentation.

3. CLUSTER-BASED SEGMENTATION

The classic segmentation procedure is to process one page at a time. After one page is segmented and we start with another page, the process is started from the very beginning again. The obvious shortcoming of this approach is scaling. For example modern web search engines don't want to process just one page, but potentially thousands of web pages. This means the entire process of rendering a web page has to be gone through thousands of times. Even with great optimizations in the rendering engine, that will take unacceptably long time.

This paper proposes a new way how to deal with performance problems when processing a set of web pages. The bigger the set of web pages is, the greater optimization this approach achieves. Our method can be summarized in following proposition.

Proposition 1. When processing more pages within the same site, it is possible to increase performance of segmentation algorithm by performing the actual segmentation only for a limited number of pages and make these pages representatives of their respective template-based clusters. Instead of segmenting each page on the site, an isomorphic mapping between the page and corresponding Cluster Representative can be used to retrieve the same information from any web page in the cluster.

In practical application, it won't be convenient to store all pages from a site in a buffer, then cluster them and retrieve the information afterwards. In fact the goal will be more likely the opposite – to optimize the memory and disk consumption by having as small page set in each cluster as possible and process[2] one page at a time from that point. Taking this optimization to the maximum, each cluster of pages will be replaced by a single structure representing all the information we need for assigning a page to the right cluster and extract the desired information from it. The structure of these Cluster Representatives is described in section 4. The set of algorithms working on these Cluster Representatives is described in section 5. The high-level overview of Cluster-based Page Segmentation is summarized by figure 2.

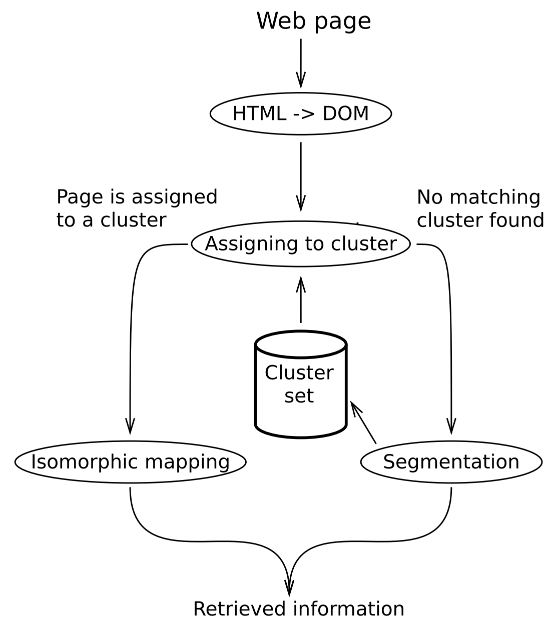

Figure 2: Block schema of the entire approach

[2]Note that processing in this context means going through the entire process displayed on figure 2

From the block schema we focus on a very specific subset of problems. Transformation from HTML code to DOM tree is done by DOM parsers. Our design starts at *assigning to cluster* where we try to identify which cluster is the right one for the page. Any algorithm measuring similarity of two web pages can be used. We use a variation of algorithm designed by Gottron in [5] and design data structures for this algorithm to be effective. 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. Together with structures directly derived from the web page we also need its segmented representation which is created in the *segmentation* process. This process is a black box for us around which we build our system. The important point of our work is that this black box basically represents any algorithm which can be used for segmentation and we can very simply adapt all other components to it. Therefore the only thing that is necessary for us are well described inputs and outputs which we can work with. For the *isomorphic mapping* of DOM nodes we offer a simple algorithm which finds corresponding nodes of one DOM tree in another DOM tree. The main focus of this work lies in the *Cluster set* and all around it. We propose data structures which together put the cluster set and are used in all other parts of the system.

