BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

WEB PAGE SEGMENTATION UTILIZING CLUSTERING TECHNIQUES

SEGMENTACE WEBOVÝCH STRÁNEK S VYUŽITÍM SHLUKOVACÍCH TECHNIK

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. JAN ZELENÝ**
AUTOR PRÁCE

SUPERVISOR **Doc. Ing. JAROSLAV ZENDULKA, CSc.**
ŠKOLITEL

BRNO 2017

Abstract

Information extraction and other techniques for mining data from the Web get more important with the development of web technologies and raising amount of information stored exclusively on the Web. However, with this information, the amount of content that is completely irrelevant in context of the presented information grows as well. That's only one of the reasons why it is so important to intensively study and develop preprocessing of information stored on the Web. Segmentation algorithms are one of the possible ways of web page preprocessing. This thesis is dedicated to utilization of clustering techniques for improving the efficiency of existing web page segmentation algorithms, as well as finding completely new ones.

Abstrakt

Získávání informací a jiné techniky dolování dat z webových stránek získávají na důležitosti s tím, jak se rozvíjí webové technologie a jak roste množství informací uložených na webu, jakožto jediném nosiči těchto informací. Spolu s tímto množství informací také ale roste množství obsahu, který není v kontextu prezentovaných informací ničím důležitý. To je jedním z důvodů, proč je důležité se intenzivně věnovat předzpracování informací uložených na webu. Segmentační algoritmy jsou jedním z možných způsobů předzpracování. Tato práce se věnuje využití shlukovacích technik pro zefektivnění existujících, ale i nalezení zcela nových algoritmů použitelných pro segmentaci webových stránek.

Keywords

web page preprocessing, document preprocessing, segmentation, clustering, template, VIPS

Klíčová slova

zpracování webových stránek, zpracování dokumentů, segmentace, shlukování, šablona, VIPS

Reference

ZELENÝ, Jan. *Web page segmentation utilizing clustering techniques.* Brno, 2017. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Zendulka Jaroslav.

Web page segmentation utilizing clustering techniques

Declaration

I hereby declare that this thesis is my own work that has been created under the supervision of Doc. Ing. Jaroslav Zendulka, CSc. It is based on my previously published research. Other information was provided to me by Ing. Radek Burget Ph.D.

.....
Jan Zelený
April 10, 2017

Acknowledgements

I would like to thank Ing. Radku Burget, Ph.D. for outstanding professional leadership and mentorship of my scientific research throughout the entire span of my studies. I would also like to thank Doc. Ing. Jaroslav Zendulka, CSc. for leadership and professional consultations.

Contents

Chapter 1

Introduction

In recent years the World Wide Web has seen such a great expansion, it has become the most important source of information in the world. Many newspapers and magazines have their electronic counterpart, there is also a large number of blogs and community portals containing vast amounts of information. Because people realize the importance of this data source, there is a lot of research and development in this area. The number of algorithms designed to process the data stored on the World Wide Web and give it some meaning grows in proportion to the growth of the Web.

The largest group of the Web processing tasks falls into the area of data mining. This includes the tasks like information retrieval, content extraction and classification and others. Another big group of tasks targets web page restructuring for small screen devices. While this thesis will focus on the former group, there are some severe issues that apply to all the algorithms working with content of the Web that need to be solved in order to make the algorithms efficient.

These issues come from the fact that each web page is a combination of useful information, navigation, related topics, comments, advertisement and many more disruptive elements. Moreover, the useful information on the web can be disseminated, there can even be more different pieces of information (or topics) on one web page.

To address these issues, new methods of preprocessing web pages have to be developed to make the data processing algorithms themselves successful. These preprocessing methods often come in some form of web page partitioning. The designation "some form of web page partitioning" means any algorithm that can split a web page into separate areas. Each of these areas will then carry a content somehow different from the content of other areas. The differentiation parameters depend on the intended usage of the preprocessed data. In case of the data mining tasks, the partitioning is often complemented by classification with the intended result being identification of the 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 or even remove some blocks that are not considered important.

Template detection methods are a good example of those that perform some form of web page partitioning. While they are usually perceived only as remotely related to this area, the definition of page partitioning fits what they do – they separate a page to useful content and the noisy remainder around it. In contrast, web page segmentation methods target specifically the task of web page partitioning. They output a set of different blocks, each block internally consistent. The consistency is usually defined either semantically (content or the structure of contained elements) or visually.

From the perspective of human intervention, there are supervised, semi-supervised and unsupervised segmentation methods. This thesis will focus on the area of unsupervised web page segmentation. Even though the area has been extensively researched, there are only a few ways how is the partitioning task approached. There are two aspects of the web page segmentation that are considered important – accuracy and speed. The existing segmentation methods emphasize one, keeping the other within the range of what is considered acceptable.

Algorithms in the group of vision-based segmentation try to perceive the page as user viewing it would perceive it and perform the segmentation based on visual cues. These methods therefore strongly prefer accuracy, with performance being only secondary. Their performance disadvantage is implied by the fact that the visual cues need to be calculated according to very complex specification.

This thesis proposes a new way how to segment web pages, using the aforementioned visual cues and clustering techniques. The motivation is to come up with such an algorithm, that will be superior to all the existing algorithms in terms of speed-accuracy balance. The proposed method is built from ground up. While it does not share almost anything with current segmentation algorithms, some of its parts are inspired by template detection algorithms which are remotely related to segmentation algorithms.

Another important difference of the algorithm proposed here and all the others that can be found in the literature is its high level design. Compared to others, the algorithm described here can be split into two parts, each one working on a different level and addressing a different use case. While the first part addresses segmentation of individual web pages, the second part takes care of the use case where multiple pages are processed over any period of time. Taking care of this particular use case significantly improves scaling of the algorithm. Note that clustering techniques are utilized in both parts of the algorithm.

This thesis has three parts which are organized as follows: The first part introduces the background and state of the art. Chapter 2 describes basic concepts of the Web, from protocols used for communication to the structure of a web page. Most concepts described there are further used throughout the entire thesis. The chapter 3 summarizes current state of the art in the area of page segmentation and template detection. The description of template detection methods focuses on those concepts that are important for the template clustering covered in chapter 7. Chapter 4 formally declares what is the scope of the thesis.

The second part covers the design of the proposed algorithm. First, its general architecture and targeted use cases are introduced in chapter 5. The segmentation core of the algorithm is closely analyzed in chapter 6. Chapter 7 explains the template clustering as a means to significantly improve scaling of segmentation algorithms. Because the description in this part of the thesis is mostly theoretical, chapter 8 fills in some specifics of the reference program that was implemented to prove the design.

The third part concludes the thesis. Chapter 9 evaluates the result got by running the reference implementation. Some discussion over the retrieved results is included. Finally, chapter 10 summarizes the content of the thesis, the contributions this thesis brings to the research field and it also outlines some plans for the future development.

Chapter 2

The World Wide Web

The original notion, from which the concept of the *World Wide Web*, or simply known as *the Web* has been derived, was introduced in late '80s of the 20th century. Since then the World Wide Web has grown rapidly, soon becoming the most important data and information source in the world. However the data stored on the Web is not organized and it is vastly fragmented.

2.1 WWW as a Distributed Source of Information

The Web is a network of interlinked documents which can be accessed via the Internet. Every document on the Web can contain external resources like images and visual style definitions. All these entities together create an oriented graph of resources, commonly known as Webgraph. Similar pages and sites are closely clustered in this graph, because they usually contain links referencing one another. In this context, "similar pages" express topic-related similarity.

Every document and every resource available on the Web has its unique identifier, so called Uniform Resource Identifier or simply URI. On the Web, URI can be used to identify either names or locations. However, in the context of this thesis, it will be used only to identify locations. It is therefore possible to use more specific type of URI, called Uniform Resource Locator or simply URL. The syntax of URLs used on the web is as follows[52]:

```
scheme://domain:port/path?query_string#fragment_id
```

Scheme is usually `http` or `https` when using encrypted communication. *Domain* can be also replaced by IP address. Both domains and addresses have to be considered since some web content is available only through IP address and on the other hand, some content is only available through the domain name. Each domain can contain several levels of subdomains. On the Web, these distinguish content either physically (on another server) or virtually separated from other parts.

Port number in the scheme is optional, standard is used when not given: 80 for HTTP and 443 for HTTPs connection. The *path* can represent either real path to some resource on the server or it can be only virtual. The latter case is more common on modern web sites, as the path is easy to read and remember and it can fully substitute the functionality of query strings. When the virtual path is used, it supports the concept of templates as presented in section 2.2. Both query strings and virtual paths are used almost exclusively by server side scripts. *Fragment id* is used to designate a certain place in the document.

The structure of the URL can be perceived as corresponding to the structure of information on the Web. In the common understanding, the Web consists of vast numbers of web sites, each web site corresponds to a certain domain. All the content within one web site usually shares some common purpose. Each site consists of multiple web pages, each one identified by the corresponding path within the site. From the information hierarchy standpoint, each web page can be considered as a unit of information. This is where the common understanding of the Web ends, however as section 3.1 explains, the modern understanding of the information on the Web further splits each web page into so called segments – atomic carriers of the information on the Web[13, 35, 50].

The contents of the Web are available by means of HTTP protocol. HTTP stands for HyperText Transfer Protocol. It is stateless protocol utilizing client-server architecture, designed specifically for the needs of the Web. Client-server architecture dictates the basic concept of communication. Clients send requests to the server and it sends back a response to each request. The communication is always client-induced. Stateless protocol means that there is no state of the communication (or context) kept neither on the client, nor on the server. This soon proved to be a problem, especially for dynamic content display. Therefore some methods have been improved to overcome these problems. Despite these methods the core of the protocol remains stateless. That means that every request has to contain all the information required from the client to complete desired action and send back its result.

When considering traversal of the graph of the Web, the HTTP allows only two methods, both directly requiring the client to explicitly traverse between nodes. One is by reading hyperlinks and following them and the other one is by means of redirection response codes. These response codes are used by servers to indicate that the client needs to take additional action to get the content, for example request it from another URL. When traversing through the graph, the client can also hand an information to the server about the previous node it visited. Discovering the graph of the Web (by traversing it) is the core of some content classification algorithms[49, 48, 32].

2.2 Web Sites

In theory, each HTML document can be stored on different server but in reality they are grouped to *web sites*. The web site is a set of web pages that present one thing, represent certain entity (like company, person, product, etc.) or serve a common purpose. The web site is contained within one domain and creates a sub-graph of the graph of the Web. In the previous section a correlation between web sites and domain names was offered. However it's important to note that even web sites can contain a certain hierarchy within themselves. News sites are a good example of such hierarchy – different areas of interest (domestic, foreign, sports, ...) often have their own subdomains and slightly different design. This hierarchy is usually accompanied by a corresponding hierarchy of subdomains. This thesis will however disregard the hierarchy and each site within this hierarchy will be treated on its own. Therefore any reference to a web site in the following text should be understood as a reference to a single web site within this hierarchy.

One of the most important things about web sites is that there are certain regularities within each site. These regularities help user to navigate and find desired information easily on the site. In his work, Nielsen [41] suggested that these regularities are needed and the Web and all pages on it should follow the trend of using them. Since the year of publishing the paper (1999) the Web has really moved in that direction and basic items like navigation are similarly placed on the web pages throughout the entire Web even though these places

change over time, adapting themselves to current trends in UI design. More than regularities on web pages, Nielsen also suggested usage of *templates* on web pages. Among other things these templates create high level of coherence within the site and thus allow even non-skilled web authors to create user friendly web.

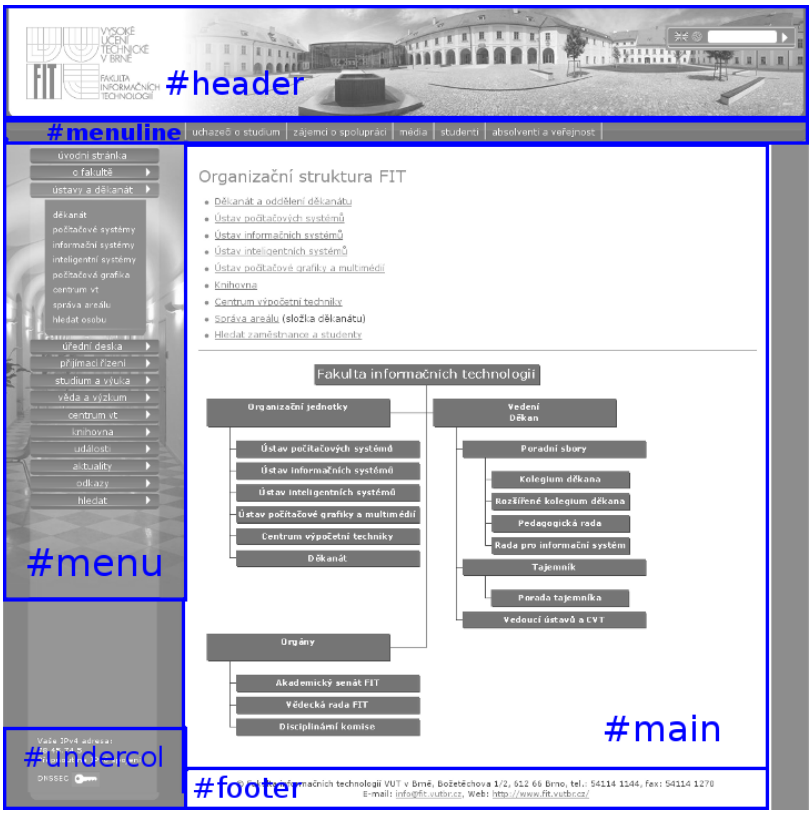

Figure 2.1: Example of web page structure

Figure 2.1 shows a web page with its basic regions outlined. The template of that very page is shown in figure 2.2. The gray parts show the template itself – the skeleton of areas common to multiple pages. White parts represent either blank places which can be replaced by a content different for each page or their ancestor nodes. In [19], Gibson et al. found that templates represent over 40% of the data on the web with additional 30% of visible items on the page appearing in the template, thus creating a large redundancy of data and making some frequency-based data mining methods inaccurate[1]. Gibson also showed that the percentage of data in templates grows each year. The concept of templates has been strongly supported by modern web content management systems.

Templates correspond to classes of pages contained within the web site. There are multiple reasons why templates are important. Their importance for web creators is in their implied capability to generate content from minimal user input (product code, category name, ...). For web-processing algorithms their importance is given by the fact they can be utilized to actually identify the useful content on the web with minimal effort. The concept of templates also allows to represent a whole set of web pages by one common entity describing the set using the template, as the following chapters demonstrate.

[1]This problem is addressed by segmenting the web page

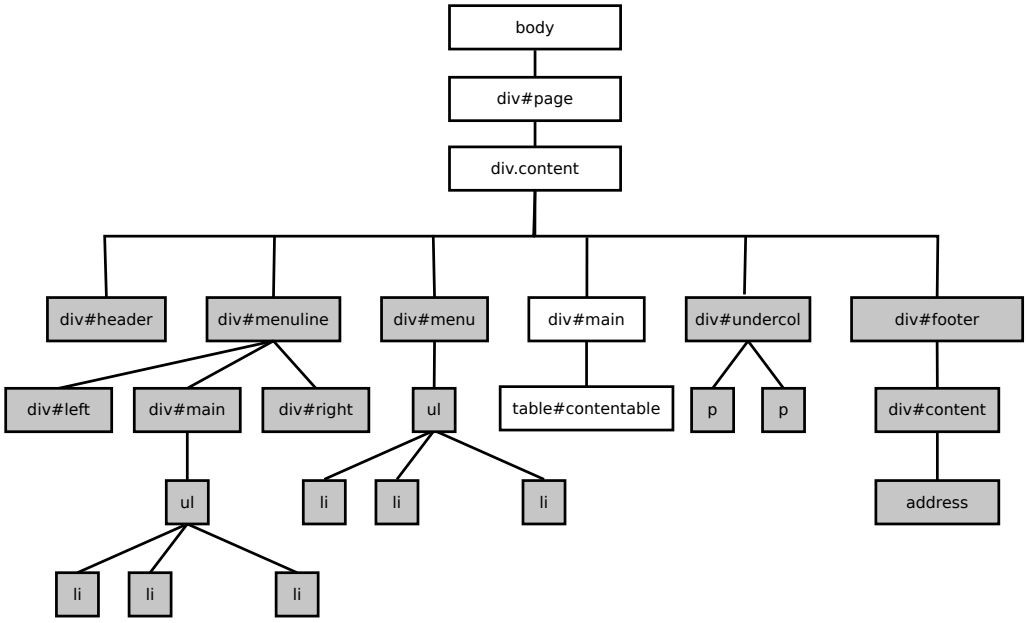

Figure 2.2: Schema of a template

2.3 Web Pages

As the previous text outlined, in the standard understanding of the Web, a web page is the basic carrier of the information. That was true for a long period of time when web pages were quite simple and coherent, containing only limited amount of information. However as one can observe, with the rising amount of information on the Web, individual pages contain much more information and structure as well. The deeper structure of web pages often has a side effect of crumbled semantics of the contained information – with the stronger structure authors often utilize the possibility to include some information that is only loosely related (sometimes even completely unrelated) to the main content. Because the topic of individual parts of the web page can vary, it can no longer be considered an atomic carrier of the information [13] and the individual parts of the page take its place as atomic units of information on the web.

The structure of a web page and the information dispersion correspond to the way how web pages are generated. The concept of templates outlined above is utilized on majority of web pages with complex structure. Figure 2.3 shows how is the final web page generated from the template. In this perspective, template is just a basic structure which, together with some additional resources, defines the layout and visual appearance of the final web page. All the useful data on the web page are taken from external data source, usually a database. In the context of data mining on the Web, the intention is to reverse the process and thus get the data as they are stored in the database before they are merged with templates.

2.3.1 Web Page Rendering

The content on the Web can be presented in several different formats, however the most common one is a classic web page with content written in HyperText Markup Language

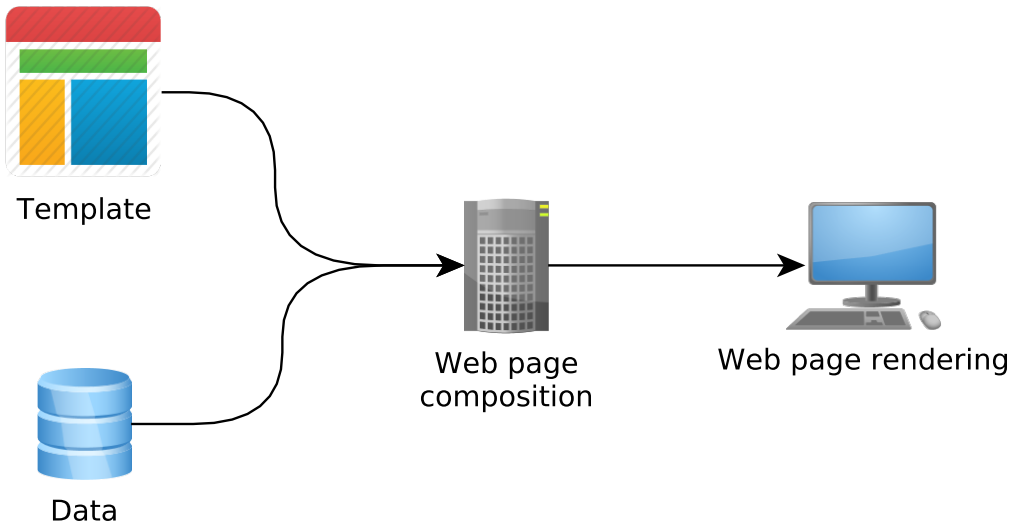

Figure 2.3: Web page life cycle

(HTML). HTML and its derivate XHTML are both languages based upon SGML (Standard Generalized Markup Language, ISO 8879). The concept of hypertext was introduced in 1965 by T. H. Nelson [40] and it refers to a text which is not strictly linear and can contain a links to other texts. In HTML, the most common links are called hyperlinks and they link the page with other pages. Other links may connect the page to its external resources like style sheets. HTML based document is basically normal text document enriched by markup strings called *tags*. These tags enrich the text by adding semantic and visual information to the part of the text enclosed by the tag.

One of the commonly known recommendations for web developers is that simple (X)HTML document shouldn't contain much visual information. If it does, this information is different in each browser since W3C recommendations don't specify the resulting look exactly. In recent years the trend is to fully separate visual information from structure of the web page and enhance the semantic information given by the markup language. The visual information should be then added in separated file called style sheet. In case of (X)HTML files, the content of their corresponding style sheet files is written in language called Cascading Style Sheets (CSS). Typical CSS file contains a set of declaration blocks where each block looks like the outline below. The bloc typically starts with selector specifying which parts of the (X)HTML document will be affected by the block and continues with a set of declarations that modify visual appearance of the affected parts of the (X)HTML document.

```
selector {
  visual-feature: setting;
  visual-feature-2: setting-2;
  ...
}
```

Rules how to process CSS are defined in various W3C recommendations[10]. CSS is cascading because style definitions can create cascades (more rules can match one tag). In these cascades, the more specific rule always takes precedence over the less specific. For

example `tag#id` takes precedence over `tag`, therefore visual feature settings which are in both rules are taken from `tag#id`. In combination with the possibility of multiple CSS files attached to one (X)HTML document and the possibility to create inline style definitions in the document or even within each tag, this makes computation of resulting visual output very complex. This computation is called *rendering* and the big picture of the process is displayed in figure 2.4.

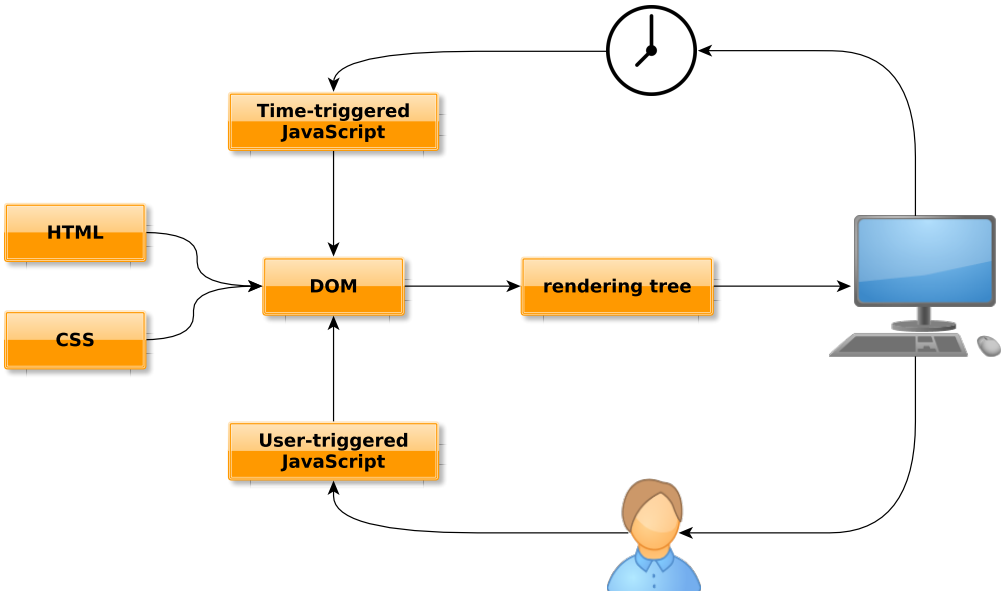

Figure 2.4: Rendering cycle of the web page

As figure 2.4 shows, the rendering process has multiple steps. The first one is transformation of HTML and CSS source to corresponding Document Object Model (DOM). The DOM is architecture independent model used to represent SGML-based documents. More detailed description of DOM is provided in the next section. DOM is especially used in the area of World Wide Web as a means to work with XHTML and HTML documents. The model can be used to describe all content, structure and visual style of the web page. The most important thing about DOM is that it is the only representation of web page that is unified across all client programs and is always up-to-date. That is particularly important for current web pages since the layout of many of them is influenced by JavaScript and some web applications even render entire pages with JavaScript. Both these cases are displayed in figure 2.4. The latter one is time based, as it is triggered at the time of displaying the web page to the user.

