Web page segmentation, evaluation and applications

Andrés Sanoja Vargas

Thése de Doctorat de l'Université Pierre et Marie Curie

Specialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Andrés Fernando SANOJA VARGAS

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Segmentation de Pages Web, Évaluation et Applications

Web Page Segmentation, Evaluation and Applications

---

soutenue le 22 janvier 2015, devant le jury composé de :

Elisabeth MURISASCO	Rapporteur	Université de Toulon
Marta RUKOZ	Rapporteur	Université de Paris Ouest Nanterre
Matthieu CORD	Examinateur	UPMC Paris 6
Luc BOUGANIM	Examinateur	INRIA Paris-Rocquencourt
Pierre SENELLART	Examinateur	Télécom ParisTech
Stéphane GANÇARSKI	Directeur de thèse	UPMC Paris 6

"Satisfaction lies in the effort, not in the attainment, full effort is full victory."

Mahatma Gandhi

"La utopía está en el horizonte. Camino dos pasos, ella se aleja dos pasos y el horizonte se corre diez pasos más allá. ¿Entonces para que sirve la utopía? Para eso, sirve para caminar."

Eduardo Galeano

Abstract

Web pages are becoming more complex than ever, as they are generated by Content Management Systems (CMS). Thus, analyzing them, *i.e.* automatically identifying and classifying different elements from Web pages, such as main content, menus, among others, becomes difficult. A solution to this issue is provided by Web page segmentation which refers to the process of dividing a Web page into visually and semantically coherent segments called blocks. The quality of a Web page segmenter is measured by its correctness and its genericity, *i.e.* the variety of Web page types it is able to segment. Our research focuses on enhancing this quality and measuring it in a fair and accurate way. We first propose a conceptual model for segmentation, as well as Block-o-Matic (BoM), our Web page segmenter. We propose an evaluation model that takes the content as well as the geometry of blocks into account in order to measure the correctness of a segmentation algorithm according to a predefined ground truth. The quality of four state of the art algorithms is experimentally tested on four types of pages. Our evaluation framework allows testing any segmenter, *i.e.* measuring their quality. The results show that BoM presents the best performance among the four segmentation algorithms tested, and also that the performance of segmenters depends on the type of page to segment. We present two applications of BoM. Pagelyzer uses BoM for comparing two Web pages versions and decides if they are similar or not. It is the main contribution of our team to the European project Scape (FP7-IP). We also developed a migration tool of Web pages from HTML4 format to HTML5 format in the context of Web archives.

Keywords: Web page segmentation, Web applications, Evaluation, Web page analysis

Résumé

Les pages web sont devenues plus complexes que jamais, principalement parce qu'elles sont générées par des systèmes de gestion de contenu (CMS). Il est donc difficile de les analyser, c'est-à-dire d'identifier et classifier automatiquement les différents éléments qui les composent. La segmentation de pages web est une des solutions à ce problème. Elle consiste à décomposer une page web en segments, visuellement et sémantiquement cohérents, appelés blocs. La qualité d'une segmentation est mesurée par sa correction et sa généricité, c'est-à-dire sa capacité à traiter des pages web de différents types. Notre recherche se concentre sur l'amélioration de la segmentation et sur une mesure fiable et équitable de la qualité des segmenteurs. Nous proposons un modèle pour la segmentation ainsi que notre segmenteur Block-o-Matic (BoM). Nous définissons un modèle d'évaluation qui prend en compte le contenu ainsi que la géométrie des blocs pour mesurer la correction d'un segmenteur par rapport à une vérité de terrain. Ce modèle est générique, il permet de tester tout algorithme de segmentation et observer ses performances sur différents types de page. Nous l'avons testé sur quatre segmenteurs et quatre types de pages. Les résultats montrent que BOM surpasse ses concurrents en général et que la performance relative d'un segmenteur dépend du type de page. Enfin, nous présentons deux applications développées au dessus de BOM. Pagelyzer compare deux versions de pages web et décide si elles sont similaires ou pas. C'est la principale contribution de notre équipe au projet européen Scape (FP7-IP). Nous avons aussi développé un outil de migration de pages HTML4 vers le nouveau format HTML5.

Mots clés : segmentation des pages Web, applications Web, Evaluation, l'analyse des pages Web

Acknowledgements

How can I describe my journey known as PhD? Let's say: a plethora of emotions and cacophony of situations that have made me a stronger and wiser person. It has been, certainly, the most important accomplishment of my professional life. What a rush!, it was full of science, knowledge, friendship, culture, love and solidarity. All these years have been shared with many wonderful people that I would like to thank.

I would like to express my gratitude and appreciations to my supervisor. Stéphane Gançarski, who is the person who believed in me, encouraged me, and advised while I was going though difficult moments. Thanks for always teaching me giving the example, your scientific rigor, your commitment, your humanity and solidarity.

Special thanks for Stéphane Gançarski and Ahlem Abbaci who struggled to find grants funding my work. I am really grateful.

I thanks Elisabeth Murisasco, Marta Rukoz, Pierre Senellart, Luc Bougamin, Matthieu Cord for accepting to be part of my PhD jury.

Thanks also to Stéphane Gançarski, Matthieu Cord and Zeynep Pelhivan working with you in the EU project was a wonderful time.

Thanks to the Université Pierre et Marie Curie (UPMC), the Laboratoire d'Informatique de Paris 6 (LIP6) and all their members and authorities.

Anne Doucet, has been a pillar for us during these years. Her advice and support were invaluable and crucial for us. We will be always grateful of that.

It is a great privilege to spend these years in the Database group at LIP6, everybody will always remain dear to me. You are, no doubt about it, the coolest team. My special thanks for Bernd Amann, Hubert Naacke, Mohamed-Amine Baazizi, Camélia Constantin and Benjamin Piwowarski. To current and past PhD students Miriam, Roxana, Zeynep, Clément, Yifan, Kun, Ndiouma, Ibrahima, and especially my fellow office-mates Jordi and Nelly. Nelly thank for your friendship, support and advice, I am really grateful.

I would like to thanks Claudia Leon for her advice and friendship, and her encouragement to come to France for my PhD. Thanks to Eugenio Scalise for his advice in the beginnings of my PhD. I thanks the Universidad Central de Venezuela (UCV), its members and authorities.

I thank Martine Movosine for helping me and my family in our adaptation to the Parisian life.

I wish to thank my parents. You have been a role model for each one of us. Always caring for us, teaching us, encouraging us to be better persons, the family values and the commitment. Thanks for being the coolest parents ever!

To my dear wife Carolina Sierraalta, my soul mate, my friend, my "media naranja", this achievement is also yours. I know these three years has been tough, but we demonstrate that our love is more powerful that anything. There are more things to come in our future but the key is to keep confronting them together as a team. Thanks to my children, Fernando and Mariana, for your patience, love and been the coolest kids!

Andrés

Contents

List of Figures

List of Tables

A mis queridos hijos Fernando y Mariana

Introduction

The main focus of this PhD thesis is to study the *Web page segmentation* and its *correctness evaluation*. In this section, we describe our motivations and the research issues. At the end of the chapter, we present its overall organization.

Web pages are becoming more complex than ever, as they are usually not designed manually but generated by Content Management Systems (CMS). Thus, analyzing them automatically (*i.e.* identifying and classifying different elements from Web pages, such as main content, menus, user comments, advertising among others), becomes difficult. A solution to this issue is provided by Web page segmentation. Web page segmentation refers to the process of dividing a Web page into visually and semantically coherent segments called *blocks*.

Detecting blocks in a Web page is a crucial step for many applications, such as mobile applications, information retrieval, Web archiving, among others. For instance, in the context of Web archiving, segmentation can be used to extract interesting parts to be stored. By giving relative weights to blocks according to their importance, it also allows the detection of relevant changes (changes in important blocks) from distinct versions of a page. This is useful for crawling optimization, as it allows tuning of crawlers so that they will revisit pages with important changes more often [BSG11]. It also helps controlling curation actions such as migrating a Web archive from ARC to WARC format, by comparing the page version before and after the action. If the segmentation of the after-version is equal to the before-version, then there is a high probability that the action is performed correctly. Mobile applications use segmentation to optimize the visualization of a Web page in small screen devices [CXMZ05]. For instance, mobile devices use the zooming technique to show details of a Web page to the user. This technique is time consuming and the time response is high. Using the segmentation, instead of zooming the device presents to the user relevant blocks found in the segmentation, decreasing the response time and user experience. Web archives can exploit the Web page segmentation for migrating from one format to another. For instance migrating Web pages from

HTML4 to HTML5 format in order to reduce the dependency of emulation.

One of the main issues in Web page segmentation are the precision and genericity. A segmentation is *precise* if its granularity is equal (or very close) to the granularity of an ideal segmentation. A segmentation is *generic* if it performs well on (almost) all the different types of Web pages. The *granularity* of a segmentation represents to which extent a segmentation divides a Web page into blocks.

Figure 1 shows an example of a segmentation which is not precise. The ideal segmentation (at left) shows that the Web page should be divided in six blocks: block 1 the header, block 2 the title and identification of the author, block 3 the social media, block 4 the main article, block 5 the related links and 6 the footer. The computed segmentation (at right)[1] has four blocks, two of them, blocks 1 and 4 being equals to the ones of the ideal segmentation. The computed segmentation has merged the title and social media blocks, and the main article with the related links. Applications that depend on segmentation will not have precise information since, for example, the title of the page is mixed with noisy social media and the main article is associated with links that are not relevant to the content.

FIGURE 1: Example segmentation with poor precision

[1] VIPS algorithm is used to obtain the computed segmentation

Figure 2 shows two segmentations using the same segmenter[2] in two different types of pages. The Web page at the left is a forum segmented with high precision. The segmenter is able to find all the elements relevant in a forum: header, question post, global announcements, all the replies and the footer. At the right of the figure a blog page is segmented with poor precision, using the same segmenter. The header and the footer are correctly detected but the navigation, the main content and the related information are merged into one big blog. This is an example of a segmenter which is not generic, because it is precise only for certain type of pages.

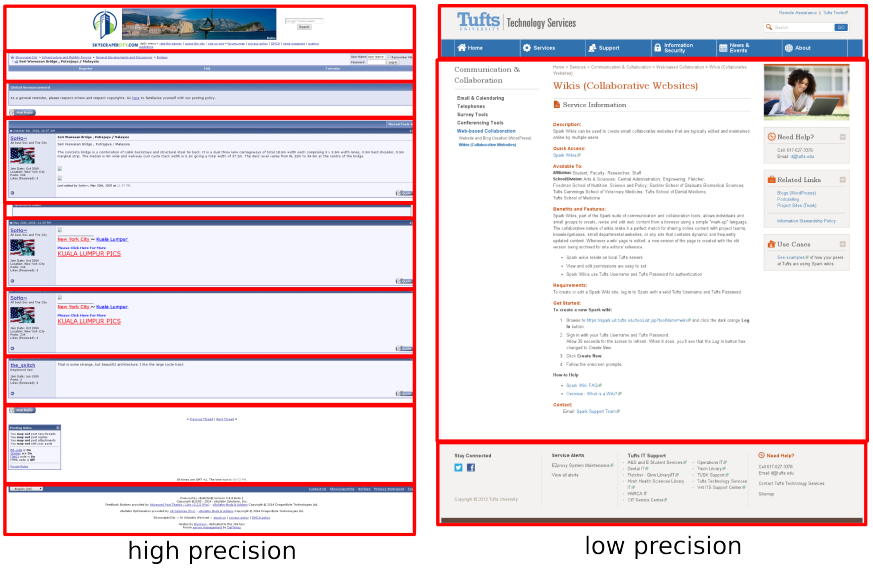


FIGURE 2: Example segmenter not generic

Most of existing segmenters pretend to emulate the user perspective by the means of heuristic rules. Even formal approaches relies on these rules to find blocks. State of the art algorithms are devoted to particular application needs and have customized heuristic rules to find blocks. As a consequence there is a risk that they do not process properly all the elements in the content of a Web page. This leads to lack of precision and genericity.

In order to solve these issues we developed Block-o-Matic (BoM), a new segmenter which takes these two characteristics, precision and genericity, into account. We designed the algorithm using the bottom-up strategy and following the vision-based approach. We base our work on the W3C set of heuristic rules inherent for Web pages, particularly using the W3C standard content categories. Using these rules for detecting and classifying blocks gives genericity to our approach. To give a solution to the precision issue, blocks are merged considering their size, their position and their label. They are finally organized in composite blocks (which define the layout of the page). Intuitively, this approach result in a more precise and generic segmentation. However, intuition it is not

[2] JVIPS algorithm is used to obtain both segmentations

enough. We need to perform an evaluation which allows us measuring these two aspects of the BoM algorithm.

When studying the literature about Web page segmentation, we noticed that there is no full comparative evaluation of Web page segmentation algorithms. This is due to a wide diversity of goals, a lack of a common dataset, and the lack of meaningful quantitative evaluation schemes. Thus, we decided to define common measures for better evaluating the correctness of Web page segmentation algorithms.

Different interpretations of correctness can be used with respect to segmentation. As defined in the literature, the correctness of an algorithm is asserted when it complies with a given specification. In the case of Web page segmentation, such a specification cannot be established *a priori*, without a human judgment. Thus, we focus on evaluation approaches based on a ground truth and developed a framework that includes correctness metrics, an aggregated score and a tool that eases the manual design of a ground truth by assessors.

We check if the ground truth matches with the computed segmentation given a vector of metrics. These metrics are devoted to measure to what extent the segmentation differs from the ideal segmentation. In the evaluation we consider the content and the blocks rectangles given by segmenters. We built a dataset of 125 Web pages organized into five type of pages. The pages were segmented by assessors and we tested four state of the art Web page segmentation algorithms. The implementations of these algorithms were adapted in order to fit into our evaluation framework. This approach to Web page segmentation evaluation allows us measuring segmenters quality and giving observations in terms of precision and genericity.

When exploring the connex domain of scanned document image segmentation, we found that there are common problems in the segmentation and evaluation of Web pages and scanned pages segmentation algorithms. These methods can not be used in a straightforward way with Web pages, however they gave us ideas and inspiration. Adapting this methods to Web pages is challenging, since both document types are different.

Our contribution consists in Block-o-Matic (BoM), a new approach to Web page segmentation as well as a framework for its evaluation and two applications of BoM. We give below a summary of those contributions, indicating the chapters where the details can be found.

Contribution 1 - Block-o-Matic We propose an original Web page segmentation model based on the heuristic rules found in the W3C standard specification. This contribution allows solving the issues of precision and genericity in Web page segmentation. It is presented in detail in Chapter 2.

Contribution 2 - Segmentation evaluation We propose an evaluation model that exploits the content as well as the geometry of blocks in order to measure the correctness of a segmentation algorithm according to a predefined ground truth. This contribution is studied in detail in Chapter 3. The correctness of four state of the art algorithms (including BoM) is experimentally tested on four types of pages (blog, enterprise, forum, picture and wiki). We give the results and discuss them in chapter 4.

Contribution 3 - Applications We present two applications of the Block-o-Matic Web page segmentation algorithm. We present an application, Pagelyzer, that uses BoM for comparing two Web pages versions and decides if they are similar or not. Pagelyzer is a tool developed in the context of the European project SCAPE. We present the use of BoM for the migration of Web pages from HTML4 format to HTML5 format in the context of Web archives. This is useful as this relieves archivist of keeping old HTML4 rendering engines. These two applications are presented in detail in Chapter 5.

This thesis is organized in 5 chapters. Chapter 1 presents the state of the art in Web page segmentation and its evaluation. Chapter 2 presents Block-o-Matic, a new Web page segmentation algorithm. Chapter 3 presents the evaluation model. Chapter 4 presents the evaluation results. Chapter 5 presents the applications. We conclude with a contribution summary and the outlook which presents some possible future works.

Chapter 1

Web Page Segmentation and Evaluation

In this chapter we present the state of the art in Web page segmentation and its evaluation. Web page segmentation consists in identifying and categorizing the regions (or blocks) of interest in a Web page. The Web page segmentation is divided into two main areas: detection and labeling of the different blocks, and the classification of the logical role they play inside the Web page (header, footer, navigation, etc.). Web pages are analyzed in a similar way as scanned documents in the optical character recognition (OCR) domain though both sources are different (elements/text vs. pixel/colors). We took advantage of this similarity in our work.

We describe in this chapter the Web page characteristics (1.2), the Web page segmentation (1.3) and its evaluation (1.4). The Web page characteristics describe useful elements that are taken into account when analyzing a Web page. We present how Web page segmentation algorithms find blocks in the page based on those characteristics. Algorithms can follow two main approaches: basic or hybrid. They can also be designed using the top-down or bottom-up strategies. We also give an introduction to the evaluation of Web page segmentation algorithms: state of the art, evaluation features, and a classification of evaluation methods.

1.1 Preliminars

In this section we describe the Web applications in terms of their structure and their rendering. These concepts are crucial to understand the Web page analysis. First, we describe what is a Web application, how it is structured (1.1.1). Second, we present an overview about the rendering of a Web page (1.1.2). Finally, we give details about the rendered DOM version of a Web page (1.1.3 and 1.1.4).

1.1.1 Web applications

A Web application is an application that runs in a Web browser. It is developed using Web technologies such as *JavaScript*, *Hypertext Markup Language* (HTML) and *Cascading Style Sheets* (CSS). It relies on a Web browser to render the application.

Web applications are structured with the *n-tiers* architectural style [Con02]. The most common structure is the *3-tier* application. The first tier is the Web browser (Firefox, Google Chrome, etc). The second, or middle-tier, is an engine using some Web content technology (PHP, Rails, etc.). The third tier is data storage (databases, documents, etc).

The Web browser sends a request to the middle tier, which runs the queries, possibly updates the data sources and generates a Web page.

Web applications may use more complex structures. However as the Web page segmentation occurs commonly in the Web browser, the way a Web page is build is not relevant for our work and we focus on the first tier only.

1.1.2 Rendering

An HTML document contains instructions for the browser on how the Web page has to be presented to the user.

In order to materialize the Web page, a browser needs to download Web contents from one or several websites. As we mentioned above, content can be HTML code, JavaScript scripts, CSS styles, among others. The role of the rendering engine then is to display the formatted content to user.

Besides the formatted content, the rendering engine produces a Document Object Model[1] (DOM) tree of the page. It is an interface that allows programs and scripts to dynamically access and update the content, structure and style of a rendered page.

An image of the formatted content can be obtained from the rendered DOM. This image is called *rendered image* or *screenshot*.

1.1.3 Rendered DOM

When the rendering is done, a Web page can be build based on the produced DOM tree. This document includes all content, updates and styles processed in the rendering process. We call this document *Rendered DOM* [Goo02]. At this point, the original HTML document and the rendered DOM of the same page are not identical. The rendered one is the HTML source code of the formatted content, while (obviously) the original HTML document only contains the original HTML source code.

Thus, our work is based on this rendered DOM. Building it produces an overhead and thus increases the analysis response time, but the analysis of a Web page is more complete and accurate.

1.1.4 Element positioning

While constructing the rendered DOM, the engine determines the position of elements based on their style properties. An element that is not a text element, determines its own width and that of its children. Children must fit inside the parent box. In some cases, this rule can be broken by positioning elements such as *div* in absolute (or static) way. As a consequence, the order of elements in the DOM tree does not correspond to the order of the formatted content when displayed. In other words, a child can be rendered out of the parent box. That can lead to errors in the segmentation because the content that a block is supposed to cover (or part of it), is rendered in a different region of the page.

[1] http://www.w3.org/DOM

1.2 Web Page Characteristics

In this section we describe the Web pages characteristics. These characteristics are key elements for Web page segmentation algorithms. They are related to the rendering (1.2.1) or to websites (1.2.2). A glossary of terms is presented at the end of the section (1.2.3).

1.2.1 Web page characteristics from the rendered DOM

Web pages are specified by an HTML document (source code), the associated Cascading Style Sheets (CSS) and embedded JavaScript scripts. The rendered DOM W of a Web page is obtained by a rendering engine (*e.g.* WebKit and Gecko) processing the specification (*cf.* 1.1.2). The characteristics of W are: the content, the visual presentation and the positioning scheme. The visual presentation and positioning scheme are described in detail in [CYWM04].

1.2.1.1 Content

An element is an object in the rendered DOM of a Web page. It always has an element name and may also have attributes, child elements and text. The content of a Web page is the root of the rendered DOM tree ($W.root$) and the text ($W.text$). $W.text$ is the result of concatenating all the text of the descendant of $W.root$. The *CDATA* sections (textual part of a document that is not parsed) are not considered. Usually these sections include, for instance JavaScript code or other information not directly related to the textual content of the page.

According to the HTML5 specification, each element belongs to a *content category*, or simply *category*. For instance, an element either belongs to the *flow content* or *phrasing content* (*block-level* category and *inline-level* category respectively in HTML 4.01 specification). Appendix A list all the elements, content categories and the exceptions.

A category describes how the element is rendered, how it should be processed and the type of content it can have. For instance, the element <article> belongs to the category *sectioning content*. That makes it not visible in the formatted content (*cf.* Section 1.1.2) but it organizes the content. Now consider, an element <p> of the *flow content* category. Its rendering affects the formatted content by reserving some space for itself

and its children elements (*cf.* Section 1.1.4). Elements belonging to *sectioning content* are interpreted as blocks explicitly coded by the Web developer.

1.2.1.2 Visual Presentation

To facilitate browsing and attract attention, Web pages usually contain much visual information in the HTML tags and attributes [YZ01]. The visual presentation is the visual style of elements (visual cues). Typical visual cues include lines, blank areas, colors, pictures, fonts, etc. Visual cues are very helpful to detect the segments in Web pages. Usually, they are good candidates to be borders of blocks. Visual cues are related to the CSS styles used in elements.

1.2.1.3 Positioning Scheme

Each element in the rendering of a Web page can be related with other elements from up to four directions (up, down, left and right) and may contain (be contained in) some other ones.

At rendering time, each element is materialized as a rectangular *box* [2] following the rules defined in the Visual Formating Model [3]. Block-level elements are containers for other elements. From these two models we enumerate three concepts that helps to define the geometric characteristics of a Web page:

- *Viewport*: the rectangle associated with the *body* element which works as initial container.
- *Box*: the rectangle corresponding to an element.
- *Scheme*: describe how boxes of a viewport are located with respect to each other.

1.2.2 Characteristics related to the website

A website has a function as a whole. Each page makes a contribution (blog, forum, etc.) to this function. This implies that the page has to cover certain functionalities, such

[2] http://www.w3.org/TR/CSS2/box.html
[3] http://www.w3.org/TR/CSS2/visuren.html

as the type of navigation used, or the type of layout according to that function. There are several tools that help developers reaching the expected goal of a website, mainly: *templates* and *Content Management Systems* (CMS).

A template is a pre-designed Web page, or set of Web pages, that anyone can modify/fill with its own content and images to setup a website. Templates allow to setup a website making sure that it has all the functionalities required by its function. A CMS is a Web application that allows editing and publishing website content from a central interface like a Web browser. Its setup can be more complex than a template but it has the advantage of being dynamic and a variety of templates can by applied to the same content.

Whatever the site type (template-based or CMS-based), we agree with [FdMdS+11] that there are three important characteristics related to websites in the context of Web page segmentation, we enumerate them:

- *Function*: is the purpose or objective of the website (*i.e.* a blog, a forum, ...)
- *Template elements*. Elements of a Web page that are present in other pages of the same website. All of them together form the layout of the page.
- *Label* represent the role of these elements in the website. For instance, navigation bars, copyrights, main content and advertisements.

The template elements are important to the segmentation since usually these elements are used for organizing content, therefore it is highly probable that it organize smaller blocks as well.

1.2.3 Glossary

The table below presents the terms described in this section and used along the whole document.

Term	**Description**
Elements	The objects in the rendered DOM W.
$W.root$	The root element of the rendered DOM.
$W.text$	The result of concatenating all the text of the descendant elements of $W.root$
Category	The content category of elements in the Web page. For example, flow content, phrasing content, sectioning content, etc. Appendix A has a complete description of HTML5 content categories, elements and their exceptions.
Visual cues	Formatting properties of elements obtained at rendering time. For instance: lines, blank areas, colors, pictures, fonts, etc
Box	How the element's rectangle shape is to be drawn and how its size is computed
Viewport	Define the size of the *body* element which works as initial container
Function	The website goal. For instance: blog, wiki, etc.
Scheme	Defines how a box's coordinates are computed with respect to another container box.
Template elements	Elements of a Web page that are repeated in several pages of the same website.

1.3 Web page segmentation

The segmentation of a Web page into meaningful components has emerged as an important Web document analysis task, since it supports many important applications.

Web page segmentation refers to the process of dividing a Web page into visually and semantically coherent segments called blocks. Detecting these different blocks is a crucial step for many applications, such as mobile devices [XTL08], information retrieval [CYWM03], Web archiving [SG10], Web accessibility [MBR07], evaluating visual quality (aesthetics) [WCLH11], among others. In the context of Web archiving, segmentation can be used to extract interesting parts to be stored. By giving relative weights to blocks according to their importance, it also allows for detecting important changes (changes in important blocks) between pages versions [PBSG10]. This is useful for crawling optimization, as it permits tuning crawlers so that they will revisit pages with important changes

more often [SG10]. It also helps for controlling preservation actions, by comparing the page version before and after the action.

We first define the concepts and notation (1.3.1 and 1.3.2), the Web page segmentation approaches (1.3.4 and 1.3.5), the relationship between document processing and Web page segmentation (1.3.7), a summary table of segmentations algorithms (1.3.8). Then we end with a discussion (1.3.9).

1.3.1 Concepts

Inspired by the concepts presented by Tang [TS94] and Nie [NWM09], we describe the Web page segmentation with the following abstractions:

- *Page* is a special block that represents the whole Web page and covers the whole Viewport.
- *Simple block* is an element or a group of elements. It is also denoted simply as *Block*. It is represented as a rectangular area resulting of merging the boxes of elements. Each block has a label related with those of the underlying elements. It is also associated with the text of those elements.
- *Composite block* is a special block that can contain other blocks. Usually such blocks correspond to template elements.
- *Block graph* is a connected planar graph representing the blocks and their relationships (*e.g.* parent/child). It can be an edge-weighted graph (each edge has a weight), or a vertex-weighted graph (each vertex has been assigned a weight). A weight associated with a vertex usually represents how coherent a blocks is, while a weight associated with an edge usually represents the cost of merging two blocks, distance or similarity between blocks.
- *Geometric model* represents the set of blocks as a set of rectangles in a plane. They are obtained from the scheme of the Web page. All rectangles are modeled as quadruples (x,y,w,h), where x and y are the coordinates of the origin point and w and h are the width and height of the rectangle. Blocks can be represented in the plane as a hierarchy or a set of non-overlapping rectangles, called Manhattan layout [TS94]. It can be hierarchical [CYWM03] or non-hierarchical [CKP08, KN08]. The latter can be obtained from the former by only considering the leaves.

