Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 3, No. 1, pp. 23-37, Junio 2016
Selección de los Mejores Artículos del SCTC 2016

Block-based Migration from HTML4 Standard to HTML5 Standard in the Context of Web Archives

Andrés Sanoja[1], Stéphane Gançarski[2]
andres.sanoja@ciens.ucv.ve, stephane.gancarski@lip6.fr

[1] Escuela de Computación, Universidad Central de Venezuela, Caracas, Venezuela
[2] Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie, Paris, France

Abstract: Web archives are not exempt of format obsolescence. In the near future Web pages written in HTML4 format, could be obsolete. We will have to choose between two preservation strategies: emulation or migration. The first option is the most evident, however due to the size of the Web and the amount of information that Web archives handle it is not practical. In the other hand migration to HTML5 format seems plausible. This is a challenge because we need to modify a page (in HTML4 format) and include elements that not even exists in this format (as the HTML5 semantic elements). Using the Web page segmentation we show that, with the appropriate granularity, blocks look alike these semantic elements. We present the use our segmentation tool, BoM (Block-o-Matic), for helping achieve the migration of Web pages from HTML4 format to HTML5 format in the context of Web archives. We also present an evaluation framework for Web page segmentation, that helps to produce metrics needed to compare the original and migrated version. If both versions are similar the migration has been successful. We show the experiments and results obtained on a sample of 40 pages. We made the manual segmentations for each page using our MoB tool. Results shows that in the migration process there is no data loss but in the migrated version (after adding the semantic elements) the margin is changed. This is, it adds whitespace that change the elements position, shifting elements slightly on the page. While this is imperceptible to the human eye, for systems it is difficult to handle without previous knowledge of this situation.

Keywords: Migration; Web; Segmentation; Blocks; HTML5; Web Archive; Format Obsolescence.

I. 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[1]. 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 [1] 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? The W3C and the WHAT group are figuring out how Web browsers can be compatible with older versions of the specification. They say that is necessary to evolve HTML incrementally into XML[2]. The strategy is to process this pages differently. So far, Web browsers have been very permissive with malformed documents. In general, Web archives store pages along with all their dependencies. We agree with Rosenthal [2] that eventually, modern browsers will no longer be able to render document in HTML4 or XHTML formats in a proper way (i.e they will not be very permissive). Thus, a strategy for their preservation must be taken. Archivists must decide to perform either a emulation or migration.

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" [3]. 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 [4]. This includes converting a Web page file from one file format

[1] The proposed recommendation is out September 2014

[2] http://diveintohtml5.info

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.

In this article we present how we use Web page segmentation to perform the migration of HTML4 pages to HTML5 format. We think that a block-based solution is more effective than a tag-by-tag approach, since we must differentiate between "regular" tags and "semantic" tags.

Semantic tags (in theory) 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.

To measure the correctness of a migration we perform an Web page segmentation evaluation with a set of predefined manual segmentations and the corresponding migrated versions.

To this end, we present Block-o-Matic (BoM), our segmentation approach, and the model for evaluating segmentation algorithms. We apply both in this work to measure the correctness of the migration. The manual segmentation is made using the Manual-design-Of-Blocks tool (MoB) and a computed one, made with BoM. In this process we give a score based on the geometry of both segmentations. In addition to this the labels of each block are also compared.

The document is organized as follow. In Section IV we present the Web page segmentation concepts and notation. In Section V is presented BoM our approach to Web page segmentation. In Section VI we present our evaluation framework. In Section VII our solution, while in Section VIII the experiments and in Section IX the results. We conclude in Section X with the perspectives and outlook.

II. Related Work

Several efforts have taken place in order to make uniform the migration from one format to another [5]. 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.

There is a lot of online references to perform the tag-by-tag migration[3] however, as far as our knowledge goes, there are very few systematic and automatic approaches to solve the problem described above.

As an example, Park [6], 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.

Roshental [2] present and describe the design and implementation of a transparent, on-access format migration capability for the LOCKSS system for preserving Web content. Their implementation is capable of transparently presenting content collected in one Web format to readers in another Web format, with no changes needed to browsers. They present an user case of this type of migration on GIF image format migrated to PNG format. They identify the practical difficulties that face any implementation of emulation; they led them to choose the migration strategy

Conversely, Jackson [7] describe a method to identify how HTML and PDF formats changes in Web archives though time. They conclude that software obsolescence is rare on the Web and uncover evidence indicating that network effects act to stabilise formats against obsolescence.