DOM is not the final representation of the document from user's perspective. The DOM model only defines how the web page should be modeled in order to be represented accurately. But to render the web page or to represent its visual layout accurately, the DOM has to be converted to a *rendering tree*[2]. Garsiel in 2009 [18] inspected and described the rendering process including its data structures, rendering tree being one of them. After the

[2]Rendering tree is not widely recognized term, as each rendering engine calls it differently. This term was chosen as it accurately describes its purpose.

rendering tree is composed, the page can be displayed to user and if there are some scripts on the page, they can be executed.

2.3.2 DOM and Rendering Trees

The previous section outlined roles of both the Document Object Model and the rendering tree in the process of displaying the web page. DOM is a very complex model as specification [56] and its predecesors prove so for simplification only the features that are important for this thesis follow.

The central entity in the Document Object Model is a DOM node. When converting the source code to DOM, every element of the page is converted to a DOM node. When nesting of elements in the HTML source is taken into account, all nodes together form a DOM tree that represents the web page as a whole. In this representation, every DOM node can be considered to be root of a subtree representing all the content within the corresponding tag in the HTML source and linked CSS.

The DOM model contains only a few basic data types like string, time stamp (represented by integer value) and user data blob. All other entities in the model are understood to be of an object data type. The particular type of an entity is defined by the interfaces it implements – each entity can implement one or more interfaces that are declared by the DOM specification.

From this perspective, the DOM node is just a basic interface in the DOM model, all entities are derived from it by inheriting additional interfaces. As the tree representation of the data in DOM model implies, each DOM node can have, $0..N$ child nodes. The DOM specification precisely defines what entity types can contain which child types. For instance, text nodes cannot contain any children but nodes representing HTML attributes can contain text or references to other entities. There is one interesting case of the parent-child relationship: attributes of HTML elements. As per the specification, these can be represented by one of the basic data types as properties of the DOM node representing the element. However this approach is deprecated, standard child nodes of the `Attr` type should be used instead.

To accommodate all the relationships between nodes within the tree, the DOM model defines an ordered collection data type. That is necessary so each DOM node could store the information about its child nodes. The fact that the collection is ordered shows an important feature of the DOM tree – the sibling nodes are ordered and this order matters. Position of node among its siblings is important for calculation of its visual position and it is also used in some algorithms that analyze web pages.

While the DOM tree is directly derived from the HTML source code and thus represents semantical structure of the document, the rendering tree on the other hand represents the visual aspect of the page. As it was stated above, its design is purely implementation specific, some of the aspects are common across all the implementations:

- it is derived from the DOM tree
- by default, the nesting (and thus the entire structure) corresponds to the DOM tree
- its structure is highly influenced by style definitions
- it is used to put all the elements on the canvas correctly

Because it represents the page as user would see it rendered, the rendering tree is important for vision-based algorithms processing the page. It contains the calculated visual information which would have to be re-calculated by the processing algorithm which would be neither efficient nor accurate (unless the calculation would be exactly the same as used by the rendering engine).

Chapter 3

Web Page Preprocessing: State of the Art

First, it is important to define what should be understood by web page preprocessing. There are many techniques how to preprocess the web page. Their goals and results strongly depend on the main tasks that the preprocessing is preparing data for. The two most common task classes are data mining operations and adaptive view of the web page.

The latter one is used for example for small-screen devices. It's about re-ordering elements on the web page[1, 25] where we need to detect areas that should not be broken apart when performing the reordering itself. For data mining applications, the goals include simple cleanup of the noisy and irrelevant information and detection of smaller and internally consistent areas where the internal consistency can be defined either visually or logically. Nowadays the data mining tasks are more common so the context of this thesis will be in that area from now on, unless specified otherwise.

The motivation of web page preprocessing in the area of data mining is usually to allow more precise results of the subsequent mining techniques such as information retrieval and content classification. The page preprocessing also enables the concept of splitting web page into smaller segments that has been described above. That allows refining results of techniques that were seemingly unrelated before. These include algorithms that perform clustering on the graph of the Web[49].

This chapter will introduce some preprocessing algorithms that can be found in the literature. While *page segmentation algorithms* primarily compete with the algorithm proposed in this thesis, *template detection algorithms* are here mostly because they introduce general mindset and some concrete concepts that are utilized in the design described in the following chapters.

3.1 Page Segmentation

As defined in chapter 1, page segmentation is a process of partitioning the web page into smaller, internally coherent parts, called segments[13, 35]. It is important to note that generally the entire web page does not have to be covered by the set of segments. Considering that, the page segmentation can be more precisely defined as a process of identification of coherent or significant parts within the page.

There are two basic groups of methods how to perform page segmentation – code-based and vision-based methods. In general, code-based methods are much faster but quality of

their results highly depends on heuristics they use and type of page they inspect (some heuristics perform well only on some types of pages, e.g. news articles). Because they don't need the page to be rendered, they rely on inspecting either the HTML code or (more often) the DOM tree which is what makes them fast and scalable.

Vision-based methods on the other hand have greater potential for accuracy[1]. This potential comes from the fact that large portion of information from the web page, such as computed CSS styles, is not evaluated by code-based methods. The approach of vision-based methods is to render the page first and then utilize the calculated visual information. That means these methods try to view the page as user would. By this perspective it is then possible to produce very good results in cost of great computational complexity. This complexity is obviously caused by time consumption of the rendering process, as described in section 2.3.1.

3.1.1 Code-based Segmentation

As said before, code-based segmentation methods are further divided based on the type of heuristics that are used. In this thesis, the distinction is based on representation of the web page that is being inspected. *Text-based methods* inspect textual representation of the web page, whereas *DOM-based methods* first transform the textual representation to a DOM tree which is then inspected.

Text-based methods

Text based methods[33, 15, 11, 26] fall into the code-based method category. They utilize features of the text content of the web page and build heuristics on them. An example of such is a method called NCE, proposed by Laber et al. in [33]. NCE is an algorithm intended to identify and extract the relevant content (i.e. the article) from news web pages.

Finding the location of the article on the web page is based on density of links in the text. Assuming the inspected page is described by a DOM tree D, T is a subtree of D, function $A(T)$ returns concatenated content of all the tags `<a>` in subtree T, $TEXT(T)$ returns the entire content of subtree T and $chars(t)$ returns the number of characters in a text t, the link density of the subtree T is calculated as:

$$linkdensity(T) = \frac{chars(A(T))}{chars(TEXT(T))} \tag{3.1}$$

the link density is practically a measure specifying how much non-navigational text is contained in subtree T. The core assumption for locating the article is that in every page's DOM tree there is a node N which with a positive real number γ has a corresponding forest $F = T_v | visachildofN; linkdensity(T_v) \leq \gamma$ which has very high F_1 score. The $F_1 score$ (also called the $F - measure$) is often used in statistical analysis of binary classification, Laber et al. just define new metrics `precision` and `recall`.

Assuming C is a chunk of text, $words(C)$ is a bag of words corresponding with that text, $|A|$ denotes the cardinality of the bag and $rel\_words(D)$ is a bag of all relevant words[2] on

[1] The accuracy is usually defined as a consistency between the result and human perception of the page.
[2] defined by human observer

the web page represented by by DOM tree D, the metrics are defined as:

$$precision(C) = \frac{|words(C) \cap rel\_words(D)|}{|rel\_words(D)|} \tag{3.2}$$

$$recall(C) = \frac{|words(C) \cap rel\_words(D)|}{|words(C)|} \tag{3.3}$$

The algorithm itself just performs depth-first-search traversal of the DOM tree and tries to find a node N which fits the conditions described above. To sum up, this heuristic is almost purely based on quantity of text and hyperlink content of DOM nodes. Laber et al. also propose other two heuristics used for identifying comments for the article and the title of the article. Both heuristics are again based on counting characters of DOM subtrees. In addition to these text-based heuristics, the algorithm for finding comments uses one DOM-based heuristic for finding a repetitive patterns in the DOM tree.

DOM-based methods

Another group of code-based segmentation algorithms performs general inspection of the DOM tree and segments the content using various heuristics[22, 50, 24]. Some related information extraction [53] and document cleaning [54, 61] approaches share the same concept too. In their extensive article about information extraction[22], Hong et al. introduce their technique for traversing the DOM tree and selecting relevant content. They focus on retrieval of repeating pieces of web page with different content by means of algorithm called *WISH*, which will be briefly described below. The algorithm is divided in several steps. In the first step, they extract content candidates.

The following stages only filter results the algorithm identified during the first stage. Output of the first stage is a list of all groups of repeating content (data records) on the page. All heuristics used in these stages are based on observations, which are summarized in [22]. The observations are as follows:

1. Repeating data records take large portion of the page
2. Data records are usually repeated more than three times on a page
3. A regular expression can be devised for description of single data record. Since all data records share the same template, it will apply on all of them
4. Data records usually consist of a small amount of distinct HTML tags

The WISH algorithm recognizes four types of data regions: the main content, advertisements, menus and menu bars. Only the first one is deemed relevant in the paper and the purpose of filtering stage is to get rid of the others. Each of these groups contains data records, which have slightly different characteristics. First, the filtering targets advertisements. Data records containing advertisement are observed to contain less than three HTML tags in each data record, so the filter traverses the data region list and deletes all data regions which correspond to this rule. The second filter targets those data regions, in which records don't contain similar tree structures. This filter is based on observation number three. Similarity of tree structures can be inspected for example by means of tree edit distance algorithms as described later in section 3.2.3, but these algorithms are too complex and they would make this method very inefficient. So instead a simple heuristic is used, which compares a number of distinct tags on each level of both trees. The last step

is to filter out those regions with less than three records, as it was observed that they are irrelevant on the most pages. After the list of data regions is filtered, the most important part comes. Now every data region has to be assigned its relevancy score. The scoring function described in [22] is partially based on observation one. It determines the size of area taken by data records by counting characters and images each data record contain. For better precision, it also takes line breaks and similar elements representing free space in records into account. Besides these heuristics, a depth within the DOM tree is taken into account as well. Data region with the best score is considered to be the main content of the page. This very sophisticated method is based on many observed and empirically proved values and it is the best example how various heuristics are used in content detection.

There are three main categories of features that can be used for heuristic:

- *Markup-based features* — perhaps the most extensive category. It's possible to analyze parents of the node, the node itself and its child nodes. Some special nodes like `<strong>` and `<em>` are usually interesting, as they designate higher importance of the contained text and child nodes. Another possibility is to investigate line breaking elements (usually used for detection of content separator).
- *Content-based features* — these represent features gathered from analyzing characters, words and sentences.
- *Document-related features* — features in this category include position in the document or a content-based features related to the context of the entire document.

3.1.2 Vision-based Segmentation

Vision-based methods are all those that use visual features of elements in a web page to determine the partitioning. In the literature, the dominating method is the first one described here: VIPS. It is the pioneering method of this approach which has not been unambiguously surpassed by any other algorithm. That's also where this group of methods got its name.

VIPS

A pioneering work in vision-based page segmentation was laid down by Cai et al. in [13]. In this work an algorithm called *Vision-Based Page Segmentation* or more commonly *VIPS* was proposed. This algorithm gave name to the entire family of segmentation methods which followed this approach. A large portion of subsequent work was derived from the very concept this algorithm introduced. It's worth noting that while the algorithm is the first one that uses visual information contained in the web page, the originally described implementation is not entirely vision-based when strictly following the description in section 2.3.2 and 3.1, as it does not use the rendering tree, it only uses the information stored within the DOM tree.

The algorithm is built on the concept of *Degree of Coherence* (or simply *DoC*). It is a metrics, represented by integer or real number, defined for each block. As the name suggests, it is directly proportionate to the internal visual consistency of that block. Intuitively, parent node cannot have greater DoC than any of its children. A threshold value affecting granularity of the result is related to DoC. It's called Permitted Degree of Coherence or simply PDoC.

The VIPS algorithm processed the web page in several steps. The first step is an extraction of visual blocks. It consists of a top-down tree traversal through the DOM

tree and inspection of visual cues. The traversal is iterative – in each iteration a new node representing visual block is detected in the DOM tree. After this detection a decision is made whether the block shall be recursively segmented further or not.

The second step is separator detection. *Separator* is defined as horizontal or vertical line or rather rectangular area which does not intersect any of previously detected blocks. The algorithm is initialized by a single separator covering the whole page. Then for each block we perform a detection of its relation to each existing separator (relations are visually outlined on figures 3.1, 3.2 and 3.3 respectively):

- if the block is fully covered by area of a separator, divide this separator into two
- if the block partially intersects a separator, shrink that separator, so the intersection is eliminated
- if the block fully intersects with a separator (i.e. covers entire height of a horizontal separator or entire width of vertical separator), remove the separator entirely

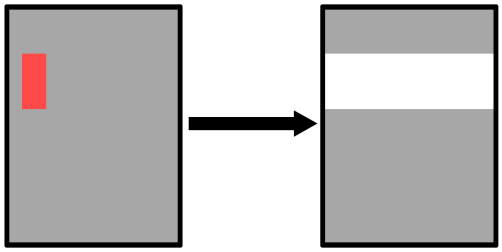

Figure 3.1: Divide the separator

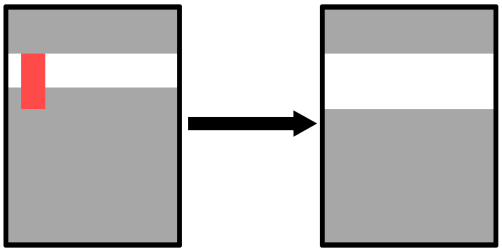

Figure 3.2: Shrink the separator

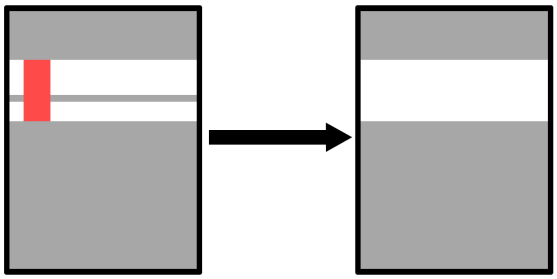

Figure 3.3: Delete the separator

The algorithm is finished by removing separators at the edges of document. After we have all separators, we assign the weights based on visual difference of adjacent blocks. Note that on each level of the block tree this algorithm produces either only vertical or only horizontal separators, but the direction is changing in each level of the block tree.

The final step of VIPS is content structure construction. In this step we iterate through a list of previously found separators and merge visual blocks adjacent to them. It's important to merge blocks adjacent to separators with the smallest weight first. The merging continues until all blocks meet the granularity requirement $DoC >= PDoC$.

Besides the shortcoming of not fully rendering the web page, there are a few others:

- In some cases direct division of a visual block is impossible and utilization of virtual blocks is required. This can have negative impact on further processing, because blocks are not really present in the document.
- Resulting tree represents page segmentation but some information such as mutual position of blocks is missing. That information might be useful for results refinement or by some subsequent algorithms.

Even though VIPS is the oldest algorithm and has its shortcomings, it has not been unequivocally superseded in both accuracy and processing time. There has been a lot of work in the area that improved its accuracy [35, 34, 68, 59, 60] and adapted it to modern standards in HTML[3, 2] but that work has been derived from the original approach and still uses the original VIPS as a core. Some alternative algorithms that rival VIPS exist as well, those will be covered in the following sections.

Box Tree Analysis

Burget in [12] introduced a bottom-up method based on some concepts introduced by VIPS. The algorithm he described has a goal to deal with some VIPS shortcomings. It creates a tree of visual areas in four steps:

1. Create temporary tree of boxes
2. Find boxes which represent standalone visual areas
3. Detect continuous areas
4. Find significant areas

The *box* is a basic compound of the web page created by rendering it according to specifications. It is defined as a rectangular area with defined position and width/height and visually containing either another set of boxes or a content of the web page.

The first step of the algorithm creates a tree of these boxes, computing their like dimensions, position, visual features and nesting in the process. The structure of the tree defines visual nesting of the boxes. There are three possible nesting relations that two boxes can have:

- *Boxes don't intersect.* In that case they are not related.
- *One box is completely nested in another.* In that case the larger one is considered a parent of the smaller one.
- *Boxes have a partial intersection.* This is detected by mutual position of boxes' corners. In this case a z position is important to calculate the visibility. If we consider that higher z position means that the box is on top of the box with lower z position, then a box with lower z is considered to be a parent. Also we have to extend boundaries of the parent box to accommodate the entire child box.

The whole page is representing a root node of the box tree. Only two rules apply in the tree: parent box contains all child boxes and all child nodes on the same level of the tree mustn't have intersection with each other.

In the second step a tree of visual areas is created by merging boxes created in step one. Visual area contains exactly one box which is visually distinct from adjacent boxes. The distinction is defined by one or more of the following three features:

- The box directly contains just textual content
- The box contains nested boxes and it is separated by border at least on one side
- The color of the box is set and it's different from the color of parent box

If the box is not visually distinct, it's merged with adjacent boxes and the whole process is repeated. The final tree of visual areas corresponds with the tree of visual boxes.

The third step brings more merging. This time visually similar nodes (e.g. adjacent paragraphs of text) are merged into one continuous block. In this step the information about mutual block position is required.

Significant areas are looked for in the last step. Optical separators like borders, lines or just big spaces between blocks are used for this. An extended version of the method for separator detection presented in VIPS algorithm is used to achieve this. The first part corresponds to the VIPS separator detection. The second part then applies inverse approach. The division works the same way as the division of separators in VIPS does, the only difference is the opposite role of separator and a block of content – in the beginning the entire page area is considered to be a content and it is then divided by particular separator blocks (e.g. big spaces between text, borders or horizontal rulers).

This method demonstrates some advantages over VIPS. The most important one is that it's strictly focused on visual information rather than the DOM tree. That can bring better results e.g. for absolute positioning on the web page.

Web Content Clustering

Another method entirely independent on VIPS was presented by Alcic et al. in [6]. While the paper does not offer any particular design, it brings up and analyzes the idea of segmenting the web page using clustering algorithms. The concept is based on the observation that each web page is just a set of pieces of the content and that these pieces can be grouped together in such a way that each group represents one topic. The underlying assumption is that each such group can be created by clustering the right content pieces. The paper offers validation of that assumption by presenting an overview of both parts necessary – clustering algorithms and metrics.

The first part of the paper covers the metrics that can be used. A few very basic metrics are explored individually:

- mutual position of nodes within the DOM tree of the web page (DOM-based distance)
- semantic distance of the content using WordNet-based generalization of concepts and TF-IDF[3]
- mutual position of the nodes on the rendered web page (vision-based distance)

The vision-based distance is outlined in figure 3.4. While the formal specification is not fundamental, it's worth stating that the distance is calculated as euclidean distance without the square root between the closest points of the two boxes.

Three clustering algorithms are compared in the second part of the paper: partitioning algorithms represented by k-means and k-medoids, hierarchical agglomerative clustering and

[3]Term Frequency-Inverse Document Frequency is distance metrics used in plain text data mining

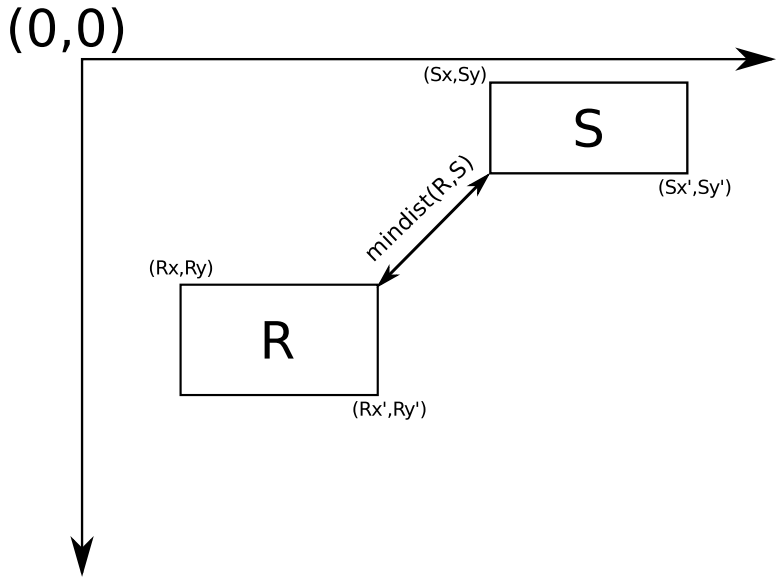

Figure 3.4: Vision-based content distance

density-based clustering represented by DBSCAN. Of these three the DBSCAN was proved to be the most convenient.

The most interesting conclusion of the paper was that using the vision-based distance didn't bring results that would be significantly better than the other two metrics, no matter which clustering algorithm was used. Instead, the DBSCAN in combination with the DOM-based distance showed the best average statistical results.

Other Vision-based Segmentation Methods

In the latest research, authors often avoid using the HTML-based heuristics by using alternative ways of block detections. Sections 3.1.2 and 3.1.2 both describe methods going in that direction and some more examples follow. Liu et al. [36] constructs a graph of spatial relationships among the rendered elements that is later partitioned with a Gomory-Hu clustering algorithm and Zeng et al. [66] compute a seam degree and content similarity of the individual blocks in order to divide larger visual blocks to smaller parts. Finally, Xu et al. [62] applies the Gestalt laws of grouping on the extracted visual blocks.

Other vision-based algorithms use entirely graphical representation of the input document that allows to abstract from the HTML-related implementation details. Cormier et al. [14] uses an edge detection algorithm for detecting the visual separators between the content blocks. Similarly, Wei et al. [58] uses Hough transform for the same purpose and Kong et. al [27] recognize atomic objects using image processing methods and perform their grouping by using a spatial graph grammar.