- *Stop condition* is a predefined value (real number) used by algorithms that indicates when a segmentation is achieved. It its based on the edge/vertex weights of the block graph. An algorithm may have one or more stop conditions.
- *Label* is the role that a block plays in the Web page such as navigation, content, header, footer, etc.

1.3.2 Notation

We present in this section several definitions, in order to have an uniform presentation of Web page segmentations algorithms: all Web page segmentations algorithms presented in the chapters 2, 3, 4 and 5 are described using this notation.

1.3.2.1 The segmentation function

The segmentation function Φ is described as follows:

$$\Phi_A\,(W,\,SC) \longrightarrow (W'_A, GM_A) \tag{1.1}$$

where A is a Web page segmentation algorithm, W is the rendered DOM of a Web page, SC is a set of stop conditions. W'_A is the block graph defined just below and GM_A is a set of rectangles representing the geometric model of the segmentation.

1.3.2.2 The block graph

The block graph is defined as a planar graph $W'_A = (Blocks, Edges)$. Each vertex B in $Blocks$ corresponds to a rectangle in GM_A (denoted $B.rect$) and a label (denoted $B.label$). It is associated with a function $weight$ on the edges and vertices, and two subset of vertices: $SimpleBlocks \subset Blocks$ (also called terminals), $CompositeBlocks \subset Blocks$, which includes a special vertex $Page$, labeled as the root of the graph.

The rectangle of the vertex $Page$ covers the whole viewport of the Web page W and all the blocks fit in. Thus,

$$\forall\, B \in Blocks,\, B.rect \subseteq Page.rect$$

The weight of a vertex B is noted as $B.weight$. The weight of an edge E is noted as $E.weight$

Usually the block graph is a tree. However, some algorithms such as Homory-HuPS [LLT11] and GraphBased [CKP08] define it as a general planar graph.

In the following section we describe the different strategies used to design Web page segmentation algorithms: top-down and bottom-up.

1.3.3 Top-down versus bottom-up

There are mainly two kinds of strategies: top-down page segmentation and bottom-up page segmentation. Each strategy guides the blocks extraction, and defines the way the rendered DOM tree is traversed. These strategies are briefly described in [AC11].

1.3.3.1 Top-down strategies

Top-down strategies start with the complete Web page as a block (Page) and partition this block iteratively into smaller blocks using different features of the content of the Web page.

A good example of a top-down algorithm is the VIPS algorithm [CYWM03], detailed in Section 1.3.5.1. It describes the block graph as a vertex-weighted tree. The algorithm assigns to each block a weight value (Degree of Coherence or DoC), between 1 and 10, indicating how coherent blocks are. The algorithm defines the stop condition as the *Permitted Degree of Coherence* (PDoC) that is established a priori and is used as parameter. The algorithm stops if $DoC > PDoC$.

The example of Figure 1.1 has a PDoC value of 5. The Web page is first divided into three big blocks (blocks 1,2 and 3) and the block graph is set accordingly. Figure 1.1a shows the weight values assigned to each block .

For block 2, the same segmentation process is carried out recursively until we get blocks where the value of DoC is less than PDoC. As a consequence, block 2 becomes a composite block, and it is divided in 5 sub-blocks (blocks 2.1 to 2.5). Blocks 1 and 3 are left untouched (Figure 1.1b). The algorithm stops because the stop condition is met, i.e. the DoC of each block is greater than the PDoC.

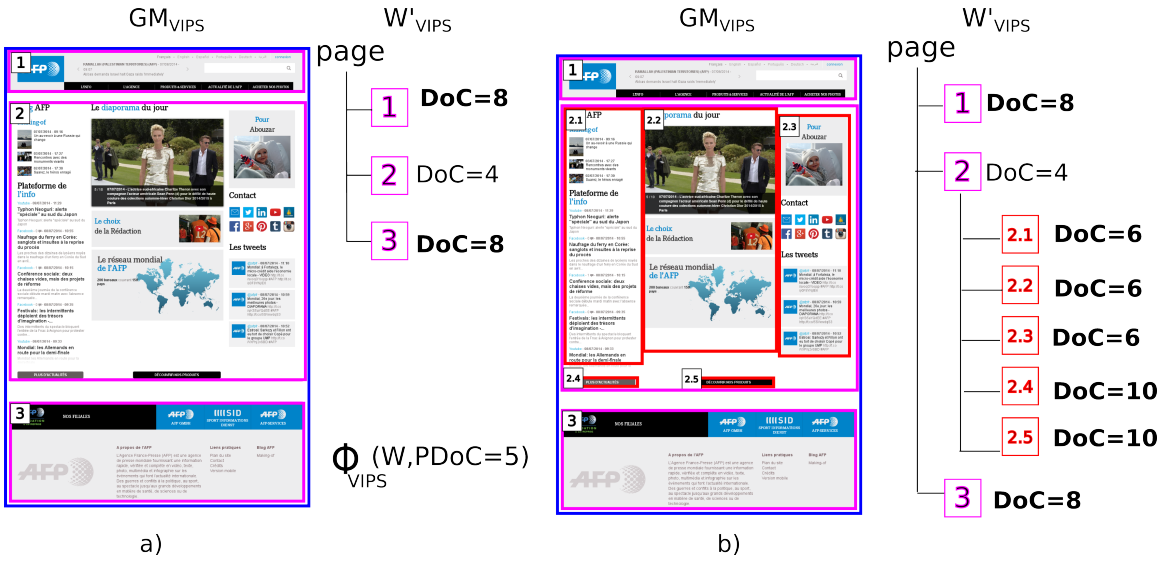


FIGURE 1.1: Top-Down Segmentation strategy used in VIPS algorithm

1.3.3.2 Bottom-up strategies

Bottom-up strategies start by selecting the leaf nodes in the DOM tree, which are considered as atomic units. They are aggregated with other nodes, in a iterative manner, until the stop condition is met.

One example of this approach is the Gomory-HuPS Algorithm [LLT11]. The block graph is represented as an edge-weighted planar graph. First the graph is constructed. Each graph vertex (block) represents content information based on the DOM structure and the visual information. DOM nodes are selected as blocks if they contain text, links and pictures as children. In most of the cases these elements are leaves in the DOM tree. Edges are added based on the location of blocks in the geometric model, *i.e.* an edge is added between two blocks if they are closest neighbors. The weight of a block is the normalized path similarity of two DOM nodes, using the edit tree distance in both paths. This means that DOM nodes in the same subtree will get less weight than if they would be located in a different branch of the tree. Figure 1.2a shows the example Web page and the DOM nodes selected. Figure 1.2b shows the initial block graph with its weights.

The algorithm iteratively evaluate two blocks i and j. If the weight of the edge between these two vertex is below a parameter, both blocks are grouped. Figures 1.2c and 1.2d show the final segmentation and the block graph assuming that the parameter value is 0.1.

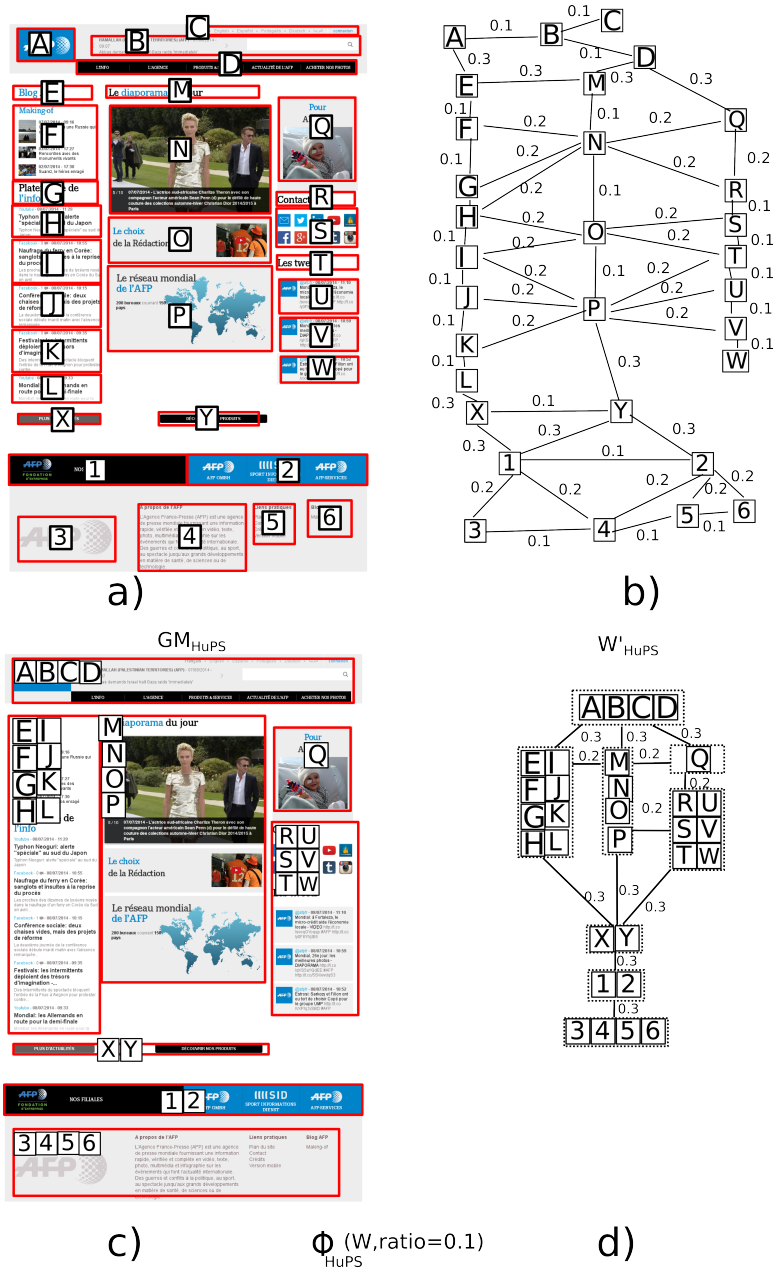


FIGURE 1.2: Bottom-up segmentation strategy used in Gomory-HuPS algorithm

1.3.4 Basic Approaches

This section presents the basic approaches for Web page segmentation: text-based (1.3.4.1), image-based (1.3.4.3) and TagName-based (1.3.4.2). Basic approaches only take one characteristic of Web pages into account.

1.3.4.1 Text-based

In the field of text analysis, words, syllables and sentences have been widely used as statistical measures to identify structural patterns in textual parts of Web documents. It can be seen as a special form of segmentation [CM00]. Regular expressions have been used to parse a Web page looking for blocks. Even if it is possible to use regular expressions for this task, it is known that they are not the best choice, because HTML is not a regular language. In text based approaches, the text can be extracted either from the HTML source code of a Web page such as in [KN08] or from the rendered DOM such as in [SSL11]. In contrast to regular expressions, the concept of text density and link text density [SSL11, KN08] have been more accepted in the Web page segmentation community.

Sun et el. [SSL11] work with the DOM nodes in order to figure out the number of characters and tags that each node contains. Then, they define the *Text Density* of a node as the ratio of the number of all characters to the number of all tags in a DOM subtree. Furthermore, they define the *Composite Text Density* as the same ratio, but with the number of all hyperlinks characters and the number of all hyperlink tags in the DOM subtree. The authors argue that a node with too many hyperlinks and less text is less important, thus getting a low density value. A node that contains much non-hyperlink text and few hyperlinks is more important, and receives a high density value. Appendix D shows details of Sun method.

On the other hand, Kohlschuetter [KN08] defines the text density, based on word-wrapping the page text at a constant line width w_{max} (in characters). The density of a the block is the ratio of the number of tokens found to the number of lines in the block. The wrapping width serves as a discriminator between sentential text (high density) and template text (low density). The author proposes a value of $w_{max} = 80$ as optimal. The task of detecting block-separating gaps on a Web page can be seen as finding sibling text portions with a significant changes in the block-by-block text density. The decision of when to merge two adjacent blocks is made by comparing them with respect to their text densities. Kohlschuetter defines a threshold to determining when two blocks should be merged. Appendix D shows more details on Kohlschuetter approach.

Text based approaches do not fully use all Web page characteristics. Therefore there are some types of pages where those approaches fail. Using only text for segmenting a Web page is incomplete because there are other important elements to take into account, such as images, formatted content, among others.

1.3.4.2 TagName-based

Although in the literature this approach is called DOM-based, the name that fits the best is TagName-based approach. Indeed, this approach analyses a Web page based on its source code, *i.e.* the page is not rendered. In general, the blocks produced by these methods tend to partition pages based on their pre-defined syntactic structure, *i.e.* the HTML tags.

Several works following this approach have been published. Lin and Ho [LH02] propose a simple partitioning method based on HTML table structures. Afterwards they compute a content entropy to estimate the importance of each block. Li et al. [LPHL02] propose improving Web search quality by segmenting Web pages into cohesive micro-units. The segmentation procedure involves creating a tag tree which is similar to DOM and then applying two heuristics to aggregate tree nodes into segments: merge headings with the following content, and merge adjacent text paragraphs. Hattori et al. [HHMS07] combine two different methods for page segmentation. In the first method, a content-distance based on the order of HTML tags is defined and the initial block is iteratively separated at positions where the content-distance exceeds a dynamically estimated threshold. The second method applies heuristic rules that are based on HTML tags. Vineel [Vin09] defines a content size and an entropy value that measures the strength of local patterns within the subtree of a node. Threshold values are defined for both measures to perform page segmentation. Crivellari [CM00], descrie the DOM approach as finding blocks can be reduced to find sub-trees tagged with <TITLE>, <P>, <H1> <H3> and <META> tags.

The advantage of this approach is that the analysis is very fast. However, looking only at the tag names of a page may not provide enough information to segment a Web page. The reasons lies in the following two aspects. First, the visual presentation is not taken into consideration, so visually adjacent blocks may be far from each other in the structure and detected wrongly. Second, tags such as <TABLE> and <DIV> are used not only for content presentation but also for layout structuring. It is therefore difficult to obtain the appropriate segmentation granularity. For instance, it is sometimes not possible to determine if a table cell <TD> holds a value or page content, if we only take its tag name into consideration.

1.3.4.3 Image-based

Image segmentation algorithms are applied to obtain the layout of a Web page from its rendered image, *i.e.* on a snapshot of the rendered page (*cf.* Section 1.1.2). They do not use neither the source code nor the rendered DOM of the Web page. Rendered images of a Web page are different from natural images in the sense that, it is easier to detect objects in rendered images because they have sharp edges or color transitions.

This approach is motivated by the fact that the DOM does not always have direct access to all the visible information. For example, if the page was formed using one flash object or is based on Java applets.

Pnueli et al. [PBSB09] propose a segmentation algorithm using edge analysis. It looks for long edges horizontal or vertical, and then selects the rectangles that do not lie within any other rectangle in the image. After this stage, the algorithm seeks for areas containing information, and groups them into distinct layout elements. This process goes recursively down until the level of individual elements, which may be text areas, images, videos, buttons, edit boxes, etc. This approach has some difficulties to detect elements that do not have rectangular shape, such as : radio buttons and text detectors. Once the text areas have been detected, OCR is applied in order to obtain information about the meaning of a layout object. For instance, a text can be a heading, a paragraph or a hyperlink.

Cao et al. [CML10] propose a segmentation method where a Web page image is divided into visually consentaneous sub-images by shrinking and splitting iteratively. First, the Web page is saved as an image that is preprocessed by an edge detection algorithm such as Canny [Sze10]. Then dividing zones are detected and the Web image is segmented repeatedly until all blocks are indivisible.

This approach is limited as it does not access DOM information. For instance, in the DOM it is clear when a text belongs to a heading or to a paragraph without no further analysis.

1.3.5 Hybrids Approaches

This section presents the hybrid approaches for Web page segmentation: vision-based (1.3.5.1), template-based (1.3.5.2), and graph-based (1.3.5.3). Hybrid approaches try to

overpass the limitations of basic approaches, by taking into account several characteristics at the same time. This can be seen as mixing several several approaches.

1.3.5.1 Vision-based

This approach mixes the following basic approaches and uses the following characteristics (*cf.* Section 1.2):

Basic approaches	**Page characteristics**
TagName-based	visual cues
Text-based	content
	scheme

According to human perception, people view a Web page as a set of different semantic objects rather than a single object. Some research efforts show that users always expect that certain functionalities of a Web page (e.g. navigational links, advertisement bar) appears at certain position of that page [Ber03]. Actually, when a Web page is displayed, the spatial and visual cues can help the user to (unconsciously) divide the Web page into several semantic parts. Therefore, it might be possible to automatically segment the Web pages by using the spatial and visual cues. The vision-based content structure of a page is obtained by combining the DOM structure and the visual cues.

The most known algorithm that follows this approach is VIPS, described by Cai et al. in [CYWM03] already introduced in Section 1.3.3. They define a Web page as a recursive structure of non overlapping blocks. Using the scheme of the page a set of separators is obtained. The Web page is first fragmented into several big blocks and the hierarchical structure of this level is recorded. For each big block, the same segmentation process is carried out recursively until we get sufficiently small blocks, *i.e.* blocks with a *DoC* value greater than *PDoC*. The *DoC* is obtained from a set of heuristics rules. The value depends on the children and on the text of the element under evaluation. Appendix D gives more detail on the VIPS algorithm.

Another algorithm that follows this approach is presented in Zhang et al. [ZJKZ10]. They focus on finding the set of nodes that are labeled as *Content Row*. A content row is a set of leaf nodes of the rendered DOM tree which are horizontally aligned and are siblings. Content rows are merged if there is an overlap between them. As a second step, the *block headers* are detected. A content row is a block header except under certain conditions. For instance if the content row contains a paragraph of text which breaks a line or two vertically adjacent content rows share the same CSS style. The other heuristic

rules for this algorithm can be consulted in Appendix D. Each detected block header is a separator of two semantic blocks. A semantic block is a stack of vertically aligned content rows. Although Zhang algorithm is very simple and efficient (for wikis, forums and blogs), it is not suitable for a general use. There are several Web pages where this algorithm will fail, mainly pages designed in a not uniform way. For instance, artistic designs where elements break the constraints of rendered positioning (*cf.* Section 1.1.4).

Several works following the vision-based approach have been published. Baluja's segmentation system [Bal06] divides a Web page into nine segments using a decision-tree which uses an information gain measure and geometric features. [CMZ03] proposes a Web page analysis method based on support vector machine rules to detect parts of the page that belongs to high level content block (header, body, footer or sidebars). For each, it applies explicit and implicit separation detection heuristics to refine blocks. Vineel [Vin09] defines a content size and an entropy value that measures the strength of local patterns within the subtree of a node. Threshold values are defined for both measures to perform page segmentation. [YS09] presents the Web page segmentation in terms of representing how humans usually understand Web pages. It is based on the Gestalt theory, a psychological theory that can explain human's visual perceptive processes. Four basic laws, namely proximity, similarity, closure, and simplicity, are deduced from the Gestalt theory and then implemented to simulate how human understand the layout of Web pages. [Pop12] presents an adaptation of the VIPS algorithm using Java. It follows the same heuristics as the original algorithm. However the results are not exactly equals because there are differences in the vertical separator detection. Akpinar et al. [AY13] present a technical improvement of the VIPS algorithm, adapting it to current Web standards and use them in the context of new applications.

The main issues with this approach is the possibility of ambiguous rules and an incomplete set of visual rules. For instance, this approach suffers of the same problem as TagName-based approaches, special rules are needed in order process all elements in Web pages, such is the case of <TABLE>, <UL>, <P>, <LI> elements.

1.3.5.2 Template-based

This approach mixes the following approaches and uses the following characteristics (*cf.* Section 1.2):

A large number of Web pages have a common template, which includes composite blocks such as header, footer, left side bar, right side bar and a body. These content blocks

Basic approaches	**Page characteristics**
TagName-based	visual cues
Vision-based	content
	scheme
	template elements

typically follow certain layout design conventions which can be used to derive heuristic rules for their classification, based on the information of the rendered DOM tree. A good example is the work of Chen et al. [CXMZ05] which consists of the following two steps: high-level content block detection and further segmentation of content body. The rendered DOM is traversed to classify each element as belonging to one of the predefined high-level blocks category. To further identify finer blocks within each content body, a set of explicit and implicit separators are identified. Explicit separators are HTML elements such as <p>, <hr>, <ul> or <h1>, while implicit separators are the gaps along the projection on the horizontal and vertical axes of blocks.

Fernandes et al. [FdMdS+11] present the segmentation from the website perspective. They define the block graph as an auxiliary tree called SOM_{tree}, for *Site Object Model*, which is the result of aggregating all the rendered DOM trees of the website to one structure. The nodes of the SOM_{tree} have the same attributes as the DOM elements but with two extra attributes: a counter, with the number of pages where the element occurs in the site and the list of pages where it occurs. For instance, if the website has 3 pages (*p1*,*p2* and *p3*), the SOM element <html> will have the $counter = 3$ and $pageList = \{p1, p2, p3\}$ The SOM tree is refined applying heuristic rules to merge those elements, conforming blocks where they difference in their depth is below to a threshold α (the stop condition).

There are some issues with the final segmentation. Elements tag names are not always used accordingly to what they are defined for. Another issue is that a block not always covers the content that lies in the same DOM sub-tree. This can lead to incorrect segmentation on some pages.

This approach add large overhead to the segmentation process. If only one page has to be segmented, the whole site need to be segmented also. Despite there exists several scenarios where this behaviour is relevant, in the majority of the cases it is not.

1.3.5.3 Graph-based

This approach mixes the following approaches and uses the following characteristics (*cf.* Section 1.2):

Basic approaches	Page characteristics
DOM-based	visual cues
Text-based	content
	scheme

As explained above, relying on heuristic rules rise problems. To overcome those limitations formal approaches have been developed for Web page segmentation. Chakrabarti et al. [CKP08] propose an approach based on a weighted graph built over nodes of the rendered DOM tree, where the cost of assigning pairs of DOM elements to different segments can be explicitly coded. This allows the definition of a global cost of a segmentation, which can be minimized using well established graph clustering algorithms.

The approach is based on the following formulation. Given a rendered DOM tree, let N be the set of nodes. A graph is constructed whose node set is N and whose edges have a weight that represents a cost of placing the adjacent nodes in different segments.

Hu et al. [HL14] defined two extra features to Chakrabarti's algorithm. The visual features including the node's position, shape (aspect ratio), background color, types, sizes, and colors of text. The content-based features capture the purpose of the content and include the average size of sentences, the fraction of text within anchors, and tag names. The edge weights are derived from these features using a regression function learned from a set of manually labelled Web pages, where each DOM node is assigned a segment ID.

Other authors have explored different clustering algorithms. Liu et al. [LLT11] present a Web page segmentation algorithm based on finding the Gomory-Hu tree in a planar graph. The algorithm first gets the rendered DOM of a Web page to construct a weighted undirected graph, whose vertices are the leaf nodes of the DOM tree and the edges represent the proximity relationship between vertices. Then it partitions the graph with the Gomory-Hu tree based clustering algorithm. The task of Web page segmentation is essentially to partition the constructed graph into groups such that the inner group similarity is maximized while the inter group similarity is minimized. An example of this algorithm is shown on Figure 1.2. It shows the grouping of blocks which edges in the graph have a weight less than a parameter (0.2 in the figure), forming seven groups corresponding to the blocks.

Algorithms following this approach have the advantage that they can be analytically evaluated. However comparing them to others algorithms following other approaches is not an easy task, as no available implementation exist.

1.3.6 Conclusion on Web page segmentation algorithms

A lot of approaches have been developed for page segmentation, most of them described in [Yes11]. There are several approaches and interesting solutions. In general the algorithms are designed with a given application in mind. That makes them efficient only for some particular application domains. The VIPS and GraphBased algorithms present a correct approach to Web page segmentation, because the rules and logic applied are based on some of the Web standards and characteristics intrinsic to Web pages. But, there is a weakness in these approaches: the block detection depends heavily on tag names (*e.g.* <table> and <img>) and textual content. While heuristics rules have reasonable limitations, formal approaches can handle a broader set of documents. However, the graph-based approach needs test data to compute edge weights that might be expensive in creation, and the densiometric-based approach is limited to textual contents while images are not considered.

As seen in section 1.3.4.2, relying on tag names and textual information can lead to unexpected results in some cases and implies defining special heuristics rules.

Our goal is to design a more general segmenter complying with the Web standards. Thus, we must minimize the dependency of tag names or special heuristics rules. This may lead to some accuracy loss for some page types, but should give better overall results.

1.3.7 Document processing and Web page segmentation

We observe that there is a clear relationship between page segmentation and the field of computer vision. Segmenting and understanding scanned documents images is a very well studied subject in the Document Processing domain [TS94, TCL+99]. In Document Processing systems, the concepts of objects, zones or blocks are applied to define the geometric and logic structure of a document where each block is associated with a category, its geometric information and a label. Processing a document comprises the document analysis and understanding phases. Document images are analyzed to detect blocks based on pixel information. Blocks are categorized, their geometric information

is extracted and, in function of both features, a label is assigned. Moreover, by understanding what blocks contain (label) and where they are located (geometric model), it is possible to merge them [FM81] and give them a reading order, or flow [Meu05]. For example, assume we have a title block and two paragraph blocks. The two paragraph blocks should be merged, the title block should appear first, followed by the merged paragraph block.

There are algorithms for Web page segmentation that include aspects of the Document Processing approach. The first to do it is VIPS [CYWM03]. Its segmentation model is based on the recursive geometric model proposed by Tang [TS94] for recognizing regions in a document image. VIPS itself focuses mainly on the content and geometric structure of a Web page. Although they do not explicitly include a logic structure, they understand the document by extracting the blocks and by grouping them based on separators. Kohlschütter [KN08] uses the relation between the pixel concepts with text elements in the Web page domain. They transfer the concept of pixel to HTML text as character data (the atomic text portion), an image region is translated to a sequence of atomic text portions (blocks). They measure the density of each block and merge those which are below a threshold tolerance using the BlockFusion algorithm. [NWM09] uses segmentation, structure labeling and text segmentation and labeling to define a random field that leads the extraction.

The Web page segmentations algorithms presented in this section adapt methods from the document processing domain. Actually, there is a vast knowledge in this domain that we can exploit to define new methods and techniques adapted to Web pages. We think that adapting existing document processing methods and techniques to Web page segmentation allow increasing the quality of the results that can be obtained from the segmentation itself. The vision-based approach exploit this relationship, such as the VIPS algorithm. The classification and labeling of blocks are defined as posterior task not included in the segmentation. In our work we also use document processing concepts but within the bottom-up strategy (VIPS follows a top-down strategy). We are inspired in a classification method as described in [LPH01]. Every block corresponds to an item in a predefined taxonomy based on the most basic categories. For scanned documents we can cite some of them: paragraphs, title and figures. In Web pages their equivalent are the HTML5 content categories: phrasing, heading, embedded.