There are some conditions that need to be kept for Cluster-based Page Segmentation to work. These conditions are most obvious when implementing the system. First of all it is important to state that even big web sites consist of a very small number of templates compared to the number of pages they contain. We confirm this claim in section 6. A small number of templates means that memory consumption is not a problem. The next constraint of our approach is that it is only good for processing one site at a time. That means if there are links on that site leading outside, these have to be stored for later processing and only after processing on one site is done, one of those in queue can start. Again, more details about this will follow in section 6.

4. CLUSTER SET DATA STRUCTURES

We will be describing the maximally optimized cluster set as introduced in section 3. That means each cluster consists only of the one representative and structures related to it. There are three parts we have to consider:

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

Before going more deep, let's formally define the cluster set in general, based on previous brief overview.

Definition 1. Let the cluster set be defined as a set of Cluster Representatives $S = \{C\}$ where each Cluster Representative is defined as an n-tuple $C = (V, D, M_{VD})$. V represents a result of segmentation performed on the Cluster Representative, D represents its pruned DOM tree and M_{VD} represents mapping between V and D.

Each of these parts will be explained in detail in following parts of the paper. Section 4.1 focuses on the DOM tree as a whole and on individual nodes of the tree (DOM nodes). Since only pruned trees are part of Cluster Representatives, some specifics of both the original tree in conformance with the W3C specification and the pruned tree are pointed out. In section 4.2 two different vision-based segmentation algorithms and most importantly their output formats are described and then a generic tree design is proposed. Finally, section 4.3 describes the mapping between pruned DOM tree and tree of visual areas.

4.1 DOM tree

The first part of each Cluster Representative is a representation of its DOM tree, designated D. The Document Object Model (DOM) is both language and architecture independent model used to represent SGML-based documents. DOM model is basically a definition of API which consists of many partial interfaces, some of them are even overlapping. These interfaces can be used to describe not only content, but also structure and visual style of the a page. The important thing is that representations of DOM tree can be different, depending on interfaces which are supported for each type of element by used implementation of the DOM tree. Therefore it's necessary to make our design as simple as possible while being sufficiently generic.

This paper doesn't focus on parsing HTML documents into a DOM tree. However for further definitions, it is important to note following properties related to HTML-DOM transformation[6]. Other than that we just assume a valid DOM tree on the input.

- node of a DOM tree can represent various items on the page or even only their parts.
- The order of elements in HTML code matters.
- No element overlaps are allowed. Next sibling element can be opened only after the previous one was closed.
- No parent-child overlapping is allowed. Parent element must be closed only after the last child element is closed.

Definition 2. Let the 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[8, 14]. There are four basic data types in the DOM tree: string, timestamp represented by 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 the 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 does usually matter for rendered output. That means the same order has to be preserved in the DOM tree. 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 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 attribute `id`. All other can be dropped from the set A_v. The formal definition of set A_v is following.

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 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 in an unordered set of paths $P_D = \{p\}$.

Note that the path set is not multiset. Even though some paths in DOM tree can be identical even though they lead to different elements. However this is not the case in our representation of path set.

Each piece of text on 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 HTML element. As it will be explained in section 5, DOM nodes representing text are exluded from paths in path set, however we still need them for mapping of DOM nodes between Cluster Representative and inspected page. Therefore we are keeping 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.

4.2 Tree of visual areas

For the purpose of this paper let's consider the output of every visual segmentation algorithm to be the Tree of Visual Areas. Different algorithms have different output formats, but they all have similar characteristics, for example the tree structure.

After being processed by **VIPS**[2], the web page is represented by a set of blocks O, a set of separators Φ and a relation between blocks δ(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:[2] $\Omega = (O, \Phi, \delta)$ where $O = \Omega_1, \Omega_2, \ldots \Omega_n$ and every Ω_i is defined in the same way as Ω. This definition implies the tree structure of the whole construct. It also means that the web page is considered and treated the same as any other visual block. Leaf nodes of the resulting tree are called *basic objects*. Besides being a leaf node in the tree of visual areas, it is also a leaf node in the DOM tree. Therefore each visual block can contain one or more nodes of the DOM tree. Note that the Tree of Visual Areas and the DOM tree don't have to correspond, i.e. a non-leaf visual block doesn't have to correspond to a particular node in the DOM tree.