3.1.3 Hybrid Segmentation

The hybrid approaches combine the DOM-based and vision-based ones in order to obtain higher segmentation accuracy or for specific applications. Sanoja et al. [45] and Manabe et al. [39] both combine the content structure (DOM) with the visual information obtained

from a web browser in order to increase the accuracy in comparison to the VIPS algorithm. Safi et al. [44] process the input document in two steps: first, a visual information analysis is performed and in the second step, DOM tree filtering is performed based on the analysis results with the aim of supporting visually impaired users. Finally, Fumarola et al. [16] combine the DOM with a visual information model in order to extract visually presented lists from the input documents. More generic content extraction use case is presented by Song et al. [51]

3.2 Template detection

The goal of template detection methods such as [37, 31, 30, 17, 4] is to find and filter redundant information (noise) on the web page[9]. Because they do not try to separate individual segments on the page, they are generally faster than segmentation methods. The only expception are those template detection algorithms that detect templates using segments[5, 29] instead of DOM nodes. However, that group is not interesting for the purpose of this thesis and therefore will not be considered further. The reason is that these algorithms are not suitable for template clustering (lower speed, unsuitable input) where they are being used in this thesis.

The goal of template detection algorithms implies that the format of their results is slightly different from the format of segmentation methods. Among other things it means these methods are only loosely related to this thesis from the result-related perspective. Their importance however lies in their speed. This thesis will inspect the means these algorithms use to provide the result quickly. It will be later demonstrated that some of the features of template detection algorithms can be used to improve the performance of segmentation algorithms in certain use cases.

Besides operating on DOM trees rather than render trees, another key factor that makes these methods fast is the fact that they process multiple pages at once. For this approach to work, the pages that are processed at once have to be based on the same template. The detection of whether or not are two pages based on the same template is the part that will be utilized to boost the page segmentation.

3.2.1 Site Style Tree

A work performed by Yi et al. [63] uses a tree structure called *Site Style Tree*. Its basic idea is that the DOM tree can describe layout of the page, but it is difficult to use it for studying presentation style of the web site. The Site Style Tree, also referred to simply as SST, is derived from a set of DOM trees which share the same template. The usual experience is that these DOM trees represent web pages that all come from the same site.

The Style Tree is composed of nodes similar to nodes in DOM tree. They contain tag designation and attributes relevant to visual representation of the node. No other attributes are present, but one additional is needed – a counter of pages which contain that particular node on that particular level of a DOM tree. The construction of SST is not complicated – it consists only of merging of DOM trees of pages using the same template. For each node we try to find whether its child nodes are already in the SST as children of that node. For every child that is we only increment its counter in SST node. Those that aren't in SST are converted to SST nodes and incorporated to the tree. This is just the basic concept. Some modifications can be performed in order to achieve better results:

1. *node merging* – is it possible that two sibling nodes in SST are in fact one node displayed differently on some pages. We can merge them in one in case their tags and attributes are the same and their content is similar. This similarity is described in detail in [63].

2. *virtual leaf nodes* – leaf nodes tend to fragment the page, therefore it is convenient to use their parents as virtual leaf nodes instead

The goal of the following algorithm is to clean the page of content which is not important and is considered to be a *noisy information*. In this context, noisy information is defined differently than in the previous sections. To define noisy information, it is first necessary to define node importance. For each node N we can define its importance as its entropy E_N:

$$E_N = \sum_{i=0}^{child\_count} -p_i \cdot \log p_i \tag{3.4}$$

where p_i is a probability of i-th child node appearing in the node `N`. This metrics alone is very inaccurate, as it would for example imply that the page as a whole is always very noisy. An extended metrics is used instead: the importance of non-leaf nodes is derived not only from their own entropy but also from the average entropy of their child nodes. Based on this importance metrics, it is possible to distinguish four notable element categories. Based on the noise level associated with the element it is possible to classify element as:

- *noisy element* element that has importance under a defined level or
- *maximal noisy element* noisy element, which does not have noisy parent

And based on the noisiness of element's child nodes, it is possible to classify element as:

- *meaningful elements* elements, which have no children marked as noisy or
- *maximal meaningful elements* meaningful element, which does not have meaningful parent

It is possible to reduce the SST significantly by deleting all child nodes elements that are classified as maximal meaningful, maximal noisy, or both. Consider SST outlined in figure 3.5 as an example. In this example, if the gray-shaded are classified as noisy, the resulting simplified SST is displayed in figure 3.6. The simplified SST can be then considered as a template to which a DOM tree of every page is mapped and can be filtered accordingly. The process is very simple and very fast. However there are some situations when this algorithm is not completely convenient. Sometimes a situation occurs when meaningful element contain noisy descendants (which are of course not its children), which can't be filtered by simplified SST and full SST has to be used.

3.2.2 Pagelets

Yossef et al. in their work [8] introduced a concept of utilizing *pagelets* in template detection. Pagelets are defined as a logical regions within the web page, which have a single function (e.g. navigational function or a text topic). What is also important is that a pagelet can't be a part (child) of another pagelet. That means its parent can't have the same function.

Yossef builds on the following principles to find pagelets, known not only in web page processing context:

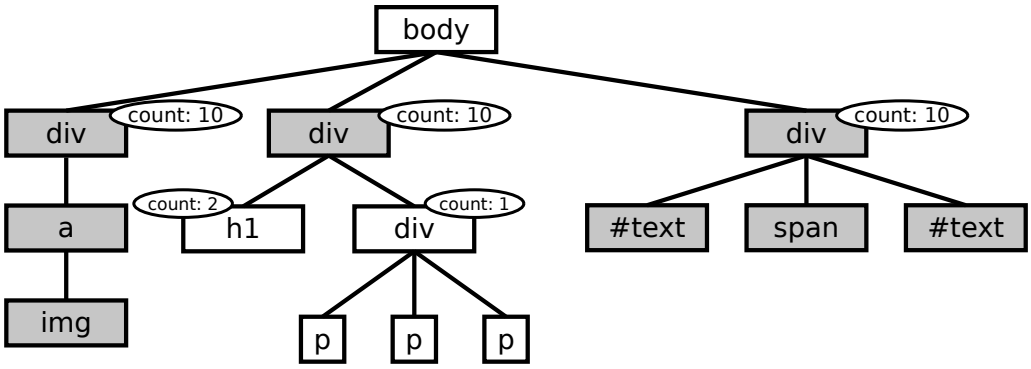

Figure 3.5: Outline of SST

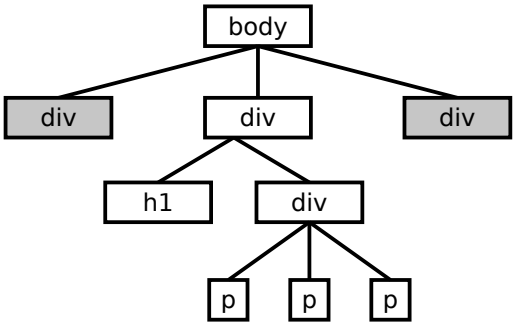

Figure 3.6: Simplified SST

1. *Relevant linkage principle* – links point to relevant resources (the principle can be also applied on citations in scientific work)

2. *Topical unity principle* – documents often linked from the same document are related

3. *Lexical affinity principle* – proximity of the link and text (or two links or blocks of text) on the page usually means that they are related

To find a template, it is first necessary to compute pagelet *shingles*. These are defined as text fingerprints which aren't influenced by small changes in the text. Yossef introduces two algorithms for finding a templates. The first one, the *Local Template Detection Algorithm* has very simple principle: `M` pagelets gathered from a set of distinct pages are grouped using clustering algorithm based on their shingle and each group represents a part of the template. The *Global Template Detection Algorithm* adds one more step. Grouping of pagelets is again done based on their shingles. After this step is done, a graph is created where pagelets represent vertices and connections between parent pages of pagelets are represented by edges between vertices representing those pagelets. The resulting template is then created by every component of the graph which is not a singleton.

3.2.3 Tree Mapping and Tree Edit Distance

The tree edit distance problem has been known for several decades[46]. It has been used for evaluation of structural differences between generic tree structures. In the context of data mining on the Web, it's used to evaluate differences between DOM trees. Note that further text will always refer to the tree edit distance between two DOM trees unless explicitly stated otherwise. In particular, algorithms working with the tree edit distance can tell how similar are two given trees (i.e. to determine if they belong to the same template) or which parts do the two trees share (i.e. to identify the template itself).

The tree edit distance comparison uses three basic operations with nodes in the tree: insert, replace and remove. Each operation type can have its cost defined but usually they all have the same value. The problem of tree edit distance can be defined as finding a mapping from tree A to tree B with minimal cost. The mapping has three important rules: (1) only one operation can be performed on each node, (2) the order between siblings has to be preserved and (3) the ancestor-successor relation between any two nodes has to be preserved.

The generic tree edit distance problem on unordered trees is proved to be NP-complete [67] and algorithms solving generic tree mapping weren't much efficient. In the practical application, however, it is possible to define restricted formulation. By imposing some restrictions (e.g. [55]), four new definitions of the problem are formulated and fast algorithms utilizing them are proposed: *alignment*, *distance between isolated trees*, *top-down* and *bottom-up*. The latter two are used in template detection and will be briefly described further.

Top-down Approach

When having two DOM trees T_1, T_2 and nodes t_1 and t_2 belonging to trees T_1 and T_2 respectively, the basic principle of a *top-down mapping* can be informally described as: if mapping of node t_1 to node t_2 exists, mapping of parent of node t_1 to parent of node t_2 also has to exist.

Restricted top-down mapping (RTDM) follows the same principle, but it also restricts replacements of different nodes to the leaves of the trees. If the mapping between two DOM trees T_1 and T_2 is defined as a relation designated M, the restricted top-down mapping can be described as: if there are two non-root non-identical nodes t_1 and t_2 in trees T_1 and T_2 respectively and $(t_1, t_2) \in M$, then there can be no descendants of theirs present in M

The restricted top-down mapping is outlined in figure 3.7. The DOM trees T_1 and T_2 are given, the mapping is indicated by the dashed lines. It is possible to see that for every node, the conditions above are met.

The RTDM algorithm uses matrix representation of child nodes of roots of T_1 and T_2. For each couple $(t_1, t_2); t_1 \in children(T_1), t_2 \in children(T_2)$ it evaluates cost of substitution, deleting children of t_1 and inserting all children of t_2 to be the new children of t_1. From these three options it always chooses the one with minimal cost. If both t_1 and t_2 have descendants and the subtrees aren't equal, the algorithm computes their minimal mapping cost recursively. To speed things up, RTDM uses pre-step based on ideas presented in [55].

The algorithm expects input consisting of a set of pages that are based on the same template. The template structure is constructed by comparing DOM trees of all these pages and expanding them by wildcard nodes (by either adding them or replacing/removing original DOM nodes). A complete table of rules how to expand the tree is described in [43]. The resulting tree is called *node extraction pattern tree*. This tree with wildcards removed

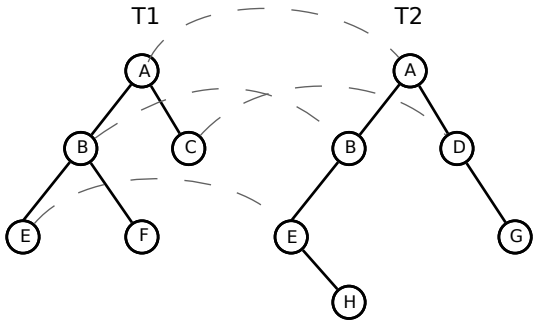

Figure 3.7: Restricted top-down mapping

represents the template. Filtering content from web pages is then just about removing all parts which don't correspond with wildcards in the node extraction pattern tree.

Bottom-up Approach

The bottom-up tree mapping was described by Valiente in [55]. As its designation says, it is based on a tree traversal from its bottom. The definition is exactly inverse compared to top-down mapping definition. The mapping M between trees T_1 and T_2 is bottom-up if for every couple $(t_1, t_2) \in M$, where t_1 and t_2 are nodes of T_1 and T_2 respectively, the following condition is valid: $\forall c_i \in children(t_1) : \forall c_j \in children(t_2) : (c_i, c_j) \in M$. Less formally it is possible to say that a node can be mapped only if all of its children are mapped as well. A graphical example of bottom-up mapping is shown in figure 3.8

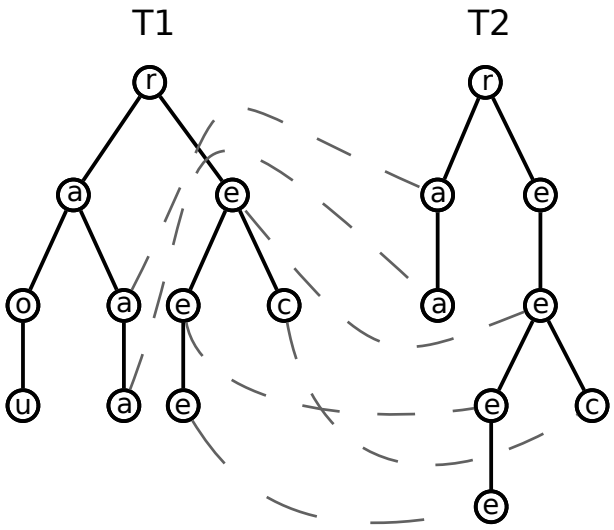

Figure 3.8: Bottom up mapping

Valiente proposed an algorithm, which has linear time complexity and is therefore amongst the best in the literature[57]. He transforms input DOM trees into directed acyclic graph G. Vertices of the graph are representing equivalence classes of nodes in both trees. The equivalence relation is defined as follows: *node t_1 and node t_2 are equivalent if and only*

if subtrees rooted in t_1 and t_2 respectively are isomorphic. Directed edges in graph G are between those vertices u and v for which applies that *there are nodes x and y in either of the trees T_1 and T_2 which are in parent-child relationship and $x \in u$ and $y \in v$.*

Finding the mapping between input trees utilizes walking through one tree, searching for nodes in the second tree which are equivalent to it and checking their subtrees for isomorphism. Building a template then consists of following steps:

1. create a list of mapped nodes contained in T_1
2. sorting the list by nodes' height (node closer to the root goes first)
3. retrieve a path from each node in the list to the root of T_1 and append the path extended with the subtree to the template tree
4. delete all nodes in the appended subtree from the list

Tree Paths Approach

While the similarity of two pages is described in the previous sections as a mapping distance between the DOM trees, Gottron suggests[20] other distance measuring methods which he proves to be significantly faster than RTDM. His work can be considered somewhere between page segmentation and template detection, though it is closer to the latter one. For segmenting a page he retrieves other pages through hyperlinks on the first page. On these retrieved pages he performs template-detection and then uses the template to segment the original page. On the beginning of the process every text node is considered to be a page segment. For each such segment its frequency of appearing in different pages is counted. If this frequency is high, that very likely means that it is a part of the template. Basically the algorithm evaluates informational gain (Entropy) for each block and it discards those with low gain.

Gottron describes three methods to calculate structural distance between two pages:

1. *Common paths distance*
2. *Common path shingles distance*
3. *Common tag sequence shingles distance*

The first one is based on enumerating all root-leaf paths in the tree (i.e. paths that begin at the root of the tree and end with a leaf node – for each leaf in the tree there is one path). The distance is then measured as a size of intersection of sets S_1 and S_2containing all paths from trees T_1 and T_2 respectively.

The second method improves precision of the first one. It considers splitting paths to a smaller pieces, called *path shingles*. The advantage of this approach is that the first algorithm would detect two paths non-equal even with one node differentiating whereas this algorithm would only evaluate the particular shingle as non-equal and the rest would be included in the intersection.

The third method solves the computational complexity of comparing two sequences of tags each representing the entire web page. Instead the sequences are again split into shingles and shingles are then compared to each other with much less demands on computer power.

3.2.4 Template Detection Algorithms as Clustering Metrics

When inspecting template detection algorithms, one prevailing feature emerges. It has been said in the previous text that the same methods that can identify common parts on the web pages can be used to determine how similar two web pages i and j are. In fact, these algorithms meet all the conditions that distance metrics for clustering algorithms have to meet:

$$d(i,j) \geq 0 \tag{3.5}$$
$$d(i,i) = 0 \tag{3.6}$$
$$d(i,j) = d(j,i) \tag{3.7}$$
$$d(i,j) \leq d(i,h) + d(h,j) \tag{3.8}$$

That effectively means it is possible to write a clustering algorithm that would group together web pages for further processing that are based on the same template. From all the algorithms listed above, the tree-mapping algorithms might seem to be the best option for this task, as they tend to be as precise as possible in practical application. However as Gottron demonstrated in [20], the tree path method is comparably precise while being much more efficient, even in the most simple variant. That's why it's the best candidate for utilization in this thesis.

Chapter 4

Motivation and Goals of the Thesis

Various methods for preparing web pages for data mining procedures have been introduced in the previous chapter. These methods approach the problem of web page preprocessing from several completely different directions, focusing on different problems which can be encountered during different data mining tasks.

In the initial phases of the research it became clear that the concept of page segments being atomic carriers of information on the Web is becoming increasingly important. However while there has been some effort in the area of web page segmentation, the vision-based page segmentation has not progressed much further since its discovery. Several algorithms were created, but most of them are built on top of VIPS algorithm. Many of them propose just some heuristics to improve its results, but the fundamental design remains the same. The research in this area has been stagnant most likely because of large time complexity of visual segmentation. Especially in the area of the Web, the time complexity is very important considering the need for algorithm application on a large scale. The problem is that this time complexity is not balanced out by equally significant gain in accuracy.

As a consequence of lack of development in recent years, vision-based page segmentation algorithms often don't fully take modern technologies like HTML5 and CSS3 into account. These technologies make the web pages more dynamic and, when rendered, often very different from the DOM tree. This thesis focuses on solving both mentioned areas. Its goal is to design a new vision-based segmentation method, which:

1. Will consider new technologies used in the process of development of web pages
2. Will be optimized from the perspective of time complexity

The task of designing a new vision-based segmentation method can be divided into four parts, each one being an important partial goal that needs to be fulfilled in order to consider the design acceptable.

1. Formal specification of the problem
2. Design of new segmentation method and its formal specification
3. Design of data structures and algorithm optimization with focus on time complexity
4. Experimental evaluation of the new method's properties

Chapter 5

Web Site Processing Using Clustering Techniques

As the previous chapters show, there are many ways how to approach web page segmentation. Some of them focus on speed, others aim for precision and some try to optimize both but they sacrifice granularity of the result. In the context of data mining, the area of main focus shifts depending on the ultimate objective of the application that performs the segmentation. The ultimate objective is meant in terms of the intended subsequent task that is to be performed on the data. The most common objectives are:

- identification of the most relevant content on the page
- identification of individual building blocks of the web page
- identification of similar pages within the web site

This thesis focuses on two particular use cases of web page segmentation. In both cases the segmentation is expected to be only the first step in the web page preprocessing. In other words, some subsequent treatment is expected to take place in order to finish the preprocessing. One example of such subsequent treatment is classification of segments identified on the page, as the segmentation itself will not tell what kind of content do these segments contain. Therefore the segmentation can't be used to replace template detection by itself. Instead, it offers solid foundation for other algorithm to achieve that.

Single-page segmentation: this is very generic use case. It covers virtually any scenario where segmentation can be used. In context of this thesis it is rather marginal but it is included to demonstrate the quality of the segmentation core that is described in this thesis. URL address of the page that is to be segmented is the only input in this use case. The scenario, displayed in figure 5.1, is very simple:

1. A web page is fetched from the Internet in a form of its source code
2. The code of the web page is transformed to the corresponding DOM tree
3. All the linked resources are fetched from the Internet and the information they contain is merged into the DOM tree
4. The DOM tree is transformed to a rendering tree by a browser rendering engine[1]

[1] CSSBox rendering engine is used in the design described in this thesis (http://cssbox.sourceforge.net/)

5. Segmentation is performed on the rendering tree
6. A set of visual areas is presented on the output of the segmentation algorithm that can be used for further processing

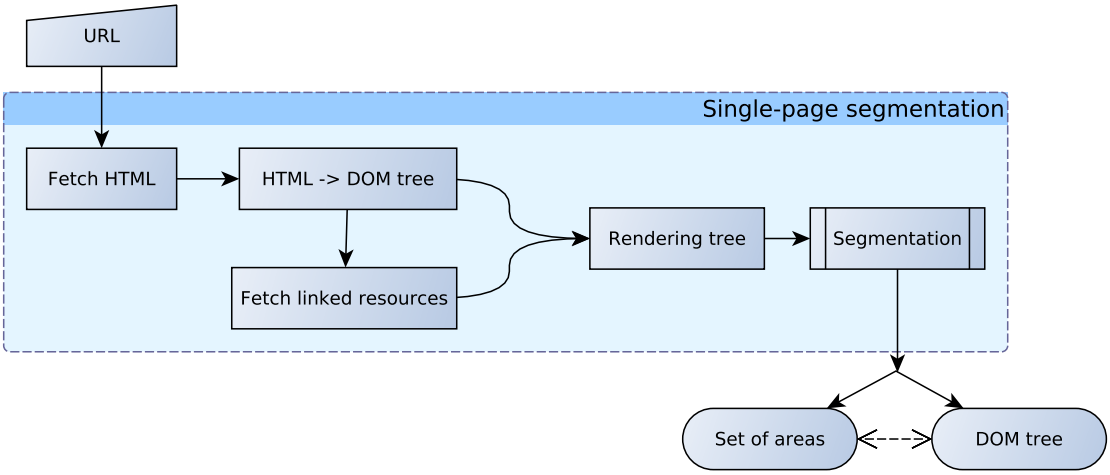

Figure 5.1: Use case: segmentation of a single web page

Multi-page segmentation: this is an extension of the previous use case. This one is based on an assumption that in practical applications users will often want to extract information from more than just one web page. Automated environment can be a really good example of utilizing this approach. While the actual implementation may differ, the proof of concept designed in this thesis is just a simple crawler mechanism. The intention of this mechanism is to process a certain number of web pages from one web site. Most likely this number will be unlimited, i.e. the intention will be to process all the web pages on the web site. If the number of processed web pages is limited, it is difficult to determine what pages exactly will be processed. The order in which the pages will be processed is also difficult to set or even determine upfront. Therefore the most likely consumers of this approach are automated applications which usually don't strictly require to know this information. This use case requires the following input: starting URL, domain name of the inspected site (if the consumers wants to limit processing to one site only) and the number of web pages that are to be processed. The use case, displayed in figure 5.2, can be described as follows:

1. A segmentation of one page is performed as described above
2. All hyperlinks are extracted from the DOM tree of the page and put at the end of a to-be-inspected queue
3. If the limit number of pages is reached, the algorithm ends
4. The first URL from the queue is taken and if it has not been inspected before, the segmentation starts from point 1.