Featured algorithm	**Approach**	**Strategy**	**Year**	**Reference**
Annotation Transcoding	DOM-based	Top-Down	2000	[AT00]
InfoDiscover	Template-based	Bottom-up	2002	[LH02]
MIU Ranking	DOM-based	Top-Down	2002	[LPHL02]
VIPS	Vision-based	Top-Down	2003	[CYWM03]
TOC-Adaptation	Vision-based	Top-Down	2003	[CMZ03]
DOMEntropy	DOM-based	Top-Down	2006	[Bal06]
Content-Distance	DOM-based	Top-Down	2007	[HHMS07]
Blockfusion	Text-based	Bottom-up	2008	[KN08]
GraphBased	Graph-based	Bottom-up	2009	[CKP08]
Node Entropy	DOM-based	Top-Down	2009	[Vin09]
E-GESTALT	Vision-based	Top-Down	2009	[YS09]
HPImage	Image-based	Top-Down	2009	[PBSB09]
Shrinking&Dividing	Image-based	Top-Down	2010	[CML10]
ContentRow	Vision-based	Top-Down	2010	[ZJKZ10]
Distance Clustering	Graph-based	Bottom-up	2011	[AC11]
Homory-HuPS	Graph-based	Bottom-Up	2011	[LLT11]
CETD	DOM/Text-based	Top-Down	2011	[SSL11]
SOMtree	DOM-based	Top-Down	2011	[FdMdS+11]
jVIPS	Vision-based	Top-Down	2012	[Pop12]
Improved-VIPS	Vision-based	Top-Down	2013	[AY13]
Block-o-Matic	Vision-based	Top-Down	2014	*cf.* Chapter 2
EVBE, EIFCE & EICTE	DOM-based	Top-Down	2014	[WT14]

TABLE 1.1: Summary table for Web page segmentation algorithms in the state of the art

1.3.8 Summary Table

Table 1.1 presents a summary of the algorithms, classified by the approach they follow, the strategy used, the year of publication and the corresponding references.

1.3.9 Discussion

Most existing approaches are devoted to some application domains or to some page types. Our aim is to produce a generic segmentation, without additional knowledge of the content of the page and its context. Adapting Document Processing concepts allow to enhance the Web page segmentation, for instance the geometry of blocks, the labels and the reading order. Using the text content gives a more detailed segmentation for certain domains, however what is gained in precision it is lost in genericity. A generic segmentation algorithm should use only the characteristics related to the Web page or website described in section 1.2 and Web content models defined in Web standards instead of tag names. In chapter 2 we present a model for Web page segmentation. It is

intended to be general and to be used as a common ground for describing segmentation algorithms. Indeed, we would like to study and compare the segmentation algorithms (including this one). Thus, in the following section we explore the evaluation of Web page segmentation. Evaluating different algorithms is a great challenge due to the lack of genericity.

1.4 Segmentation evaluation

The question we address in this section is: how well do the presented approaches correctly identify segments in nowadays Web pages? This question raises the issue of evaluating page segmentation methods.

This section presents the different methods used to evaluate segmentation algorithms (1.4.1) and our interpretation of segmentation correctness (1.4.2). We investigated also the connex domain of scanned page segmentation (1.4.3), since the issue of evaluating such systems is quite similar with our problem. We present the state of the art on evaluating Web page segmentation (1.4.4), a summary (1.4.5), concluding with a discussion (1.4.6).

1.4.1 Classification of evaluation methods

Zhang et al. [ZFG08] present a complete classification of image segmentation evaluation methods. In this work we present an adaptation of their classification to Web page segmentation. The different methods are:

1. **Analytical**: directly evaluates the segmentation algorithms themselves by analysing their principles and properties
2. **Empirical**: indirectly judge the segmentation algorithms by applying them to test data and measuring the quality of segmentation results. They can be divided into: *goodness methods* and *discrepancy methods*:
    - **Goodness methods** are methods where a segmentation is considered "ideal" if it satisfies some conditions assessed by a human judgment. The latter can be obtained from simple observation or by obtaining some features that complements the observation. There is no need to have *a priori* knowledge of the reference segmentation. The result is known as the *goodness parameter*.

- **Discrepancy methods** are methods that compare the segmentation with a "correct" or "ideal" segmentation. This ideal segmentation is also known as *ground truth*. The author clarify that the goal of these methods is to find a discrepancy parameters that allows to determine how far the two segmentation are one from the others. A ground truth is defined *a priori*.

As there are several approaches for Web page segmentation, a natural question raised is how to compare them?

As mentioned by Cardoso et al. [CCR05], analytical methods avoid the implementation of algorithms and so they do not suffer from bias induced by evaluation experiments as the empirical methods do. However, analytical methods can only be used if the models of the two segmentation algorithms are similar, or if one can be transformed into the other. For instance, the algorithms GraphBased and Homory-HuPS have a similar model. Therefore they share the same type block graph and other characteristics of Web page segmentation.

On the other hand the use of analytical methods is not so simple if we take two algorithms with different models. Consider the two algorithms, VIPS and Homory-HuPS. One follows the vision-based approach while the other follows the graph-based approach. The block graphs are different (vertex-weighted tree vs. edge-weighted planar graph) and the stop conditions are completely different, but the geometric models are similar. This implies that their can not be compared analytically because their characteristics and properties are not similar.

In this work we focus on describing empirical evaluation methods. Indeed they are are better suited for Web page segmentation. For use analytical methods a common formal model is necessary, but with the current Web standards and technologies a formal model is not possible since they rely on heuristics rules. Analytic evaluation is left for future work, probably based on analytic methods found in Zhang survey on image segmentation evaluation methods [Zha96].

In the remainder of this section we explore the Web page segmentation algorithms mentioned in the section 1.3.4 and section 1.3.5 and describe how authors evaluate their algorithms. We gather all these experiences and present them in a comparative way.

1.4.2 Segmentation correctness evaluation

Different interpretations of correctness can be used with respect to segmentation. As defined in the literature, the correctness of an algorithm is asserted when it complies with a given specification. The problem here is that such a specification cannot be established *a priori*, without a human judgment. Thus, we focus on evaluation approaches based on a ground truth. We also investigated the correctness issue in the connex domain of scanned page segmentation, since the issue of evaluating such systems is quite similar to our problem.

Segmentation issues have been addressed for almost thirty years in the optical character recognition (OCR) domain [CCMM98]. Automatic evaluation of (scanned) page segmentation algorithms is a very well studied topic. Several authors have obtained good results in performance and accuracy evaluation, as well in measuring quality assurance [HKW99, ZG94, Bre02].

There are common problems in the evaluation of Web pages and scanned pages segmentation algorithms : the lack of a common dataset, a wide diversity of goals/applications, a lack of meaningful quantitative evaluation, and inconsistencies in the use of document models. This observation led us to closely study how segmentation is evaluated for scanned pages.

There is a wide range of work around automatic evaluation based on a predefined ground truth in the literature. Although Web pages and scanned pages are different (pixels/colors vs. elements/text), the way they are analyzed and the result of their segmentation are similar. In both cases, blocks can be organized as a hierarchy or a set of non-overlapping rectangles (Manhattan layout [TS94]).

The review that we made on Web page segmentation algorithms shows that authors are interested in exploiting the geometric and visual aspects of the page. However, the evaluation of the algorithms is constrained only in either the textual aspects of the content or by using the observation. In order to have a more integral evaluation including these three characteristics above mentioned (geometry, visual aspects and content), we need another approach for example that of Shafait et al [SKB08]. They take into consideration the visual and geometric aspects of a scanned document image and the content is measure using a pixel-based representation (foreground pixels of the image are considered as the content in the document). They measure the quality of a page segmentation by analysing the errors in the size and position of the zone shapes (blocks).

In the following section we describe their approach for scanned document segmentation evaluation.

1.4.3 Correctness measures in scanned document segmentation

The evaluation of image segmentation is a very well studied area. For instance, [CCR05] describe several metrics to measure the quality of the segmentation based in the distance of each partition in an image segmentation. However, this method is designed for general images not specifically for scanned documents. Its adaptation to Web pages is thus not straightforward. [LPH01] measure the accuracy of a segmentation by determining the label assigned to entities (paragraphs, title, table, etc) by an image segmentation algorithms and their correspondence to a predefined taxonomy. This approach gets close to what evaluation of Web segmentation needs, but they do not consider the content and the geometric aspect of a scanned document as part of the evaluation. [SKB08] present a vectorial score that identifies the common classes of segmentation errors using a ground truth of annotated scanned pages. We think that this approach covers all the needs of Web page segmentation (with some adaptations and modifications) since they consider the content, geometry and visual aspect of a scanned document in their evaluation model.

The correctness measures proposed by Shafait et al. [SKB08] evaluate to what extent a set of text lines are equal to the ones of the ground truth, which ones are missing, which lie into the bounding boxes and which ones are horizontally merged. They define that a text line is significant if the amount of foreground pixels in each line is greater that two threshold parameters, one relative and the other absolute. They build a bipartite graph whose nodes represent the text lines in the ground truth, in one hand, and in the other the text lines which represent the proposed segmentation. They assign a weight to the edges of the graph according to the significance of the ground truth vertices.

Based on the number of edges in the graph, seven measures are defined. They present how far a proposed segmentation is from the ground truth. These metrics are:

- *Total correct segmentations*: the total number of one-to-one matches between the ground-truth components and the segmentation components.
- *Total oversegmentations*: the total number of significant edges that ground-truth components have minus the number of ground-truth components to which at least one significant edge is incident.

- *Total undersegmentations*: the total number of significant edges that segmentation components have minus the number of segmentation components to which at least one significant edge is incident.
- *Oversegmented components*: the number of ground-truth components having more than one significant edge.
- *Undersegmented components*: the number of segmentation components having more than one significant edge.
- *Missed components*: the number of ground-truth components that did not match any foreground component in the proposed segmentation.
- *False alarms*: the number of components in the proposed segmentation that did not match any foreground component in the ground truth segmentation.

We think that this model is well suited to Web page segmentation evaluation. As the Web content is very different from images content the rendering of both (images and Web pages) lead to different results. However with some adaptations this method is a good candidate to evaluate segmentation algorithms. Not all the Shafait metrics are relevant for Web page segmentation. Other metrics specific to Web pages are needed in order to evaluate a Web page segmentation algorithm. These new metrics are described in Chapter 3 as well as the adaptation of the model to Web pages.

1.4.4 State of the art on evaluating Web page segmentation

Some papers present the evaluation of their segmentation algorithms in an indirect way, with respect to some specific task. For instance, they test the efficiency of segmentation based on for information retrieval rather than the Web page segmentation itself [LPHL02, LH02, CYWM03]. We describe only the works where the evaluation focuses on the performance of the segmentation algorithm itself.

The works presented in the state-of-the-art evaluate their segmentation using these metrics: Rand index (Rand), Adjusted Rand Index (AdjRand), Normalized Mutual Information (NMI), Dunn index (Dunn), Nested Earth Mover's distance (EMD), Precision and Recall (Prec & Rec), F1 score (F1), and custom heuristic rules (heuristics). These metrics are detailed in Appendix C.

[CYWM03] selected 140 Web pages from popular sites listed in 14 main categories of Yahoo directory, and human assessors gave them a goodness parameter which values

are: perfect, satisfactory and failed. [CMZ03] evaluate their algorithm with the goodness parameter as *Error*, *Good* or *Perfect*. [Bal06] presents little information about the evaluation of their segmentation algorithm. They describe with examples how their results can be visually evaluated by human assessors. Hattori et al. [HHMS07] present a supervised evaluation based on precision and recall metrics. Human assessors are asked to detect "correct" segments according to a ground truth in a dataset of 100 pages. The precision is computed as the ratio of correct segments over the total segments in a Web page. The recall metric is the number of correct segments over the number of all correct segments. The same technique was applied by Vineel [Vin09]. They evaluates the algorithm over a dataset of 400 manually segmented Web pages.

[KN08] present three methods for evaluating the blockfusion algorithm. It is the first one which evaluates the segmentation accuracy, the most relevant for our study. Kohlshuetter and Chakrabarti [CKP08] use two cluster correlation metrics the *Adjusted Rand Index* (AdjRand) and *Normalized Mutual Information* (NMI). Kohlschuetter uses 111 Web pages coming from 102 different websites while Chakrabarti uses 1088 pages from 105 different websites. The build a ground truth with this set of pages to define a comparable segmentation. Their results are comparable but Kohlschuetter reports better results for both metrics.

Yang [YS09] evaluates his algorithm defining a goodness parameter with the values: error, not-bad and perfect. Human assessors were asked to evaluate the performance of the algorithm. The results were compared to the VIPS [CYWM03] algorithm using precision and recall as metrics.

The main issues with the evaluation present in the algorithms of the state of the art, is that they exploit the content, geometric, and visual aspects of the page, but their evaluation is reduced to either textual aspects of the content or by the judgment of human assessors.

Another important issue are the datasets. Besides the Web archives, there are no Web page dataset available from which rendered DOM can be build. For instance, the TREC collection only stores the HTML source code, which makes it impossible to fully render the page. We do not know about of other segmentation datasets, probably because they are not publicly available or do not exist any more. As far as we know, the only Web page segmentation dataset available is that of Kreuzer [Kre13]. They provide two datasets of annotated Web pages publicly available[4] [5]. They present also a method for

[4] `https://github.com/rkrzr/dataset-random`
[5] `https://github.com/rkrzr/dataset-popular`

Algorithm	**Type**	**Sub-type**	**Metrics**	**Year**	**Reference**
Annotation Transcoding	Empiric	Goodness	Heuristics	2000	[AT00]
VIPS	Empiric	Goodness	Heuristics	2003	[CYWM03]
TOC-Adaptation	Empiric	Goodness	Heuristics	2003	[CMZ03]
DOMEntropy	Empiric	Goodness	Heuristics	2006	[Bal06]
Content-Distance	Empiric	Goodness	Heuristics	2007	[HHMS07]
Blockfusion	Empiric	Discrepancy	AdjRand, NMI	2008	[KN08]
GraphBased	Empiric	Discrepancy	AdjRand, NMI	2008	[CKP08]
Node Entropy	Empiric	Goodness	Heuristics	2009	[Vin09]
E-GESTALT	Empiric	Goodness	Heuristics	2009	[YS09]
Shrinking&Dividing	Empiric	Discrepancy	Nested-EMD	2010	[CML10]
ContentRow	Empiric	Discrepancy	Prec & Rec	2010	[ZJKZ10]
Distance Clustering	Empiric	Discrepancy	Dunn,Rand	2011	[AC11]
Homory-HuPS	Empiric	Discrepancy	Prec & Rec	2011	[LLT11]
CETD	Empiric	Discrepancy	Prec & Rec, F1	2011	[SSL11]
SOMtree	Empiric	Discrepancy	AdjRand	2011	[FdMdS+11]
EVBE, EIFCE & EICTE	Empiric	Goodness	Heuristics	2014	[WT14]

TABLE 1.2: Summary table on Web page segmentation evaluation

quantitative comparison of semantic Web page segmentation algorithms. This approach mainly uses text content comparison in order to perform the match between ground truth and segmentations blocks, which is not enough, since geometry of blocks plays a key role in the segmentation.

1.4.5 Summary table

Table 1.2 shows the algorithms, the type and sub-type of the evaluation method used (*cf.* Section 1.4.1), the metrics applyied, the year of publication and the referenced article.

1.4.6 Discussion

As shown in this section, evaluating segmentation algorithms is a big challenge. The lack of a common base to express segmentation algorithms impacts their evaluation. Even formal approaches of segmentation use empiric evaluation. Does it means that the empirical evaluation is more natural when segmenting Web pages?

W3C standards and its technologies depend heavily on heuristics rules. As far as we know there is no formal definition or model for Web pages. This lack of formality in Web page standard and its technology impacts the way segmentation algorithms are designed and evaluated. For this reason, even formal segmentation approaches end up relying

on empirical methods to evaluate. We think that analytic evaluation methods could be used when standards and its technologies will be formally defined. Meanwhile, we rely on empirical methods to evaluate segmentation algorithms. We present an empirical evaluation method that aims at being generic, whatever the approach used by algorithms.

Web page segmentation algorithms in the state of the art exploit the content, geometric, and visual aspects of the page, but their evaluation is reduced to either textual aspects of the content or by the judgment of human assessors. In order to have a more complete evaluation these three aspects need to be considered.

Most of the metrics used to evaluate Web page segmentation algorithms are not well suited because they evaluate only textual aspect of the segmentation. For that reason we consider the metrics presented by Shafait (*cf.* Section 1.4.3) because all aspects of the segmentation are taken into account.

Chapter 2

Block-o-Matic (BoM): a New Web Page Segmenter

In this chapter we present BoM, our Web page segmentation approach. One of the main features of BoM is that we segment a Web page without having previous knowledge of its content and using only the heuristic rules defined by the W3C Web standards. For instance, we detect blocks using HTML5 content categories instead of using the tag names or text features. That gives genericity to BoM and allow it (in theory) segmenting all types of Web pages.

Another feature of our approach is the introduction of methods and techniques of document processing systems. We leverage existing techniques from the field of computer vision for segmenting scanned documents, in order to adapt them to Web pages. This produces more interesting results for the applications that depends on the segmentation, such as the order of the blocks in the segmentation and their labels.

We present the concepts used along the chapter (2.1) and an overview of the segmentation algorithm (2.3 to 2.5). Then we describe the different parts of the algorithm (2.2). We conclude with a discussion (2.6).

2.1 Preliminars

In this section we present the concepts used along the chapter. We use the notation introduced in Section 1.3.2. We use the concepts of page, block, composite block and

block graph as defined in section 1.3.1.

Let W be the rendered DOM of a Web page. A segmentation Φ_{BoM} of W is defined as follows :

$$\Phi_{BoM}(W, pA, pD, pND) = (W'_{BoM}, GM_{BoM})$$

where W'_{BoM} is the block graph (a tree) of the segmentation, GM_{BoM} is the geometric model and pA, the stop condition. In BoM, the stop condition is the normalized area parameter which is the proportional size of a block respect to the page. We include other parameters used in the algorithm: pD is the Distance parameter used for merging blocks. pND which is used to compute the normalized area and the weights of blocks. The pA and pD parameters are described on detail in section 2.4 and 2.5. The pND is described at the end of this section for computing the weight of a block.

Each block B is associated with its rectangle ($B.rect$), its label ($B.label$), its weight ($B.weight$) as defined in Section 1.3.2, and a set of DOM elements ($B.elements$).

Consider W'_{BoM} as a rooted, planar and vertex-weighted tree. The root vertex is the *Page* block, inner vertices are the composite blocks, terminal vertices are the simple blocks.

The edges between blocks represent a hierarchical relationship of geometric containment. In other words, consider $Page$, B_c and $B_p \in Blocks$, the following constraints apply:

1. For every pair of blocks (B_c, B_p), where B_p is the *parent* of B_c in the W'_{BoM} tree, we write B_c *child* of B_p and B_p *parent* of B_c.
2. For every block B_c, child of B_p, $B_c.rect$ is contained in $B_p.rect$

$$\forall\, B_c, B_p,\ B_c \text{ child of } B_p \Rightarrow\ B_c.rect \subset B_p.rect$$

3. The *Page* rectangle cover the whole page and all blocks fit inside it.

$$\forall\, b \in Blocks,\ b.rect \subseteq Page.rect$$

Only simple blocks are associated to DOM elements, thus for the page and composite blocks the $B.elements$ is an empty set.

The weight of a block is the normalized area of its rectangle. It is used to check the stop condition (*cf.* section 1.3.1). Thus, the weight of a block B is:

$$B.weight = 0.1 \times \frac{B.rect.w \times B.rect.h \times pND}{Page.rect.w \times Page.rect.h}$$

where pND is the predefined constant. In this work we fix this value to pND=100, so that both $B.weight$ and pA belongs to the interval [0,10].

2.2 Overview

In this section we present the Web page segmentation model. It is an hybrid approach, and it follows the bottom-up strategy, as defined in Chapter 1.3.3.

First, we describe the segmentation as a black box indicating its input and output. A more detailed explanation follows, describing the three sub-processes that achieve the final segmentation.

We define the Web page segmentation as the process of finding coherent regions of content (blocks) into the rendered DOM (W) of a Web page. As a result, the block graph W'_{BoM} and the geometric model GM_{BoM} are produced. The block graph is a tree structure as defined in section 2.1. Blocks of W'_{BoM} are ordered considering the *reading order* of the page. This order is based on the rectangles of the geometric model of the segmentation (*cf.* Section 1.3.1) rather than the DOM tree[1] structure. In other words, the order of blocks is more likely to have the same order as blocks are found in the segmentation rather than the order found in the source code.

Figure 2.1 shows how a rendered Web page W is segmented. The output is the block graph W'_{BoM} shown on the right side of the figure and the geometric model in the center of the figure.

The sub-processes of the segmentation are:

1. **Fine-grained segmentation construction**. Builds the fine-grained segmentation of W producing W'_{BoM} and GM_{BoM}.

[1] In the whole section "DOM tree" stands for "rendered DOM tree" (*cf.* Section 1.1.3)

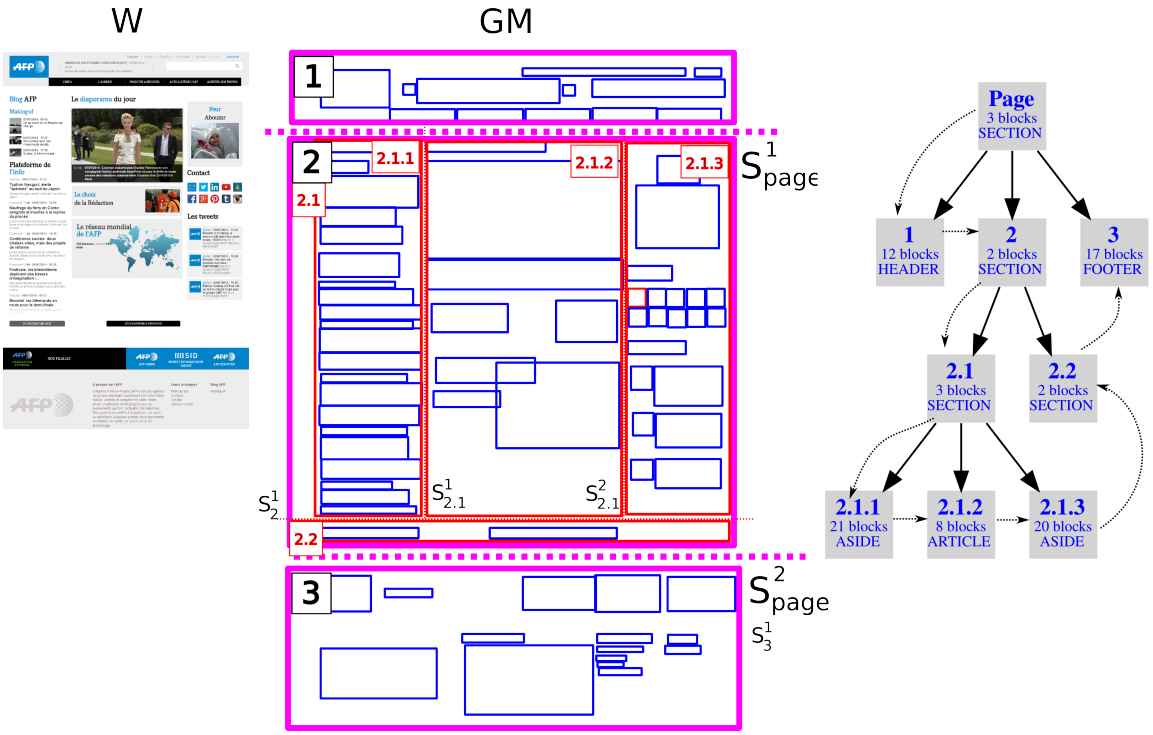


FIGURE 2.1: Segmentation model example

2. **Composite block and flow detection**. Detects the composite blocks and the flow of blocks. This sub-process updates W'_{BoM} and GM_{BoM}

3. **Merging blocks**. Merges blocks according to their area, distance, alignment, labels and content categories. This sub-process produces the final version of W'_{BoM} and GM_{BoM}.

In the following sections we detail the three sub-process of the segmentation.

2.3 Fine-grained segmentation construction

The idea of the fine-grained segmentation is to find coherent blocks as small as possible. It serves as a starting point for the whole process by creating a first version of the block graph W'_{BoM} and the geometric model GM_{BoM}. The condition C that a DOM element must satisfy to be considered as a block is that it does not belongs to the following content categories: text, phrasing, embedded, interactive or form-associated elements. Appendix A has a complete description of HTML5 content categories, elements and their exceptions. The value to the label ($B.label$) is the most inclusive content category of its elements ($B.elements$). For instance, if the block has one element which content

category is *flow* the label of the block is the same. If the block is associated with two elements, one element in the *embedded* category and the other in the *heading* category, the most inclusive category is *flow*. Figure 2.4 shows which content category includes other content categories.

The process begins from the leaves of the DOM tree, towards the *W.root* (*cf.* Section 1.2.1.1). If an element is found that meets the condition C above defined, the process stops for this branch. Figure 2.2 shows how an element is selected as a block. Element *li* is the first element that does not belong to the categories above listed, then it is marked as a block and the label *flow* is assigned. From the information obtained during this sub-process a geometric model (cf. section 1.3.1) and a first version of the block graph are built (cf. section 1.3.1).

FIGURE 2.2: Block detection based on content categories

Algorithm 1 shows the steps to build the fine-grained segmentation. First, the rendered DOM tree W is traversed and leaves elements are selected (line 5). If a selected element does not match the condition C its parent become the current element (line 7-8).

The same process continues until either the *W.root* element (*i.e.:* the *body* element) is reached or the current element meet the condition C. If the condition C is met a new block is created (lines 10-11). The element becomes the block's element (line 12), the block label is the element category (line 13), a new rectangle is created (line 14), the geometric model is updated (line 15) and the weight is computed (line 17). The rectangle is based on the box of the element (*cf.* section 1.2) and it is associated to the block (line 16). The block graph is updated with the new block b, adding an edge between the Page

block and block b (lines 18-19)

```
   Data: Rendered DOM : W
   Result: block graph W'_BoM, geometric model GM_BoM
1  Blocks = {Page};
2  E = {};
3  W'_BoM = (Blocks, E);
4  GM_BoM = {};
5  Terminal ← getTerminalElements(W);
6  foreach element ∈ Terminal do
7      while element ≠ W.root and ¬C(element) do
8          element ← element.parentElement;
9      end
10     if element ≠ W.root then
11         create block b;
12         b.elements ← element;
13         b.label = element.category;
14         rect = createRectangleFromElement(element);
15         add rectangle rect to GM_BoM;
16         b.rect = rect;
17         b.weight = normalized_area(b);
18         add vertex b to W'_BoM;
19         add edge (Page, b) to E;
20     end
21 end
```

Algorithm 1: Fine-grained segmentation construction algorithm

The fine-grained segmentation form a flat segmentation, that is $height(Page) = 1$.