For each block an information about its position and size is absolutely essential. These properties can be expressed as absolute numbers or relative to the parent block[11]. Also the alignment with its parent (for example float left) is used[11]. If considering how VIPS works, an information about the *Degree of Coherence* as defined in [2] could be stored for each block as well. For separators it is important to store their visual impact, which can be in form of width or visibility defined e.g. by borders of adjacent blocks. Relations between blocks have one feature and that is the degree of visual similarity of blocks in relation. This information is not a part of VIPS output, but it is added in some other algorithms using it[11].

Burget in his work [1] focuses on similar problems as VIPS but structures he uses are slightly different. The tree produced by his 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. To define both of these, a special topographical grid is constructed for each non-leaf visual area. An example of this grid is displayed on figure 3

All child areas are then placed on the grid. A position of each area is represented by the cell of grid in which is the top-left corner of the area. Dimensions are represented by number of rows and columns the area takes. Every non-leaf visual area in the tree can contain only other visual areas. Each leaf visual area contains one or more boxes, but no grid is necessary for them since no further segmentation is performed on top of them.

Leaf visual areas contain one or more boxes which, concatenated to a string, create a single continuous area of the document. There are two types of content elements: *images*

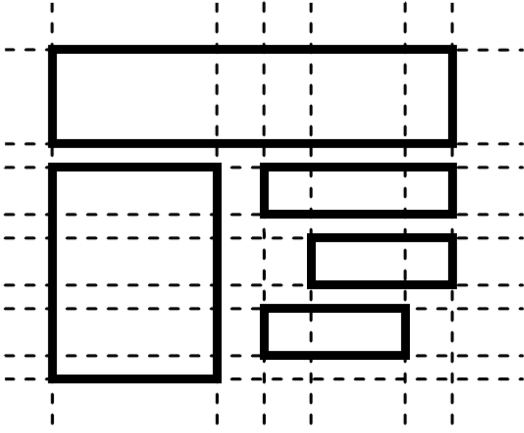

Figure 3: An example of the topographical grid

and *text*. Each of them contains different attributes describing appearance. Taking description of text nodes in section 4.1 into account, it is possible to consider text and image nodes in the DOM tree as content elements, because they are basically equal.

4.2.1 Design of the tree of visual areas

From section 4.2 it is obvious that different methods have very different representation of the tree of visual areas. Yet, for storage, a single unifying representation is needed.

Definition 7. 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.

The definition is derived from the definition of DOM tree. The difference between a DOM tree and a tree of visual areas lies in the definition of their nodes.

Definition 8. 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.

The set of attributes can contain attributes like position, visual features, size and other attributes which can be used for example by classification algorithm bound to the segmentation method that was used. 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 for the condition of the design being 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 both segmentation algorithms described in section 4.2.

4.3 Tree mapping

The description in sections 4.1 and 4.2 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, 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[11, 3], 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 go towards more clear definition of a visual area'by following definition:

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

5. 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 plain segmentation algorithm while algorithm 2 displays the Cluster-based Page Segmentation algorithm, utilizing structures defined in section 4. We mention both algorithms so they can be easily compared. Note that Algorithm 2 is more formal expression of figure 2.