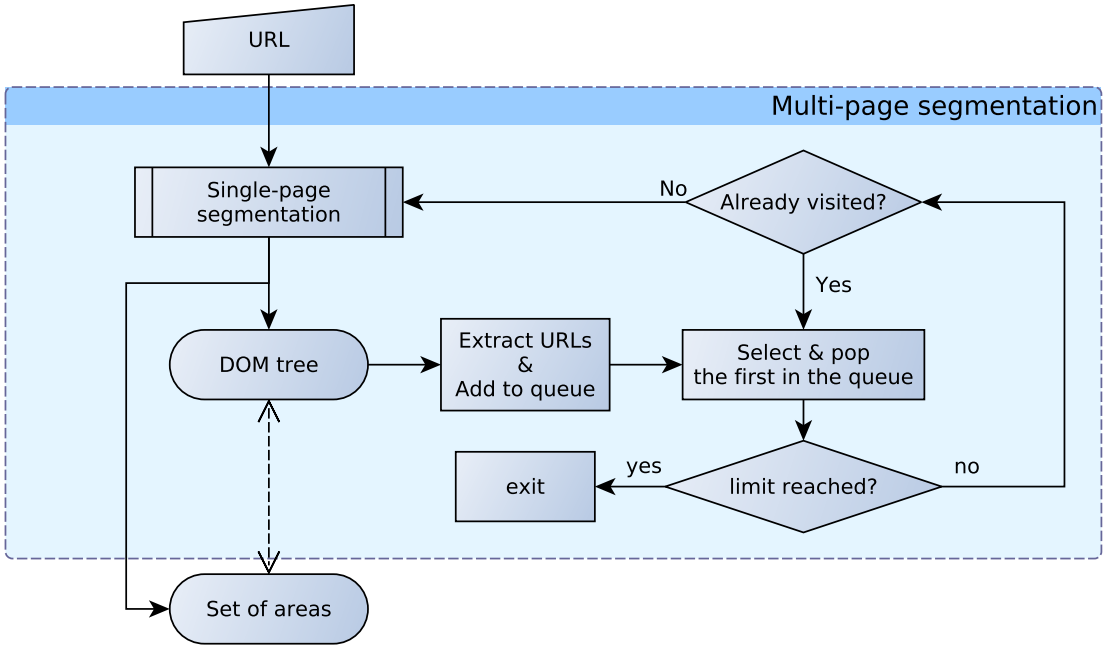

Figure 5.2: Use case: segmentation of multiple web pages in one run

5.1 High level design

By its design, the system is intuitively adapted to the two use cases described above. Graphical representation of the entire system is displayed in figure 5.3. The diagram shows high level view of the individual components and the data as they flow between them. The components with doubled side lines are the key parts of the system that will be closely described in the following chapters. This section will cover the rest of the components, as they are rather straightforward in the proof-of-concept implementation. Obviously the implementation of all these components can be more complex if the intended use of the system is different.

The very first component processing the data is *URL selector*. The trivial but sufficient implementation uses just a simple queue. Before the processing begins, the initial URL specified by user is put into the queue. The selector always selects the first URL in the queue and sends it to the next component for processing.

The next components in the pipeline are *URL fetcher* and *DOM parser*. Their job is to fetch the web page represented by URL and to transfer it to DOM tree respectively. There are many implementations of these components that handle most of the work. The only thing that has to be checked explicitly is content type, as the URL fetcher fetches any document from the Web. By definition, this document does not have to be a web page, therefore it's crucial to check for that and if the document is not a web page, it has to be skipped.

One aspect of the system that is not directly visible is the iterative nature of the internal functionality. This iterativeness is implied by the feedback of URL addresses via *URL extractor*. This component scans the DOM tree for any links to other documents it contains.

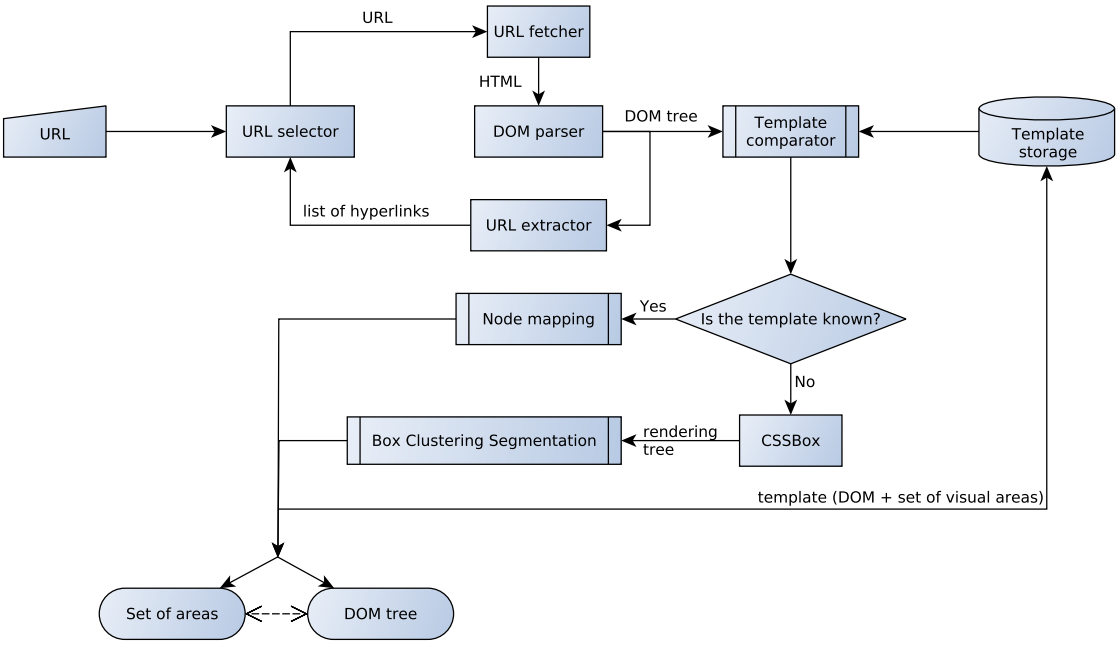

Figure 5.3: Architecture of the proposed segmentation system

Once the list is extracted, it is pushed to the end of the queue in URL selector and the addresses are therefore scheduled for inspection in the following iterations.

The last component that is not straightforward but is handled like a black box is the *CSSBox* rendering engine. In theory any rendering engine can be used, however the CSSBox has been designed from ground up to be used in segmentation algorithms and as a consequence it offers variety of information on the output that other engines don't. One component that poses to be simple one in the diagram is Template storage. In theory this can be any external database but there are some design requirements that have to be met. The complete design of the storage will be depicted in detail in chapter 7.

Chapter 6

Box Clustering Segmentation

The Box Clustering Segmentation (BCS) is one of the core pieces of the segmentation method presented in this thesis. From the perspective of functionality, it's the most important component. It is the only component that influences precision of the result and it has great effect on the performance as well. BCS is designed to be a pure vision-based method. Also, in contrast to other segmentation methods presented in chapter 3, BCS is designed to give flat results. That means, it produces a tiled arrangement of segments rather than their hierarchy, which is the usual layout of segmentation results.

The internal composition of the BCS process is displayed in figure 6.1. The algorithm takes rendering tree produced by CSSBox as an input. This rendering tree contains all the information about visual features of the web page but it also contains all the information that was stored in the original DOM tree, as it stores references to its individual DOM nodes. This, however, does not mean that the DOM tree is taken as input of the algorithm, just that all of its information was absorbed by the rendering tree in the most convenient way.

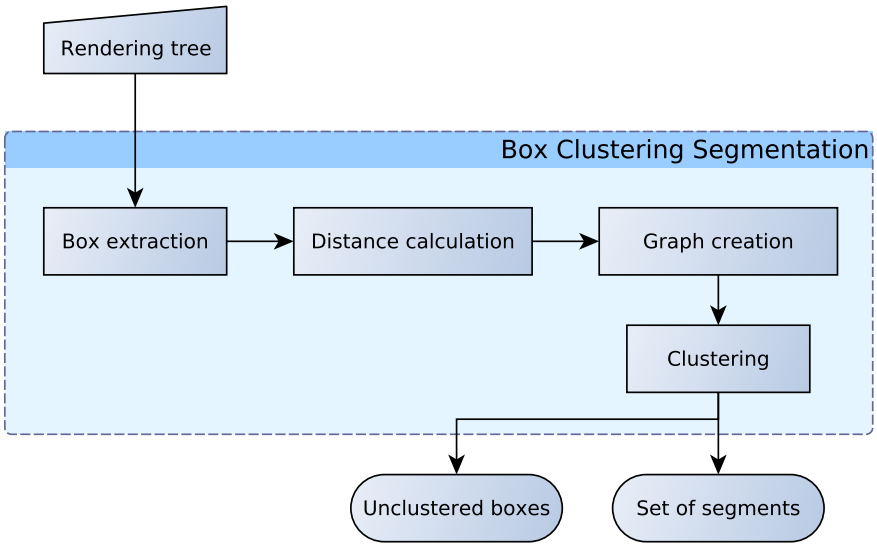

Figure 6.1: Internal composition of Box Clustering Segmentation

Even though the main part of the output data is designated *Set of segments*, it's important to realize that some boxes on the web page might not be a part of any segment in the set. Such boxes are assumed unimportant and it is therefore acceptable not to cluster them. However for further processing these boxes still might be important, that's why they are a part of the output as well. And while they are intentionally disregarded in the big picture displayed in figure 5.3, including them there is trivial.

The BCS algorithm works in several phases. As the name suggests, the algorithm aims to cluster the smallest visual elements on the web page into coherent groups. For this to work, the smallest visual elements have to be identified first, as they don't always correspond to leaves in the rendering tree. Then the distance between individual elements has to be calculated using criteria described in section 6.4. In further text, this distance metrics is also called *dissimilarity*, as the distance between two elements on the web page is determined by their visual dissimilarity. After the dissimilarity is calculated, it is used to create and then iteratively extend clusters.

6.1 Extracting boxes

The box extraction is the first step of the Box Clustering Segmentation. Even though it can be perceived just as a preprocessing, it is an important part of the BCS algorithm, as it influences both precision and performance.

In order to properly understand the preprocessing nature of the box extraction, it's important to understand how the rendering tree produced by CSSBox looks like. It is basically a tree of *boxes* where each box represents a rectangular area in the rendered page. Each box in the rendering tree corresponds to a particular node (or its part) in the input DOM tree; i.e. it is always possible to identify the source DOM node that generated a particular box. A reference to the corresponding DOM node is contained in the data structure representing the box. Cardinality of the relation is $1..N$, as one DOM node can generate multiple boxes. Elementary visual boxes that represent atomic units of the content are leaf nodes in the rendering tree. Lines of text and images are the most common examples of the atomic units of the page content. However the atomic units in general are not always convenient for clustering, as they can sometimes have slightly larger bounding boxes.

The box extraction algorithm traverses the rendering tree and selects boxes that should be used for further processing. All the boxes are transformed to simplified data structures during the selection process. In the following text, the designation *box* will be used for these simplified data structures. In the data model used by Box Clustering Segmentation, these simplified boxes are the basic building blocks of the segmented web page. Each box contains just minimal amount of information – only the data required for subsequent segmentation steps (the dissimilarity measurement in particular).

- Box position in the page
- Dimensions of the box
- The averaged color of the box

In the following sections, various properties of boxes will be referenced. Therefore a proper formal definition of the simplified box structure follows. Note that this thesis uses a coordinate model that is common in web design. The basic unit in that model is a pixel and coordinate system starts in the top left corner of the web page. When talking about

position, terms like top, down, above, below and others have their intuitive semantics, that is e.g. top edge means "top from user's perspective".

Definition 1 (Box structure) *Let the box m be defined as 7-tuple $m = (left, right, top, bottom, width, height, color)$ where $left$, right, top and bottom are integer values that represent positions of the respective edges of the box; $width = right - left$; $height = bottom - top$ and color is a composite value representing mean color of the box.*

Assuming the 7-tuple will be named, each element in that 7-tuple will be referred to by its name as well, separated from the . This can be illustrated by example: having box m defined, the pair representing its top-left corner would be defined as $(m.top, m.left)$.

The boxes are extracted from the rendering tree using a recursive algorithm performing a pre-order traversal of the rendering tree. The general idea is to consider the nodes of the tree that are actually visually accessible in the page, i.e. the user looking at the page can see them. This refers to the fact that many tricks are used on modern web pages to achieve the intended layout and some boxes on the web page might not actually be visible.

The rest of the content, i.e. the boxes that are visible, is usually represented by leaf nodes in the rendering tree (that represent the actual content of the page). However, as mentioned above, there are some exceptions from this rule. The following list provides a deeper look into the box extraction process:

1. **Text nodes** are always leaf nodes. They contain a line or its part. Graphically, each text box is a minimal bounding box of the text contained. These nodes are directly transformed to box areas.

2. **Image nodes** are always leaf nodes. They represent a particular image and therefore, they share its properties like position and size.

3. **Childless boxes** that don't fall into previous categories are omitted.

4. **One-child boxes** that don't fall into previous categories are viewed as subtrees rooted at the child box. These subtrees are then inspected for branches and if there are none, the smallest box in the subtree with non-transparent background is returned. If there is no such box, the leaf box is returned.

After the box extraction is completed, it is still possible that some extracted boxes will visually contain other boxes. Therefore, an additional detection and filtering is performed. In this additional step the extraction algorithm tries to construct a tree structure based on visual nesting of previously extracted boxes. If any any visual nesting is detected, only the nested boxes are kept, the ones containing them are filtered out.

The output of the box extraction algorithm after the filtering step can be described as a set of boxes where no two boxes overlap. The overlap is defined as follows:

Definition 2 (Box overlap and containment) *Two boxes m and n overlap if $m.right \geq n.left \wedge m.left \leq n.right \wedge m.bottom \geq n.top \wedge m.top \leq n.bottom$. Box m visually contains box n if $m.left \leq n.left \wedge m.right \geq n.right \wedge m.top \leq n.top \wedge m.bottom \geq n.bottom$*

Unless specified otherwise, the rest of chapter 6 assumes that boxes that are being worked with are non-overlapping.

6.1.1 Box color

While other values are straightforward to get, the box color has to be calculated during the extraction process. In the set of boxes B, there are three types of boxes from the perspective of what they represent: lines of text, images and other content (for example boxes with no content, their purpose being only visual on the web page). For the color component of the dissimilarity model, it is crucial to represent each box with the right color. Because one of the key features of the Box Clustering segmentation is its speed, the goal is to pick just one – visually the most significant – color per box.

Boxes with no content have the most simple color representation, as their background color is picked (or the background color of their parent in the rendering tree if they are transparent). For every image the average color is calculated. All the pixels in the image are iterated over and the mean values of the red, blue and green components put together the average value. If the image is not using RGB color model, proper transformation is performed first.

For text, the situation is much more complex. The color of the text box has to take various circumstances into account, for example background color, font weight and slant and text decorations. The rules below are original to the BCS and were specifically designed to accommodate all the different combinations of font decorations that can be specified in HTML and CSS. HSV color model is used, as it simply allows modifications of the color brightness. When calculating the text color, the following rules are evaluated (in the specified order):

1. The specified color of the text is taken as the starting value that is modified in the following steps
2. If the text is white, set hue to the hue of background, saturation to a value > 0. Saturation is selected as the component to be modified (decreased) in the following steps.
3. If the text is gray or black, brightness is selected to be modified (decreased) in the following steps.
4. If the text is black, set brightness to value as small as possible but > 0.
5. If the text is colored, saturation is selected as the component to be modified (increased) in the following steps.
6. If the text is decorated, modify the selected color component by 20%.
7. If the text is italic, modify the selected color component by 20%.
8. If the text is bold, modify the selected color component by 30%.
9. If the selected color component is out of range $< 0, 1 >$, set it to the nearest value in that range.

Once calculated, only the color value will be carried by the box 7-tuple, no additional information is included.

6.2 Connecting the boxes

The previous step produces a set of boxes which are atomic elements of visual content of the web page. The goal of connection-creating algorithm is to iterate over this set and for each box to identify other boxes that will be preferentially evaluated for clustering with that box.

This part of the segmentation process is crucial for performance of the entire algorithm, as the number of connections between boxes strongly influences the computational time of the clustering step. A naive approach would be to create a connection between every two boxes. However, with thousands of boxes being on a web page (which is not uncommon), the size of the box set increases so much it raises time and space demands of the algorithm beyond practical usability. It is therefore desirable to optimize the number of connections as much as possible.

To understand the selected optimization, it is first important to understand what is called projected overlap. Projected overlap can be best described as overlap of boxes in one-dimensional space, i.e. when the boxes were projected to individual axes.

Definition 3 (Projected overlap and semi-alignment) *Let m and n be two boxes on a web page. The projected overlap of boxes m and n is defined as a function $pov : (m, n) \to \{x, y, o\}$ where x and y indicate projected overlap on the respective coordinate axes of the web page and o designates "no projected overlap". The following rules apply:*

$$pov(m,n) = \begin{cases} x & if \quad m.right \geq n.left \wedge m.left \leq n.right \\ y & if \quad m.bottom \geq n.top \wedge m.top \leq n.bottom \\ o & \quad otherwise \end{cases} \tag{6.1}$$

The two boxes m and n are in semi-alignment if $pov(m, n) \neq o$.

There is also finer grained version of projected overlap that is used when determining the distance between boxes. Is is called mutual position.

Definition 4 (Mutual position of two boxes) *Let a set of possible positions be $P = \{a, b, l, r, o\}$ where a, b, l, r designate position above, below, left and right respectively and o designates "other position". The position of box m relative to box n is defined as a function $pos : (m, n) \to P$ where the following rules apply:*

$$pos(m,n) = \begin{cases} a & if \quad m.bottom \leq n.top \wedge pov(m,n) = x \\ b & if \quad m.top \geq n.bottom \wedge pov(m,n) = x \\ l & if \quad m.right \leq n.left \wedge pov(m,n) = y \\ r & if \quad m.left \geq n.right \wedge pov(m,n) = y \\ o & \quad otherwise \end{cases} \tag{6.2}$$

Note that as per section 6.1, overlapping boxes are not allowed in the model, thus all the conditions of $pov(m, n)$ are mutually exclusive. The same applies to $pos(m, n)$.

Figure 6.2 graphically depicts semi alignment of boxes. The dark blue box is being inspected, the light blue boxes are semi-aligned with it while the red boxes are not.

The chosen optimization of the connection-creating algorithm is to connect only boxes that are a) adjoined and b) semi-aligned. There are two reasons why this representation was chosen:

- It's easier to extract directly adjoined boxes during subsequent processing.

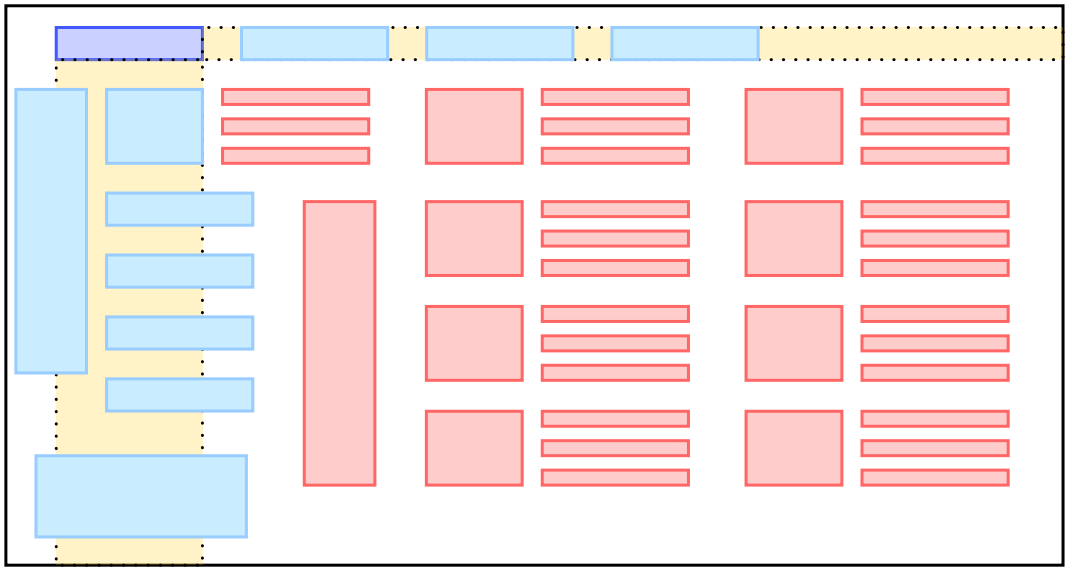

Figure 6.2: Boxes that are selected to be connected

- Experimental observation showed that visually related boxes are always organized like this

6.3 Composite Dissimilarity Model

A proper model for evaluating the dissimilarity between the elements on the web page, that will be described more closely in this section, is one of the main pillars of any segmentation algorithm. There are two types of entities featuring in the dissimilarity model – *boxes* and their *clusters*. The relation between these two types is denoted in figure 6.3.

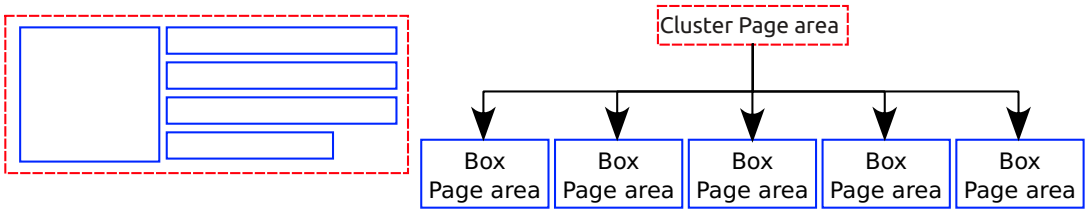

Figure 6.3: Hierarchy of visual areas representing boxes and their clusters

The model is called composite because it contains two main dissimilarity components, based on the type of entities between which the dissimilarity value is calculated. The first compound, called *base dissimilarity*, is based on visual features of individual boxes and is therefore defined only between two boxes. On the other hand, the second one, called *cluster*

dissimilarity, is used to express the dissimilarity when at least one of the two entities is a cluster.

Generally, the value of dissimilarity and/or its compounds is floating-point number in the range of $< 0, 1 >$ with straight proportionality semantics, i.e. the higher the value gets, the higher the dissimilarity is.

6.4 Base Dissimilarity

Base dissimilarity can be theoretically calculated between any two boxes on a web page. But to follow the optimization rule established in section 6.2, it is calculated only between those pairs of boxes that are connected.

The model of base dissimilarity can be also perceived as a composite one but from a different perspective. It does not combine multiple measurement methods but it measures three different visual aspects of the dissimilarity between two boxes – *relative distance* between the boxes, their *shape* dissimilarity and the dissimilarity of their *colors*. Even though *alignment score* is taken into account as well, it can't be considered related strictly to the dissimilarity between two boxes. All these components will be detailed further in this section. The final value of base dissimilarity, designated as $bdis(m, n)$ in the previous text is calculated as follows:

$$bdis(m,n) = \begin{cases} 0 & \text{if } distance(m,n) = 0 \\ 1 & \text{if } distance(m,n) = 1 \\ \dfrac{\left(\begin{array}{l} distance(m,n) + \\ dis\_shape(m,n) + \\ dis\_color(m,n) \end{array} \right)}{3 * alignment\_score(m,n)} & \text{otherwise} \end{cases} \tag{6.3}$$

There are two reasons why only three visual aspects of boxes are used: 1) all the three aspects are defined for all the boxes on every web page that exists and 2) it makes the dissimilarity model more simple. That is good for the performance of the algorithm. On the other hand it opens the possibility to argue that the precision of the algorithm might be affected. That argument might be true to a certain degree however, against the expectations, not having overly sophisticated dissimilarity model made it possible to fine tune the clustering step significantly. If a research is conducted in the future that finds other aspects that can be included in the dissimilarity model, it will be very cheap improvement of the Box Clustering Segmentation precision, when compared to improvements of the clustering algorithm

6.4.1 Relative Distance

Distance calculation is based on obvious fact that the closer two boxes are, the higher is the likelihood of them belonging to the same cluster. The distance measurement is based on relative distances between adjacent boxes. To calculate relative distance, it is first necessary to know absolute distances. This starts with identifying what we call a *direct neighborhood* of each box. The direct neighborhood of box X includes just those boxes that are closest to it in terms of absolute distance. The way how the absolute distances are computed is graphically expressed in figure 6.4.