2.4 Composite block and flow detection

Composite blocks usually are Web page regions that lie along separation lines. A separation line is the space that goes from one limit of the page to another without crossing any block. A horizontal separation line S in a block is represented by the line formed by the points (x_1, y_1) and (x_2, y_2), where $y_1 = y_2$ if it is horizontal, $x_1 = x_2$ if it is vertical. The spaces found either at the beginning or at the end of the document are omitted.

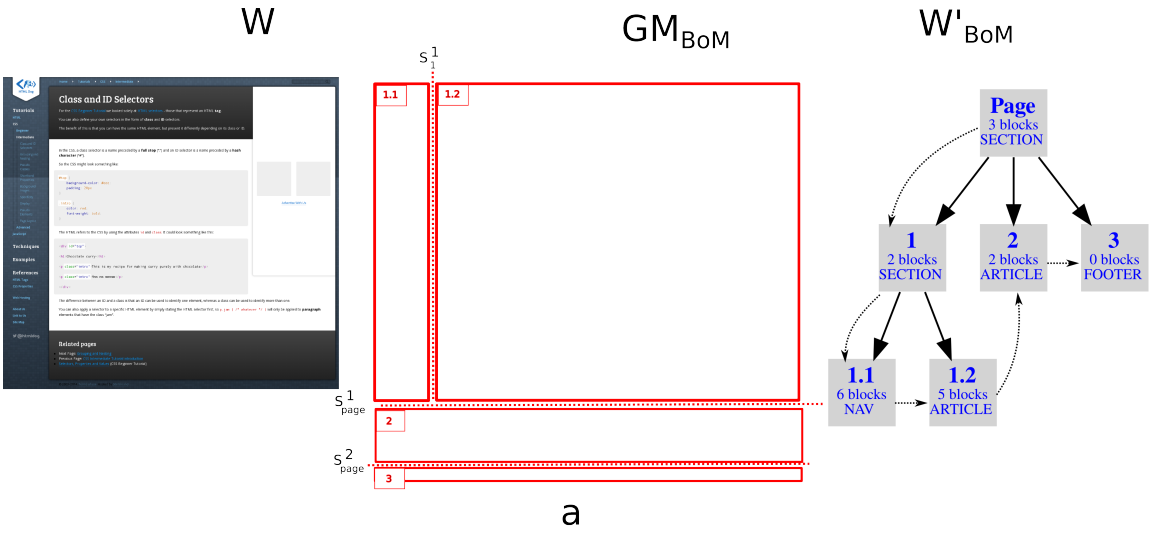
Algorithm 2 shows the *CompositeBlockDetection* function in order to find the composite blocks and the flow of a segmentation. It accepts a composite block as input and outputs the W'_{BoM} graph and the geometric model GM updated with new blocks (if any) and including the computed order.

We start finding the composite blocks in the Page block itself, considered as a composite. Two composite blocks are formed on both sides of the separation line (line 12). All simple blocks that are covered by these new blocks are aggregated accordingly and become their children blocks (line 22). The process stops if it is met one of two conditions: their weights are below the predefined stop condition parameter (pA) or the horizontal or vertical limits of the block are not those of the Page (line 1), *i.e.* if $B.rect.x > Page.rect.x$ and $B.rect.w < Page.rect.w$ (respectively $B.rect.y > Page.rect.y$ and $B.rect.h < Page.rect.h$).

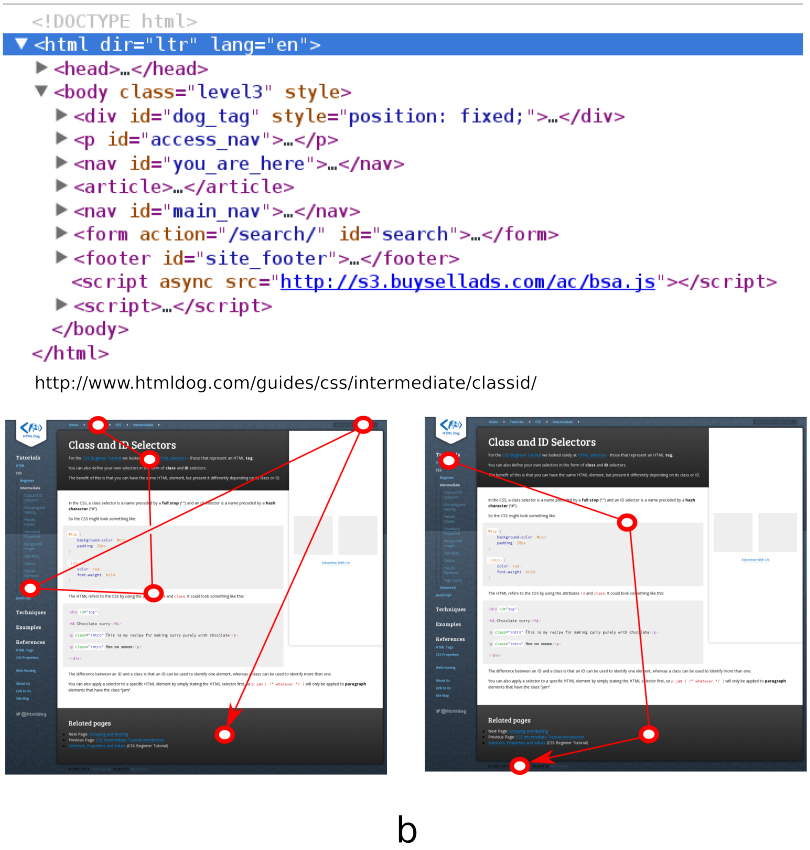
Figure 2.1 shows the separation lines, S^1_{page} and S^2_{page}, found in the Page block, generating blocks 1, 2 and 3. On the same figure, block 1 and 3 are not processed because their weights are higher than pA, but the same process is applied to block 2. First the horizontal separator S^1_2 is discovered, generating the composite blocks 2.1 and 2.2. We assume that the weight of block 2.2 is below the predefined stop condition parameter, thus no further processing is needed. However, in block 2.1, two vertical separators $S^1_{2.1}$ and $S^2_{2.1}$ are found.

The reading order of blocks is build while detecting composite blocks. Following the spirit of Meunier's algorithm [Meu05], we define the flow of blocks in the same order as separators are detected and according to how the composite blocks are divided.

Figure 2.3 shows another example, which allows comparing the order induced by the DOM structure and the order resulting from the composite block detection. Figure 2.3a shows the composite blocks detected by the algorithm. The dotted lines in the W'_{BoM} graph denotes the reading order of blocks. Figure 2.3b (top) shows the DOM elements of the page, we see that the *main_nav* element (menu at the left of the page) is defined after the *article* element (main content), but in the rendering of the page it appears at its left. This example page uses the absolute position of elements (*cf.* Section 1.1.2) so their position in the DOM does not correspond with the formatted content. In the same figure (left) we see the elements order after the rendering and (right) the order of blocks found by the segmentation.



a



b

FIGURE 2.3: Web page segmentation example showing the DOM based flow and BoM block flow

```
Data: block b
Result: W'_BoM and GM_BoM updated
if b limits equals to Page and b.weight > pA then
    Separators ← findSeparatorsIn(b);
    foreach s ∈ Separators do
        if s is horizontal then
            rect_1 = {b.rect.x, b.rect.y, b.rect.w, s.y_1};
            rect_2 = {b.rect.x, s.y_1, b.rect.w, b.rect.h};
        else
            rect_1 = {b.rect.x, b.rect.y, s.x_1, b.rect.h};
            rect_2 = {s.x_1, b.rect.y, b.rect.w, b.rect.h};
        end
        add rectangles rect_1, rect_2 to GM_BoM;
        create blocks b_1, b_2;
        b_1.rect = rect_1;
        b_2.rect = rect_2;
        add vertices b_1, b_2 to W'_BoM;
        add edge (b, b_1) to E;
        CompositeBlockDetection(b_1);
        add edge (b, b_2) to E;
        CompositeBlockDetection(b_2);
    end
else
    update W'_BoM and GM to associate blocks covered by b
end
```

Algorithm 2: Composite blocks detection algorithm

2.5 Merging blocks

Once composite blocks are created, the merging process starts. This process allows obtaining simple blocks the weight of which is greater than the predefined stop condition parameter (pA). Two blocks are merged if the following heuristic rules are all satisfied:

1. Their weights are less than the the predefined stop condition parameter.
2. The distance between them is below a predefined distance parameter pD.
3. Both blocks are horizontal or vertical aligned with a tolerance than no more that pD pixels.
4. They are not aligned but one's rectangle covers completely the other's one.
5. Their label is not *sectioning* (*cf.* Section 1.2.1.1).

The rules are checked in the given order for efficiency purpose: the first rules are most discriminant.

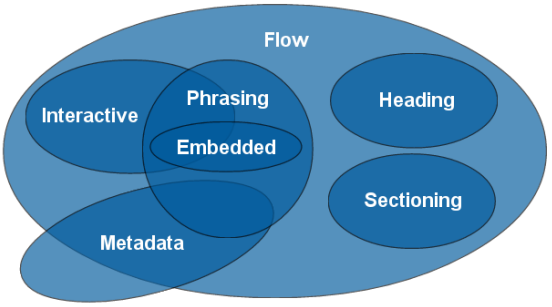


FIGURE 2.4: HTML5 content models. Source: `http://www.w3.org`

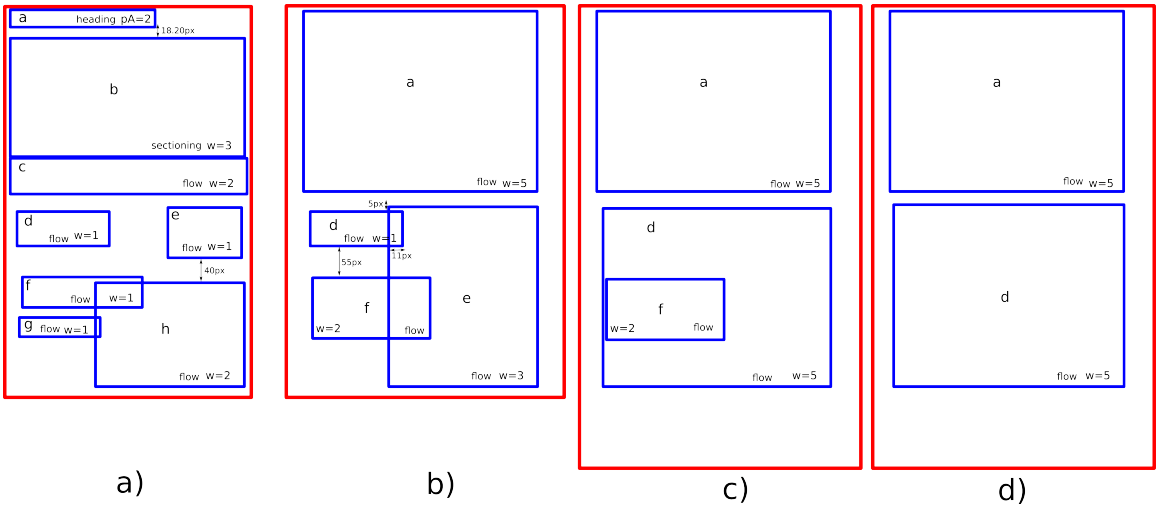


FIGURE 2.5: Merging blocks and labeling

This process is repeated until no more merges are possible. Then we check if the proportion of blocks with a weight less than pA is greater than a constant (for instance 75%). If it is the case, all the children of the composite blocks are removed. If the composite block has only one child, this latter is removed.

To illustrate the merging process, let $pA = 4$, $pD = 50$ and $pND = 100$. Figure 2.5 shows the merging process for the block 2.1.2 of an example page. Each blocks has its weight and its label. In Figure 2.5a blocks a, b and c are merged because they are aligned and the distance between them is less than pD. The label *flow* is assigned. The same applies for blocks e and h. However blocks d and f are too far. Blocks f and g are not aligned. Figure 2.5b shows the result of merging those blocks and in a second round the blocks d and e are merged because their distance is below the parameter pD and they are aligned using the tolerance. Figure 2.5c show the merged blocks. Block f is contained into block d, so they are merged and the label *flow* is assigned. Figure 2.5c shows the final merging, the process stops because the weight of both blocks a and d is greater than the predefined stop condition $pA = 4$.

Algorithm 3 presents details about the algorithm for merging blocks. We only consider the composite blocks that have simple blocks as children and the weight of which is greater than the predefined stop condition parameter (pA). If it is the case we try to merge the children.

```
Data: composite block b
Result: W'_BoM, GM_BoM updated
if b.weight > pA then
    Children ← getChildren(b);
    foreach child ∈ Children do
        if child.weight < pA then
            Siblings ← getSiblings(child);
            foreach sibling ∈ Siblings do
                if child and sibling are aligned then
                    if distance between child and sibling less than pD then
                        if labels of child and sibling are not sectioning then
                            merge sibling with child as child;
                            label child from both labels;
                        end
                    end
                else
                    if child covers sibling then
                        merge sibling with child as child;
                    end
                end
            end
        end
    end
    if |getChildren(b)| = 1 then
        remove child of b;
    end
    if proportion of non merged small children is superior to 75% then
        remove children of b;
    end
else
    remove children of b;
end
```

Algorithm 3: Merging algorithm

2.6 Discussion

In this section we presented our approach to Web page segmentation. We aim segmenting a Web page without previous knowledge about its content. This allows segmenting different type of Web pages. The heuristic rules are based solely on rules defined in the Web standards, such as content categories.

We do not do any assumption about the text. However, this can be a weakness because in some cases analyzing the text can be relevant. For instance, two consecutive blocks that talk about different subjects should not be merged. Solving this issue would imply studying the semantics of the block content and is out of the scope of this thesis.

There are three different implementations of the BoM algorithm. One version is developed as a Ruby application, the second as a Java application and the third as a JavaScript library. The Ruby version is intended as functional prototype, the Java version to production environments for the European project SCAPE[2] and the JavaScript version for the open source community[3].

Introducing concept and techniques from the computer vision field of scanned document image segmentation allow having a more complete segmentation, as it contains more useful information for applications than most of the other segmenters.

[2] http://www.openplanetsfoundation.org/blogs/2014-02-12-scape-qa-tool-technologies-behind-pagelyzer-ii-web-page-segmentation

[3] https://github.com/openplanets/pagelyzer/tree/master/SettingsFiles/js

Chapter 3

Segmentation evaluation model

Evaluating web page segmentation algorithms is not an easy task. Usually, each algorithm proposes its own *adhoc* validation mechanism that can not be really applied to other approaches. This chapter attempts to close this gap by proposing a number of evaluation metrics that essentially measure how well the generated segmentation maps to a ground truth segmentation. This can be formulated as a graph matching problem, and we propose a number of metrics based on the generated matching to assess the quality of the generated blocks.

In this chapter, we present our evaluation model in order to measure the quality of a segmentation according to a discrepancy parameter (*i.e.:* determine how far the two segmentation are one from the others). The goal of the evaluation model is to compare an automated segmentation of a web page W with the corresponding ground truth, in order to determine its quality. Both segmentations are organized as non-hierarchical Manhattan layout (*cf.* section 1.3.1), in other words, they are flat segmentations. Our evaluation model is an adaptation to web pages of the model presented by [SKB08] for scanned page segmentation evaluation (see Section 3.1). The ground truth is manually designed, we explain in Section 4.4.2 how it was built for the evaluated collection. The comparison focuses on block geometry and content. The quality of a segmentation is evaluated by using the block correspondence and text covering measures. The block correspondence measures allows knowing to what extent the generated blocks match those of the ground truth. The text covering indicates to what extent the global content (expressed as a number of words) of the generated blocks is the same as the content of the page. At the end of the chapter an example is given to illustrate our approach.

We present the evaluation model adaptation (3.1), the representation of a segmentation (3.2), the representation of the evaluation (3.3), an example (3.4), finishing with a discussion (3.5).

3.1 Model adaptation

In order to adapt to web pages the model presented by Shafait et al. [SKB08] (*cf.* section 1.4.2) for scanned page segmentation evaluation we need to identify the different aspects of both type of documents. Shafait represent a segmentation of scanned documents images using a pixel-based representation. Each foreground pixel belongs to a zone or region. The evaluated documents (and the ground truth) must have the same dimension.

Their evaluation model defines several performance metrics to evaluate different aspects of the behavior of a scanned page segmentation in image form. These metrics allow measuring the correspondence of each pair of rectangles the segmentation and the ground truth. A region (or block) is significant if it the amount of foreground pixels associated with it is greater than a parameter.

By analogy, web pages consist of elements and text. In our adaptation, a block is significant if the amount of elements and text is greater than a parameter. Other features of our model are intrinsic to web pages, such as the block importance and the text coverage. They are explained in this chapter.

3.2 Representation of segmentation

In this section we model a segmentation in order to describe its evaluation. We describe the absolute and normalized representation of a segmentation (3.2.1 and 3.2.2), as well as the importance of blocks and how it is computed (3.2.3).

We present the concepts used along the chapter. We use the notation described in Section 1.3.2. We use the concepts of page, block and block graph based on the concepts described in section 1.3.1.

3.2.1 Absolute representation of a segmentation

Each block B is associated with its rectangle ($B.rect$), its label ($B.label$) and its weight ($B.weight$) (*cf.* Section 1.3.2). To each B we add three values: the amount of elements it covers ($B.ec$), the text associated to the block ($B.text$) in the original page W and the importance ($B.importance$). Note that $B.ec = |B.elements|$.

The importance of a block depends on the area covered by its rectangle. Section 3.2.3 explain how it is computed.

An absolute segmentation for the rendered DOM W, using the algorithm A and SC a set of stop conditions, is defined by the following function Φ (*cf.* Section 1.3.2):

$$\Phi_A(W, SC) \longrightarrow \left(W'_A, GM_A\right)$$

where W'_A is the block graph and GM_A is a set of rectangles representing the geometric model of the segmentation.

Consider W'_A as a rooted, planar and vertex-weighted tree. The root vertex is the *Page* block and the terminal vertices are the simple blocks. We consider the segmentation as flat, that is the $height(Page) \leq 1$. GM_A is the geometric model of the segmentation consisting of a set of rectangles.

3.2.2 Normalized Segmentation Representation

In order to compare two segmentations, we need to normalize the rectangles.

Given an absolute segmentation Φ_A, the geometric model of its normalized version $N\Phi_A$ fits in a $ND \times ND$ square, where ND is a fixed value, called Normalized Document Size. In our experimentation, we fixed this value to 100. Thus if $N\Phi_A$ is the normalized segmentation of Φ_A:

$$N\Phi_A(W, SC) \longrightarrow \left(NW'_A, NGM_A\right) \tag{3.1}$$

where NW'_A is the block graph of the normalized segmentation, NGM_A is the normalized geometric model. All the segmentation rectangles are normalized. Thus, the *Page* block

rectangle is normalized as:

$$NW'_A.Page.rect = \{0, 0, ND, ND\}$$

Each block rectangle is then normalized according to the stretch ratio of the page, *i.e.*

$$\forall\, b \in NW'_A,\ b.rect.x = \frac{ND \times W'_A.Page.rect.x}{W'_A.Page.rect.w}$$

The other values of the block rectangle (y, w and h) are normalized in the same way.

3.2.3 Block importance

The regions in a web page are not all equally important. A block is more important than another block if it contains more important information. Usually, important blocks are located in the most visible part of the page. A good segmentation algorithm must mostly find important blocks.

The block importance is obtained from the geometric model of the segmentation, that is the spatial features. A segmentation is mapped to a grid of $NP \times NP$, where NP is the Normalized Partition Size. This grid can be represented as a matrix *IM(NP,NP)*. Each cell of the matrix (im_{ij}) is assigned with a value representing the importance that a block has if it lies within this area. For instance, with the *window spatial features* defined by Song et al. [SLWM04], a highest importance is assigned to blocks found in the middle of the visible part of a web page, and a lower importance to blocks found outside of this area.

The computed importance of a block is the sum of the cell values obtained by mapping the block rectangle over the grid. The rectangle coordinates are divided by the constant NP. This defines two intervals, one for each dimension. If i and j respectively belong to those intervals, then the cell value im_{ij} is taken into account. Thus the computed importance of a block $B \in W'_A.Blocks$ is:

$$computed\_importance(B) = \sum_{ij} im_{i,j} \tag{3.2}$$

where

- $i \in \left[round(\frac{B.rect.x}{NP}), round(\frac{B.rect.w}{NP})\right]$ and,

- $j \in \left[round(\frac{B.rect.y}{NP}), round(\frac{B.rect.h}{NP})\right]$

In order to uniformize the importance we define $B.importance$ as the average importance of a blocks in a segmentation. The computed importance of each block is divided by the sum of all the computed blocks importance in a segmentation. Thus the importance of a block $B \in W'_A.Blocks$ is:

$$B.importance = \frac{computed\_importance(B)}{\sum_{b \in W'_A.Blocks} computed\_importance(b)} \tag{3.3}$$

3.3 Representation of the evaluation

In this section we model the evaluation itself, described in terms of input and output. We describe also the metrics used in for measuring the text covering (3.3.1) and the block correspondence (3.3.2).

The evaluation is described as a function that takes two segmentations and four constants as parameters. The two segmentations Φ_G and Φ_P are absolutes segmentations as described in section 3.2 producing the block graphs W'_G and W'_P. The four parameters are the relative tolerance (t_r), the importance tolerance (t_i), the Normalized Document size (ND) and the Normalized Partition size (NP) as defined in section 3.2.1 and 3.2.2. These parameters are described in detail in the following sections. The evaluation function returns a vector of metrics representing the quality of Φ_P with respect to Φ_G.

$$evaluate(\Phi_G, \Phi_P, t_r, t_i, \text{ND}, \text{NP}) = (\text{text coverage metric}, \text{correspondence metrics}) \tag{3.4}$$

The quality of a segmentation is measured in two complementary ways:

- Text covering : measures to which extent the global content (here expressed as the number of words) of the blocks in W'_P is the same as the content of the page W.
- Block correspondence : measures how well the blocks of W'_P match with the ones of W'_G.

The text coverage allow determining whether the segmentation has taken into account all the parts of the web page in terms of textual content. The block correspondence takes

into account the location and geometry of block. It allows for detecting which blocks were correctly discovered and which ones raised issues.

3.3.1 Measuring text coverage

The intuitive idea of evaluating the covering is to know if there is some content from the original page not taken into account by the segmentation. The covering of a segmentation Φ_A is given by the TC function, which returns the proportion of words of $W.text$ (*cf.* Section 1.2.1) that appear in the blocks of W'_A, as follows :

$$TC(\Phi_A, W) = \frac{\sum_{b \in W'_A.Blocks} words(b.text)}{words(W.text)}$$

For simplicity we denote the function $TC(\Phi_A, W)$ as TC. More complex functions can be used to measure the text coverage, but this is left for future work.

3.3.2 Measuring block correspondence

The block correspondence indicates whether the blocks rectangles of a segmentation match those of the ground truth.

Consider two normalized segmentations for a page W: a computed one $N\Phi_P$ and the ground truth $N\Phi_G$. The associated normalized block graphs are NW'_P (denoted P in the rest of the section) and NW'_G (denoted G). Figures 3.1(a) and (b) give respectively an example for G and P.

To compute the block correspondence, we build a weighted bipartite graph called *block correspondence graph* (BCG). We start with an example and then give the algorithm.

As seen on Figure 3.1(c), nodes of the BCG are the blocks of P and of G. An edge is added between each couple of nodes n_i and n_j such that the weight $w(n_i, n_j)$ of the edge is equal to the number of underlying HTML elements and text in the intersection of the regions covered by the rectangle of each of the blocks corresponding to the two nodes. If the blocks rectangles do not overlap in P and G, no edge is added.

Algorithm 4 shows how is built the BCG. If the blocks in P fits perfectly with the

Data: nodes $n_i \in G$,$n_j \in P$
Result: vertex (n_i,n_j) and its weight (if apply)
if $n_i.rect$ *is contained in* $n_j.rect$ **then**
    create vertex (n_i,n_j);
    $w(n_i, n_j) = n_i.htmlcover + n_i.textcover$;
else if $n_i.rect$ *contains* $n_j.rect$ **then**
    create vertex (n_j,n_i);
    $w(n_i, n_j) = n_j.htmlcover + n_j.textcover$;
else
    /* no vertex is created */
    $w(n_i, n_j) = 0$;
end

Algorithm 4: Algorithm for building the BCG graph

ground-truth blocks G, then the BCG will be a perfect matching. That is, each node in the two component of the graph has exactly one incident edge. If there are differences between the two segmentations, nodes of P or G may have multiples edges. If there is more than one edge incident to a node n in P (resp. in G), n is considered oversegmented (resp. undersegmented). Using these definitions, we can introduce several measures for evaluating the correspondence of a web page segmentation algorithm.

Intuitively, if all blocks in G are in P, this means that the algorithm has a good quality. If one set of blocks in G are grouped into one block in P or if one block in G is divided in several blocks in P then there is an issue with respect to the granularity but no error. We determine a segmentation error if one block in the ground truth is not found in the computed segmentation or if there are blocks that were "invented" by the algorithm.

The metrics for block correspondence are defined as follows:

1. **Correct segmentation** $C_c(\Phi_A)$, C_c for short. The number of one-to-one matches between P and G. A one-to-one match is defined by a couple of nodes (n_i, n_j), n_i in P, n_j in G, such that $w(n_i, n_j) \geq t_r$, where t_r is a threshold that defines how well a detected block must match to be considered as correct. For instance, in Fig. 3.1, there is an edge between node 2 and node B and another one between node 2 and node C. However, as the weight $w(2, C)$ is less than t_r, and the weight $w(2, B)$ is greater than t_r, B is considered as a correct block. The metric value for the example is $C_c = 2$. C_c is the main metric for measuring the quality of a segmentation.

2. **Oversegmented blocks** $C_o(\Phi_A)$, C_o for short. The number of G nodes having more than one edge. This metric measures how much a segmentation produced too small blocks. However, those small blocks fit inside a block of the ground truth.

In the example of Fig. 3.1, node 6 of the ground truth is oversegmented in the proposed segmentation. In the example, the metric value is $C_o = 2$ because nodes 6 and 2 are both over-segmented.

3. **Undersegmented blocks** $C_u(\Phi_A)$, C_u for short. The number of P nodes having more than one edge. The same as above, but for big blocks, where blocks of the ground truth fit in. For instance, on Fig. 3.1, node D of the proposed segmentation is undersegmented with respect to the ground truth, and the value for the metric is $C_u = 1$.