Algorithm 1 Plain segmentation algorithm

```
def segment_plain(page):
  dom = parse_page(page)
  visual = segment_page(dom)
  return visual
```

Algorithm 2 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
```

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 non-trivial parts that we need to attend to.

1. Creating a Cluster Representative
2. Matching DOM tree to the cluster set
3. Matching nodes of DOM tree on input to corresponding tree D

Creating a Cluster Representative

This task consists of series of small transformations of DOM tree and tree of visual areas of the original page that serves as Cluster Representative origin. This transformation leads to the Cluster Representative's n-tuple $C = (V, D, M_{VD})$.

Before the tree 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 in section 4.1. That is straightforward – a simple recursive, post-order tree traversing algorithm can be used for this. After transforming the DOM tree, creating the graph structure as defined in section 4.1 is trivial.

Since V is very similar to the output returned by segmenting algorithm, the only thing 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 knows this information the best. 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 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 is part of. 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. Note that there is only a small number of clusters for the site in memory so simple iteration is sufficient to gain considerable performance boost against the plain page segmentation assuming the matching algorithm is fast.

For this matching, we use modification of Common Paths Distance measuring algorithm[5] as it has been proven significantly faster than tree-edit-distance algorithms while still being precise enough. However in our practical evaluation we needed to adjust the original algorithm for better precision. Our modifications are following:

- 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 the name concatenated with the id

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

When a web page has been matched to a D belonging to some Cluster Representative, it is possible to use the tree of visual areas V associated with it. 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 upon it and results have to be used to create a new Cluster Representative.

Matching nodes of both DOM trees

One last problem remains when web page has been matched to its Cluster Representative's D and corresponding tree V is retrieved. When processing a web page by plain segmentation algorithm, the next step after segmentation is classification of created visual blocks. That is not necessary in Cluster-based Page Segmentation, as the classification could have been already performed on the retrieved V. However the step after classification, most likely an identification and retrieval of all DOM nodes within blocks of certain class, has to be considered. This step is simple for Cluster Representatives, since 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 are we interested and corresponding set of Cluster Representative's DOM nodes. Now we need to find corresponding DOM node of the input page for each DOM node of the Cluster Representative. If we designate the DOM tree on input as D_I, we are looking for a *Tree mapping between D_I and D*. Note that this is not the main objective of this paper but its needed for the output to be equally usable as output of normal segmentation. Also note that the tree mapping problem is very extensive in its generic form so we focus on solving only a very specific scenario of this generic problem.

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 that root we use a *distinguished path* to a node. In its simplest version, this path is defined as follows.

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 4.1 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 3 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 4 for their construction first. After construction of each path, algorithm 3 can be be used again for finding corresponding DOM node within inspected web page using the path.

Algorithm 4 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
```

6. EXPERIMENTAL EVALUATION

An experimental implementation has been designed and realized as a proof of concept. We used our own Java implementation of VIPS. It gives comparable results as the original with slightly worse performance.

Implemented scenario

The following scenario has been considered: we have a program, outlined by figure 4, which is given a web site to process in a form of starting URL[3]. It is processing the site page by page and printing out important content of each page. In our specific implementation, headings of articles are considered to be the important content but the algorithm can be easily adjusted. Besides performing the content extraction, the algorithm stores all links found on the site. Three lists of links are used: (1) *visited links*, (2) *extracted links* and (3) *outgoing links*.

The program on figure 4 uses very simple breadth-first-search crawler implementation. Everything outside of the dashed box can be considered a part of the crawler. For that we need a global list of all links which are planned to be inspected. This list is designated *extracted links*. 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. In reality, *extracted links* are separated into two lists – there is the global list and a small local list that is composed when parsing a page. After the page is processed, the local list is merged with the global one.

Visited links are used as a hash table for convenience so we can quickly filter out already visited pages. Not performing this would cause deviations in our measurement.