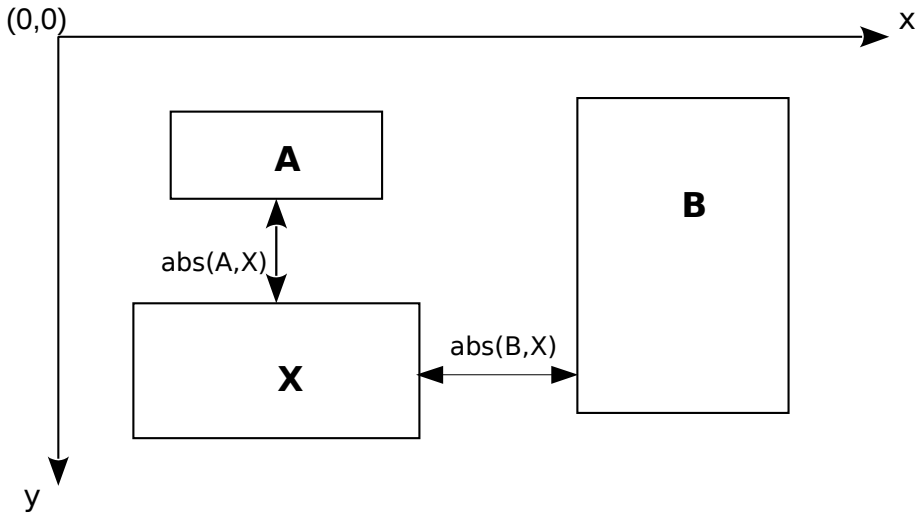

Figure 6.4: Absolute distance measurement between boxes

Definition 5 (Absolute Distance and Direct Neighborhood of a Box) *Let B be a set of boxes on a web page and let $m, n \in B$. Absolute distance between two boxes is a function $abs : B \times B \to \mathbb{R}$:*

$$abs(m,n) = \begin{cases} n.top - m.bottom & \text{if } pos(m,n) = a \\ m.top - n.bottom & \text{if } pos(m,n) = b \\ n.left - m.right & \text{if } pos(m,n) = l \\ m.left - n.right & \text{if } pos(m,n) = r \\ \infty & \text{otherwise} \end{cases} \tag{6.4}$$

.

Direct neighborhood of box m is defined as $N_m = \{n | n \in B \wedge pos(m,n) \neq o \wedge \nexists k \in B : (pos(m,k) = pos(m,n) \wedge abs(m,k) < abs(m,n))\}$

Using the direct neighborhoods and absolute distances, it is possible to calculate the relative distance, denoted *distance*():

Definition 6 (Relative distance) *Let B be a set of all boxes on a web page. For each box $m \in B$ and its direct neighborhood N_m, there is a maximal neighborhood distance $maxd(m) = abs(m,k)$ where $k \in N_m \wedge \nexists l \in N_m : abs(m,l) > abs(m,k)$. For each $n \in N_m$ the relative distance, designated $distance(m,n)$, is calculated as:*

$$distance(m,n) = \frac{\frac{abs(m,n)}{maxd(m)} + \frac{abs(m,n)}{maxd(n)}}{2} \tag{6.5}$$

With this definition, the relative distance represented by the function *distance*() expresses how far the two boxes are from each other in the context of their direct neighborhoods. There are two things to be noted here:

1. The connections between boxes, as described in section 6.2, have formal equivalent in the set of all direct neighborhoods

2. The semantics of the distance value is exactly the same as semantics of the dissimilarity values. It can assume values in the range $< 0, 1 >$ with direct proportionality.

6.4.2 Shape

While the reason for distance measurement is obvious, shape comparison might not be at first. The premise here is that two adjacent boxes that have similar shape are more likely to belong to the same cluster than two boxes with completely different shapes. The most common case where the shape dissimilarity plays significant role are menu items, especially in vertical menus. Another very common situation is demonstrated by a simple paragraph of text where each line creates a separate box and all such boxes have very similar shape.

To properly evaluate shape dissimilarity, it's important to pay attention to two visual features – aspect ratio and surface of the two boxes. Let's have two boxes $m, n \in B$. The following equations define how the aspect ratio dissimilarity is calculated. There is a condition for the ratio dissimilarity to have the same semantics as other dissimilarities. This condition determines how the function $ratio()$ should look like. Using simple absolute value of difference between the two ratios is not possible, as it can result to values higher than 1. To get these values into the range, it has to be divided by the maximal possible value this difference can result in. In the following equations, variables r_m and r_n represent aspect ratios of the respective boxes whereas $ratio(m, n)$ is a formulation of the ratio dissimilarity. Note that while each aspect ratio is expressed as ratio of width to height, having it vice versa does not change the result.

$$r_m = \frac{m.width}{m.height} \tag{6.6}$$

$$r_n = \frac{n.width}{n.height} \tag{6.7}$$

$$ratio(m,n) = \frac{max\{r_m, r_n\} - min\{r_m, r_n\}}{\frac{max\{r_m, r_n\}^2 - 1}{max\{r_m, r_n\}}} \tag{6.8}$$

The second part of shape dissimilarity measurement is the size dissimilarity. Symbols used in the following equations correspond to the symbols in the previous ones: s_m and s_n represent surface size of the respective boxes and $size(m, n)$ expresses the size dissimilarity itself. The same condition that determines the design of the ratio function also determines the design of the $size()$ function. Knowing this condition, the design of the function is straightforward.

$$s_m = m.width * m.height \tag{6.9}$$

$$s_n = n.width * n.height \tag{6.10}$$

$$size(m,n) = 1 - \frac{min\{s_m, s_n\}}{max\{s_m, s_n\}} \tag{6.11}$$

The final shape dissimilarity is simply calculated as a mean value of $ratio$ and $size$:

$$dis\_shape(m,n) = \frac{ratio(m,n) + size(m,n)}{2} \tag{6.12}$$

6.4.3 Color

Color comparison is again an obvious one. Boxes with the same color are likely to belong to the same cluster than boxes with completely different colors. The process of assigning

each box a color is described in section 6.1.1. Color dissimilarity of two boxes is based on color distance calculation. The color distance is commonly used term in computer graphics. It is a metrics used to quantify the difference between two colors. It is commonly denoted as ΔE and there are many formulas to calculate it. The actual choice of the most suitable formula depends on the application[47].

The International Commission on Illumination came up with several formulas that work on *Lab* and *LCH* color spaces. They are all based on the fact that the human eye is more sensitive to changes in chroma than to changes in lightness[47]. As opposed to *RGB* color space, both *Lab* and *LCH* color spaces allow a separate calculation for lightness and chroma. More recent formulas[7, 38] compensate for perceptional non-uniformities in some areas of the color space.

Both *Lab* and *LCH* based color distances were evaluated for their potential utilization in the BCS algorithm. However an observation has been made during the experiments that simple euclidean *RGB*-based difference offers much better results. The most likely cause for this observation is that on most web pages, hue is used much more often than chroma to visually distinguish components on the web page that belong to different semantic blocks of the web page (e.g. navigation vs. article heading).

Because the results of the color distance have to match all the other components of the box comparison, a normalization has to be carried out. This is done by dividing the euclidean distance by the maximal diagonal distance in the *RGB* color space. In the standard *RGB* model which is used here, each color channel can assume values in the range $< 0, 1 >$, thus the maximal diagonal distance is 1.732. Using $C_m = (R_m, G_m, B_m)$ and $C_n = (R_n, G_n, B_n)$ as the color representations of boxes m and n respectively, the formula to calculate the color distance is the following:

$$dis\_color = \frac{\sqrt{(R_n - R_m)^2 + (G_n - G_m)^2 + (B_n - B_m)^2}}{1.732} \tag{6.13}$$

6.4.4 Alignment

Alignment is not strictly one of the components of the dissimilarity model, as it is not a visual aspect of either the boxes or their dissimilarity. It is however used as adjusting value that is applied as the last step of dissimilarity calculation. From the more formal perspective, alignment should be perceived as a way of scoring the dissimilarity between the two boxes in the context of their surroundings. The rationale to use this adjustment came from the observation how VIPS algorithm works – thanks to the top-down approach, the VIPS algorithm implicitly puts stronger separators around content that is aligned. Because clustering algorithms by design don't support such behavior, it has to be emulated, as the final result is very desirable.

Alignment is represented by an integer value that specifies how many boxes are aligned with the two that are being inspected. Obviously for this integer to be higher than one, the two inspected boxes need to be aligned themselves (either their top or left edges need to have the same coordinate). Formally, the value can be described by definition

Definition 7 (Alignment score) *Let B be the set of boxes on a web page and let $m, n \in B; m \neq n$ be two boxes. The alignment score is a function $B \times B \to \mathbb{N}$ that is defined as:*

$$alignment\_score(m,n) = \begin{cases} |\{b|b \in B \wedge b.left = m.left\}| & if\ m.left = n.left \\ |\{b|b \in B \wedge b.top = m.top\}| & if\ m.top = n.top \\ 1 & otherwise \end{cases} \quad (6.14)$$

6.5 Cluster Dissimilarity

Aside from the Base Dissimilarity, Cluster Dissimilarity is the second component of the compound dissimilarity model used in BCS. It is used only during the clustering step of the algorithm. It is different from the base dissimilarity because clusters can't inherit features of the boxes they contain. The only option how to do that reliably would be to determine how exactly individual boxes contribute to the appearance of the entire cluster. However this task would be too time consuming for the algorithm to be practically usable.

Cluster dissimilarity is a model that works solely with dissimilarity values and connections calculated in the previous steps. Before the cluster dissimilarity model can be explained, the structure of a cluster has to be defined.

Definition 8 (Cluster) *Let B be a set of all boxes on a web page. Cluster c is a designation of a set of boxes $B_c \subseteq B$.*

Cluster dissimilarity model is demonstrated in figure 6.5. As a prerequisite of this model, the definition of direct neighborhood must be extended so it can accommodate both clusters and boxes. The model derives direct neighborhood of each cluster from direct neighborhoods of all the boxes contained in that cluster. Informally, it is possible to say that direct neighborhood of cluster c includes boxes that are in direct neighborhood of any of the boxes within the cluster c and all clusters containing at least one box that is in direct neighborhood of any box within the cluster c.

Definition 9 (Unclustered Boxes and Direct Neighborhood of a Cluster) *Let B and C respectively be sets of boxes and clusters on a web page. Furthermore, let $B_c = c$ be a cluster, $m \in B$ be a box and N_m its direct neighborhood. A set of unclustered boxes on the page is defined as $B_U = \{b|b \in B; \nexists c \in C : (b \in B_c)\}$.*

For cluster c, its direct neighborhood is defined as $N_c = \{m|m \in B_U \wedge \exists n \in B_c : n \in N_m\} \cup \{d|d \in C \wedge \exists m \in B_c, n \in B_d : n \in N_m\}$.

For dissimilarity, the situation is similar – the value of dissimilarity between a cluster and any entity in its direct neighborhood represents the mean value of dissimilarities between that entity and all the boxes contained in the cluster. In figure 6.5, the box-cluster connection `X` represents connections `A`, `B` and `C` when cluster `c` is created. The dissimilarity assigned to `X` therefore is derived from their respective dissimilarities. The optimal way how to derive the value would be via some weights based on the visual importance of each box within the cluster. However, that would be too time-consuming so a simple mean value is used instead.

The previous text only uses connections as simple pairs of boxes that are adjacent where each such pair has a corresponding dissimilarity value defined by the *cdis* function. However, the concepts of connection cardinality and cumulative dissimilarity introduce additional functions *card*() and *cumul*() which are defined as follows:

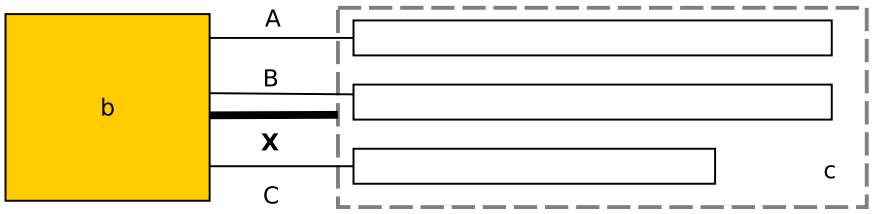

Figure 6.5: Demonstration of cluster dissimilarity model

Definition 10 (Connection cardinality and cumulative dissimilarity) *Let B, B_U and C respectively be sets of boxes, unclustered boxes and clusters on a web page. Also, let N_e designate direct neighborhood of entity e. Functions $card : C \times C \cup B_U \to \mathbb{N}$ and $cumul : C \times C \cup B_U \to \mathbb{R}$, which respectively represent connection cardinality and cumulative dissimilarity, are defined as:*

$$card(c, e) = |\{m | m \in B_c \wedge e \in N_m\}| \tag{6.15}$$

$$cumul(c, e) = \sum_{\forall m \in B_c} d(m, e) \tag{6.16}$$

Note that function $d()$ is described in definition 11. When a cluster is being created, all the unclustered boxes are scanned and those that are about to become cluster neighbors are selected for processing. For each of these future neighbors the value of cumulative dissimilarity and cardinality is calculated and the final dissimilarity *cdis* is then calculated as their quotient:

$$cdis(c, b) = \frac{cumul(c, b)}{card(c, b)} \tag{6.17}$$

6.6 Entity Dissimilarity

Now when both compounds of the dissimilarity model are formally defined, it's also possible to define how are they combined. Section 6.3 offers informal description. Formally, the value of dissimilarity when both compounds are combined is called *Entity dissimilarity* and it's defined as follows:

Definition 11 (Entity dissimilarity) *Let B and C respectively be sets of boxes and clusters on a web page and let $e_1, e_2 \in B \cup C$ be two entities. The entity dissimilarity d is defined as:*

$$d(e_1, e_2) = \begin{cases} bdis(e_1, e_2) & \text{if } e_1 \in B \wedge e_2 \in B \\ cdis(e_1, e_2) & \text{if } e_1 \in C \\ cdis(e_2, e_1) & \text{if } e_1 \notin C \wedge e_2 \in C \end{cases} \tag{6.18}$$

6.7 Box Clustering

The clustering algorithm implements bottom-up hierarchical clustering where the depth of the hierarchy is limited to two levels – clusters at the top level and boxes at the bottom level.

Definition 12 (Goal of the algorithm) *Goal of the algorithm is to find such a cluster set C that will minimize the set of unclustered boxes B_U where the following rule applies:* $\nexists e_1, e_2 \in C \cup B_U : (d(e_1, e_2) \leq CT)$.

The algorithm is iterative, exactly one new cluster is created by merging two existing entities in every iteration. Several steps take place for that to happen:

1. Selection of the most similar pair of entities
2. Mergeability testing
3. Candidate creation
4. Overlap testing and cluster extending
5. Cluster verification and commission

All of these steps will be closely analyzed in the following text and then the algorithm will be described in detail.

Selection of the most similar pair of entities

Selection of the most similar pair of entities is straightforward, as all the information required in directly stored in the graph of entities.

However, the component handling the selection has one important non-trivial task: to stop the clustering algorithm when appropriate. Obviously, the algorithm will stop once the set of relations is empty. However that's not the most common case. In order to set a stopping point, a border has to be established that the algorithm will not cross. In case of BCS, that border is represented by a *Clustering Threshold*, designated also as CT. The CT is a numeric value that can assume values between 0 and 1. Its purpose corresponds to that of Permitted Degree of Coherence (PDoC) used in VIPS algorithm[13]. It has to be set in advance and it stays the same for the entire page. Picking the right value of CT is a difficult task and every web page has a different optimal value. If it's too low, many boxes will end up unclustered. On the other hand, if picked too high, some clusters that should be separate are merged instead. Compared to VIPS, there is also another consequence: The results can also look completely different with different values of CT. That's caused by the phase of overlap testing and cluster extending.

Selecting the right value of CT for the web page is application specific so deeper analysis is out of scope of this thesis, much like selecting PDoC is out of scope of the VIPS algorithm. In practical applications, several approaches can be taken, for example an iterative or bi-sective approach with the number of unclustered boxes being used as an indicator when to stop. This solution is acceptable in the use case of batch processing, as the template clustering step will compensate the performance impact.

Mergeability testing

Mergeability testing is important especially in later iterations of the clustering algorithm. The idea of this testing is to prevent merging clusters that should not be merged because of their visual inconsistencies. These inconsistencies are based on the shape of clusters, their density and their disjunction. Before describing the individual constituents, it's important to establish a few auxiliary definitions.

Minimal bounding rectangle of each cluster is important for cluster density but it's also convenient for detecting overlap between the cluster and other entities on the web page.

Definition 13 (Minimal Bounding Rectangle) *Let B_U and C respectively be a set of unclustered boxes and a set of clusters on a web page. Furthermore, let $c \in C$ be a cluster. In this subset, there is one or more boundary defining boxes: the topmost box $b_t \in B_c; \nexists b \in B_c : (b.top < b_t.top)$ establishes top boundary of the set, the bottommost box $b_b \in B_c; \nexists b \in B_c : (b.bottom > b_b.bottom)$ establishes the bottom boundary of the set, the leftmost box $b_l \in B_c; \nexists b \in B_c : (b.left < b_l.left)$ establishes the left boundary of the set and the rightmost box $b_r \in B_c; \nexists b \in B_c : (b.right > b_r.right)$ establishes the right boundary of the set.*

Minimal bounding rectangle function $MBR : B_U \cup C \to \mathbb{N}^6$ produces six-tuples $(left, right, top, bottom, width, height)$ where individual components of $MBR(e)$, $e \in B_U \cup C$ are defined as:

$$left = \begin{cases} b_l.left & \text{if } e \in C \\ e.left & \text{if } e \in B_U \end{cases} \tag{6.19}$$

$$right = \begin{cases} b_r.right & \text{if } e \in C \\ e.right & \text{if } e \in B_U \end{cases} \tag{6.20}$$

$$top = \begin{cases} b_t.top & \text{if } e \in C \\ e.top & \text{if } e \in B_U \end{cases} \tag{6.21}$$

$$bottom = \begin{cases} b_b.bottom & \text{if } e \in C \\ e.bottom & \text{if } e \in B_U \end{cases} \tag{6.22}$$

$$width = right - left \tag{6.23}$$

$$height = bottom - top \tag{6.24}$$

$$\tag{6.25}$$

Semi alignment cardinality expresses how many semi-alignments are there between boxes within a cluster and their direct neighbors that are also within the cluster.

Definition 14 (Semi-alignment Cardinality) *Let $c = B_c$ be a cluster and let N_m designate direct neighborhood of box m. Cardinality of horizontal and vertical semi-alignment relations within cluster c is respectively defined as:*

$$hc(c) = |\{\{m, n\} | m, n \in B_c \wedge n \in N_m \wedge pov(m, n) = y\}| \tag{6.26}$$

$$vc(c) = |\{\{m, n\} | m, n \in B_c \wedge n \in N_m \wedge pov(m, n) = x\}| \tag{6.27}$$

Cluster shape is the first indicator evaluated when testing for mergeability. It is defined differently than the box shape as described in section 6.4.2.

Definition 15 (Cluster shape) *Let C be a set of clusters and let $c \in C$ be a cluster. The function $cshape : C \to \{col, row, blob\}$ which describes the cluster shape is defined as*

follows.

$$cshape(c) = \begin{cases} col & \text{if } hc(c) < 0.5 \cdot vc(c) \\ row & \text{if } hc(c) > 2 \cdot vc(c) \\ blob & otherwise \end{cases} \tag{6.28}$$

The purpose of the cluster shape is to identify columns and rows in a web page. Informally, each row in the web page consists of a number of boxes that are placed side by side. Similarly, each column in the page consists of a number of boxes being placed one below another. While the Definition 15 partially covers this, it only says which connection direction (horizontal or vertical) prevails between the boxes within the cluster. It does not say anything about the number of boxes that are aligned in the prevailing direction. This information is covered by cluster density.

Definition 16 (Cluster density) *Let C be a set of clusters and let $c \in C$ be a cluster. The function $cdens : C \to \mathbb{R}$ which describes the cluster density is defined as follows.*

$$cdens(c) = \begin{cases} \frac{hc(c)}{MBR_c.width} & \text{if } cshape(c) = row \\ \frac{vc(c)}{MBR_c.height} & \text{if } cshape(c) = col \\ undefined & otherwise \end{cases} \tag{6.29}$$

The last indicator used in the process of merge testing is cluster disjunction, which is defined in Definition 17. This definition itself is quite straightforward but it first requires entity overlap to be set. With the definition of cluster corresponding to the definition of box, it is possible to use Definition 2 for both boxes and clusters. Now back to the cluster disjunction: simply put, the cluster created by merging the two entities must not overlap with any other entity on the web page.

Definition 17 (Entity overlap and Cluster disjunction) *Let B_U and C respectively be set of unclustered boxes and a set of clusters on a web page. Furthermore, let $c \in C$ be a cluster and let $e_1, e_2 \in B_U \cup C$ be two entities.*

Entities e_1 and e_2 overlap if $MBR(e_1).right \geq MBR(e_2).left \wedge MBR(e_1).left \leq MBR(e_2).right \wedge MBR(e_1).bottom \geq MBR(e_2).top \wedge MBR(e_1).top \leq MBR(e_2).bottom$. The cluster c is disjunct if $\nexists e \in B_U \cup (C - c) : (e \text{ overlaps with } c)$.

The merge test of two entities contains the following set of conditions.

- If the two entities are y-projection-overlapped columns and their density ratio is not in the interval of $< 0.5, 2 >$, forbid the merge
- If the two entities are x-projection-overlapped rows and their density ratio is not in the interval of $< 0.5, 2 >$, forbid the merge
- If the shapes of the two entities differ and cluster created by their merge is not disjunct, forbid the merge
- Otherwise allow the merge

The idea behind the interval $< 0.5, 2 >$ can be best demonstrated on a part of a web page displayed in figure 6.6. A human observer recognizes there are two columns of content on this piece of page. Assuming the clustering algorithm has already clustered both columns and is now evaluating their potential merge, it's desirable to stop it as each of the columns

is likely to contain a different type of information. In this case, the first rule applies and the interval specifies to stop the merge if either of the columns has twice as many rows as the other one. While other values can be considered as borders of the interval, this one was empirically verified to produce good results.