4. **Missed blocks** $C_m(\Phi_A)$, C_m for short. The number of G nodes that have no match with any in P. This metric measures how many blocks of the ground truth are not detected by the segmentation. One example is node 3 shown in the Fig. 3.1 and the value of the metric is $C_m = 1$.

5. **False alarms** $C_f(\Phi_A)$, C_f for short. The number of P nodes that have no match with any in G. This metric measures how many blocks are "invented" by the segmentation. For instance, in Fig. 3.1 node I has no correspondent in the ground truth making the metric value as $C_f = 1$.

Each metric C_x has a version, noted IC_x, that takes the importance of the blocks into account. In other words, C_x can be seen as the metric when all the blocks have the same importance. C_c is a positive measure, C_m and C_f are negative measures. C_o and C_u are "something in the middle", as they count "not too serious" errors : found blocks could match with the ground truth if they were aggregated or split. Note that the defined measures cover all the possible cases when considering the matching between G and P.

Thus, the evaluate function returns a vector made of all the computed metrics, *i.e.*

$$evaluate(\Phi_G, \Phi_P, t_r, t_i, \text{ND}, \text{NP}) = (TC, C_x, IC_x) \tag{3.5}$$

To evaluate the quality of the segmentation we define a score C_q, as the total number of acceptable blocks discovered, *i.e.* $C_q = C_c + C_o + C_u$ and $IC_q = IC_c + IC_o + IC_u$. Note that C_m is the complement of C_q where $C_q + Cm = |G|$.

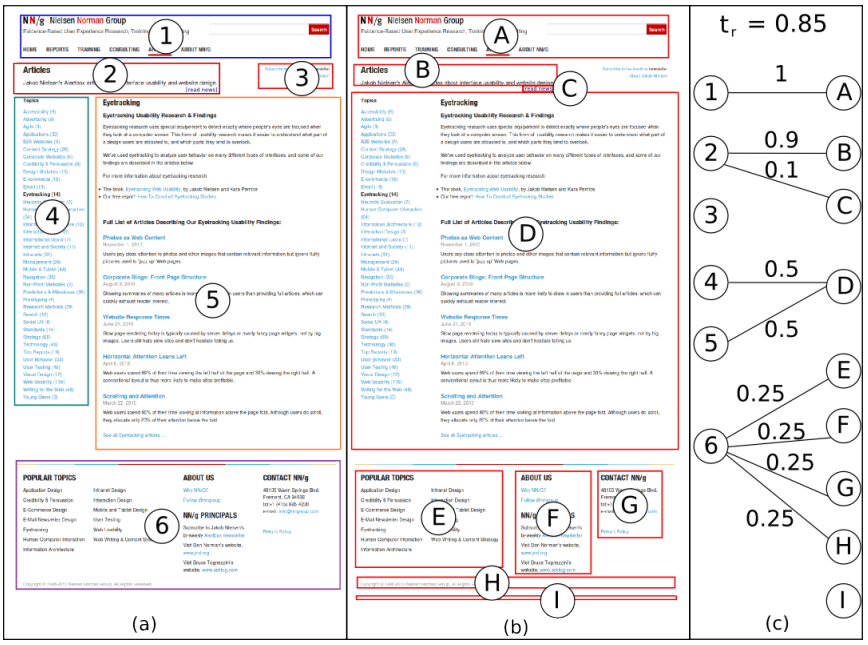


FIGURE 3.1: (a) Ground-truth segmentation. (b) Computed segmentation. (c) BCG.

3.4 Example

In this section web show how for an example how the importance is computed (3.4.1), the text coverage (3.4.2) and the correspondence measures (3.4.3).

In order to illustrate our approach let us assume four algorithms producing four example absolute segmentations (Φ_{P_1}, Φ_{P_2},Φ_{P_3} and Φ_{P_4}) and a ground truth (Φ_G), and the corresponding block graphs denoted P_1, P_2, P_3, P_4 and G, for the sake of readability .

The evaluation function is represented as $evaluate(\Phi_G, \Phi_{P_i}, 0.1, 0.3, 100, 10)$. Where G is the ground truth graph, P_i is any of the tested block graphs, $t_r = 0.1$, $t_i = 0.3$, $ND = 100$ and $NP = 10$.

Figure 3.2 shows the different normalized segmentations obtained with the four tested algorithms. The Pages and blocks are normalized to fit in a ND×ND = 100×100 square. The example Web page has four valid blocks: the logo, the search form, a set of images and the footer. However there is, at the top of the page, some text colored with white, therefore not visible to human assessor and it was not taken into account in the manual segmentation.

FIGURE 3.2: Normalized segmentations for ND=100 for an example web page

Importance	G1	G2	G3	G4
Computed	33	26	15	2
Average	0.43	0.34	0.19	0.02

TABLE 3.1: Computed and average importance values with $t_i = 0.3$

3.4.1 Computing the importance

We compute the importance for the ground truth. Figure 3.3 shows the grid *IM* over the normalized segmentation. The matrix *IM* is build following the recommendation of Song, where blocks in the center of the visible area get highest values than those in the exterior of this area. Based on IM we compute the importance of each block as follows: the absolute importance for block *G1* is the sum of cells $im_{1,4}$, $im_{1,8}$, $im_{2,4}$ and $im_{2,8}$. Table 3.1 shows the computed and average importances for Figure 3.3.

3.4.2 Computing text coverage

For each segmentation, we get the amount of words in the page as well as for each block. Table 3.2 shows the different word count for the four tested segmentations and the value of the *TC* function. Each B_i column represents the word count for a block in

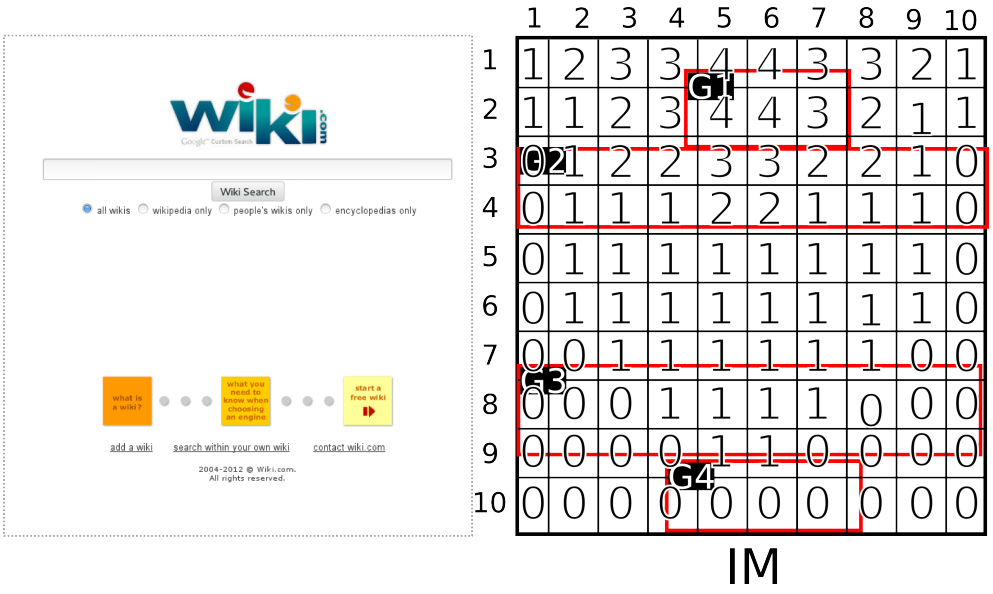


FIGURE 3.3: Grid for determining the importance on segmentation G of the example page

Segmentation	B1	B2	B3	B4	B5	W	TC
P_1	11	0	10	1	18	40	1.00
P_2	11	10	18	-	-	40	0.97
P_3	11	29	-	-	-	40	1.00
P_4	11	11	18	-	-	40	1.00

TABLE 3.2: Text coverage for segmentations in the example

the corresponding segmentation. Column W is the word count for the web page and TC the proportion of text coverage for each tested segmentation.

All the segmentations present a good text coverage but segmentation P_2 has missed two regions of the page (the logo and the images), therefore it has some problems partitioning the page.

From a human point of view, there is clearly an error in segmentation P_2, because it misses the block on the logo and the block over the images in the web page, and the logo is an important region of the web page. Computing the block correspondence allows giving more details on this error.

3.4.3 Computing block correspondence

The measures defined in Section 3.3 are represented on Table 3.3 and considering the importance, in Table 3.4. The two first columns are respectively and, with respect to

Algorithm	GTB	PTB	C_c	C_o	C_u	C_m	C_f	C_q	$TextCover$
P1	4	5	4	0	0	0	1	4	1.00
P2	4	4	2	0	0	2	1	2	0.97
P3	4	2	0	0	1	0	1	1	1.00
P4	4	3	1	0	1	0	1	2	1.00

TABLE 3.3: Block correspondence measures to segmentations in Figure 3.2 with $t_r = 0.1$

Algorithm	$IGTB$	$IPTB$	IC_c	IC_o	IC_u	IC_m	IC_f	IC_q
P1	2	3	2	0	0	0	1	2
P2	2	2	1	0	0	1	1	1
P3	2	2	0	0	1	1	1	1
P4	2	2	0	0	1	1	1	1

TABLE 3.4: Block correspondence measures (with importance)

the numbers of blocks in the ground truth (GTB) and the number of blocks obtained in each segmentation (PTB) ($IGTB$ and $IPTB$ respectively considering the importance). Figure 3.4 shows the four BCG graphs associated to each possible segmentation. The solid lines represent the significants edges, *i.e.* their weight is greater than the parameter t_r. The dotted lines represent the non significant edges. For instance in segmentation P_1 node B1 has no edge, it is counted as false alarm. The other nodes have exactly one edge, they are counted as correct block. In segmentation P_2, nodes G1 and G3 have no edges and they are thus counted as miss. B1 is a false alarm and G2 and G4 are correct blocks. In segmentation P_3, nodes G2, G3 and G4 of the ground truth have an edge with the node B2, making B2 undersegmented. The block G1 is missing because the edge between G1 and B2 is not significant. In segmentation P_4, node B2 is undersegmented (corresponds to G2 and G3), B3 is correct (corresponds to G4) and G1 is missing.

If we take the importance into account we note that segmentation P_1 has the best performance with a score of $IC_q = 2$. It matches the two important blocks in the ground truth. Segmentation P_2 is able to match one, but missed one. The segmentations P_3 and P_4 present the same performance: one undersegmented loc and one important missed block.

P_2 and P_3 have the same IC_q score. However, one would prefer P_2 to P_3 since it matches one important.

All segmentations present a false alarm. It corresponds to the white coloured block found at the top of the page.

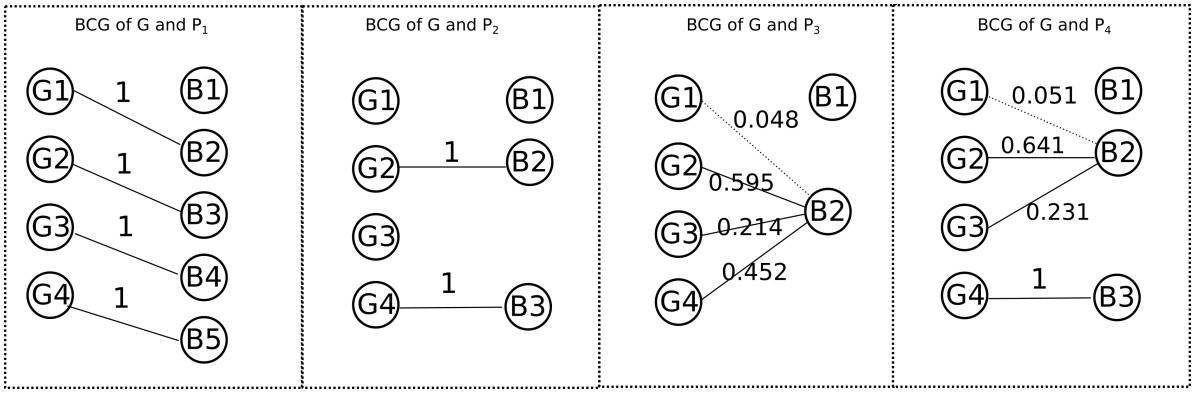


FIGURE 3.4: BCG for the four tested segmentations with $t_r = 0.1$

3.5 Discussion

In this section we presented our approach to web page segmentation evaluation. It is designed with the aim of evaluating web pages algorithms regardless of the approach they follow and their internal specification. We only consider the geometric model and the content to perform the evaluation.

We adapt some of the metrics defined by Shafait (*cf.* Section 1.4.3) to web page segmentation evaluation. We introduce an new metric, the C_q score. It is the aggregation of the correct, under and over segmented blocks. Under and over segmentations are not considered as an error, as it is in Shafait model. It indicates that the algorithm parameters are not set properly. We include also a version of these metrics considering the importance.

Representing web page segmentation algorithms by their outcome (the block graphs and their rectangles) allows us producing comparable versions on which we can compute the text coverage and block correspondence measures. For each measure, we produce a version that takes the importance into account. This allows favouring algorithm that correctly detect important blocks.

Chapter 4

Experimentation

In this section we present our experimental evaluation of BoM and other algorithms, according to the evaluation method described in section 3.3. A dataset composed of 200 pages annotated by human assessors is used as ground truth. Four algorithms (among the ones introduced in Chapter 1) were evaluated, based on the measures defined in Section 3.3. These measures evaluate different aspects of a segmentation algorithm for a given quadruple (page, render engine, algorithm, stop condition) as a parameter. In section 4.1 we present an overview of the experimentation. In section 4.3 we present the segmentations algorithms and how they are used in the experiments. In Section 4.2 the block descriptors are introduced. Section 4.4 presents the dataset construction. Section 4.5 presents the experiments and the results. We conclude with a discussion in Section 4.6.

4.1 Overview

Our evaluation framework allows running different Web page segmentation algorithms on a collection of Web pages and measuring their correctness, as defined in Section 3.3.

Four algorithms are tested, adapted in such a way that it was possible to extract the page, the block rectangles, the HTML and the word counts. At a glance, the framework gets an URL, a collection and a predefined stop condition and produces the vector with the scores described in Section 3.3.2, using the ground truth.

A set of block descriptors are extracted from each evaluated algorithm. A block descriptor describes the block in terms of its attributes, its geometry and other information necessary to build the segmentation dataset.

4.2 Block descriptors

A block descriptor is used to describe the blocks into a segmentation. It contains the algorithm name, the url, the document size, the amount of words in the Web page, the stop condition, the id, its rectangle coordinates, the amount of elements and the text it covers (*cf.* Section 3.2.1).

Each block B is described by a register with the following format:

Algorithm	a string identifying the algorithm
url	the Web page url
Document size	The document size is expressed as width and height
words(W.text)	number of words in the Web page
Stop Condition	The parameters of the algorithm
Block ID	String identifying the block.
B.rect	The block rectangle expressed in absolutes coordinates x,y and width and height
B.ec	The number of elements covered by the block
words(*B.text*)	the number of words present in a block

Algorithms implementation must compute this information for evaluation purpose. We choose the comma separated values (CSV) format for block descriptors.

4.3 Tested segmentation algorithms

In this section we give a short description of the algorithms we evaluated and how they were adapted to obtain the block descriptors needed to evaluate them. We choose those algorithms as a representative sample of the state of the art. We would have like to include the GraphBased algorithm but no implementation is supplied by the authors.

We adapted the implementation of the tested algorithms in order to get a flat segmentation and then the block descriptors needed for the comparison.

For algorithms where the source code was available (BoM, BlockFusion and JVIPS), the adaptation has been made on the source code. For VIPS, the adaptation has been made on the output.

In Sections 4.3.1 to 4.3.4 we describe the algorithms and their adaptation. In Section 4.3.5 we show a summary with technical details of the adaptation.

4.3.1 BF (BlockFusion)

In this section we give a short description of the Blockfusion (BF)[KN08] algorithm and how we adapt Boilerpipe (which implements BF) to our experiments.

4.3.1.1 BF Description

The BlockFusion algorithm uses the text density as a valuable heuristic to segment documents. The text density is calculated by taking the number of words in the text and dividing it by the number of lines, where a line is capped to 80 characters. A HTML document is then first preprocessed into a list of atomic text blocks. The density is computed for each atomic block. Blockfusion use an HTML file as input, not necessarily rendered.

Iteratively, two adjacent blocks are merged if their text densities are below a certain threshold ϑ_{max}. The value of this threshold represents the stop condition of the segmentation. The authors report that its optimal value is $\vartheta_{max} \approx 0.38$ and we take it as is. This algorithm use the bottom-up strategy (*cf.* section 1.3.3).

4.3.1.2 BF Adaptation

BF does not take the DOM into consideration during the segmentation. In order to get the rectangles of the segmentation, we thus need to modify its implementation. The BoilerPipe [1] application is modified, changing the input and the output of the

[1] `https://code.google.com/p/boilerpipe`

application and modifying the *TextBlock* class merging procedure. Each text element in the input Web page is wrapped into a *span* tag, with two extra attributes (*rect* and *words*). These attributes represent the bounding box of that text element and its word count, respectively. The method *merge()* of class *TextBlock* was modified in order to consider the *span* and both attributes. The output document then contains the *span* elements contained in the text blocks chosen by BF with the corresponding *rect* and *words* attributes. The $B.rect$ and $words(B.text)$ are taken from the attributes while the $B.ec$ has the value of one (1), representing the text element.

4.3.2 BoM (Block-o-Matic)

In this section we present the adaptation of the JavaScript implementation of the Block-o-Matic segmentation algorithm[2], described in Chapter 2.

The rectangles are taken from the simple blocks of the block graph W'_{BoM}. For each terminal block the *words(B.text)* and the $B.ec$ are extracted from the text elements and from the associated DOM elements, respectively. The $B.rect$ is extracted from the *dim* attribute of the block class.

4.3.3 VIPS (Vision-based Web Page Segmentation)

In this section we give a short description of the VIPS Web page segmentation algorithm [CYWM03] and its adaptation to obtain the block descriptors.

4.3.3.1 VIPS Description

The VIPS algorithm segments Web pages by analyzing their rendered version. It first develops a vision-based content structure, which analyses the page with visual cues present in the rendered page instead of the HTML source code. This structure is built by splitting a page into a 3-tuple consisting of a set of visual blocks, a set of separators, and a function that describes the relationship (shared separators) between each pair of blocks of a page. Separators are for example vertical and horizontal lines, images similar to lines, headers and white-space. This structure is built by going top-down (*cf.* Section 1.3.3)

[2] `https://github.com/asanoja/web-segmentation-evaluation/tree/master/chrome-extensions/BOM`

through the DOM tree and taking both the DOM structure and the visual information (position, color, font size) into account.

VIPS detects separators by splitting the page around the visual blocks so that no separator intersects with a block. Subsequently, it assigns weights to the separators, according to certain predefined heuristic rules. From the visual blocks and the separators it can then build the vision-based content structure of the page, using the Degree of Coherence (*DoC*) of each block for determining the stop condition.

4.3.3.2 VIPS Adaptation

Using the Dynamic Linked Library (DLL) provided as implementation of VIPS[3], we parse the output XML document to obtain the four values.

The *B.rect* are obtained from the leaves nodes (LayoutNode elements) of the XML document using the ObjectRect's attributes. The *B.ec* count is taken from the *DOMCldNum* attribute and the *words(B.text)* from the *content* attribute of each *LayoutNode*.

4.3.4 jVIPS (Java VIPS)

In this section we describe the Java version of the VIPS algorithm [Pop12] and its adaptation to our experiments.

4.3.4.1 JVIPS Description

jVIPS is another implementation of the VIPS model proposed by Cai [CYWM03]. Hence, the predefined stop condition parameter is the same as VIPS : *pDoC*. JVIPS is implemented in Java using the CSSBox rendering engine. The difference between VIPS and jVIPS resides in two of the heuristic rules, the version of jVIPS prohibiting splitting some blocks that VIPS would split. This implies that jVIPS often generates blocks as wide as the Web page width.

[3] `http://www.cad.zju.edu.cn/home/dengcai/VIPS/VIPS.html`

This algorithm has been referenced and used in several projects, as an alternative to VIPS in open source environments, so we think it is worthy to include it in our evaluation. This algorithm uses the top-down strategy (*cf.* section 1.3.3).

4.3.4.2 JVIPS Adaptation

With jVIPS, obtaining the required data for evaluation was straightforward because the source code is publicly available[4]. When the visual content structure is completed (the stop condition is met) the *B.rect*, *words(B.text)* and *B.ec* are obtained from the *VisualStructure* class attributes.

4.3.5 Summary

In Table 4.1 it is resumed the availability of an executable, the source code and technical details of the implementation for each algorithm.

Algorithm	**Executable**	**Source code**	**Technical remarks**
Blockfusion	Yes	Yes	It is integrated deep inside BoilerPipe application.
BoM	Yes	Yes	Cross-browser implementation. Ruby, Java and JavaScript version available. Can work also as browser extension
VIPS	Yes	No	Only for Microsoft operating systems and for Internet Explorer version 6
JVIPS	Yes	Yes	Java version of VIPS

TABLE 4.1: Segmentation algorithms been evaluated

4.4 Dataset construction

In this section we describe the method to build the dataset of annotated Web pages (4.4.1) that serves as ground truth for the evaluation of segmentation algorithms. The main motivation of this task is to evaluate the performance of our segmentation algorithm Block-o-Matic (BoM) and compare it with state of the art algorithms. To accomplish

[4] `https://github.com/tpopela/vips_java/`

that, a set of pages have been crawled and annotated with the Manual-Design of Blocks (MoB) tool to conform a Ground Truth (4.4.2).

The dataset can be consulted in `http://www-poleia.lip6.fr/~sanojaa/BOM/inventory/`.

4.4.1 Dataset organization

The dataset holds the offline version of Web pages, together with their segmentations obtained by the different algorithms (including the ground truth), organized in categories.

It is designed as a Web application organized in different levels of detail. Figure 4.1 shows the general architecture of the dataset repository. Within a collection, each page

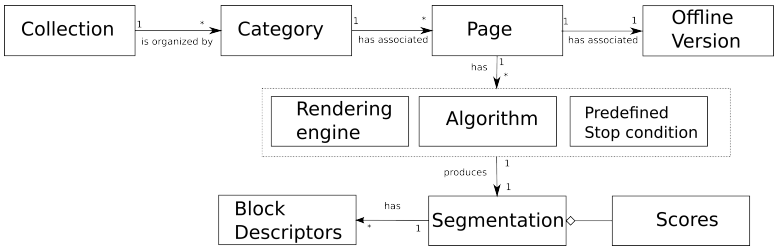


FIGURE 4.1: Dataset architecture

is rendered with different rendering engines with different predefined stop conditions values. To each quadruple (page, render engine, algorithm, predefined stop condition) corresponds a segmentation performed on that page, and rendered by that engine, using one algorithm with a predefined stop condition. A set of block descriptors (*cf.* Section 4.2) and the vector scores are associated to the segmentation (*cf.* Section 3.3.2) Web pages are taken from the GOSH (GOogle SearcH) collection that we built. It is described below.

4.4.1.1 GOSH Collection

Web pages in this collection are selected with respect to their category. This selection is based in the categorization made by Brian Solis [Sol14], "The Conversation Prism". It depicts the social media landscape from ethnography point of view. In this work, we considered the five most common of these categories, namely Blog, Forum, Picture, Enterprise and Wiki. For each category, a set of 25 sites have been selected using Google

search to find the pages with the highest *PageRank*. Within each of those sites, one page is crawled [5]. The GOSH collection contains 125 pages.

4.4.1.2 Rendering

Different rendering engines are used. They are encapsulated using Selenium WebDriver[6]. The Selenium Chrome driver is used for BoM and BF implementation while Internet Explorer driver is used for VIPS implementation. The CSSBox rendering engine for JVIPS.

4.4.1.3 Collection post-processing

A Web page rendered with different engines may result in differences in the display. The most common case are the white spaces between the window borders and the content.

We must assure that all renders of the same Web page have the same dimensions. For that reason, we check the above mentioned white space and remove them.

The average importance is computed for all segmentations in the dataset (*cf.* Section 3.2.3)

4.4.2 Ground truth construction

The human assessor selects a set of elements that compose a block. Then we deduce the bounding rectangle of the block and compute the word and elements count. This is a time consuming and error prone task. To speed up the process we have developed the tool *MoB* (Manual-design of Blocks)[7]. It assists human assessors to select the elements that form a block and automatically extract all the information needed.

The tool provides to the user a partial segmentation with candidates blocks. These blocks corresponds to the DOM elements that have content (e.g.: text and images). It provides the following operations:

[5] `https://github.com/asanoja/web-segmentation-evaluation/tree/master/dataset`
[6] `http://docs.seleniumhq.org/projects/webdriver`
[7] http://www-poleia.lip6.fr/ sanojaa/BOM/

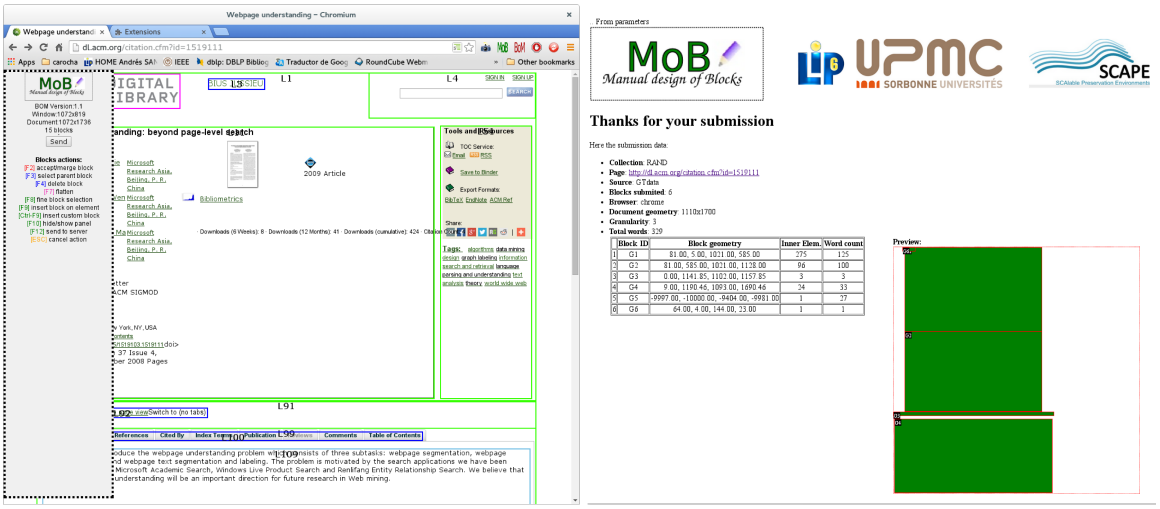


FIGURE 4.2: Screenshot of the MoB tool

- *Block selection.* Click on the area of a block to select it.
- *Fine block selection.* Select blocks that are covered by other ones.
- *Accept block.* The selected block becomes a terminal block (leaf).
- *Merge two blocks.* After selecting two blocks make them one.
- *Delete a block.* Remove the selected block, its children are passed to the parent of the deleted block.
- *Insert block on element.* Create a new block based on element clicked.
- *Insert custom block.* Given two coordinates draw a block.
- *Add label.* Assign a label to a block.
- *Flatten segmentation.* Remove non-terminal blocks.
- *Resolve overlapping.* Try to adjust blocks geometry to avoid overlapping (experimental)
- *Send to repository.* Send current segmentation to the repository server becoming part of the dataset.