Outgoing links are links that lead outside of the site. In most cases of our definition of a web site, the site is defined as set of pages within the same second-level domain. Note that there are some domains where top-level-domain contains two parts (like `.co.uk`), it is important to take this into account when writing algorithm for detecting whether the target page is on the same site or not. 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 most consistent sites can have as few as 5 templates on the entire site. However even complex web sites have at most several tens of templates (usually depending on granularity of the template-matching algorithm). 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 Cluster Representatives in memory while inspecting the web site they belong to. Such approach is highly con-

[3]For example http://www.slashdot.org

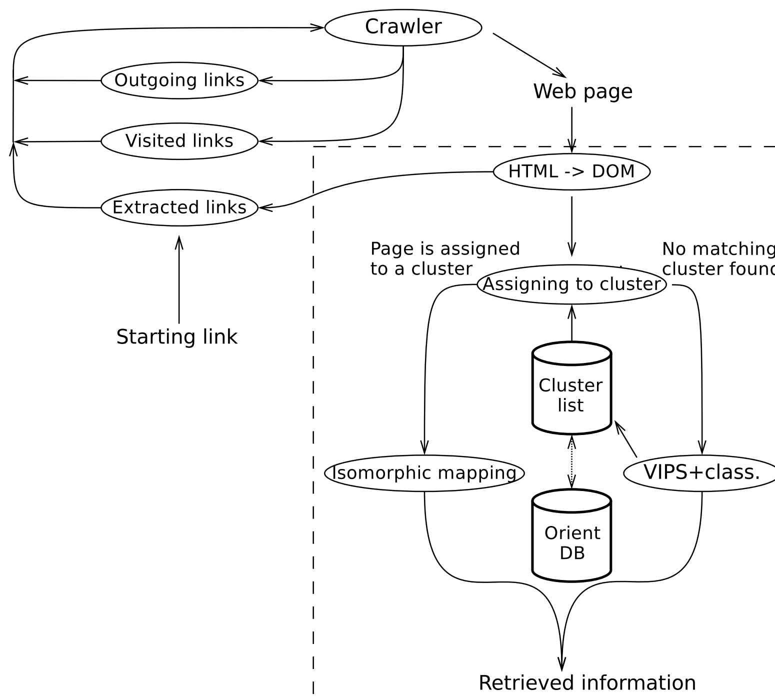

Figure 4: Implementation schema of the CPS algorithm

venient 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. This is displayed on the figure 4 by dotted line between OrientDB and Cluster list.

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[4] before processing any pages on the site. Our object hierarchy has been loosely based on the description in section 4. For matching DOM trees against Cluster Representatives we use the algorithm described in section 5. One modification has been performed in the implementation for the sake of simplicity – section 5 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. As suggested in section 5, the algorithm just iterates over all Cluster Representatives from the web site and compares input DOM tree with them one-by-one. After the DOM tree is matched, its segmented counterpart is fetched and desired content is extracted as described in section 5.

[4]http://code.google.com/p/orient/

Testing and measurements

The implementation has been tested on several Czech and world-wide web sites. 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 5 demonstrates graphically that the number of templates on a single site converges fast to a relatively small number.

Table 2 demonstrates time difference between standard VIPS segmentation and Cluster-based Page Segmentation for each web site. The *VIPS* 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 time necessary to retrieve data with

site	clusters	hit ratio
iDnes.cz	42	91.6%
novinky.cz	15	97%
telegraph.co.uk	32	93.6%
slashdot.org	18	96.4%
finmag.cz	5	99%