Figure 6.6: Columns on the web page

Candidate creation

During the candidate creation process a temporary cluster is created that has all the relevant features as the real cluster would have but is not yet committed, i.e. it's not a part of set of all the clusters on the web page C.

The reason to take this intermediate step is that the overlap testing can potentially forbid the creation of the cluster. When such case happens, the algorithm needs to have an option to revert back. So in essence the candidate cluster exists only so that all the features that are required for extending the cluster during the overlap detection are present. These features include:

- position on the page
- dimensions
- references to all the boxes that would be contained within the cluster

There is no need for formal definition of candidate cluster, as the cluster definition accommodates all the features. Note that if either of the two entities that are being processed in the currently running iteration is a cluster, it's boxes are considered instead of the cluster itself. This is crucial for the flat output of BCS.

Overlap testing and cluster extending

This is perhaps the most complex part of the clustering algorithm. The point of this part of the algorithm is to adjust clusters so they don't overlap with other clusters and/or boxes that are not part of the cluster. If the candidate cluster cannot be adjusted to match the conditions, its creation is prevented and the clustering algorithm continues by starting another iteration (selection of the most similar pair of entities, ...).

The Algorithm 1 shows the process of iterative overlap testing and cluster extending. It only needs a set of boxes B, a set of clusters C, the current candidate cluster cc and the two entities m, n that were merged into the candidate cluster. The algorithm returns true and updates the cluster candidate cc if it can be extended. Otherwise it returns false.

Algorithm 1 Overlap testing and cluster extending

```
1:  function extendAndTest(IN: B, C, m, n, INOUT: cc)
2:      repeat
3:          B_O ← {b|b ∈ B ∧ b ∉ B_cc ∧ b overlaps cc}
4:          if ∃k ∈ C − {m, n} : B_k ∩ B_O ≠ ∅ then
5:              return false
6:          end if
7:          if ∃b ∈ B_O : (∄b_cc ∈ B_cc : (b overlaps b_cc)) then
8:              return false
9:          end if
10:         B_cc ← B_cc ∪ B_O
11:     until B_0 = ∅
12:     return true
13: end function
```

Cluster verification and commission

Because the candidate cluster was used in the previous steps, it has to be transformed to a standard cluster that can be used in further iterations of the clustering algorithm. Several partial actions have to be carried out during this operation. From the formal perspective, the only interesting one is updating the set of clusters C on the web page. If either of the two original entities m and n was a cluster, it must be removed and the cluster candidate cc must be added:

$$C = (C - \{m, n\}) \cup \{cc\} \tag{6.30}$$

BCS main loop

The main loop reflects the description throughout the entire chapter. The input of the algorithm is the threshold value CT and a set of boxes B.

Algorithm 2 BCS main loop

```
1: function BCS(IN: B, CT) C ← ∅
2:     loop
3:         if |B_U| < 2 then
4:             return C
5:         end if
6:         m, n ← m, n ∈ B_U ∪ C : ∄x, y ∈ B_U ∪ C : (d(x, y) < d(m, n))
7:         if d(e) > CT then
8:             return C
9:         end if
10:        ratio = cdens(m) / cdens(n)
11:        if cshape(m) = cshape(n) then
12:            if cshape(m) = column then
13:                if pov(m, n) = y ∧ ¬(0.5 ≤ ratio ≤ 2) then
14:                    continue
15:                end if
16:            else if cshape(m) = row then
17:                if pov(m, n) = x ∧ ¬(0.5 ≤ ratio ≤ 2) then
18:                    continue
19:                end if
20:            end if
21:        else
22:            if m ∪ n is not disjunct then
23:                continue
24:            end if
25:        end if
26:        create cluster candidate cc
27:        if EXTENDANDTEST((B, C, cc, m, n)) then
28:            continue
29:        end if
30:        COMMITCLUSTER((B, C, m, n, cc))
31:    end loop
32: end function
```

Chapter 7

Template clustering

The previous chapter described the core of the segmentation algorithm that is used in both use cases specified in chapter 5. In figure 5.3, it was designated Box Clustering Segmentation. This chapter will focus on the rest of the non-trivial components that were not covered in previous chapters. In figure 5.3, these are designated *Template storage*, *Template comparator* and *Node mapping*.

These components are used mostly in the multi-page segmentation scenario displayed in figure 5.2. Because all the components are fixed parts of the architecture and their functionality is the same in both segmentation scenarios, they affect the single-page segmentation scenario displayed in figure 5.1 as well – for example when performing segmentation repeatedly on similar pages after some time (assuming the Template storage is permanent). However the multi-page segmentation use case is still the primary target of the template clustering, therefore this chapter will focus on in.

7.1 Template clustering overview

First let's emphasize focus of the crawler algorithm as stated in chapter 5. The focus of the algorithm is to process one site at a time. If there are any link leading to other sites, they should be scheduled for later inspection and revisited only after processing of the current site is finished. Even though this scenario is considerably specific, the template clustering algorithm is generic and can be used in virtually any scenario or crawler design. However, this specific scenario perfectly demonstrates potential of the template clustering and that's why the following text will assume it if not specified otherwise.

In the multi-page segmentation scenario, the standard approach is to segment every web page separately before sending it to further processing. For some large servers like world-wide news servers, this means performing the segmentation task hundreds of thousands of times. Even though this example might be extreme, it's obvious that this approach does not scale at all and the time required to process even mid-sized web sites is unacceptably long. The template clustering algorithm is designed specifically to address this scaling issue.

The general idea of the template clustering can be presented as follows. When processing more pages within the same site, it is possible to indirectly increase the performance of segmentation by actually performing it only on a limited number of pages and transform these pages so they can represent their respective template-based clusters. When, in one of the following iterations of the multi-page segmentation scenario, the inspected page is 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.

One of the advantages of this approach is that the optimization achieved is in direct correlation with the number of web pages processed – the more pages are processed, the greater the optimization actually is (measured in both relative and absolute scale). Another advantage is that the template clustering algorithm does not require all the pages to be processed at once. Instead, partial data set can be retrieved, the process interrupted and continued any time later. That is also why this approach works on the single-page segmentation scenario.

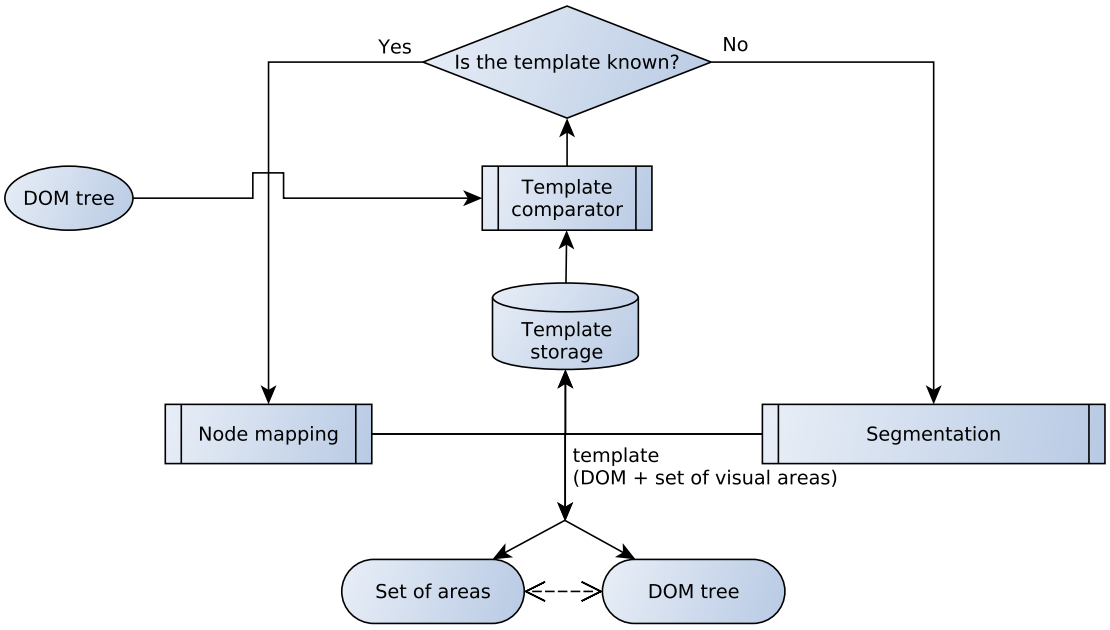

Figure 7.1: Block schema of the template clustering

The high-level overview of the Cluster-based Page Segmentation is outlined in figure 7.1. It presents the relevant parts extracted from figure 5.3. As the outline demonstrates, there are three main parts of the template clustering system. The *Template comparator* is the first point of contact for incoming data. The comparator basically employs clustering metrics to find out if the incoming DOM tree matches to any of the previously known templates that are stored in the *Template storage*. The purpose of the last component, designated *Node mapping*, is to identify similarities between the incoming DOM tree and the matched DOM tree loaded from the storage in order to assign the right DOM nodes to the right visual areas.

Before going deeper into the details of template clustering, it is appropriate to mention some of the design specifics, as they are important for understanding some of the details. Most often, the clustering is performed on a complete set of values. However this approach is not practically usable in any of the targeted use cases. There are two reasons for that:

- It is not possible to estimate upfront how many pages will be in the set.

- Large web sites like news servers keep publishing new content and therefore, the set of web pages will never be completed.

Both these reasons imply that some form of stream clustering algorithm is required. That aligns with another preferred feature – to store as few data as possible in order to optimize the memory and disk space consumption in practical applications. To achieve this goal, none of the segmented web pages are stored in the existing clusters. Only minimal structures represent the individual clusters. These structures will be further called Cluster Representatives.

7.2 Template storage

This component has to be explained first, as the definition of the data structures stored in it is crucial for understanding the Template comparator. As stated in section 7.1, once the web page is segmented it is transformed so it can represent an entire cluster of pages.

To avoid confusion created by combination of box clustering and template clustering, the clusters as defined in the previous chapter will be referred to as *visual areas* in this chapter. The reason for introducing this alias is to clearly distinguish between clusters of boxes and clusters of web pages.

Let's focus directly on the cluster set consisting only of the minimal clusters as introduced in section 7.1. That means, each cluster consists only of its Cluster Representative. There are three parts that need to be considered:

- Templates represented by structures derived from DOM trees
- Set of the visual areas that are returned by the Box Clustering Segmentation algorithm
- Mapping between the previous two.

The set of visual areas, denoted V, is equal to the set of clusters defined in chapter 6, therefore its definition will not be repeated here. The first part of each Cluster Representative that is to be described is a template representation in a form of DOM tree based structures. Even though an unmodified DOM tree would serve the intended purpose as well, its lightweight abstraction is used instead. This lightweight abstraction is called *pruned DOM tree*. Using it saves space both on the disk and in memory. It also improves time performance as the redundant data don't have to be loaded, stored and repeatedly inspected. The following definitions therefore define the pruned version of the DOM tree, as used in each Cluster Representative.

Definitions 18 and 19 formally introduce the structure of reduced DOM node and pruned DOM tree. Identically to the structure of DOM tree itself, the structure of its abstraction is carried within each reduced DOM node. Therefore a) no set of edges is required in the DOM tree, even though it is essentially a graph and b) the most important information in the DOM tree pair is a reference to its root node.

Definition 18 (Reduced DOM node) *Let N be a set of all DOM nodes in a web page. A reduced DOM node $n \in N$ is defined as a pair $n = (C_n, A_n)$ where $C_n \subseteq N$ is an ordered set of its child nodes and $A_n = \{(key, value)\}$ is a set of its attributes.*

Definition 19 (Pruned DOM tree) *Pruned DOM tree is defined as a pair $D = (N, n_r)$ where N is a set of all DOM nodes in the tree and $n_r \in N$ is a root node of the tree.*

Definition 18 expresses how does the reduced DOM node look. Similarly to standard DOM model, this is a key entity in the pruned DOM tree. It comprehensibly defines structure of the tree in a way that allows easy handling by algorithms. Defining the structure is the main goal of reduced DOM nodes. However, to be useful, they need to contain some information as well. All the information is actually not contained in the node itself. Instead, every piece of information that is used is transferred to the set of attributes which effectively condenses it all into one point, keeping the node structure simple.

Structure of the DOM tree is defined by relations between the DOM nodes. One of the key features pointed out in section 2.3.2 that contributes to setting the structure is the order of sibling nodes. The same order has to be preserved in the pruned DOM tree so this important information is not lost in the transformation. The following definition specifies the relation on top of the elements of C_n which makes the set ordered.

Definition 20 (Order of the sibling nodes) *Let $n = (C_n, A_n)$ be a reduced DOM node and $c_i, c_j \in C_n, i \neq j$. The element order in C_n is defined as a condition $i < j \leftrightarrow$ c_i precedes c_j in the HTML code.*

Attributes are the smallest units of information in the pruned DOM tree. The design proposed here establishes a middle ground between the two approaches allowed in the DOM model and enumerated in section 2.3.2. That supports flexibility while keeping the overhead at minimum. Because the set of attributes does not contain only a set of what is considered to be an attribute in the standard DOM model but it also contains other information, such as style definitions, the flexibility is its key feature.

Now that the structure of the pruned DOM tree is defined and tangible, it's possible to use it as a base of structure that is important for the clustering algorithm – DOM path. The clustering algorithm works with sets of these paths. Note that these sets are not multisets. The DOM tree itself can contain multiple paths leading to different nodes of the tree but having the same representation (as specified in Definition 21). However, the intended application of the path sets does not require this property to be preserved so only one instance of each path is kept in the set.

Definition 21 (DOM path and set of DOM paths) *Let $D = (N, n_r)$ be a pruned DOM tree. A DOM path in the tree D is defined as a k-tuple $p = (n_0, n_1, \ldots, n_k)$ where $n_0 = n_r$ and $\forall 0 < i \leq k : (n_i \in C_{v_{i-1}})$ and $C_{n_k} = \emptyset$. A set of DOM paths is designated P_D.*

The format of the pruned DOM tree in combination with how was structure of a cluster and the set of clusters defined in chapter 6 implies an image where each Cluster Representative can be viewed as a combination of pruned DOM tree D and a set of visual areas V. This combination is disunited, as the individual components are only loosely coupled and there are no direct links between them. When mapping between the two is included, this changes significantly. The reason for storing this information is purely practical. When performing the Box Clustering Segmentation, the algorithm exactly knows which DOM nodes belong to every cluster that is returned. Storing this information for the future reference eliminates the need to heuristically (and very probably inaccurately) analyze all the relations later.

The mapping between the pruned DOM tree D and the set of visual areas V starts at relations between individual elements of these two entities. Definition 22 declares how the visual containment is determined and then definition 23 explains how is the mapping defined.

Definition 22 (Visual containment) *Let $D = (N, n_r)$ be a pruned DOM tree representing a web page and let $n = (C_n, A_n)$ be a node within that tree. Furthermore, let a be a*

visual area bounding a part of the web page. The visual area a visually contains the node n if and only if the textual and media content of DOM subtree rooted at node n is a part of textual and media content of the part of the web page that is bounded by area a and a does not visually contain the parent node p_n. The relation of a containing n will be denoted as $a \odot n$.

Definition 23 (Visual to DOM Mapping) *Let $D = (N, n_r)$ be a pruned DOM tree and let a be a cluster representing one visual area in the set of visual areas V. The Visual-to-DOM mapping M_a for cluster a is defined as a set of DOM nodes visually contained by the cluster $M_a = \{n | n \in N \wedge a \odot n\}$. The mapping M between the set of visual areas V and DOM tree D is defined as a set of Visual-to-DOM mappings of all clusters in the set V: $M = \{(a, M_a) | a \in V\}$.*

Now that all the components of Cluster Representative and relations between them are defined, it's possible to define the Cluster Representative itself.

Definition 24 (Cluster Representative) *Let D be a pruned DOM tree, V a set of visual areas and M a Visual-to-DOM mapping between D and V. The Cluster Representative r is defined as a 4-tuple $r = (u, D, V, M)$ where u denotes URL of the original web page.*

7.3 Working with the Cluster Set

Now that all data types in the Template storage are specified, the algorithm behind the Template comparator can be designed. Let's repeat some of the specifics of the comparator. Its goal is to determine if the Template storage contains any record of previously segmented web page that the incoming one resembles. The record will be stored in form of a Cluster Representative. If such record exists, the entire Cluster Representative shall be returned for further processing. If such record does not exist, the page shall be sent for segmentation and stored afterwards.

This description outlines the entire big picture that is shown in figure 7.1. Algorithm 3 shows the orchestration of the process more formally. As the figure 7.1 shows, there are two inputs of the algorithm: the incoming web page represented by its DOM tree and the contents of the Template storage, represented by a set of Cluster Representatives R. The orchestration algorithm returns a Custer Representative, as it contains all the components displayed in figure 7.1.

Algorithm 3 Orchestration of the BCS and Template clustering

```
1:  function processPage(DOM_input, R)
2:      D_in ← pruneDom(DOM_input)
3:      r ← matchDom(R, D_in)
4:      if r = nil then
5:          V ← BCS(DOM_input)
6:          M ← mapVisualToDom(D_in, V)
7:          r ← (D_in, V, M)
8:          R ← R ∪ {r}
9:      end if
10:     return r
11: end function
```

The streaming nature of the algorithm is clearly visible, as the result is returned immediately and no processing is scheduled for later. There are some functions which have not yet been described, nor explained. Function `pruneDom` on line 2 transforms the standard DOM tree to the pruned one according to definitions 19, 18 and 20. The process is straightforward, therefore it won't be described more closely. Function `BCS` represents the Box Clustering Segmentation as described in chapter 6. Functions `matchDom` and `mapVisualToDom` will be both described more closely.

7.4 Matching DOM Tree to the Cluster Set

This section will cover internal functionality of the `matchDom` function. Since it is a stream clustering function, its internal process can be as simple as matching the incoming tree to each Cluster Representative that is stored from previous segmentation runs and returning the best match.

In its most crude design, the function just iterates over all the records and compares the incoming DOM tree with the DOM tree of each Cluster Representative. However this approach does not scale at all. There are several possible ways how to address the scaling issue. One of them would be some sort of indexing, based on the DOM tree layout. However designing proper indexing data structures and algorithms is well beyond the focus of this thesis. Therefore an alternative approach was selected.

This alternative approach is based on the assumption that there is only a very limited set of templates on every web site. Iterating over this limited set is much more scalable, mainly because the set of templates does not extend much over time. The only thing that can happen is that the design of the web site is changed in which case the old template set for that site can be replaced entirely by the new set and in the end the algorithm has to iterate over approximately the same amount of Cluster Representatives.

This approach lies in pre-selecting the set of templates for the site that the incoming web page comes from, thus filtering out the irrelevant Cluster Representatives. The default behavior of the preselecting algorithm is to take the second level domain of URL of the incoming web page and filter out those that don't match. Since the URL is stored as a plain string, this is pretty simple, as string indexes are a common feature of almost every database engine.

Algorithm 4 illustrates how the function `matchDom` works. The process of filtering out the irrelevant cluster representatives is represented by function `filterByDomain` on line 5. One of its inputs is URL of the page on the input which is obtained by function `getPageUrl` on line 4. This does not need to be described, as the information about URL can be associated with the DOM tree in several different forms and its extraction is always trivial, regardless of the implementation. The filtering process itself will also not be described closely, as it is covered by external database engine in practical applications and its value to this thesis is minimal.

After the set of Cluster Representatives for the site is selected, the algorithm simply iterates over the set and checks similarity of the incoming web page to each Cluster Representative. This process is covered by lines 6 to 21 in the algorithm 4. Only one requirement that has to be met remains and that is fast DOM-to-DOM matching algorithm. Now the template detection algorithms become important, because the design of some of them allows their usage as metrics in the clustering algorithm as section 3.2.4 explains.

The best algorithm for the task, as reasoned in section 3.2.4, is the Common Paths Distance algorithm. It was proven to be significantly faster than standard tree-edit-distance

Algorithm 4 Finding the corresponding Cluster Representative

```
1: function matchDom(IN: R, D_in)                    ▷ D_in = (N_{D_in}, n_r, P_{D_in})
2:     bestMatch ← 0
3:     ret ← nil
4:     u_{D_in} ← getPageUrl(D_in)
5:     R_filtered ← filterByDomain(R, u_{D_in})
6:     for all r ∈ R do
7:         match ← 0
8:         sum ← 0
9:         for all path_in ∈ P_{D_in} do
10:            for all path_r ∈ P_{D_r} do
11:                sum ← sum + 1
12:                if path_r = path_in then
13:                    match ← match + 1
14:                end if
15:            end for
16:        end for
17:        match ← match/sum
18:        if match ≥ 0.7 ∧ match > bestMatch then
19:            ret ← r
20:            bestMatch ← match
21:        end if
22:    end for
23:    return ret
24: end function
```

algorithms while still being precise enough to match a DOM tree to the right template (if such template exists). The chosen algorithm is also the reason why a tree of paths is stored as a part of each Cluster Representative. A modified version of the original algorithm is used in this thesis. The core is essentially the same and it is outlined on lines 8 to 19 in algorithm 4. The adjustments made to the original algorithm are not visible in the algorithm 4, as they lie in modification of the paths in the path set:

- All the nodes that do not represent any HTML element (e.g. attribute nodes, text nodes, etc.) are filtered out
- If an element has an `id` attribute, its name is used in combination with the `id` value in the path. This is important considering that the set of paths is not a multiset.

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

The threshold of 0.7 was taken from the original paper and experiments prove that it is applicable even when taking the modifications introduced above into account.

The best match that is above this threshold (if there are any such matches) is returned as the template that the incoming page is based on. When such match is found, it is possible to use the corresponding set of visual areas and map them to the DOM tree of the incoming web page. If no matching Cluster Representative is found, the page is considered to be based on a new template that the algorithm has not encountered yet.

7.5 DOM Tree Mapping

Mapping the nodes of the processed page to those stored in the matched Cluster Representative is the last step in the process when a match is identified. This step is necessary for retrieving the actual contents of the visual areas and producing the segmentation result. The mapping procedure is trivial for Cluster Representatives themselves, as the mapping is already a part of their stored structure.

In case of any other page that matches the cluster, the process is more complex. Again, we have a set of visual areas of the matched Cluster Representative, the corresponding set of Cluster Representative's DOM nodes D_r and the mapping between the two sets. We also have a content stored within set of nodes of the DOM tree of the incoming web page D_{in}. Because it's impossible to directly adjust the mapping so it connects visual areas with nodes of the incoming DOM tree, the only option is to replace nodes of the Cluster Representative's DOM tree with nodes of the incoming DOM tree in a working copy of the mapping M. The first step in this process is to find an isomorphic mapping between the two DOM trees in order to know which nodes of the D_r are to be replaced and which nodes of the D_{in} will be replacing them.

The tree-mapping problem for two DOM trees is quite complex in general, as many examples demonstrate [43, 55]. However, as I explain in my previous work [64], this scenario is very specific so it's possible to introduce some simplifications. Most of them are based on features of DOM trees that are stated in section 2.3.2.