Figure 4.2a shows the editing environment of the tool while in Figure 4.2b the final segmentation (with block descriptors) is sent to repository.

The ground truth was assessed by human assessors at LIP6 laboratory. We plan to crawl a bigger set of pages and to include other assessors to do this task.

4.5 Experiments and results

In this section, we present the results of evaluating the four segmentation algorithms described in Section 4.3. In Section 4.5.1 (respectively 4.5.2) we present how the parameters of the tested algorithms are set (respectively of the evaluation). Sections 4.5.3 to 4.5.4 present the results of the evaluation: block correspondence and text coverage.

The algorithms were evaluated on the GOSH collection based on the measures defined in Section 3.3. These measures evaluate different aspects of a segmentation algorithm for a given quadruple (page, render engine, algorithm, granularity) as parameter.

4.5.1 Setting the stop condition parameters

The accuracy of the measures directly depends on the way the ground truth is built. If the human assessors defined blocks of a certain granularity in the ground truth, the stop condition of each algorithm need to be adjusted accordingly.

In the present work our goal is to detect blocks of medium size. We do not focus neither in detecting only large blocks, such as header, menu, footer and content, or in detecting blocks at a too high level of detail (sentences, links or single images). Instead, we focus on detecting part of the page that represent significant pieces of information, such as a blog post, table of content, image and caption, set of images, forum response, and so forth. This is a most challenging task for segmentation algorithms.

Thus, in the following experiments, the parameters were set accordingly, so that each algorithm produces medium size blocks. In table 4.2 are listed the parameters used for each algorithm.

Algorithm	**Parameter**
Blockfusion	$\vartheta_{max} = 0.38$
BoM	$pA = 5$, $pD = 50px$, $pND = 100$
VIPS	$DoC = 4$
JVIPS	$DoC = 4$

TABLE 4.2: Segmentation algorithms parameters

4.5.2 Setting the thresholds

Setting the relative threshold t_r (*cf.* Section 3.3.2) is not so obvious, as the notion of "good block" is quite subjective.

In this work, we fixed t_r to 0.1 as we observed, on a significant number of example, that it corresponds to our notion of good block. In the future, we plan to perform supervised machine learning with a large number of users to determine the right value. Each user will annotate the segmentation blocks with the corresponding block in the ground truth if (s)he thinks that the blocks sufficiently match. The t_i parameter is set to 0.1, based on our experience with the collection. Because rendering engines may produce some small differences in their rendering, we introduce a geometric tolerance t_t to help in the comparison of the rectangles. The value of this parameter is fixed based on the experience in working on the collection. It is category-dependent. In general cases blocks rectangles do not differs in more than $\pm$ *2* pixels. For the whole collection, the best value appears to be *1* pixel. In table 4.3 are listed the parameters used for the evaluation.

Algorithm	**Parameter**
Relative threshold	$t_r = 0.1$
Importance threshold	$t_i = 0.1$
Geometric tolerance	$t_t = 1$

TABLE 4.3: Segmentation evaluation parameters

4.5.3 Computing block correspondence

We computed the different metrics for block correspondence, as defined in Section 3.3.2. Table 4.4 shows the scores (average of the metrics on all the documents of the collection) obtained by the different algorithms on the GOSH collection. The *GTB* column represents the total blocks in the ground truth. Table 4.5 shows the the average of the metrics taking the importance into account. The *IGTB* column represents the total important blocks in the ground truth.

Algorithm	C_c	C_o	C_u	C_m	C_f	C_q	GTB
BF	1.10	0.20	0.29	4.74	1.06	1.59	7.08
BoM	3.34	0.49	0.54	1.76	1.68	4.37	7.08
JVIPS	1.71	0.36	0.81	2.64	6.95	2.89	7.08
VIPS	1.55	0.40	0.73	2.56	2.53	2.69	7.08

TABLE 4.4: Correspondence metrics for the global collection with $t_r = 0.1$ and $t_t = 1$

Algorithm	IC_c	IC_o	IC_u	IC_m	IC_f	IC_q	$IGTB$
BF	0.57	0.15	0.28	1.57	0.64	1.00	2.76
BoM	1.41	0.18	0.32	0.70	0.77	1.91	2.76
JVIPS	0.43	0.17	0.48	1.47	1.06	1.08	2.76
VIPS	0.49	0.23	0.50	1.06	0.94	1.21	2.76

TABLE 4.5: Correspondence metrics with importance for the global collection with $t_r = 0.1$, $t_t = 1$ and $t_i = 0.1$

Several observations can be done:

- BoM obtain the best result for score C_q, as it is more accurate, thanks for its high values of correct blocks C_c. It produces very few serious errors (C_m, C_f) with respect to the other algorithms. It deals good with the chosen the stop condition parameter, as indicated by its low values for C_o and C_u. BoM present the highest value for the IC_q score, indicating that it does not miss many important blocks.

- BF obtain the worst result for C_q, but with a low level of false alarms. In other words, BF does not detect all the correct blocks (mainly, it misses the blocks that are not located in the center of the page) but detects good blocks, with a rather good granularity. This is mainly due to the fact that BF uses the text density for determining blocks. As the blocks on the sides of the pages have a low text density, it is hard for BF to detect them. Important blocks are located in the center of the visible area, that is where BF finds the majority of blocks. For that reason the score IC_q is slightly better than the C_q score.

- VIPS and JVIPS have comparable results in terms of correct blocks and missed blocks. However, JVIPS generates a lot of false alarms. This is due to a specific heuristic rules used in JVIPS that tends to detect blocks as wide as the page width, as mentioned in section 4.3.4. This is relevant for blocks like headers or footers, but not for the content located in the center of the page. JVIPS present a better C_q score that VIPS because it find slightly more correct blocks than VIPS. Conversely, VIPS has better performance than JVIPS considering the important blocks. That is maybe because JVIPS finds important blocks as headers, navigation but misses important blocks located at the visible part of the content.

In order to study the adequacy between segmentation algorithms and Web page categories, tables 4.6 and 4.7 list the values for the C_q and IC_q metrics by category. Figure 4.3 and 4.4 show both scores obtained by the different algorithms for the five categories above mentioned. Each algorithm is represented by a color bar, the dashed lines are the

Category	BF	BoM	JVIPS	VIPS	AVG	GTB
blog	1.32	4.11	2.79	2.68	2.72	7.26
enterprise	1.58	4.00	2.58	2.54	2.68	6.17
forum	2.75	6.38	4.19	3.50	4.20	10.69
picture	1.32	3.64	2.20	2.28	2.36	5.32
wiki	1.29	4.38	3.14	2.71	2.88	7.29

TABLE 4.6: C_q average values by categories for the global collection with $t_r = 0.1$ and $t_t = 1$

Category	BF	BoM	JVIPS	VIPS	AVG	IGTB
blog	0.63	1.37	0.74	1.00	0.93	2.11
enterprise	1.13	2.13	1.38	1.46	1.52	2.88
forum	1.25	2.50	0.94	1.13	1.45	3.63
picture	1.04	1.92	1.00	1.24	1.30	2.68
wiki	0.95	1.71	1.24	1.14	1.26	2.67

TABLE 4.7: IC_q average values by categories for the global collection with $t_r = 0.1$, $t_t = 1$ and $t_i = 0.1$

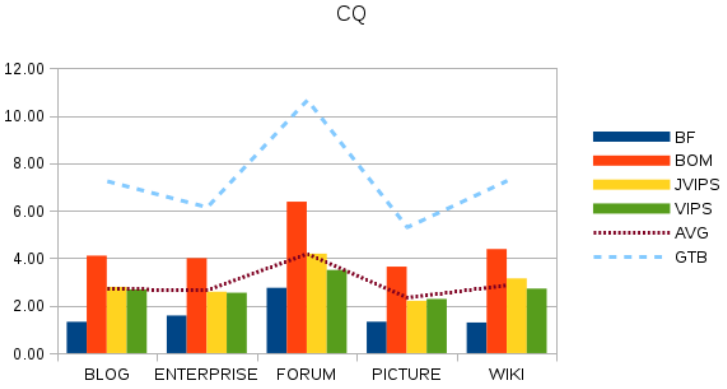


FIGURE 4.3: Average C_q score by categories for table 4.6

averages over all the collection. The *AVG* line represent the average correct blocks while *TAVG* represent the average of expected blocks in the ground truth.

We make the following observations:

- The best results for C_q are obtained for the Picture collection. The reason is probably because picture pages have a regular and simple structure. This observation also holds, though attenuated, for the Enterprise category. For the metric IC_q the best performance is for the blog category. The visible part of these kind of pages are commonly formed by a header, lateral menus and the beginning of the blog post. They are standard in almost all blogs.

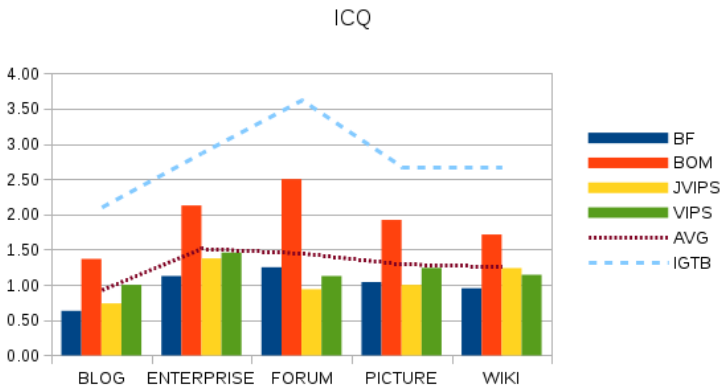


FIGURE 4.4: Average IC_q score by categories for table 4.7

- The worst results for the C_q metric are obtained for the Forum category. The reason for this, is probably that forum pages are constituted of several question/answers blocks, each of them having a complex structure (including avatars, email addresses, and so on) which is not easy to detect by algorithms. For the IC_q metric the worst performance is also for the forum category, probably because blocks that identify the forum and the question are missed.

- BF performs well for Forum. As those kinds of pages contain many text (question/responses) blocks, the text density is sufficient to detect most of them, but not those surrounding the main content. Algorithms miss a lot of blocks in this category. Even BoM, which performs over the average, still misses the half of the blocks. The worst performance for the IC_q score for BF is in the blog category. It misses blocks of the top of the visible part of pages, which are usually considered as important. This is mainly due to the fact that BF uses the text density for determining blocks. As the blocks on the top of the pages have a low text density, it is hard to BF to detect them.

- JVIPS has problems with the Blog collection. These pages do not have blocks that occupy the whole width of the page. Instead, they have many small blocks allocated horizontally that JVIPS cannot detect. This observation also holds, though attenuated, for the IC_q category. Probably it misses the menus and the blog title.

4.5.4 Computing text coverage

We computed the text coverage as defined in Section 3.4.2. Table 4.8 gives the (rounded) values for the whole collection and for each category of pages. The first observation is that the coverage obtained by all the algorithms are quite high. This means that each

Algorithm	all	forum	blog	wiki	picture	enterprise
BF	56	42	85	91	14	37
BoM	69	75	87	91	61	71
JVIPS	86	100	80	98	87	92
VIPS	95	96	95	94	95	95

TABLE 4.8: Text coverage values for each algorithm

of them is able to perform the basic task of text extraction. It appears that Blockfusion does not perform well for Blog and Wiki. However, this is mainly due to the fact that, for those categories, BF misses a lot of blocks (as seen above), thus misses their content.

4.6 Discussion

In this section we presented the experiments and results of applying our approach to Web page segmentation evaluation. The experiments are designed in such a way that segmentation algorithms must find midsize pieces of information in Web pages. The results show that BoM presents the best performance among the four segmentation algorithms tested. This is due to the fact that the combination of both strategies (composite block detection and merging) presented used by BoM algorithm (*cf.* Chapter 2) is close to the way assessors build and discover blocks in the ground truth.

After applying the evaluation model defined in Section 3 we observe that metric C_c for the correct block has very low values. The concept of correct block is very strict (same rectangle and content significantly similar). This observation is shared with that of Kreuzer et al. [Kre13]. They discuss the low values of the F-score measure of correct blocks in their random and popular datasets. They report that for almost every evaluated algorithm the correct blocks are very few.

However, we think that the model and the measures presented in this work helps to have a more integral comparison of the segmentations. As we mentioned, an over-segmentation is not an serious error, neither an under-segmentation. In fact, this appears frequently in Web page segmentation. Kreuzer et al. only consider a block as correct or missed. By taking into account the over and undersegmentation, it is possible to have a score (C_q) that better represents the performance of a segmentation.

However, some precisions must be given concerning the adaptation of algorithms to our model. Algorithms such as BF do not include any other information than text, thus it is hard to adapt. It has some problems when a complete segmentation is expected, not

only the main content text. The other three algorithms was not that hard to adapt, and their segmentations are complete and comparables.

We do not include algorithms following the graph theoretic approaches since they are hard to implement. Only the specifications are published in the articles. There is no public implementation available. We tried to contact them but the responses were either vague or evasive. Adding more algorithms is left as future work. However, the chosen algorithms allow us testing our evaluation framework.

The inclusion of two versions of the metrics, one considering the importance and the other not, allows us having a better understanding of the performance of a segmentation. That allow us to observe how effective an algorithm can be under this particular situation. We plan, as a future work, to extend the model with more metrics.

Chapter 5

Applications

In this chapter we present two applications of the BoM algorithm. We present the Pagelyzer tool for Web page version comparison, which is the main contribution of the LIP6 to the European Project SCAPE[1] (5.1). Then we describe the migration of archived Web pages from HTML4 to HTML5 format, in order to avoid emulation due to format obsolescence (5.2).

5.1 Pagelyzer

Pagelyzer is a tool developed in the context of the European project SCAPE. It compares two Web pages versions and decides if they are similar or not. Figure 5.1 shows two versions of an example Web page. The blocks marked in green color are blocks the content of which has changed from one version to another. Blocks with unchanged content are in red.

Applications of Pagelyzer are mainly:

- *Web harvesting*: check if a crawl is correct and adjust crawl frequency (higher frequency if page changes).
- *Migration*: check if migration (e.g. arc to warc) operation works correctly.

[1] `http://www.scape-project.eu/`

FIGURE 5.1: Change detection example in two Web page versions

5.1.1 How does it work?

Pagelyzer takes two urls and two browsers types (e.g. Firefox as default and chrome) and one comparison type as input (image-based, hybrid or content-based) and output a score indicating if they are similar or dissimilar.

It can be described in three steps:

1. For each url given as input, it captures the screen in PNG format and also produces an HTML document integrating the visual cues, called Decorated HTML. This allows saving the state of a browser at capture time and make the solution independent from a particular browser.

2. In a second step, each page is segmented using the BoM algorithm. At the end of this step, two XML trees, representing the segmented Web pages are returned. The XML format of such trees is called ViXML [PBSG10]. The Web page segmentation is considered only for the structure and hybrid comparison types. For more details about Web page segmentation, the reader can refer to Section 2.

3. In a third step, visual and structural descriptors are extracted. Images (snapshots) are first described by color descriptors and also by SIFT descriptors. For image representation, Bag of Words (BoWs) representation is used. Structural descriptors are based on Jaccard indices and also based on the Vi-XML files differences. The structural and visual differences are merged to obtain a similarity vector used to determine if the two urls are similar or disimilar, according to [LTGC12].

5.1.2 Implementation

There are two releases: the functional prototype and the final release. The first version of the tool was developed as a Ruby application while the final release was developed as a Java JAR package.

5.1.2.1 Functional prototype

This version is composed of three components: capture, analyser and change detection tools. It uses the Ruby version of BoM (*cf.* Section 2.6).

The capture tool, performs the first step of the Pagelyzer process. Web pages are processed using Selenium Web driver. The visual cues are obtained through JavaScript script that are injected to the browsers and the screen-shots are obtained using selenium features.

In earlier versions of the tool, VIPS [CYWM03] was used to segment Web pages. Using Block-o-Matic removes the VIPS restriction of using Internet Explorer as a Web browser and also enhances the precision of visual block extraction and the hierarchy construction. Ruby 1.9.2 was used as programming language for implementing the segmentation algorithm, Nokogiri libraries was used for HTML/XML manipulation.

Figure 5.2 shows the change detection process for each comparison type. For the *image* comparison (considering only the activities within the long dashed boxes) the process starts with two urls and two browsers for each url. For both url, the screenshot is taken and passed as input to the *Marcalizer* component which finally gives a score. Marcalizer was developed by the MLIA team at LIP6.

In the same figure, we describe the *structure* comparison (activities within the dotted boxes). For the two urls, the decorated HTML is captured and passed as input to the *pageanalyzer.rb* component. The latter produces as output two ViXML files, one for each url. They are the input for the *ViDIFF.jar* which produces a *Delta file*, describing the changes between the to versions according to [PBSG10]. This delta file, together with both xml files, is the input to the *Marcalizer* component which gives the final score.

The *hybrid* comparison involves all the activities within the solid lined box. It shows how both image and structure comparison type are merged using all components of the system.

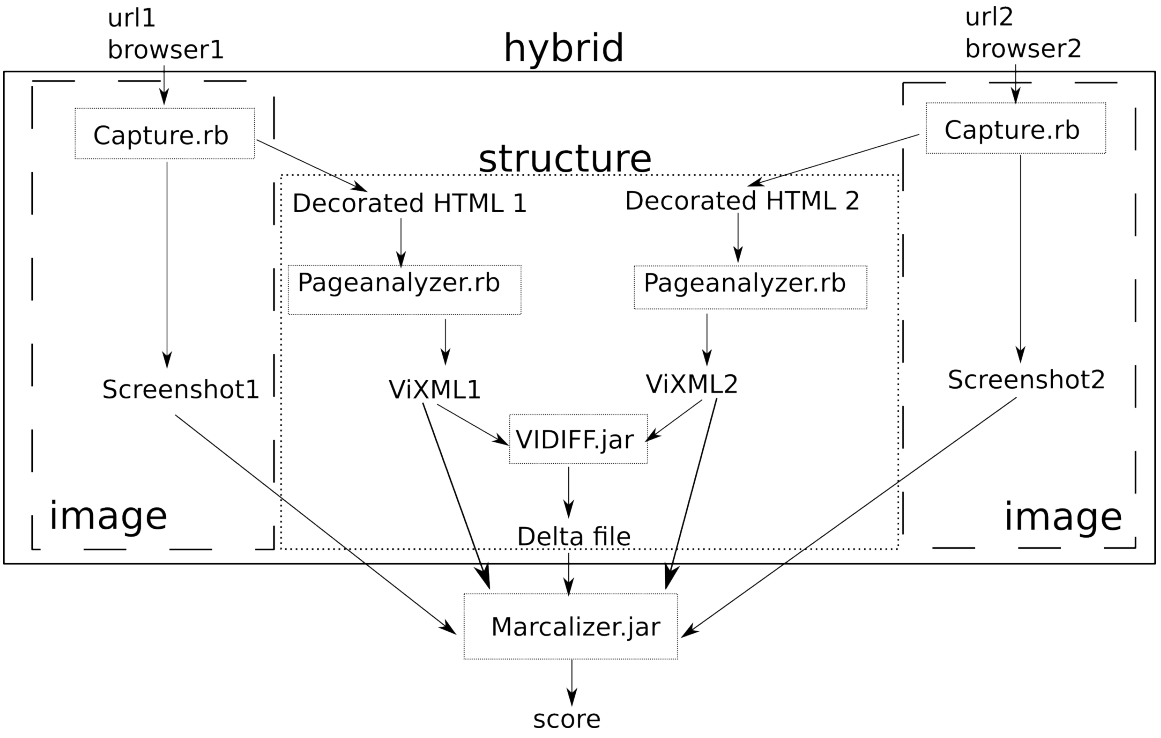


FIGURE 5.2: Change detection flow for image, structure and hybrid comparison types in the prototype version

We chose the Ruby language because it allows producing a prototype rapidly. Indeed, it was important to get feedback from other project contributors. This prototype has been used for early testing, but it is not suitable as a product because it uses files to interchange data between components. While in a standalone configuration this is no major problem, in a high performance environment (like Hadoop) it adds an unnecessary overhead and inefficient use of resources.

5.1.2.2 Final release

In order to improve efficiency (use of resources) in high performance computing environments, the option to use input data as streams and headless browsers has been chosen. Screenshots and source code are captured directly from their hosting sites and processed internally with no extra files needed. Moreover, the Pagelyzer tool was rewritten with the Java programming language in order to ease the integration of *Marcalizer* and with other components developed by other teams in the SCAPE project. This version of Pagelyzer uses the Java version of BoM (*cf.* Section 2.6).

Figure 5.3 shows the modifications made to the architecture of Pagelyzer for the final release version. The ViDIFF component was removed and its functionality was included

in the Marcalizer component. One of the major changes in this release is the replacement of the capture component with a fork of the *browser-shot-tool* developed by the Internet Memory Foundation (IM)[2]. The decorated HTML was replaced with by the JSON version of a selenium webdriver instance. In this way the segmentation tool interact directly with a browser instance. In order to optimize the configuration in high performance computing environments a configuration file was included. This minimize the parameters needed to be passed for each invocation of the tool.

A demo of the final release is in `http://scape.lip6.fr/scape-demo-sites/pagelyzer/`. It has been presented at the Scape final workshop joint with the DL'14 conference in London. The source code can be accessed in the Github website `https://github.com/openplanets/pagelyzer`

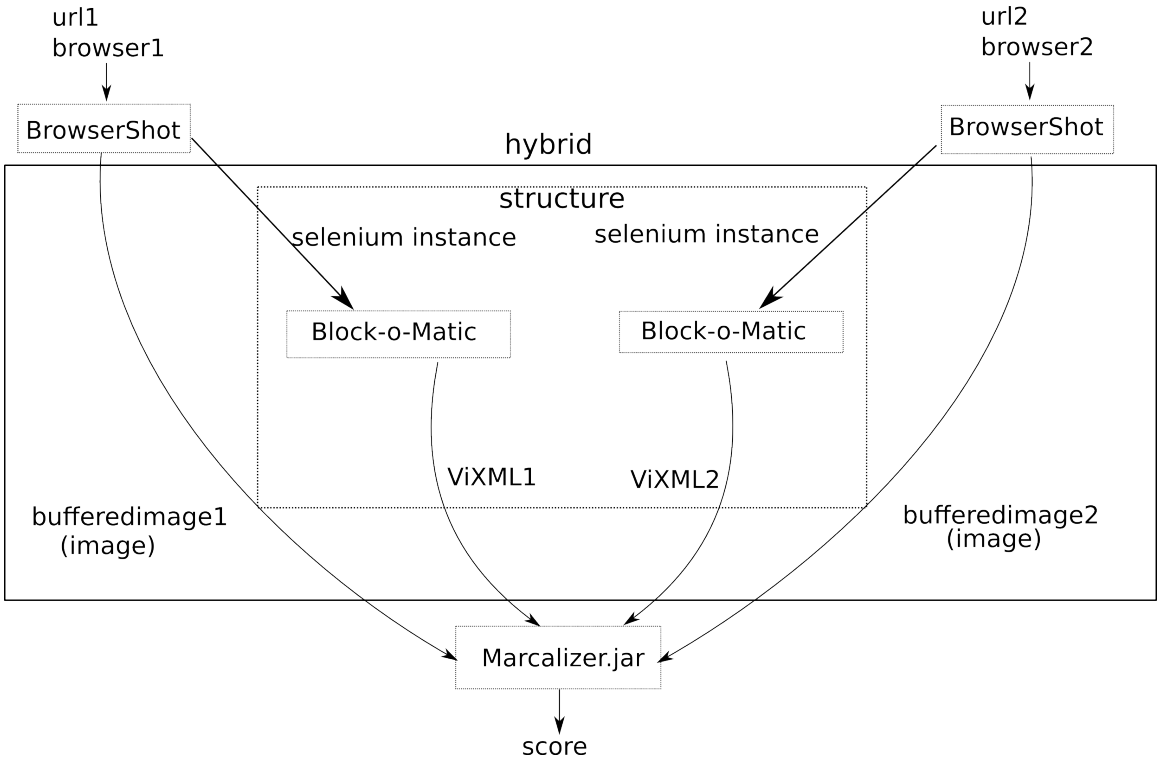


FIGURE 5.3: Change detection flow for image, structure and hybrid comparison types in the final release

5.1.3 Practical application

The final release was used to perform the correctness based benchmarks and validation test for the SCAPE project Web QA workflow. A dataset of 449 pairs of urls was used,

[2]https://github.com/sbarton/browser-shot-tool-mapred

annotated by the IM[3].

Figure 5.4 shows the result of this process according to the different annotations (Similar, Dissimilar) and according to the page type (Blank page, HTML, image, etc). Total values are calculated as a weighted mean by type. For HTML pages, visual and content comparison result that Pagelyzer agreed with the manual annotation by 68% for visual and 76% for content, which are promising results.

However, the hybrid approach, which is still on beta version, stays behind the other two approaches while it should give better results.

bechmarks.png bechmarks.png

Type	Similar			Dissimilar			Overall		
	Visual	Content	Hybrid	Visual	Content	Hybrid	Visual	Content	Hybrid
Blank page	0,98	1,00	1,00	-	-	-	0,98	1,00	1,00
Doc file	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
Error message	0,27	0,13	0,40	-	-	-	0,27	0,13	0,40
Error page	0,42	0,08	0,17	-	-	-	0,17	0,08	0,17
HTML	0,75	0,83	0,60	0,78	0,89	0,76	0,75	0,84	0,63
Image	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
Image png	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
Index page	0,70	0,00	0,00	0,03	1,00	1,00	0,26	0,65	0,65
Javascript file	0,83	0,41	0,24	-	-	-	0,83	0,41	0,24
PDF file	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
XML file	0,00	0,00	0,00	-	-	-	0,00	0,00	0,00
Zip file	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
TOTAL	**0,75**	**0,71**	**0,57**	**0,45**	**0,94**	**0,87**	**0,68**	**0,76**	**0,62**

FIGURE 5.4: Benchmarking results for Web QA

5.1.4 Perspectives and outlook

The development of Pagelyzer is a successful story in the context of the Quality Assurance Workpackage of SCAPE project. It can be used in other domains or context of applications. Several members of the project contact us to help them to use this tool (or part of it) to their own projects. A interesting application could be the project of a national Web archive in Venezuela [San12]. This is still a draft project, but all the

[3]http://www.scape-project.eu/deliverable/d11-2-quality-assurance-workflow-release-2-release-report

insights and experience gained along this last three years can be very helpful to develop a Web archive system adapted to this country.