Table 1: Cluster counts for different sites

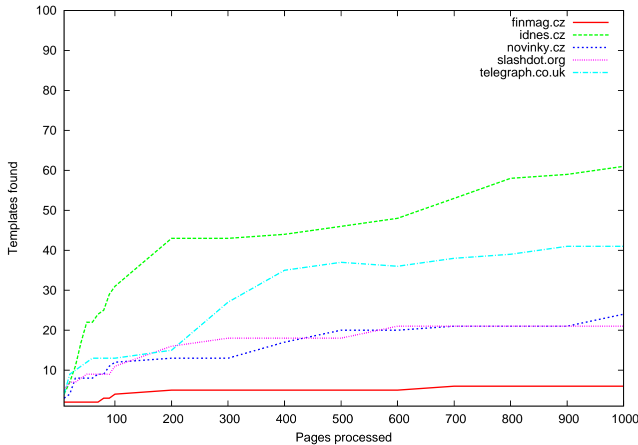

Figure 5: Dependency of cluster count on page count

Cluster-based Page Segmentation. This time includes segmentation of pages when creating new cluster, comparison of incoming pages against existing Cluster Representatives and retrieval of desired content from the page through node matching. The time represents the worst case scenario for CPS, i.e. the first processing of the site. This scenario is worst case because if the site is already processed and 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. Back to the table, the *time saved* intuitively demonstrates how many percent of time necessary for VIPS segmentation of each page on a site is saved when using Cluster-based Page segmentation. Both VIPS and CPS times are measured as a sum of times necessary to retrieve the data from all 500 processed pages.

Our results clearly prove that Cluster-based Page Segmentation offers high performance boost while keeping the same accuracy compared to vision based page segmentation. This is confirmed by one result not visible in table 2. 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.

site	VIPS	CPS	time saved
iDnes.cz	776 s	92 s	88.1%
novinky.cz	1 529 s	41 s	97.3%
telegraph.co.uk	5 476 s	267 s	95.1%
slashdot.org	1 172 s	74 s	93.7%
finmag.cz	7 283 s	101 s	98.6%

Table 2: Performance measurements of the caching algorithm

Note that some times in VIPS column are significantly higher than others. This is caused by complexity of web pages. For example some blogs have quite extensive JavaScript menus which are not displayed but still occupy segmentation algorithm quite a lot.

7. 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 has been used for that. 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 therefore compensates for its greatest disadvantage, making it a superior to other methods assuming that the original precision of algorithms in this family is better than precision of other methods.

While there is still some room for improvement, a solid foundation for further research has been laid down by work in this paper. Our future work shall focus on post-matching processing. In this area it is important to identify what actions are usually performed on a page after it is segmented and adapt the clustering algorithm and structures for these actions.

Acknowledgement

This work was supported by the research programme MSM 0021630528, the BUT FIT grant FIT-S-11-2 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

8. REFERENCES

[1] 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.

[2] D. Cai, S. Yu, J. rong Wen, and W. ying Ma. Vips: a vision-based page segmentation algorithm. Microsoft technical report MSR-TR-2003-79, November 2003.

[3] P. 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.

[4] D. Gibson, K. Punera, and A. 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.

[5] T. 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.

[6] W. H. W. Group. Xhtml 1.0 the extensible hypertext markup language. W3C Recommendation, August 2002.

[7] J. L. Hong, E.-G. Siew, and S. Egerton. Information extraction for search engines using fast heuristic techniques. *Data Knowl. Eng.*, 69(2):169–196, February 2010.

[8] A. L. Hors, P. L. Hegaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document object model (dom) level 3 document object model core. W3C Recommendation, April 2004.

[9] E. S. Laber, C. P. de Souza, I. V. Jabour, E. C. F. de Amorim, E. T. Cardoso, R. P. Rentería, L. C. Tinoco, and C. D. 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.

[10] B. Liu, R. Grossman, and Y. 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.

[11] W. Liu, X. Meng, and W. Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Trans. on Knowl. and Data Eng.*, 22(3):447–460, March 2010.

[12] 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.

[13] M. Spousta, M. Marek, and P. Pecina. Victor: the Web-Page Cleaning Tool. In *Proceedings of the 4th Web as Corpus Workshop, LREC*, 2008.

[14] J. Stenback, P. L. Hegaret, and A. L. Hors. Document object model (dom) level 2 document object model html. W3C Recommendation, January 2003.

[15] G. 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.

[16] K. Vieira, A. L. Costa Carvalho, K. Berlt, E. S. Moura, A. S. Silva, and J. Freire. On finding templates on web collections. *World Wide Web*, 12(2):171–211, June 2009.

[17] 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.

[18] L. Yi, B. Liu, and X. 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.

[19] S. Yu, D. Cai, J.-R. Wen, and W.-Y. 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.