The most important simplification, however, is that for purposes of locating the right content within D_{in}, the algorithm does not have to perform full node-to-node mapping. Instead, it is sufficient to locate a subtree located at a specific node. The assumption is that once the root is found, all descendant nodes correspond in both trees. To verify this assumption, the mapping algorithm performs a validation of the found root node by comparing the selected subtree of D_{in} with the corresponding subtree in D_r.

The mapping algorithm can be divided into two steps. When a node belonging to D_r is selected from the mapping M, the first step is to find the corresponding node within D_{in}. This step is called *Node identification*. Because the first step just select a node from the tree D_{in}, the mapping algorithm must then verify in step two that the selected node is the right one.

Node identification

In this step the goal is to find the corresponding node in the DOM tree D_{in}, having a node from D_r as input. Because the number of nodes that are to be mapped can be potentially very high, the main requirement for the node identification process is speed. There are also some secondary requirements like uniqueness of results and reliability (i.e. the method must return the corresponding node or nothing if the node is not found).

In this step, there is one key feature of DOM trees that is utilized and that is the order preservation, i.e. the fact that order matters in DOM trees. Therefore it's safe to consider that DOM node of D_{in} and a corresponding DOM node of D_r will be on the same position within their siblings, the same applies for its parent and the same applies recursively up to the root node of a DOM tree. The structure that utilizes this feature is called *path of positions*. It is formalized in definition 25. Basically it's a string of numbers that identifies the path from DOM tree root to the searched node.

Definition 25 (Path of positions) *Let the path of positions p_n from root of given DOM tree to a node in that DOM tree n be defined as a sequence :*
$p_n = (p_1, p_2, \ldots, p_k)$ *where* $1 \leq p_i \leq k$ *is a position of a node withing its siblings. i is an index designating a level of each node on the path in the DOM tree and k is a level of the searched node n.*

This representation is just the simplest form of the path of positions. While the tree mapping algorithm is not concerned with verification in this phase, some degree of error detection is desirable. A mechanism called *feature fingerprinting* is implemented for this error detection. The path of positions extended by feature fingerprints is called *path of distinguished positions.*

Feature fingerprint can be perceived as a combination of various attributes that DOM nodes can have. These attributes can be either visual (like size or position of rendered box corresponding to the DOM node), tree-based (e.g. tag name or value of any specific attribute) or content-based (e.g. word count). Specific combination of these features can be used to detect if each node on the path of positions is the one that is supposed to be there. For example Hao et al. in [21] use combination of position within one level of the tree combined with visual position and size.

While feature fingerprinting constitutes a good way to improve reliability of the node identification process and uniqueness of its results, it is still not reliable enough so the need for subtree verification in step two of the tree mapping process is still valid.

Considering the requirements for node identification, visual feature can be ruled out, as calculating them would cause performance degradation of the entire process. One additional requirement for the features that are used in fingerprints is for them to be as generic as possible – it is pointless to use a feature that only a fraction of DOM nodes contain. That's why the final design uses another feature of DOM trees mentioned in section 2.3.2 and that is the fact that each element in the HTML code has a name and therefore each node in the pruned DOM tree has a label assigned to it that represents that name.

Definition 26 shows the fingerprint that is a part of node descriptor used in the path of distinguished positions. It utilizes the DOM node label together with `id` attribute (if the element in question has one) and the number of nodes on the level of the node (i.e. the number of siblings including the node itself). The path of distinguished positions is then a sequence of these descriptors as definition 27 describes.

Definition 26 (Node descriptor) *Let d_n be a descriptor of node n. The descriptor is an 4-tuple $d_n = (p_n, cnt_n, id_n, lbl_n)$ where cnt_n and p_n designate the total count of nodes within the sibling array and the position of node in this array respectively and id_n and lbl_n represent feature-fingerprint of the node n.*

Definition 27 (Path of distinguished positions) *The path of distinguished positions dp_n from root of given DOM tree to a node in that DOM tree n is defined as a sequence of descriptors :*
$p_n = (d_1, d_2, \ldots, d_k)$ *where* $d_i : 1 \leq i \leq k$ *is a descriptor of a node that is on the path on i-th level of the DOM tree hierarchy. k is a level of the searched node n.*

Algorithm 5 demonstrates how the node is located within the incoming DOM tree. It shows that the approach is really straightforward and therefore really fast. In fact, thanks to utilizing the sibling order feature of DOM model, the time complexity of this algorithm

Algorithm 5 Locating DOM node using path of distinguished positions

```
1: function LOCATENODE(D_in, dp)                          ▷ D_in = (N_in, n_r, P_in)
2:     n ← n_r                                            ▷ n = (p_n, C_n, A_n)
3:     i ← 0
4:     if |dp| = 0 then
5:         return n
6:     end if
7:     repeat
8:         d_i ← dp[i]                                    ▷ d_i = (p_i, cnt_i, id_i, lbl_i)
9:         if |C_n| ≠ cnt_i then                          ▷ test for sibling count match
10:            return nil
11:        end if
12:        n1 ← C_n[p_i]                                  ▷ n1 = (p_n1, C_n1, A_n1)
13:        if id_i ≠ nil ∧ ('id', id_i) ∉ A_n1 then       ▷ test for ID match
14:            return nil
15:        end if
16:        if id_i = nil ∧ ∃X : ('id', X) ∈ A_n1 then     ▷ 2nd test for ID match
17:            return nil
18:        end if
19:        if ('lbl', lbl_i) ∉ A_n1 then                  ▷ test for label match
20:            return nil
21:        end if
22:        n ← n1
23:        i ← i + 1
24:    until i < |dp|
25:    return n
26: end function
```

is $O(n)$ where n denotes the depth of the DOM tree. Conditions on lines 9, 13, 16 and 19 of the algorithm 5 also show that the algorithm also cares about reliability of results.

There are two ways how to make use of paths of distinguished positions. In practical applications, they are most likely to be calculated during the construction of pruned DOM tree and stored within DOM nodes themselves. Another option is to construct them afterwards, using references to reduced DOM nodes as starting points. The algorithm to do that is trivial, as it basically behaves like reversed algorithm 5.

Subtree verification

While feature fingerprinting is somewhat reliable, it has been clearly stated its that their reliability is not 100%. There is a method for verification of the result that offers even greater degree of reliability for most subtrees that can be located within a DOM tree. The verification is based on simple premise that two nodes, each in a different DOM tree, are equivalent if their content is equivalent (mostly in terms of its structure).

This premise comes back to the concept of templates where the core structure of two web pages based on the same template is the same, therefore the position of all nodes belonging to the template will be the same on both pages. This rule does not apply for the actual content of the two web pages, as it is not part of the core template structure. For the same reason the content of the two subtrees is not important for the verification, as

properly working segmentation algorithms do not select just parts of a content block to be in a visual area. In this context, definition 28 is used as a formal specification of the subtree equivalence.

Definition 28 (Subtree equivalence) *The content of two nodes, each in a different DOM tree, is considered equivalent if subtrees rooted at each of the two nodes are equivalent.*

Because the resemblance between subtrees comparison and some template detection algorithms is used to establish the content equivalence, it is possible to seek solution in the area of template detection again. The solution resides in utilizing the same algorithm that is already used in this thesis. The subtree equivalence is again determined using comparison of path sets and the same modified version of the algorithm introduced by Gottron in [20] is used.

While it would be theoretically possible to use this algorithm for every node on the path of distinguished positions, the demands for both time and space complexity would be too high to do that. That's why this is only used to verify that the node identified in the first step of the algorithm is the right one.

Chapter 8

Experimental Implementation

This chapter will provide the information that is not important for the designed segmentation algorithm but it clarifies some of the practical aspects of the proof of concept implementation that was used for experimental evaluation of the algorithm.

The experimental program was written in Java. That is the only way how to reliably compare the algorithm with other algorithms in existence. The original VIPS algorithm[13] is closed source and based on a proprietary library but there is an alternative available that was written in Java and based on CSSBox. Writing the Box Clustering Segmentation in Java and base it on CSSBox is very convenient for subsequent comparison, as some disturbing influences like using different rendering core don't have to be taken into account when evaluating the results. Besides that, CSSBox also provides a great amount of information about structure of the rendered page that other engines either don't provide at all or they have to be worked around to get the information.

The implementation will be now analyzed component by component, using the system component diagram (figure 5.3) as a reference. For every component the specifics that differ significantly from the design laid down in previous sections will be closely explained and the difference justified.

The architecture represented by the component diagram starts with URL selector. In practical implementation, this component uses three lists of URLs instead of one. While the list (used as queue) of URLs scheduled for processing would be sufficient, it's also a good idea to include a list of visited URLs. Because pages within web sites are deeply interconnected by hyperlinks, the crawler might start looping, never to inspect the entire site (or go out of a small set of interlinked web pages for that matter). The third list used is for URLs that go out of the web site. This has practical implications that will be covered in the text below.

For the next two components in the chain (URL fetcher and DOM parser), the reference implementation uses functionality offered by Java language itself. Since there is nothing special about these tasks, it is possible to use the standard Java library.

The subsequent processing, however, diverges from the schema in figure 5.3. The divergence is shown in figure 8.1. In the original schema, the DOM tree is passed as input to two components – URL extractor and template comparator. However, as algorithm 3 demonstrates, template comparator actually needs pruned DOM tree to compare it to stored Cluster Representatives. Considering the wider context, it means that two traversals through the DOM tree are required. To save some computation time, it's convenient to traverse through the tree just once and extract both of the information at the same time.

That's exactly what DOM browser does. Because it's just an implementation detail, it is not included in the schema in figure 5.3.

8.1 Template clustering specifics

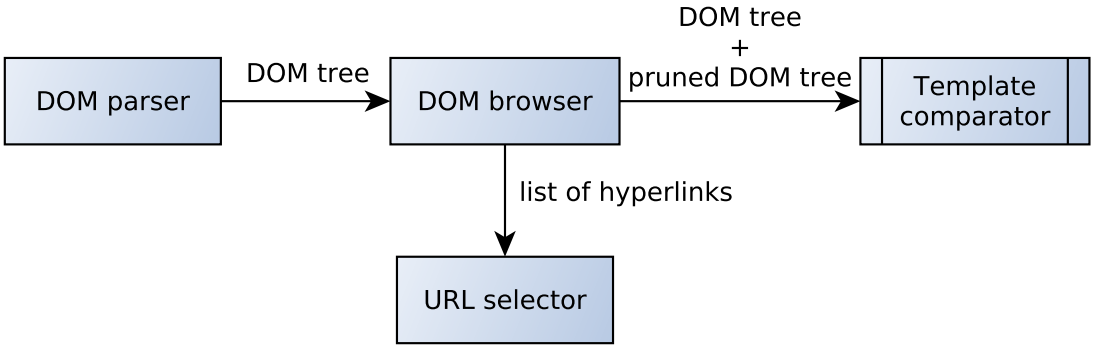

Figure 8.1: DOM browser

In the next step the pruned DOM tree is compared to the templates that are stored in the database. The reference implementation has some specifics here as well. OrientDB [1] is used as database engine for the template storage. It is object-oriented database with persistent storage. That means the data is stored on disk. The problem is that searching the data on the disk is too slow and comparing all Cluster Representatives with the incoming pruned DOM tree would have a significant performance impact, one that would marginalize benefits of the entire template clustering, especially with higher numbers of templates stored in the database. Therefore a subset of the data has to be loaded to memory. The pre-selecting technique described in section 7.4 was designed specifically to address this issue. Before a web site processing starts, all the templates that belong to that web site are loaded into memory and that in-memory instance is used from that point until the site is processed.

This is where the third queue of URLs is important. It is used specifically for the crawler to stay on one web site for as long as possible. Performed measurements confirm that loading the data from the database is slower by as much as several orders of magnitude compared to the in-memory processing. Using the queue of outbound links makes sure the data has to be loaded only so many times which further improves the performance of the algorithm.

Another important implementation detail is that every piece of data described in section 7.2 is stored in the database, no computation has to be performed on the data once it's stored in the database. Even the mapping relation M is stored as a set of references to other objects, which makes the implementation quite fast: once the data is loaded from the database, all object references are loaded as well and traversing through the data is very simple.

[1] http://www.orientechnologies.com/orientdb/

8.2 Box Clustering specifics

The implementation of the Box Clustering Segmentation matches the detailed description given in chapter 6. There are just a few remarks that are worth mentioning, as they were not important in that chapter but are useful in a real world application.

The first thing to note is creation of mapping between clusters and DOM nodes. That is quite simple, as each cluster carries references to all boxes that are within it and each box has a reference to the DOM node which produced it. This is a feature of CSSBox and while the BCS algorithm internally uses its own structures to represent boxes, the reference to the DOM node is preserved there. In effect, the mapping is transparently moved from CSSBox all the way to Cluster Representatives, no further adjustments are made by the program.

Going through the process of Box Clustering Segmentation, the first step is cluster creation. One thing that was not mentioned in section 6.1 is that CSSBox produces some faulty boxes. These can be for example boxes with no content and not actually existing on the page or boxes which have coordinates placing the box outside of the page. There are some heuristics in the reference implementation that try to find these and filter them out but occasionally these boxes are missed by the heuristics and the segmentation results are then distorted.

Connection creation and similarity calculation are straightforward and no traps are there. Then there is the clustering step. Because the definitions in section 6.7 are declarative, the imperative nature of the algorithm is somewhat hidden. In reality, the algorithm uses a structure called *list of relations* which can be described as the set of edges in a graph, enhanced with some extra information to save computational time. This information includes pieces like cardinality and direction of each edge and similarity of the connected boxes. To ensure that the order of elements in the list is correct, a sorting operation has to be performed in every iteration of the clustering algorithm. This brings the performance down quite a bit. The edge pruning demonstrated in figure 6.2 was devised specifically to reduce the effect of the sort operation. Measurements showed that building the relation set using only the limited set of boxes improves the performance of the entire BCS by approximately 30%.

Another potential performance issues are brought by the fact that a lot of testing for overlap happens during the entire clustering process, especially when extending the cluster to accommodate overlapping boxes. To make this process as efficient as possible, an external library was used to perform the overlap testing. The library is called Java Spatial Index Library [2]) and it is open source, licensed under LGPLv2. The library uses R-Tree structures so its implementation is quite efficient.

[2]http://sourceforge.net/projects/jsi/

Chapter 9

Evaluation

This chapter will summarize and evaluate various aspects of the proposed segmentation algorithm and it will compare it with the VIPS algorithm which is still the most widely used segmentation algorithm. Both the quality of results and the performance boost will be evaluated here; the data presented is the same as it is in the original paper[65].

Experimental type of evaluation is the most suitable, as the time complexity analysis does not offer clear picture. The complexity of the BCS algorithm is polynomial, with asymptote at $O(n^4)$ where n designates the number of boxes on the web page. However that case is so extreme that it cannot be encountered on real life web pages. The actual computational time is so dependent on all visual attributes of individual boxes that it cannot be determined. The problem is that the the number of boxes does not influence the computational time that much, the visual attributes are at least equally important and no single one of them can be taken as one attribute of the time complexity evaluation. Making the asymptote a function of all these attributes would make the comparison to other algorithms impractical. The experimental comparison therefore offers a better way how to compare the proposed algorithm with existing solution.

This chapter provides comparison with the VIPS algorithm. The implementation used here was written as master's thesis by Tomáš Popela [42] and it was designed specifically for the purpose of this comparison. That implies two things: it was written in Java and it uses the CSSBox as the rendering core. That way it's possible to make a safe comparison – otherwise the results would be influenced by both runtime environment and different results provided by the rendering core. The VIPS implementation is in compliance with the algorithm as described in [13].

9.1 Template Count

One of the key points of the template clustering described in chapter 7 is to make the algorithm as scalable as possible. However the entire concept is based on the assumption that the number of templates on one web site is very small compared to the total number of pages generated using these templates. Another important assumption was that at a certain point the number of templates does not grow any more and in general it grows only very slowly in comparison with the number of pages inspected.

Most of the performance experiments in following sections are performed on a set of 500 inspected web pages. To demonstrate how many templates are registered by the system for the web site after the inspection of 500 web pages, table 9.1 presents template count for the

web sites that will occur in further evaluations. Also to better illustrate how much does template clustering help in the process of segmentation, it also contains the information about hit ratio. The number there specifies what is the percentage of pages that didn't have to be processed by the segmentation algorithm and their content was instead retrieved via the isomorphic mapping.

One important observation was made during the testing: when the site is inspected for the first time, most of the templates are discovered quite early in the process. If the number of processed web pages was increased from 500 to 1000, the number of detected templates rose in two extreme cases by more than 33% but in all other cases by less than 22%. This demonstrates a logarithmic growth of the cluster set size, leading to confirmation of all the assumptions stated above. Figure 9.1 graphically demonstrates the growth of template count with raising number of inspected pages within a single site. It confirms that the number of templates discovered on a single site converges fast to a relatively small number.

site	**template count**	**hit ratio**
businessinsider.com	10	98%
e15.cz	25	95%
fedora.cz	20	96%
gizmodo.com	3	99.4%
idnes.cz	50	90%
lidovky.cz	33	93.4%
novinky.cz	22	95.6%
slashdot.org	18	96.4%
telegraph.co.uk	42	91.6%
reuters.com	37	92.6%
yahoon news	56	88.8%

Table 9.1: Template counts for different sites

9.2 Evaluation of the BCS

BCS is direct competitor of VIPS, it will be therefore evaluated against it. The following criteria are important in the comparison.

- performance of each algorithm
- accuracy of segmentation results
- stability of each algorithm
- general usability of results

In order not to run test separately for each of the tested criteria, the following sequence of actions is performed for every tested web page.

1. render the page, get the rendering tree
2. let a user create reference segmentation of the web page.

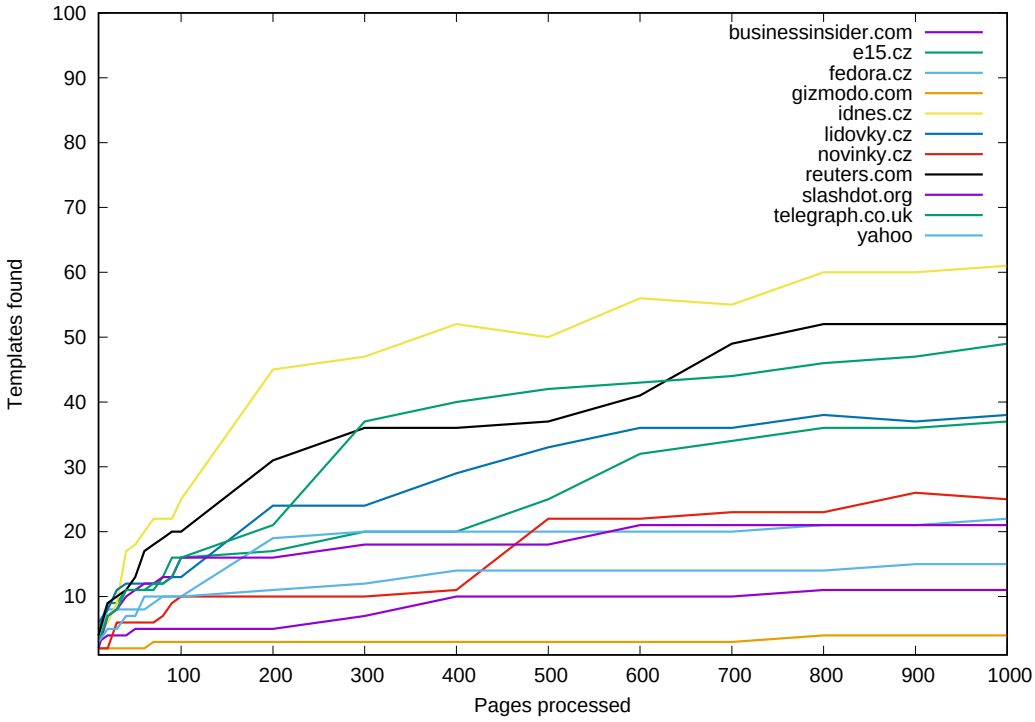

Figure 9.1: The dependency of cluster count on page count

3. run each algorithm ten times on that rendering tree, each time with different value of CT/PDoC
4. measure accuracy of the algorithm in each run
5. compute the mean run time of each algorithm
6. compare run times and the accuracy of results

Let's designate the algorithm to be stable if the quality of its results is stable across different web pages. In that context, it's necessary to pick the right web pages on which the segmentation is going to be performed. To test stability thoroughly, a wide variety of different layouts needs to be selected for segmentation. Several layout types of web pages are considered:

- *Complex index pages* – pages like news indexes are characterized as highly complex and strongly structured. There are multiple logical areas on the page, each covering different topic or announcing different article. On top of that, every area consists of a small number of elements. Another characteristic is that there are often patterns on these pages, as all the articles are displayed using the same template.

- *Articles* – pages like these contain one main block of text, usually consisting of multiple paragraphs. Besides this one big block, there are some smaller areas that are usually related to navigation and some small pieces of generic information (like contact or news on company web sites).
- *Simple web pages* – this is a good example of some minimalistic web pages, usually educational ones. The main characteristics of web pages like these is a minimal amount of logical areas with varying number of boxes constituting them.

For the evaluation of BCS, a dataset of real web pages containing all the mentioned page types was created. In this dataset, there are 8 different types of pages from 5 news web sites. These pages are listed in all the tables below. Note that the *businessinsider.com* and *yahoo news* sites don't use paging and it was therefore not possible to statistically process their respective index pages. A set of 100 pages of every type was collected. That makes 800 pages in total. Then, three volunteers independently created a reference segmentation for every page from the set. In total, that makes 2400 annotated pages from the three volunteers. To facilitate the work of the volunteers and to ensure consistent segmentation of all pages of each of the eight types by a single volunteer, we have used a semi-automatic segmentation tool. This approach is based on the fact that all the pages of the same type share the same template that is used for generating their HTML code. The tool lets the user segment one page and then maps the segmentation results to the other 99 pages of the same type and lets the user verify the correctness of that mapping for each page.

As for the usability, this thesis compares how usable is the format of results by any application that wants to use the results for further processing. This will be discussed in one of the subsections below.

9.2.1 Performance

Table 9.2 demonstrates the first part of the algorithm evaluation – the run times of both algorithms. As explained above, multiple web page types are represented in the table. As the table 9.2 demonstrates, the BCS algorithm is superior to the VIPS in terms of time necessary to process a web page. This difference gets bigger with decreasing complexity of evaluated web page.

page	VIPS time	BCS time
businessinsider.com (article)	522 ms	20 ms
idnes.cz (index)	1079 ms	39 ms
idnes.cz (article)	723 ms	53 ms
novinky.cz (index)	28126 ms	699 ms
novinky.cz (article)	390 ms	18 ms
reuters.com (index)	475 ms	15 ms
reuters.com (article)	442 ms	37 ms
yahoo news (article)	342 ms	21 ms

Table 9.2: Algorithm run time comparison

9.2.2 Accuracy, stability and usability

Evaluating the accuracy is more complex. In statistical analysis in general, F-score is a common way how to evaluate accuracy. However, no method commonly used specifically

for evaluating the accuracy of segmentation algorithms was found in the literature. In [28], the F-score is used but the underlying method for matching segments is only very crude. It matches the textual content of detected segments to the textual content of the anticipated ones. There is, however, an alternative that we can use. As this thesis proposes, the page segmentation task, regardless of how it's performed, is basically a clustering task – each segment being a cluster of page elements and the page as a whole being clustering. In data clustering, Adjusted Rand Index (ARI) [23] is being used to measure similarity between two clusterings.