5.2 Block-based migration of HTML4 standard to HTML5 standard

In this section we describe how to use the segmentation to migrate pages content from one format to another. In 5.2.1, we reference other experiences where the HTML5 standard has been chosen as a final format and we introduce the main issue of migrating HTML4 pages to HTML5 format. Finally in 5.2.2 we propose a solution to this issue (in the context of Web archives). Some experiments are reported.

5.2.1 Introduction

Obsolescence, adjustment, and renewal are necessary parts of the development cycle. Improvements usually require changes. That includes technologies, products, processes, and people, as well. In July 2012, the WWW Consortium introduced a recommendation for HTML5[4]. It represents an important change regarding the preceding version of HTML and the XHTML specification. For instance it introduces the semantic tags allowing browsers to easily access contents, audio and video among others. The first question raised by HTML5 is: why to use it? Laws [Law13] discusses this from the competition point of view and he concludes that organizations and publishers need to be ready for this technological change if they want to outperform their competitors and stay in the technological race. This raises another question: once publishers switch to HTML5, what happens with the current HTML4 content?

In the context of Web archiving we are interested in what is going to happen to archived content (HTML 4 and XHTML formats).

In general, Web archives store pages along with all their dependencies. We agree with Rosenthal [RLRM05] that eventually, modern browsers will no longer be able to render document in HTML4 or XHTML formats in a proper way. Thus, a strategy for their preservation must be taken. Archivists must decide to perform either a emulation or migration.

[4]The proposed recommendation is out September 2014

In the context of digital preservation the emulation is "the replicating of functionality of an obsolete system, but on the hard- and software environment in which the object is rendered" [VdH07]. In other words emulation consists in recreating the environment in which a Web page was originally created. This implies keeping old versions of tools or old tools. Migration refers to transferring data to newer system environments [Gar96]. This includes converting a Web page file from one file format to that another so the resource including its functionalities remains fully accessible.

Rosenthal also describes the difficulties of using only emulation. Its cost is very high in terms of storage and operation. Conversely, migration of Web content from an obsolete format to a current one seems to be a good strategy to minimize emulation, but it increases data duplication and there is the risk of loosing document information in the process. The obsolescence of Web content is usually associated with its presentation, that is, its rendering and visual aesthetic. However, the document semantic should be also taken into account also. The main goal of HTML5 is to improve the language, keeping it readable by humans and by computers and useful, and able to enrich the semantic content of documents. As an example, Park [PLR+10], present their experience in the migration of ETD (Electronic Theses and Dissertations) from the PDF format to HTML5 format. Most of ETD have linked multimedia documents and connected by hyperlinks (in PDF format). Storing them in this format, requires to have the corresponding multimedia readers, libraries and plug-in, as well. HTML5 is a convenient migration format because in this way it is possible to have one single file that has all of the content linked together, including all of the multimedia information in the ETD and metadata available for Web search indexing and other general tasks.

In the remainder of this section, we present why and how we use Web page segmentation to perform the migration of HTML4 pages to HTML5 format.

Several efforts have taken place in order to make uniform the migration from one format to another [Pfe10]. Existing methods usually perform a tag-by-tag migration, in other words they translate tags. However, it is difficult to define an appropriated translation of HTML5 semantic tags (which defines the layout of the Web page) from HTML4 pages where such tags do not exist. Semantic tags have no impact in the rendering of the page, but they help to organize the content into coherent regions. Thus, using segmentation seems relevant for the migration, which can be performed by segmenting HTML4 pages and incorporating semantic tags to the result.

5.2.2 Proposed solution

We propose to segment an HTML4 Web page, with the appropriate predefined stop condition parameter so that the resulting blocks will correspond to the semantic tags in the HTML5 format.

Then we compare the labels found by the segmentation with a manually labeled segmentation as ground truth. If both versions are similar the migration is achieved. If they are different we measure how discrepant they are in order to determine the causes and the possible actions to improve the migration method.

Finally, migration is evaluated in order to measure whether it has affected the rendering of the Web page. We use for this an adaptation of the framework of Chapter 4.

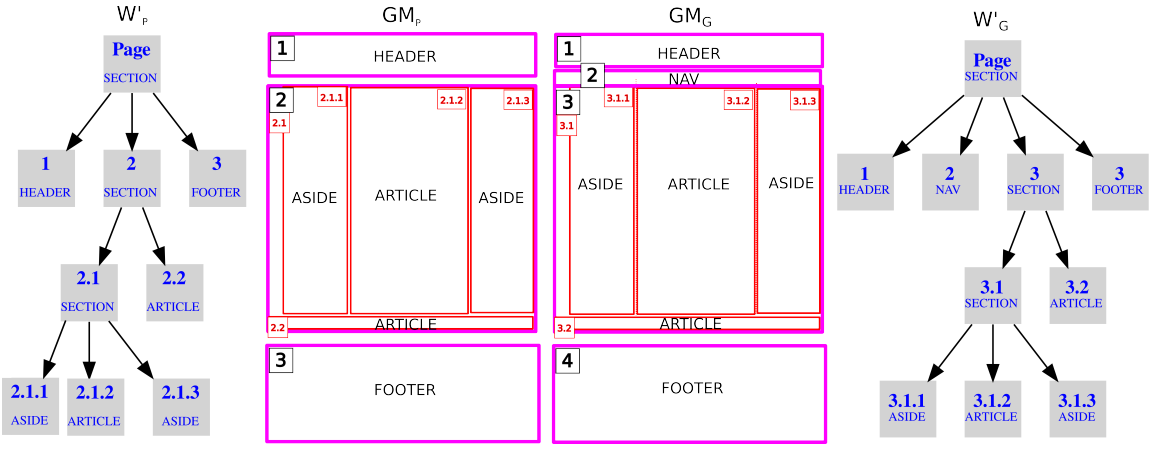


FIGURE 5.5: Labels for the manual and computed segmentation

In the following section we describe the experiments to evaluate our migration approach.

5.2.3 Experiments

In this section we present the setup of experiments, their design and the measures used.

5.2.3.1 Experimentation design

The MIG5 collection is a subset of the GOSH collection presented in Section 4.4. It only contains Web pages in HTML4 format. We keep the same categories organization (blog, enterprise, forum, picture and wiki) in this collection.

Category	*Pages*
blog	5
enterprise	9
forum	14
picture	7
wiki	5
total	40

TABLE 5.1: MIG5 pages by categories

The first experiment is devoted to measure to what extent the labels found with the Block-o-Matic segmentation algorithm match to those in a ground truth of manually labeled blocks.

The second experiment aims of measuring if including the semantic elements affects the rendering of the page. The block correspondence method, as presented in Section 3.4.3, is used for evaluating the correctness of the migration. The segmentation of the original Web page is used as a ground truth, while the segmentation of the migrated Web page is the evaluated segmentation.

5.2.3.2 Ground truth building

Table 5.1 shows the organization of the MIG5 collection. It is composed of 40 pages organized by category.

The MoB tool (*cf.* Section 4.4.2) is used to annotate the blocks. Besides specifying the blocks, assessors assign a label to each block. Labels corresponds to a subset of the semantic elements defined in the HTML5 specification (header, footer, section, article, nav, aside). Appendix B shows the complete list of semantic elements. The stop condition for all the experiments is set to $pA = 6$. Indeed, through experiments, we noticed that this value generates blocks likely to correspond to template elements (*cf.* Section 1.2.2). The separation is set to $pD = 30$ because usually these regions can be very close one to each other.

5.2.3.3 Assigning labels

The BoM labeling method is modified to support the semantic elements as labels. Heuristics rules are defined in order to determine the label of each block. These rules assign

labels depending on the position of a block and its relationship to the others blocks. A block is treated differently if it resides in the visible part of the page (*i.e.* the part of the page visible without using scrolling). For instance, a block is labeled as *header* if it is the first block found vertically (on top of the page), it resides in the visible part of the page, it is a simple block and it has siblings. A block with the same characteristics but outside of the visible area and at the bottom of the page is labeled as *footer*.

For the labels *section* and *nav*, two additional conditions are considered. If the proportion of elements a block covers is greater than a constant, it can be considered a *section*. If the proportion of hyperlinks (*i.e.* <A> elements) a block covers is greater than a constant, it can be considered a *nav*. Algorithm 5 describe the label assignment method for all possible cases.

5.2.3.4 Measuring labels

The manual segmentation Φ_G and the computed segmentation Φ_P are formal defined in Section 1.3.2. The manual segmentation, produced by assessors, takes the rendered DOM of a Web page (W) in HTML4 file format and produces the W'_G block graph. The computed segmentation takes the same rendered DOM (W) and produces the W'_P block graph.

We present the labels of a segmentation as a list of labels ($labels(W'_A)$). The order of the list follows the reading order (*cf.* Section 2.4), considering only the leaves nodes.

Using the intersection of both list we get the amount of correct labels found by the segmentation with respect to the ground truth. The $correct\_labels$ measure is defined as:

$$correct\_labels(W'_G, W'_P) = labels(W'_G) \cap labels(W'_P)$$

Figure 5.5, shows the labels for the manual and computed segmentation. The list of labels from the manual segmentation is: { header, nav, aside, article, aside, article, footer}. The list of labels for the computed segmentation is: { header, aside, article, aside, article, footer}. For simplicity, we denote the labels with one letter. Thus, the list of labels for both example segmentations are:

- $labels(W'_G) = \{H, N, D, A, D, A, F\}$
- $labels(W'_P) = \{H, D, A, D, A, F\}$

```
   Data: Block: b
   Result: B.label
 1 if b.weight > pA then
 2 |   if b in the visible part of page then
 3 |   |   if b is the first block on top then
 4 |   |   |   if proportion of elements covered by b is greater than a constant then
 5 |   |   |   |   if b is composite then
 6 |   |   |   |   |   B.label=SECTION;
 7 |   |   |   |   else if b has no siblings then
 8 |   |   |   |   |   B.label=SECTION;
 9 |   |   |   |   else
10 |   |   |   |   |   B.label=HEADER;
11 |   |   |   |   end
12 |   |   |   else
13 |   |   |   |   B.label=HEADER;
14 |   |   |   end
15 |   |   else if proportion of elements covered by b is greater than a constant then
16 |   |   |   if b is composite then
17 |   |   |   |   B.label=SECTION;
18 |   |   |   else
19 |   |   |   |   B.label=ARTICLE;
20 |   |   |   end
21 |   |   else if proportion of hyperlinks covered by b is greater than a constant then
22 |   |   |   B.label=NAV;
23 |   |   else if b is in the middle/center of the page then
24 |   |   |   B.label=ARTICLE;
25 |   |   else if b is the last block at bottom then
26 |   |   |   B.label=FOOTER;
27 |   |   else if b is at left/right of the page then
28 |   |   |   B.label=ASIDE;
29 |   |   else
30 |   |   |   B.label=ARTICLE;
31 |   |   end
32 |   else if b is the last block at bottom then
33 |   |   B.label=FOOTER;
34 |   else
35 |   |   B.label=ARTICLE;
36 |   end
37 end
```

Algorithm 5: Label assignment algorithm

The migration of Figure 5.5 is not perfect since the segmentation did not find the block labeled as *nav*. Instead, it found the block labeled as *header* covering the corresponding region of the page. We measure this error with the Levenshtein distance [Nav01].

$$error(W'_G, W'_P) = levenshtein\_distance(labels(W'_G), labels(W'_P))$$

For the example the error is 1: it is sufficient to insert 1 label (N) in the computed segmentation label list to produce the list of the ground truth.

We represent also the results in terms of precision and recall:

$$precision = \frac{correct\_labels(W'_G, W'_P) + |labels(W'_G)|}{|labels(W'_G)|}$$

$$recall = \frac{correct\_labels(W'_G, W'_P) + |labels(W'_G)|}{correct\_labels(W'_G, W'_P)}$$

5.2.3.5 Measuring rendering errors

In order to measure to what extent the migration affects the rendering of the migrated Web page, we use the correspondence measures defined in Section 3.4.3. We do not consider the metric version with importance.

We have two rendered DOM, W and $W5$, where W is the rendered DOM of a Web page in HTML4 format and $W5$ is the rendered DOM of the migrated Web page. They respectively produce the blocks graphs W'_P and $W5'_P$. Setting the parameters $t_r = 0$, $t_i = 0$, $ND = 100$ and $NP = 10$ we get the correspondence measures. We choose these parameters because we want to evaluate all blocks, so we consider all as significant and all are equally important.

If we find only correct blocks then the migration may be perfect, if both segmentations produce the same segmentation there is a high probability that their rendering is the same. If an oversegmentation or an undersegmentation occurs that means that the inclusion of semantic elements in $W5$ modified the size and position of the blocks, therefore segmentations are different. Blocks missed and false alarms are possible when the rendering changes, slightly displacing content in the migrated version.

category	$correct\_labels(W'_G, W'_P)$	$\|labels(W'_G)\|$	$error(W'_G, W'_P)$	*precision*	*recall*
blog	2.50	3.50	2.00	0.28	0.4
enterprise	2.22	3.55	2.38	0.37	0.60
forum	3.00	3.53	1.44	0.14	0.17
picture	2.55	3.00	1.55	0.14	0.17
wiki	2.20	3.00	1.90	0.26	0.36

TABLE 5.2: Average values for correct, expected labels and error for the MIG5 collection

5.2.4 Results

In this section we present the results of applying our approach to migrated Web pages from HTML4 format to HTML5 format. We present how we measure the labels found by the algorithm compared to the ground truth and the rendering errors using the evaluation model presented in Chapter 3

5.2.4.1 Measuring labels

Table 5.2 shows the average values of the metrics defined in Section 5.2.3.4 for the MIG5 collection separated by categories. In general BoM produces a list of labels similar to the ground truth. In average it adds 1.85 unexpected labels. This is probably due to the introduction of semantic elements that affect the segmentation and the stop condition, producing smaller blocks than expected. For instance, for a blog post with two paragraphs, labeled as a whole in the ground truth, each paragraph become a block in the migrated page generating one additional unexpected label. It is interesting that both rendering looks equal but the segmentations differs.

Forum category presents the lowest error rate, because in general the question/response region of the page is detected in both segmentation, as one block labeled as *article*. The worst performance is for the enterprise category, because this type of pages are structured with complex navigation and main content, and the probability of mislabeling is high.

Table 5.2 shows the precision and recall metrics. Figure 5.6 shows these metrics graphically. The BoM algorithm has a high precision for the forum and picture categories. As we mention earlier both type of pages produces small and simple list of labels, while pages in the other categories their labeling is more complex, therefore less precision. However, all results present high recall values indicating that the algorithm find enough good labels but with a considerable error rate.

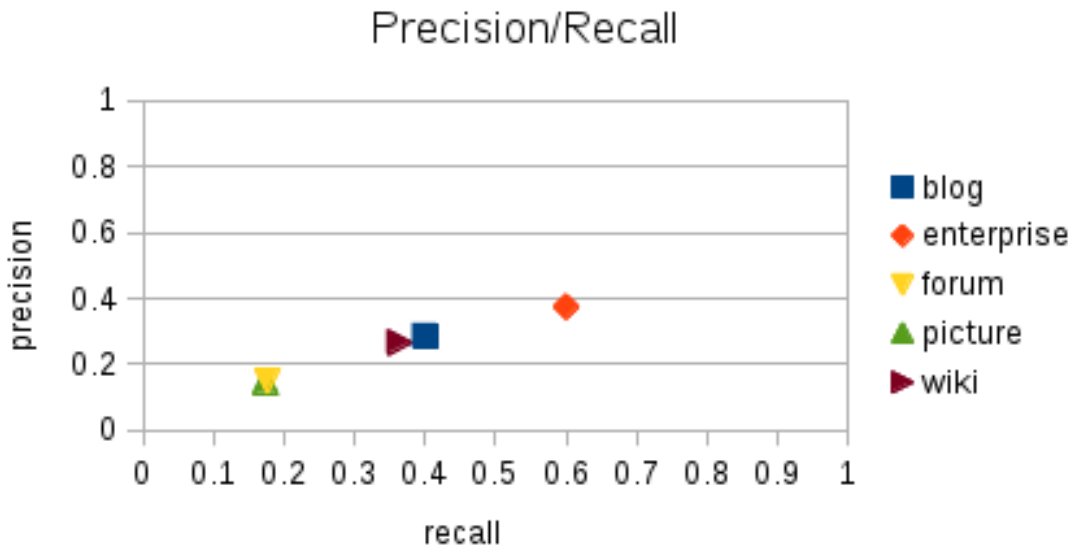


FIGURE 5.6: Precision and recall for the MIG collection

Algorithm	C_c	C_o	C_u	C_m	C_f	C_q	GTB
blog	6.50	0.50	0.00	0.00	0.50	7.00	7
enterprise	4.00	0.33	0.33	1.11	2.77	4.67	6.45
forum	3.41	0.59	0.41	2.11	1.29	4.41	6.59
picture	2.71	1.00	0.29	2.00	0.71	4.00	6.71
wiki	6.00	0.0	0.00	0.60	0.40	6.00	6.6

TABLE 5.3: Correspondence metrics for the MIG5 collection with $t_r = 0.1$ and $t_t = 1$

5.2.4.2 Measuring rendering errors

Table 5.3 shows the average correspondence metrics, by category, for the MIG5 collection. The values of the C_q metric shows that the performance of the algorithm in both versions (original and migrated) is good. However, there are some missed blocks, particularly in the enterprise, forum and picture categories because of shifting of blocks due to rendering changes. But in both cases, the formatted content displayed is equal. Blog and wiki categories present the best performance. The regions in these type of pages are simple and the position and order of blocks are standard. The regions are well separated, making it easy to segmentation algorithms like BoM to detect correct labels. For instance, almost all pages in this categories start by a *header* followed by a *navigation*, then the *aside* at left, the main *article* and the *footer* at the bottom of the page.

5.2.5 Perspectives and outlook

In this section we presented our approach to block-based migration of Web pages from HTML4 format to HTML5 format. Using the segmentation, we produce a migrated version according to the HTML5 specification. We analyzed how the algorithm assigned labels to blocks in comparison to a ground truth of manually labeled segmentation. The rendering errors were measured using the block correspondence metrics defined in Section 3.3.2. The results show that, in the context of digital preservation, migrating Web pages from one format to another is possible using the BoM Web page segmentation algorithm, minimizing the emulation in Web archives. We show that there is no data loss in the process and no important changes in the rendering (few false alarms). However the segmentation is affected by the semantic tags. For instance, some browsers have no default style for these elements, and they are taken by the algorithms as invisible or not valid elements, therefore they are ignored. The evaluation model presented in Chapter 3 is very helpful to measure the performance and detecting the rendering errors. The parameters and the stop conditions of the algorithm can be adjusted by category (using Machine Learning techniques) to have better performance depending on page category. This is left as future work.

There are still challenges to overcome. Our approach gives insights of the upcoming issue raised by the migration of Web content in the context of Web preservation.

Conclusion

This thesis studies some problems raised by the segmentation of Web pages. It focuses on different points: the precision and genericity of the segmenter, as well as the accuracy and fairness of the evaluation of segmenters. More specifically, we address the following challenges and contribute as follows:

- We propose Block-o-matic (BoM) a new Web page segmentation algorithm. This work is the first to take into account in the design of the segmenter, precision and genericity as quality criteria of the segmentation. As our results show, it allows segmenting different type of Web pages without previous knowledge of the content, with a better accuracy than the other tested segmenters. Thanks to a bottom-up strategy and heuristic rules defined in the W3C standards, we achieve genericity and precision.

- We propose a framework that allows us evaluating the correctness of segmentations algorithms whatever of the approach they follow. To the best of our knowledge, this is the first evaluation work that focuses on segmentation intrinsic properties, which are content and blocks rectangles. Existing approaches do have evaluation, but they are driven by specific applications and thus are not generic enough to compare all the segmentations algorithms. Our approach is based on a ground truth, built thanks to a tool (MoB) we developed and that substantially eases the manual design of segmentation. Our dataset contains 125 pages, covering five categories (25 pages per category).

- We present an evaluation model that defines several useful metrics for the evaluation. One metric is devoted to the text extraction task, the other ones compute how well the blocks detected by a given algorithm match the ones of the ground truth. We use this model for evaluating and comparing four segmentation algorithms, adapted in order to fit into our framework. The results show that the algorithms perform reasonably for extracting text from pages. With respect to block

correspondence, results slightly depend on the category of the pages considered. We include the importance in the evaluation model, so that algorithms that detect important blocks get a better score. Our experiments show that BoM is the most precise and generic algorithm in the evaluation. That is because it presents the lowest occurrence of oversegmentations and undersegmentations (granularity issues) and it works reasonably well in all type of pages considered in the evaluation.

- We present two applications using our segmentation algorithm and the evaluation framework. Pagelyzer is a tool developed in the context of the European project SCAPE which use BoM for comparing two Web pages versions to decide if they are similar or not. In order to minimize the emulation in Web archives, we develop a tool for migrating Web pages from HTML4 format to HTM5 format, using BoM.

In the rest of this chapter, we outline our plans for future work, and discuss possible research topics beyond what has been addressed in the thesis.

Web page segmentation There are many directions for future work related to the Web page segmentation. The labeling of blocks is a crucial task in Web page segmentation. It defines the role a block plays, therefore how it should be treated. In this thesis we show that using the labels from HTML content categories allow us being generic. However, it would be interesting to give a more precise description of the role a block plays, in order to give more interesting information to applications that relies on the segmentation. For instance, we may indicate if a block labeled as *footer* is an appendix, an index or a verbose license agreement or if a block labeled as *nav* is a navigation of first, second of third level.

The reading order is useful information given from the segmentation. As shown in Chapter 5, it is a useful information to processes segmentation output. For instance, a mobile application that relies on segmentation for visualizing a Web page, can give the option to the viewer to read the page content, block by block, following the reading order. Following intuition, in this thesis we studied the document processing domain looking for insights for implementing the reading order in the Web page segmentation. We found that usually the reading order follows the order that blocks are found. However, we think that there is still space for more improvements, perhaps related to the type of pages and include reading order measures in the evaluation.

The rectangle is the intrinsic shape of blocks. Block rectangles depend on page elements boxes which are also rectangular. However, rectangles may overlap, causing ambiguities

in the visual presentation. This is an issue since the visual presentation is more and more important for client applications. For instance in the example of the mobile application mentioned above, while showing a block, parts of other blocks (overlaping) may be included in the visualization. Thus, we plan to add support for non-manhattan layouts in the segmentation. Instead of rectangles, polygons will be used.

We will study the new visual characteristics on Web pages (CSS3): background images, animations, Ajax, among others. Intuitively, a solution to this issue is to consider the visual cues as part of the content of the page. This may not be so simple. Indeed, this is mainly the reason why the CSS exists, *i.e.* to separate the content of its presentation.

Evaluation There are many directions for future work around the evaluation. First, we plan to use machine learning (ML) techniques for learning the tolerance parameter t_r. We also plan to use ML for discovering new relevant score functions, based on the feedback of users giving a manual score to segmentations from a training set.

Second, we will continue to experiment segmentation algorithms on more pages and more page categories. Our aim is to develop a complete evaluation framework in order to help users in choosing the best segmentation algorithm depending on their application and on the category of pages they manipulate. Of course, as the results show that some algorithms have problems with some categories, they can also be used to help improving the efficiency of segmentation algorithms for those categories.

Third, we plan to evaluate the segmentation algorithms with respect to the type of task that uses the segmentation. Task types include Web entity extraction, layout detection, boilerpipe detection, visualization in small screen devices, and, in the context of digital libraries, optimization of Web archives crawling, change detection between Web page versions, among others. This implies defining scripts that perform the task (including calls to segmentation) and defining new *ad hoc* metrics for each task. Also, we would like to define a generic model for Web page segmentation that can express all the existing approaches. This would allow for an analytic evaluation of segmentation algorithms.

Fourth, we plan to include in the evaluation the reading order of the segmentation. A good candidate is the metric introduced by Liang [LPH01]. Evaluating the reading order is reduced to computing the number of moves required to obtain the reading order of the ground truth from the one of the computed segmentation.

Appendix A

HTML5 Content Categories

In this section we list the HTML5 content categories and their exceptions. Table source : http://www.w3.org/TR/html5/index.html

Table A.1: HTML5 content categories.

Category	Elements	Exceptions
Metadata content	base; link; meta; noscript; script; style; template; title	-
Flow content	a; abbr; address; article; aside; audio; b; bdi; bdo; blockquote; br; button; canvas; cite; code; data; datalist; del; dfn; div; dl; em; embed; fieldset; figure; footer; form; h1; h2; h3; h4; h5; h6; header; hr; i; iframe; img; input; ins; kbd; keygen; label; main; map; mark; math; meter; nav; noscript; object; ol; output; p; pre; progress; q; ruby; s; samp; script; section; select; small; span; strong; sub; sup; svg; table; template; textarea; time; u; ul; var; video; wbr; Text	area (if it is a descendant of a map element)
Sectioning content	article; aside; nav; section	-
Heading content	h1; h2; h3; h4; h5; h6;	-

Continued on next page

Table A.1 – continued from previous page

Category	Elements	Exceptions
Phrasing content	a; abbr; audio; b; bdi; bdo; br; button; canvas; cite; code; data; datalist; del; dfn; em; embed; i; iframe; img; input; ins; kbd; keygen; label; map; mark; math; meter; noscript; object; output; progress; q; ruby; s; samp; script; select; small; span; strong; sub; sup; svg; template; textarea; time; u; var; video; wbr; Text	area (if it is a descendant of a map element)
Embedded content	audio canvas embed iframe img math object svg video	-
Interactive content	a; button; embed; iframe; keygen; label; select; textarea;	audio (if the controls attribute is present); img (if the usemap attribute is present); input (if the type attribute is not in the Hidden state); object (if the usemap attribute is present); video (if the controls attribute is present)
Sectioning roots	blockquote; body; fieldset; figure; td	-
Form-associated elements	button; fieldset; input; keygen; label; object; output; select; textarea; img	-
Listed elements	button; fieldset; input; keygen; object; output; select; textarea	-
Submittable elements	button; input; keygen; object; select; textarea	-
Resettable elements	input; keygen; output; select; textarea	-
Labelable elements	button; input; keygen; meter; output; progress; select; textarea	-

Continued on next page

Table A.1 – continued from previous page

Category	Elements	Exceptions
Reassociateable elements	button; fieldset; input; keygen; label; object; output; select; textarea	-
Palpable content	a; abbr; address; article; aside; b; bdi; bdo; blockquote; button; canvas; cite; code; data; dfn; div; em; embed; fieldset; figure; footer; form; h1; h2; h3; h4; h5; h6; header; i; iframe; img; ins; kbd; keygen; label; main; map; mark; math; meter; nav; object; output; p; pre; progress; q; ruby; s; samp; section; select; small; span; strong; sub; sup; svg; table; textarea; time; u; var; video	audio (if the controls attribute is present); dl (if the element's children include at least one name-value group); input (if the type attribute is not in the Hidden state); ol (if the element's children include at least one li element); ul (if the element's children include at least one li element); Text that is not inter-element whitespace
Script-supporting elements	script; template	-

Appendix B

Semantic HTML5 elements

In this section we list the HTML5 semantic elements. Table source : http://diveintohtml5.info

Table B.1: HTML5 semantic elements

Element	Description
<section>	The section element represents a generic document or application section. A section, in this context, is a thematic grouping of content, typically with a heading. Examples of sections would be chapters, the tabbed pages in a tabbed dialog box, or the numbered sections of a thesis. A Web site's home page could be split into sections for an introduction, news items, contact information.
<nav>	The nav element represents a section of a page that links to other pages or to parts within the page: a section with navigation links. Not all groups of links on a page need to be in a nav element — only sections that consist of major navigation blocks are appropriate for the nav element. In particular, it is common for footers to have a short list of links to common pages of a site, such as the terms of service, the home page, and a copyright page. The footer element alone is sufficient for such cases, without a nav element.