Due to the lack of existence of commonly accepted reliable method for segmentation accuracy evaluation, both the F-score and ARI are being used in this evaluation. The evaluation is based on visual perception of detected segments. That visual perception can be understood as interpretation of what does ideally segmented page look like. Each box in such page is assigned to at most one visual area, there are no areas on the page containing no boxes and no areas overlap. Moreover, each visual area has to meet a *semantic condition*: the boxes in the area have to constitute one unit of content that is coherent visually, semantically or (preferably) both. The method description follows. For better orientation, note that the following text distinguishes *detected areas* and *selected areas*. While the former one designates an area detected by the tested algorithm, the latter one refers to an area that is manually selected by human observer.

- initialize the ARI contingency table by zeros
- manually mark all selected areas on the web page that meet the semantic condition
- for each detected area:
    1. find "detected" boxes: all rendered boxes that intersect with the detected area
    2. find all selected areas that share at least one box with the detected area
    3. for each such selected area:
        (a) find "selected" boxes: all rendered boxes that intersect with the selected area
        (b) calculate precision and recall using sets of "detected" and "selected" boxes
        (c) fill the line in the ARI contingency table that corresponds to the "detected" area
- if there are any selected areas that do not share boxes with any detected areas, set the recall value for each of them to 0
- calculate sums in the ARI contingency table
- calculate ARI and F-Score for the web page

This sequence is repeated for every segmentation result (with varying Threshold and PDoC) and for every manual selection of visual areas available for the evaluation.

To evaluate stability, we will use the results of accuracy evaluation.

Accuracy and stability

Table 9.3 shows F-score and ARI comparison of BCS and VIPS. For each comparison, the best respective values of both algorithms were chosen. The F-score value is a real number

page	BCS ARI	VIPS ARI	BCS F	VIPS F
businessinsider.com (a)	0,5704	0,7010	0,6345	0,7394
idnes.cz (article)	0,6629	0,7240	0,5570	0,5720
idnes.cz (index)	0,5954	0,7926	0,5522	0,7259
novinky.cz (article)	0,7670	0,7877	0,6446	0,7191
novinky.cz (index)	0,5303	0,9121	0,4265	0,9043
reuters.com (article)	0,6123	0,6786	0,5914	0,6943
reuters.com (index)	0,5832	0,8160	0,5145	0,7569
yahoo news (article)	0,7556	0,5626	0,7102	0,5446

Table 9.3: Algorithm accuracy comparison using the ARI and F-score metrics

between 0 and 1, where higher values are better. ARI score is between -1 and 1, higher values are better.

The results show that the accuracy of VIPS is slightly better, especially when processing structured pages. The reason is that BCS is too aggressive when creating clusters, thus effectively overlooking the structure. VIPS on the other hand does much better job in finding repeating patterns in the web page. When processing pages with less structure, the accuracy of BCS and VIPS is comparable, in some cases BCS is even better than VIPS.

Stability of both algorithms was calculated using the same data that was used to populate table 9.3. The stability was calculated for each page type in each web site. As such, each number in table 9.4 represents the stability of BCS across multiple instances of the same page (i.e. different pages with the same template but different content in that template).

page	BCS ARI	VIPS ARI	BCS F	VIPS F
businessinsider.com (a)	0,1275	0,1740	0,0358	0,0721
idnes.cz (article)	0,0646	0,0766	0,0424	0,0794
idnes.cz (index)	0,0733	0,0078	0,0108	0,0070
novinky.cz (article)	0,1274	0,1406	0,0404	0,0731
novinky.cz (index)	0,0529	0,0140	0,0558	0,0219
reuters.com (article)	0,1316	0,1687	0,0301	0,0847
reuters.com (index)	0,0328	0,0516	0,0203	0,0336
yahoo news (article)	0,2089	0,1658	0,1000	0,0539

Table 9.4: Algorithm stability comparison: Standard deviation of the results in the dataset

Again, both algorithms are comparable. In some cases the stability of BCS is almost three times better than that of VIPS, in others, it's exactly the opposite. Not looking at the degree of superiority, the stability of BCS is better in 5 data sets, i.e. 62.5% of measurements.

Usability

One problem remains in the evaluation system presented above and that is the hierarchy of visual area presented by VIPS. Note that this complication helps to prove that flat structure of results is much more usable for further processing. To eliminate the ambiguity that the hierarchy presents, only the leaf areas of the hierarchy will be used for the evaluation.

To better demonstrate the advantage of the flat output of BCS in the evaluation process, the difference between the two output models is displayed in figure 9.2. The BCS flat model

is quite straightforward – it is just a set of groups, each of which can be further processed right away. The VIPS tree model on the other hand is not that simple. Figure 9.2 visualizes the different levels of the output tree with different shades of gray and the internal consistency level by numbers in the leaf areas. It may be understandable for a human observer; however, in context of an automatic processing, one needs to perform a subsequent analysis of the segmented result to select the right area set. The simplification of picking only the leaf nodes of the tree is the only viable alternative. For that reason, it's possible to consider the flat model being the most distinct advantage of BCS when compared to VIPS and other hierarchy-producing algorithms.

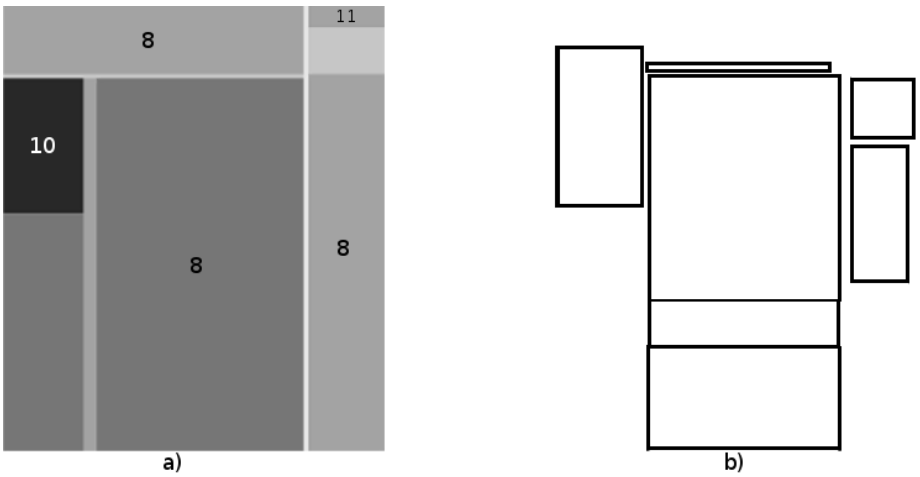

Figure 9.2: Output model comparison: a) VIPS tree model and b) BCS flat model

9.3 Evaluation of BCS with template clustering

The general evaluation focuses on running the proposed method on small chunks of web sites that were presented in section 9.1. The comparison is between the algorithm combining both Box Clustering Segmentation and template clustering and the plain VIPS algorithm. The crawler is given the URL presented in column *site* as a starting point. Since the link extraction is deterministic, the set of pages that the evaluation was performed on remains the same. To reduce the risk of processing completely different content, night hours were chosen to perform the test, as the content of the web sites does not change too much at this time.

The test consisted of running segmentation on 500 web pages and calculating total time it took to perform the segmentation. Equally to the performance test in section 9.2, the timers here were also embedded in the code, so only the real time of segmentation was counted.

Table 9.5 shows the results of the test. The last column of the table is the most informative one. The results clearly say that in the use case of multi page segmentation, the proposed method is on average more than 80 percent more efficient than plain VIPS. In combination with results described in previous section, it's visible that the difference in efficiency is directly correlated with the simplicity of the web site – both in terms of layout simplicity and site structure simplicity (i.e. the number of templates used).

site	**VIPS**	**BCS + template clustering**	savings
businessinsider.com	1 538 298 ms	33 741 ms	97,81%
idnes.cz	1 173 891 ms	262 486 ms	77,64%
novinky.cz	350 613 ms	21 791 ms	93,78%
reuters.com	882 502 ms	38 752 ms	95,61%
yahoo news	1 307 889 ms	40 955 ms	96,87%
e15.cz	669 041 ms	37 939 ms	94,33%
fedora.cz	740 950 ms	16 836 ms	97,73%
gizmodo.com	932 351 ms	31 816 ms	96,59%
lidovky.cz	759 109 ms	81 828 ms	89,22%
slashdot.org	1 990 743 ms	53 739 ms	97,30%
telegraph.co.uk	3 490 337 ms	1 080 180 ms	69,05%

Table 9.5: Performance of the proposed algorithm

Chapter 10

Conclusion

The proposed method can be used in real life applications as a preprocessor of the content on the World Wide Web. Obviously the method offers the greatest potential when used in unsupervised high volume preprocessing. In this context, it means processing thousands of web pages as a stream of data. In these volumes, even a small relative performance boost is significant in absolute numbers. The significance of performance boost this thesis offers is even higher. Even when web pages are not processed in such high volumes, the boost is considerable.

As for precision of the Box Clustering Segmentation algorithm, this thesis demonstrates that the results are comparable to the most commonly used existing method – VIPS. In some cases the precision of BCS algorithm is even better while its stability is slightly better in general.

Some issues of the proposed method remain when segmenting large and complex web pages. Considering the targeted use case, their impact can be perceived as marginal, as the complex web pages usually serve as guide posts, with minimal amount of useful content. If the significance of such pages was higher, there are techniques how to enhance the proposed method to overcome this problem.

10.1 Summary of Contributions

The proposed method presents a completely fresh perspective on the web page segmentation task. The main contributions contained in this thesis include:

1. Performance superiority: the segmentation method proposed in this thesis performs better on every level when compared to current algorithms. The combination of these individual levels multiplies the performance benefit.

2. *Quality of result:* while the overall quality of results is comparable, there are some aspects that make Box Clustering Segmentation better than the rest. Using the flat model for results is much more convenient for consumers of these results, as no further post-processing is required. Better handling of some web pages is also one of the significant contributions this thesis makes.

3. *Pure vision-based method:* even though previous methods claimed to be vision based, this is, to the best knowledge of the author, the first method that is really based purely on visual cues. This contribution is most significant on highly dynamic web pages where other cues, such as DOM tree features, might be highly misleading.

4. *Re-usability:* the fact that the Box Clustering Segmentation segmentation algorithm is not tightly bound to the DOM model implies that it can be used for other document types, if there is a convenient parser that provides the right input to the segmentation algorithm.

5. *Boost of segmentation methods in general:* the template clustering approach not being tightly bound to the Box Clustering Segmentation presents a way how to significantly improve the performance of all vision-based segmentation algorithms in the field.

Aside from these main contributions, this thesis offers some that are more general but almost equally important.

1. *Practical application:* we rarely want to segment just one page and never visit it again. While the existing methods focus on this only task, this thesis provides more complex view of the segmentation problem. When considering the big picture (i.e. how is the algorithm likely to be used), the design of the methods shifts to address the specific needs of targeted use cases.

2. *New research areas:* this thesis proofs that clustering techniques can be successfully used at different levels of the web page preprocessing. While there was some existing work in the area before, the author is not aware of any comprehensive research on this topic. The research provided in this thesis can serve as a base for future scientific work in this area.

3. *Connection of two different research areas:* to author's best knowledge, this thesis is the first one that combines web page segmentation with template detection to get the benefits of both while minimizing their drawbacks.

Finally, one of the contributions of this thesis is a working implementation of both techniques described in this thesis. The architecture of this implementation allows simple utilization of each part separately.

10.2 Possible Improvements and Future Work

The most obvious area that needs to be improved is the segmentation of complex web pages that don't contain any larger continuous regions. One way to address this is to pay attention to more visual cues, as the subset that the algorithm now uses is quite minimal.

Another way that has a potential to improve segmentation result on complex pages is pattern analysis. Most of these complex web pages consist of large number of boxes that are organized in patterns so their identification has a lot of potential. Naturally, this will have impact on all web pages, not just the complex ones.

The last potential area of research is adaptive segmentation. Such approach would mean to perform a preliminary scan of a page and then application of different algorithms or at least algorithm parameters on different parts of the web page. This would improve results on those web pages where the layout complexity varies in different areas (e.g. some structure at the top of the page, solid content in the middle and complex footer).

Bibliography

[1] Hamed Ahmadi and Jun Kong. User-centric adaptation of web information for small screens. *Journal of visual languages & computing*, 23(1):13–28, 2012.

[2] Elgin Akpinar and Yeliz Yesilada. Vision based page segmentation: Extended and improved algorithm. Technical Report eMINE Technical Report Deliverable 2 (D2), Middle East Technical University, Ankara, Turkey, 2012.

[3] M. Elgin Akpinar and Yeliz Yesilada. Vision based page segmentation algorithm: Extended and perceived success. In *Revised Selected Papers of the ICWE 2013 International Workshops on Current Trends in Web Engineering - Volume 8295*, pages 238–252, New York, NY, USA, 2013. Springer-Verlag New York, Inc.

[4] Julián Alarte, David Insa, Josep Silva, and Salvador Tamarit. Temex: The web template extractor. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 155–158, New York, NY, USA, 2015. ACM.

[5] Derar Alassi and Reda Alhajj. Effectiveness of template detection on noise reduction and websites summarization. *Information Sciences*, 219:41 – 72, 2013.

[6] Sadet Alcic and Stefan Conrad. Page segmentation by web content clustering. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, WIMS '11, pages 24:1–24:9, New York, NY, USA, 2011. ACM.

[7] David Alman. Industrial colour-difference evaluation. Technical report, Internation Commission on Illumination, 1995.

[8] 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.

[9] Jayendra Barua, Dhaval Patel, and Ankur Kumar Agrawal. Removing noise content from online news articles. In *Proceedings of the 20th International Conference on Management of Data*, COMAD '14, pages 113–116, Mumbai, India, India, 2014. Computer Society of India.

[10] Bert Bos, Tantek Celik, Ian Hickson, and Håkon Wium Lie. Cascading style sheets level 2 revision 1 (CSS 2.1) specification. W3C Recommendation, June 2011.

[11] Zhan Bu, Chengcui Zhang, Zhengyou Xia, and Jiandong Wang. An far-sw based approach for webpage information extraction. *Information Systems Frontiers*, 16(5):771–785, 2014.

[12] Radek Burget. Visual area classification for article identification in web documents. In *Proceedings of the 2010 Workshops on Database and Expert Systems Applications*, DEXA '10, pages 171–175, Washington, DC, USA, 2010. IEEE Computer Society.

[13] 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.

[14] Michael Cormier, Karyn Moffatt, Robin Cohen, and Richard Mann. Purely vision-based segmentation of web pages for assistive technology. *Computer Vision and Image Understanding*, 2016, 2016.

[15] Hassan F. Eldirdiery and A. H. Ahmed. Detecting and removing noisy data on web document using text density approach. *International Journal of Computer Applications*, 112(5):32–36, February 2015.

[16] Fabio Fumarola, Tim Weninger, Rick Barber, Donato Malerba, and Jiawei Han. Extracting general lists from web documents: A hybrid approach. In *Proceedings of the 24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems Conference on Modern Approaches in Applied Intelligence - Volume Part I*, IEA/AIE'11, pages 285–294, Berlin, Heidelberg, 2011. Springer-Verlag.

[17] Bo Gao and Qifeng Fan. Multiple template detection based on segments. In *Advances in Data Mining. Applications and Theoretical Aspects*, pages 24–38. Springer, 2014.

[18] Tali Garsiel. How browsers work: behind the scenes of modern web browsers. `http://taligarsiel.com/Projects/howbrowserswork1.htm`, October 2009.

[19] 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.

[20] 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.

[21] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 775–784, New York, NY, USA, 2011. ACM.

[22] 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.

[23] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[24] K. Jiang and Y. Yang. Noise reduction of web pages via feature analysis. In *Information Science and Control Engineering (ICISCE), 2015 2nd International Conference on*, pages 345–348, April 2015.

[25] V Kalaivani and K Rajkumar. Reappearance layout based web page segmentation for small screen devices. *International Journal of Computer Applications*, 49(20), 2012.

[26] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM.

[27] J. Kong, O. Barkol, R. Bergman, A. Pnueli, S. Schein, K. Zhang, and C. Zhao. Web interface interpretation using graph grammars. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):590–602, July 2012.

[28] RA Kreuzer, Jurriaan Hage, and Ad Feelders. A quantitative comparison of semantic web page segmentation algorithms. Master's thesis, Universiteit Utrecht, Faculty of Science, 2013.

[29] S. S. Krishna and J. S. Dattatraya. Schema inference and data extraction from templatized web pages. In *Pervasive Computing (ICPC), 2015 International Conference on*, pages 1–6, Jan 2015.

[30] AH Kulkarni and BM Patil. Template extraction from heterogeneous web pages with cosine similarity. *International Journal of Computer Applications*, 87(3):5, 2014.

[31] Harshal H Kulkarni and Manasi K Kulkarni. Template extraction from heterogeneous web pages. *International Journal of Electrical, Electronics and Computer Engineering*, 4(1):125, 2015.

[32] Kaushal Kumar and Fungayi Donewell Mukoko Abhaya. Pagerank algorithm and its variations: A survey report. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 14(1):38–45, 2013.

[33] 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.

[34] Long Li, An Min Zhou, Yong Fang, Liang Liu, and Qian Wu. An improved VIPS-based algorithm of extracting web content. In *Material Science, Civil Engineering and Architecture Science, Mechanical Engineering and Manufacturing Technology II*, volume 651 of *Applied Mechanics and Materials*, pages 1806–1810. Trans Tech Publications, 11 2014.

[35] 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.

[36] Xinyue Liu, Hongfei Lin, and Ye Tian. Segmenting webpage with gomory-hu tree based clustering. *Journal of Software*, 6(12):2421–2425, 2011.

[37] Erik Lundgren, Panagiotis Papapetrou, and Lars Asker. Extracting news text from web pages: An application for the visually impaired. In *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '15, pages 68:1–68:4, New York, NY, USA, 2015. ACM.

[38] M. R. Luo, G. Cui, and B. Rigg. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research and Application*, 26(5):340–350, 2001.

[39] Tomohiro Manabe and Keishi Tajima. Extracting logical hierarchical structure of html documents based on headings. *Proceedings of the VLDB Endowment*, 8(12):1606–1617, 2015.

[40] T. H. Nelson. Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference*, ACM '65, pages 84–100, New York, NY, USA, 1965. ACM.

[41] Jakob Nielsen. User interface directions for the web. *Commun. ACM*, 42(1):65–72, January 1999.

[42] Tomáš Popela. Implementace algoritmu pro vizuální segmentaci www stránek. Master's thesis, Brno University of Technology, Faculty of Information Technology, 2012.

[43] 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.

[44] Waseem Safi, Fabrice Maurel, Jean-Marc Routoure, Pierre Beust, and Gaël Dias. A Hybrid Segmentation of Web Pages for Vibro-Tactile Access on Touch-Screen Devices. In *3rd Workshop on Vision and Language (VL 2014) associated to 25th International Conference on Computational Linguistics (COLING 2014)*, pages 95 – 102, dublin, Ireland, Aug 2014.

[45] A. Sanoja and S. Gançarski. Block-o-matic: A web page segmentation framework. In *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, pages 595–600, April 2014.

[46] Stanley M Selkow. The tree-to-tree editing problem. *Information processing letters*, 6(6):184–186, 1977.

[47] Abhay Sharma. *Understanding Color Management*. Graphic Design/Interactive Media Series. Thomson/Delmar Learning, 2004.

[48] Aditi Sharma, Nishtha Adhao, and Anju Mishra. A survey: Static and dynamic ranking. *International Journal of Computer Applications*, 70(14), 2013.

[49] Dilip Kumar Sharma and AK Sharma. A comparative analysis of web page ranking algorithms. *International Journal on Computer Science and Engineering*, 2(08):2670–2676, 2010.

[50] Shengsheng Shi, Chengfei Liu, Yi Shen, Chunfeng Yuan, and Yihua Huang. Autorm: An effective approach for automatic web data record mining. *Knowledge-Based Systems*, 89:314–331, 2015.

[51] Dandan Song, Fei Sun, and Lejian Liao. A hybrid approach for content extraction with text density and visual importance of dom nodes. *Knowledge and Information Systems*, 42(1):75–96, 2015.

[52] Mark MCCAHILL Tim BERNERS-LEE, Larry MASINTER. Rfc 1738. `http://tools.ietf.org/html/rfc1738`, December 1994.

[53] E. Uzun, H. V. Agun, and T. Yerlikaya. Web content extraction by using decision tree learning. In *2012 20th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, April 2012.

[54] Erdinç Uzun, Hayri Volkan Agun, and Tarık Yerlikaya. A hybrid approach for extracting informative content from web pages. *Information Processing & Management*, 49(4):928 – 944, 2013.

[55] 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.

[56] Anne van Kesteren, Aryeh Gregor, Alex Russell, and Robin Berjon. W3c dom4. W3C Recommendation, November 2015.

[57] 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.

[58] T. Wei, Y. Lu, X. Li, and J. Liu. Web page segmentation based on the Hough transform and vision cues. In *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 865–872. IEEE, 2015.

[59] Daiyue Weng, Jun Hong, and David A. Bell. Extracting data records from query result pages based on visual features. In *Advances in Databases: 28th British National Conference on Databases, BNCOD 28, Manchester, UK, July 12-14, 2011, Revised Selected Papers*, pages 140–153, Berlin, Heidelberg, 2011. Springer.

[60] Daiyue Weng, Jun Hong, and David A. Bell. Automatically annotating structured web data using a svm-based multiclass classifier. In *Web Information Systems Engineering – WISE 2014: 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceedings, Part I*, pages 115–124, Cham, 2014. Springer International Publishing.

[61] Yu-Chieh Wu. Language independent web news extraction system based on text detection framework. *Information Sciences*, 342:132 – 149, 2016.

[62] Zhen Xu and James Miller. Identifying semantic blocks in web pages using gestalt laws of grouping. *World Wide Web*, pages 1–22, 2015.

[63] 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.

[64] 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.

[65] Jan Zeleny, Radek Burget, and Jaroslav Zendulka. Box clustering segmentation: A new method for vision-based web page preprocessing. *Information Processing and Management*, 2017.

[66] Jun Zeng, Brendan Flanagan, Sachio Hirokawa, and Eisuke Ito. A web page segmentation approach using visual semantics. *IEICE Transactions on Information and Systems*, E97-D(2):223–230, February 2014.

[67] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Inf. Process. Lett.*, 42:133–139, May 1992.

[68] W. Zhu, S. Dai, Y. Song, and Z. Lu. Extracting news content with visual unit of web pages. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on*, pages 1–5, June 2015.