Continued on next page

Table B.1 – continued from previous page

Element	Description
<article>	The article element represents a component of a page that consists of a self-contained composition in a document, page, application, or site and that is intended to be independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a Web log entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content.
<aside>	The aside element represents a section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content. Such sections are often represented as sidebars in printed typography. The element can be used for typographical effects like pull quotes or sidebars, for advertising, for groups of nav elements, and for other content that is considered separate from the main content of the page.
<hgroup>	The hgroup element represents the heading of a section. The element is used to group a set of h1–h6 elements when the heading has multiple levels, such as subheadings, alternative titles, or taglines.
<header>	The header element represents a group of introductory or navigational aids. A header element is intended to usually contain the section's heading (an h1–h6 element or an hgroup element), but this is not required. The header element can also be used to wrap a section's table of contents, a search form, or any relevant logos.
<footer>	The footer element represents a footer for its nearest ancestor sectioning content or sectioning root element. A footer typically contains information about its section such as who wrote it, links to related documents, copyright data, and the like. Footers don't necessarily have to appear at the end of a section, though they usually do. When the footer element contains entire sections, they represent appendices, indexes, long colophons, verbose license agreements, and other such content.
<time>	The time element represents either a time on a 24 hour clock, or a precise date in the proleptic Gregorian calendar, optionally with a time and a time-zone offset.
<mark>	The mark element represents a run of text in one document marked or highlighted for reference purposes.

Appendix C

Web page segmentation evaluation metrics

In this section we describe the metrics used for the evaluation of segmentation by algorithms in the state of the art (*cf.* Section 1.4.4)

C.1 Adjusted Rand Index

The Rand index [Ran71] is a measure of the similarity between two data clusterings.

Given a set of n elements $S = \{o_1, \ldots, o_n\}$ and two partitions of S to compare, $X = \{X_1, \ldots, X_r\}$, a partition of S into r subsets, and $Y = \{Y_1, \ldots, Y_s\}$, a partition of S into s subsets, define the following:

- a, the number of pairs of elements in S that are in the same set in X and in the same set in Y
- b, the number of pairs of elements in S that are in different sets in X and in different sets in Y
- c, the number of pairs of elements in S that are in the same set in X and in different sets in Y
- d, the number of pairs of elements in S that are in different sets in X and in the same set in Y

The Rand index, R, is:

$$R = \frac{a+b}{a+b+c+d} \tag{C.1}$$

Intuitively, $a+b$ can be considered as the number of agreements between X and Y and $c+d$ as the number of disagreements between X and Y.

The Adjusted Rand index (AdjRand) [VEB09] is the corrected-for-chance of the Rand index.

$$AdjRand = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex} \tag{C.2}$$

Higher values indicate higher quality, with a maximum value of 1.

C.2 Normalized Mutual Information

This metric was introduced by Strehl and Ghosh [SG03]. It is the mutual information between two partitioning normalized by the geometric mean of their entropies (H):

$$NMI(X,Y) = \frac{I(X,Y)}{\sqrt{H(X)\,H(Y)}} \tag{C.3}$$

where, X and Y are the two partitions and H(X) and H(Y) their entropies.

This measure has been commonly used recently for computing the accuracy of clustering algorithms.

As with AdjRAND, higher values indicate higher quality, with a maximum value of 1.

C.3 Dunn Index

The Dunn index aims to identify dense and well-separated clusters. It is defined as the ratio between the minimal inter-cluster distance to maximal intra-cluster distance. For each cluster partition, the Dunn index [Dun74] can be calculated by the following formula:

$$D = \min_{1 \leq i \leq n} \left\{ \min_{1 \leq j \leq n, i \neq j} \left\{ \frac{d(i,j)}{\max_{1 \leq k \leq n} d^{'}(k)} \right\} \right\} \tag{C.4}$$

where $d(i,j)$ represents the distance between clusters i and j, and $d^{'}(k)$ measures the intra-cluster distance of cluster k. The inter-cluster distance $d(i,j)$ between two clusters may be any number of distance measures, such as the distance between the centroids of the clusters.

Alcic et al. [AC11] use this metric over a cluster of DOM elements, that is i,j and k are blocks.

C.4 Nested Earth Mover's Distance

Given two set of features of an image, the Nested Earth Mover's Distance (Nested-EMD) [CML10, RTG00] is a distance based on bipartite graph matching, defined as the minimum cost of matching the bins (discretes intervals) of two histograms. It is accepted as a general metric between signatures for image retrieval. Intuitively, given two distributions, one can be seen as a mass of earth properly spread in space, the other as a collection of holes in that same space. Then, the EMD measures the least amount of work needed to the holes with earth. Here, a unit of work corresponds to transporting a unit of earth by a unit of ground distance.

C.5 Precision, Recall and F1 score

These are common measures in information retrieval. Here, the segmentation is seen as a task of retrieval of blocks. The precision is the fraction of retrieved instances that are relevant, while the recall is the fraction of relevant instances that are retrieved. Both precision and recall are therefore based on an understanding and measure of relevance.

In web page segmentation [SSL11, LLT11, ZJKZ10], precision is measured using block instances as follows:

$$precision = \frac{|\{correctblocks\} \cap \{blocksfound\}|}{|blocksfound|} \tag{C.5}$$

The recall is measured as:

$$recall = \frac{|\{correctblocks\} \cap \{blocksfound\}|}{|correctblocks|} \tag{C.6}$$

The F1 score is a measure that combines of the precision and recall measures.

It is the harmonic mean of precision and recall.

$$F_1 = \frac{precision \cdot recall}{precision + recall} \tag{C.7}$$

Appendix D

Web Segmentation approaches details

In this section we show details of the web page segmentation approaches: text-based and vision-based.

D.1 Text-based

In the field of text analysis, words, syllables and sentences have been widely used as statistical measures to identify structural patterns in textual parts of web documents. It can be seen as a special form of segmentation [CM00]. Regular expressions have been used to parse a web page looking for blocks. Even if it is possible to use regular expressions for this task, it is commonly accepted that they are not the best choice, because HTML is not a regular language.

In text based approaches, the text can be extracted either from the HTML source code of a web page or the rendered DOM. For instance Kohlschuetter [KN08] takes only an input web page file but Sun [SSL11] obtains the text from DOM nodes, (rendered DOM).

In contrast to regular expressions, the concept of text density and link text density [SSL11, KN08] have been more accepted in the web page segmentation community. Sun et el. [SSL11] work with the DOM nodes in order to figure out the number of characters and tags that each node contains. Then, statistical information can be added to the node :

- CharNumber: number of all characters in its subtree
- TagNumber: number of all tags in its subtree.

Then, they define the *Text Density* of a node TD_i as the ratio of its CharNumber (C_i) to its TagNumber (T_i) :

$$TD_i = \frac{C_i}{T_i}$$

where C_i is the number of all characters under i, T_i is the number of all tags under i.

Furthermore, they define the *Composite Text Density* (CTD_i), which adds additional statistical information to each node as below:

- LinkCharNumber (LC) : number of all hyperlinks characters in its subtree
- LinkTagNumber (LT): number of all hyperlink tags in its subtree.

The authors argue that a node with too many hyperlinks and less text is less important, thus getting a low density value. A node that contains much non-hyperlink text and few hyperlinks is more important, and receives a high density value.

On the other hand, Kohlschuetter [KN08] defines the text density, word-wrapping the page text at a constant line width w_{max} (in characters). The density $\rho(b_x)$ of the block b_x can be then formulated as follows:

$$\rho(b_x) = \frac{\text{Number of tokens in } b_x}{\text{Number of lines in } b_x} \tag{D.1}$$

The wrapping width is intended to serve as a discriminator between sentential text (high density) and template text (low density). The author proposes a value of $w_{max} = 80$ as optimal.

The task of detecting block-separating gaps on a web page can be seen as finding neighboured text portions with a significant changes in the block-by-block text density.

The decision of when to merge two adjacent blocks is made by comparing them with respect to their text densities. Kohlschuetter defines the slope delta between two adjacent blocks x and y as:

$$\Delta\rho(x,y) = \frac{|\rho(x) - \rho(y)|}{max(\rho(x), \rho(y))} \tag{D.2}$$

If the slope delta is below a certain threshold Θ_{max}, it means that the blocks belong to one single segment and should therefore be merged. The author reports that the optimal value is $\Theta_{max} = 0.38$.

Figure D.1 shows an example of the blockfusion algorithm.

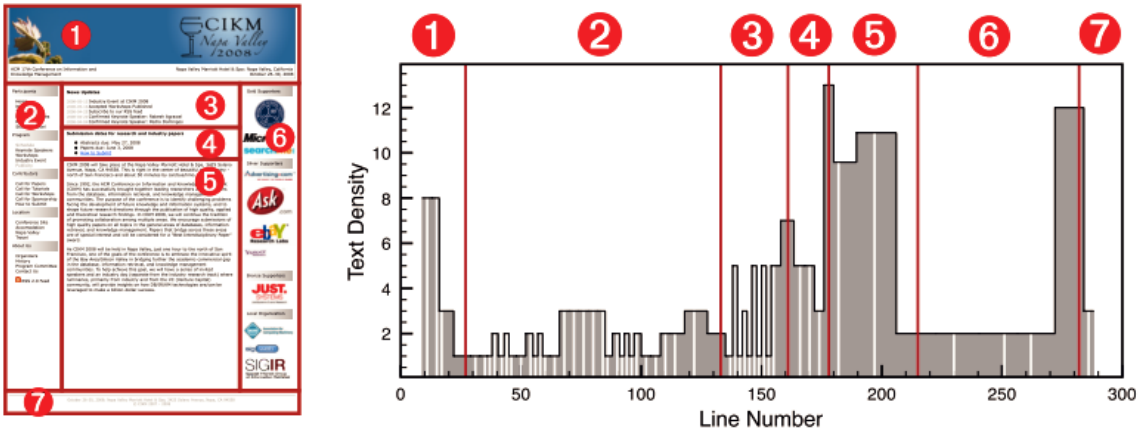


FIGURE D.1: Text-based approach example on a web page. Source: [KN08]

Text based approaches do not fully use all web page characteristics. Therefore there are some types of pages where those approaches fail. Using only text for segmenting a web page is still incomplete because there are other important elements to take into account, such as images, formatted content, among others.

D.2 Vision-based

According to human perception, people view a web page as a set of different semantic objects rather than a single object. Some research efforts show that users always expect that certain functionalities of a web page (e.g. navigational links, advertisement bar) appears at certain position of that page [Ber03].

Actually, when a web page is displayed, the spatial and visual cues can help the user to (unconsciously) divide the web page into several semantic parts. Therefore, it might be possible to automatically segment the web pages by using the spatial and visual cues

The vision-based content structure of a page is obtained by combining the DOM structure and the visual cues. The most known algorithm that follows this approach is VIPS, described by Cai et al. in [CYWM03].

They define a web page as a triple $\Omega = (O, \Phi, \delta)$. $O = \Omega^1, \Omega^2, \ldots, \Omega^N$ is a finite set of non overlapping blocks. Each block can be recursively viewed as a sub-web-page associated with sub-structure induced from the whole page structure. $\Phi = \{\varphi^1, \varphi^2, \ldots, \varphi^T\}$ is a finite set of separators, including horizontal separators and vertical separators. Every separator has a weight indicating its visibility, and all the separators in the same have the same weight. δ is the relationship of every two blocks in O and can be expressed as: $\delta = O \times O \rightarrow \Phi \cup \{NULL\}$. Since each $\Omega_i \in O$ is a sub-webpage of the original page, it has similar content structure as Ω. Recursively, we have

$$\Omega_s^t = \left(O_s^t, \Phi_s^t, \delta_s^t\right) \tag{D.3}$$

$$O_s^t = \left\{\Omega_{st}^1, \Omega_{st}^2, \ldots, \Omega_{st}^{N_{st}}\right\} \tag{D.4}$$

$$\Phi_s^t = \left\{\varphi_{st}^1, \varphi_{st}^2, \ldots, \varphi_{st}^{T_{st}}\right\} \tag{D.5}$$

In the VIPS algorithm, instead of operating solely on the DOM tree, a vision-based content structure of a page is deduced by combining the DOM structure and the visual cues.

The web page is first fragmented into several big blocks and the hierarchical structure of this level is recorded. For each big block, the same segmentation process is carried out recursively until we get sufficiently small blocks, i.e. blocks with a DoC value greater than $PDoC$.

An example of this algorithm can be seen in Figure 1.1 In the first round, the DoC blocks 1 and 3 is greater than the PDoC. An extra round is needed where block 2 is divided. The process stop when blocks from 2.1 to 2.5 meet the criteria.

Another algorithm that follows this approach is presented in Zhang et al.[ZJKZ10]. They focus on finding the set of nodes that are labeled as *Content Row*. A content row is a set of leaf nodes of the DOM tree which all the items arrange horizontally, which are all siblings. Content rows are merged if there is an overlap between them.

As a second step, the *block headers* are detected. A content row is a block header except if:

1. the height of the content row exceeds a threshold or the content row is not a block header.
2. the content row contains a paragraph of text which breaks a line.
3. the words count of the first item of the content row on the left is larger than a threshold.
4. two vertically adjacent content rows share the same CSS style.
5. its next content row does not locate beneath it.

Each detected block header detected is a separator of two semantic blocks. A semantic block is a stack of vertically aligned content rows.

Although Zhang algorithm algorithm is very simple and efficient (for wikis, forums and blogs), it is not suitable for general use. There are several web pages where this algorithm will fail, mainly pages designed in not uniform way. For instance artistic designs where elements break the constraints of the block-level and inline-level content models.

The main issues with this approach are the possibility of ambiguous rules and an incomplete set of visual rules. For instance, this approach suffers of the same problem as DOM-based approaches, special rules are needed in order process all elements in web pages, such is the case of <TABLE>, <UL>, <P>, <LI> elements.

Bibliography

[AC11] 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.

[AT00] Chieko Asakawa and Hironobu Takagi. Annotation-based transcoding for nonvisual web access. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies*, Assets '00, pages 172–179, Arlington, Virginia, USA, 2000. ACM.

[AY13] M.Elgin Akpinar and Yeliz Yesilada. Heuristic role detection of visual elements of web pages. In Florian Daniel, Peter Dolog, and Qing Li, editors, *Web Engineering*, volume 7977 of *Lecture Notes in Computer Science*, pages 123–131. Springer Berlin Heidelberg, 2013.

[Bal06] Shumeet Baluja. Browsing on small screens: Recasting web-page segmentation into an efficient machine learning framework. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 33–42, New York, NY, USA, 2006. ACM.

[Ber03] Michael L Bernard. Criteria for optimal web design (designing for usability). Technical report, University of West Florida, 2003. `http://uwf.edu/ddawson/d3net/documents/web_usability/optimal%20web%20design.pdf`.

[Bre02] Thomas M Breuel. Representations and metrics for off-line handwriting segmentation. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, pages 428–433, Ontario, Canada, 2002. IEEE.

[BSG11] Myriam Ben Saad and Stéphane Gançarski. Archiving the web using page changes patterns: A case study. In *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*, JCDL '11, pages 113–122, Ottawa, Ontario, Canada, 2011. ACM.

[CCMM98] R Cattoni, T Coianiz, S Messelodi, and CM Modena. Geometric layout analysis techniques for document image understanding: a review. *ITC-irst Technical Report*, 9703(09), 1998.

[CCR05] Jaime S Cardoso and Luís Corte-Real. Toward a generic evaluation of image segmentation. *Image Processing, IEEE Transactions on*, 14(11):1773–1782, 2005.

[CKP08] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. A graph-theoretic approach to webpage segmentation. In *Proceedings of the 17th international conference on World Wide Web*, pages 377–386, Beijing, China, 2008. ACM.

[CM00] Franco Crivellari and Massimo Melucci. Web document retrieval using passage retrieval, connectivity information, and automatic link weighting–trec-9 report. In *TREC9*, page 611, Gaithersburg, USA, 2000. TREC.

[CML10] Jiuxin Cao, Bo Mao, and Junzhou Luo. A segmentation method for web page analysis using shrinking and dividing. *International Journal of Parallel, Emergent and Distributed Systems*, 25(2):93–104, 2010.

[CMZ03] Yu Chen, Wei-Ying Ma, and Hong-Jiang Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 225–233, New York, NY, USA, 2003. ACM.

[Con02] Jim Conallen. *Building Web applications with UML*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[CXMZ05] Yu Chen, Xing Xie, Wei-Ying Ma, and Hong-Jiang Zhang. Adapting web pages for small-screen devices. *IEEE Internet Computing*, 9(1):50–56, 2005.

[CYWM03] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In *Proceedings of the 5th Asia-Pacific Web Conference on Web Technologies and Applications*, APWeb'03, pages 406–417, Xian, China, 2003. Springer-Verlag.

[CYWM04] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Block-based web search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 456–463, New York, NY, USA, 2004. ACM.

[Dun74] Joseph C Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104, 1974.

[FdMdS+11] David Fernandes, Edleno Silva de Moura, Altigran Soares da Silva, Berthier Ribeiro-Neto, and Edisson Braga. A site oriented method for segmenting web pages. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 215–224, Beijing, China, 2011. ACM.

[FM81] King-Sun Fu and JK Mui. A survey on image segmentation. *Pattern recognition*, 13(1):3–16, 1981.

[Gar96] Jhon Garret. Preserving digital information. Technical report, Commission on Preservation and Access and the Research Libraries Group, 1996.

[Goo02] Danny Goodman. *Dynamic HTML - the definitive reference: a comprehensive resource for HTML, CSS, DOM and JavaScript (2. ed.).* O'Reilly, 2002.

[HHMS07] Gen Hattori, Keiichiro Hoashi, Kazunori Matsumoto, and Fumiaki Sugaya. Robust web page segmentation for mobile terminal using content-distances and page layout information. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 361–370, Banff, Alberta, Canada, 2007. ACM.

[HKW99] Jianying Hu, Ramanujan Kashi, and Gordon Wilfong. Document image layout comparison and classification. In *1999. ICDAR '99. Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pages 285–288, Bangalore, India, Sep 1999.

[HL14] Jianying Hu and Ying Liu. Analysis of documents born digital. In David Doermann and Karl Tombre, editors, *Handbook of Document Image Processing and Recognition*, pages 775–804. Springer London, 2014.

[KN08] Christian Kohlschütter and Wolfgang Nejdl. A densitometric approach to web page segmentation. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1173–1182, New York, NY, USA, 2008. ACM.

[Kre13] R. Kreuzer. A quantitative comparison of semantic web page segmentation algorithms. Master's thesis, Universiteit Utrecht, 2013.

[Law13] Byron Laws. Seriously, another format? you must be kidding. *CSE NEWS*, 36(2):41, 2013.

[LH02] Shian-Hua Lin and Jan-Ming Ho. Discovering informative content blocks from web documents. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 588–593, Edmonton, Alberta, Canada, 2002. ACM.

[LLT11] Xinyue Liu, Hongfei Lin, and Ye Tian. Segmenting webpage with gomory-hu tree based clustering. *Journal of Software*, 6(12):2421–2425, Dec 2011.

[LPH01] Jisheng Liang, Ihsin T. Phillips, and Robert M. Haralick. Performance evaluation of document structure extraction algorithms. *Computer Vision Image Understanding*, 84(1):144–159, October 2001.

[LPHL02] Xiaoli Li, Tong-Heng Phang, Minqing Hu, and Bing Liu. Using micro information units for internet search. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 566–573, New York, NY, USA, 2002. ACM.

[LTGC12] Marc Teva Law, Nicolas Thome, Stéphane Gançarski, and Matthieu Cord. Structural and visual comparisons for web page archiving. In *Proceedings of the 2012 ACM Symposium on Document Engineering*, DocEng '12, pages 117–120, New York, NY, USA, 2012. ACM.

[MBR07] Jalal U. Mahmud, Yevgen Borodin, and I. V. Ramakrishnan. Csurf: A context-driven non-visual web-browser. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 31–40, New York, NY, USA, 2007. ACM.

[Meu05] Jean-Luc Meunier. Optimized xy-cut for determining a page reading order. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, ICDAR '05, pages 347–351, Washington, DC, USA, 2005. IEEE Computer Society.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[NWM09] Zaiqing Nie, Ji-Rong Wen, and Wei-Ying Ma. Webpage understanding: Beyond page-level search. *SIGMOD Rec.*, 37(4):48–54, mar 2009.

[PBSB09] Ayelet Pnueli, Ruth Bergman, Sagi Schein, and Omer Barkol. Web page layout via visual segmentation. Technical report, HP Laboratories, 2009.

[PBSG10] Zeynep Pehlivan, Myriam Ben-Saad, and Stéphane Gançarski. Vi-diff: Understanding web pages changes. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part I*, DEXA'10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.

[Pfe10] Silvia Pfeiffer. *The Definitive Guide to HTML5 Video*. Apress, Berkely, CA, USA, 1st edition, 2010.

[PLR+10] Sung Hee Park, Nicholas Lynberg, Jesse Racer, Philip McElmurray, and Edward A Fox. Html5 etds. In *Proceedigs of International Symposium on Electronic Thesis and Dissertations*, Austin, TX, USA, 2010.

[Pop12] Tomas Popela. Implementace algoritmu pro vizualni segmentaci www stranek. Master's thesis, BRNO University of Technology, 2012.

[Ran71] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[RLRM05] David S. H. Rosenthal, Thomas Lipkis, Thomas Robertson, and Seth Morabito. Transparent format migration of preserved web content. *D-Lib Magazine*, 11(1), 2005.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[San12] Andrés Sanoja. Web archiving y su relevancia en el contexto de venezuela. In José Luís Berroteran Nuñez-Mirian Carmona Rodríguez, editor, *Consideraciones Teórico-Políticas para la Cienca y Tecnología en la Revolución Bolivariana Venezolana*, pages 311–321. Publicaciones MPPCTI/ONCTI. Ediciones Oncti, 2012.

[SG03] Alexander Strehl and Joydeep Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, March 2003.

[SG10] Myriam Ben Saad and Stéphane Gançarski. Using visual pages analysis for optimizing web archiving. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 43:1–43:7, New York, NY, USA, 2010. ACM.

[SKB08] F. Shafait, D. Keysers, and T.M. Breuel. Performance evaluation and benchmarking of six-page segmentation algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(6):941–954, 2008.

[SLWM04] Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma. Learning block importance models for web pages. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 203–211, New York, NY, USA, 2004. ACM.

[Sol14] Brian Solis. The conversation prism, 2014. `https://conversationprism.com`.

[SSL11] Fei Sun, Dandan Song, and Lejian Liao. Dom based content extraction via text density. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 245–254, Beijing, China, 2011. ACM.

[Sze10] Richard Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[TCL+99] Yuan Y Tang, Mb Cheriet, Jiming Liu, JN Said, and Ching Y Suen. Document analysis and recognition by computers. *Handbook of Pattern Recognition and Computer Vision, edited by CH Chen, LF Pau, and PSP Wang World Scientific Publishing Company*, 1999.

[TS94] Yuan Y Tang and Ching Y Suen. Document structures: a survey. *International journal of pattern recognition and artificial intelligence*, 8(05):1081–1111, 1994.

[VdH07] Jeffrey Van der Hoeven. Emulation for digital preservation in practice: The results. *The International Journal of Digital Curation*, 2(2):123–132, Decembre 2007.

[VEB09] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1073–1080, Montreal, Quebec, Canada, 2009. ACM.

[Vin09] Gujjar Vineel. Web page dom node characterization and its application to page segmentation. In *Proceedings of the 3rd IEEE International Conference on Internet Multimedia Services Architecture and Applications*, IMSAA'09, pages 325–330, Piscataway, NJ, USA, 2009. IEEE Press.

[WCLH11] Ou Wu, Yunfei Chen, Bing Li, and Weiming Hu. Evaluating the visual quality of web pages using a computational aesthetic approach. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 337–346, Hong Kong, China, 2011. ACM.

[WT14] Chaw Su Win and Mie Mie Su Thwin. Web page segmentation and informative content extraction for effective information retrieval. *IJCCER*, 2(2):35–45, 2014.

[XTL08] Yunpeng Xiao, Yang Tao, and Qian Li. Web page adaptation for mobile device. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1–5, Dailan, China, 2008.

[Yes11] Yeliz Yesilada. Web page segmentation: A review. Technical report, University of Manchester and Middle East Technical University Northern Cyprus Campus, 2011.

[YS09] Xin Yang and Yuanchun Shi. Enhanced gestalt theory guided web page segmentation for mobile browsing. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, volume 3, pages 46–49, Milano, Italy, Sept 2009.

[YZ01] Yudong Yang and HongJiang Zhang. Html page analysis based on visual cues. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, ICDAR '01, pages 859–864, Seattle, USA, 2001. IEEE Computer Society.

[ZFG08] Hui Zhang, Jason E. Fritts, and Sally A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Computer Vision and Image Understanding*, 110(2):260–280, 2008.

[ZG94] YJ Zhang and JJ Gerbrands. Objective and quantitative segmentation evaluation and comparison. *Signal processing*, 39(1):43–54, 1994.

[Zha96] Y.J. Zhang. A survey on evaluation methods for image segmentation. *Pattern Recognition*, 29(8):1335–1346, 1996.

[ZJKZ10] Aihua Zhang, Jiwu Jing, Le Kang, and Lingchen Zhang. Precise web page segmentation based on semantic block headers detection. In *Digital Content, Multimedia Technology and its Applications (IDC), 2010 6th International Conference on*, pages 63–68, Seoul, South Korea, Aug 2010.