However, we think that obsolescence can occurs in Web environments. We observe this behaviour with old *plugins* (*e.g.* Macromedia Shockwave content) in old Web pages. We agree with Rosenthal that any format is susceptible of been obsolete, and the HTML4, and earlier formats, are not the exception.

In the following sections we present our approach to Web page segmentation and its evaluation as a preliminary to describe our migration approach.

III. Overview of the Migration Process

In this section we present an overview of the migration process. The idea is to take a Web page in HTML4 format and produce a version of the same page according to the HTML5 format. The main goal of the process described in this paper is to measure a what extent this process is correct, and how reliable it is.

The process is illustrated in Figure 1 and can be divided in five steps, describe as follows:

1) **Segmentation of the input page**: a Web page in HTML4 format is segmented using the BoM segmenter (*c.f.* Section V).

[3]Googling the term 'translating html 4 tag to html5' will give these references

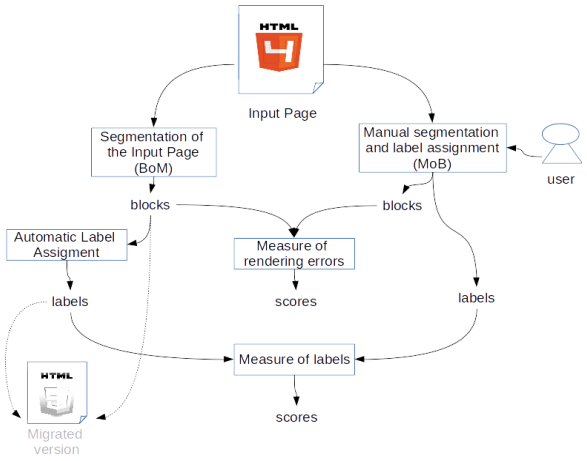

Figure 1: Migration Overview

2) **Automatic label assignment**: Based on the properties and characteristics of the blocks found in the segmentation we assign a label (*i.e.* HTML5 semantic elements tags) to each block (*c.f.* Section VIII-D).
3) **Manual segmentation and label assignment**: Using the MoB tool (*c.f.* Section VIII-B) we produce a *ideal* segmentation of the input page. In the same process the user assign a label to each block.
4) **Measure of labels**: from both segmentations (*i.e.* the manual and automatic one) we apply some measures to determine how different both assignments are. The metrics are described in detail in Section VIII-E.
5) **Measure of rendering errors**: Using the Web page segmentation evaluation framework (*c.f.* Section VI) we measure the difference on the rendering both of the automatic and manual segmentation

From the automatic segmentation outcome it is possible to produce the HTML representation, that is, the migrated Web page. This detail is not included in this paper, but technically is the transformation of a XML document into a HTML Web page.

IV. Web Page Segmentation

Web page segmentation refers to the process of dividing a Web page into visually and semantically coherent segments called blocks. For determining the coherence of each segment we relies on the content categories classifications made by the W3C for the HTML 5 specification (e.g. sectioning content). Detecting these different blocks is a crucial step for many applications, such as mobile devices [8], information retrieval [9], Web archiving [10], Web accessibility [11], evaluating visual quality (aesthetics) [12], 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 [13]. This is useful for crawling optimization, as it permits tuning crawlers so that they will revisit pages with important changes more often [10]. It also helps for controlling preservation actions, by comparing the page version before and after the action.

It is crucial for Web page segmentation to know which elements of the page are considered. For a Web page we extract visual and structural aspects found in the rendered DOM (W) of a Web page. From its structure we extract the elements in the form of a hierarchy (DOM tree), the root element ($W.root$). We obtain the text of the page ($W.text$) by recursively concatenating the the text of all elements. Each element corresponds to a HTML 5 content category. From its visual we get the *visual cues* (lines, blank areas, colors, pictures, fonts, etc) and the boxes of each element (rectangles). We have a special box called viewport representing the *body* element.

A. Concepts

Inspired by the concepts presented by Tang [14] and Nie [15], 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 modelled 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 [14]. It can be hierarchical [9] or non-hierarchical [16], [17]. 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.

B. Notation

We present in this section several definitions, in order to have an uniform presentation of Web page segmentations algorithms.

1) The Segmentation Function: The segmentation function Φ is described as follows:

$$\Phi_A\,(W,\,SC) \longrightarrow (W'_A, GM_A) \tag{1}$$

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.

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 [18] and GraphBased [16] define it as a general planar graph.

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

In this section 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 blocks labels.

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 V-C and V-D. 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 IV-A, 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 2). 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].

A. Model

In this section we present the Web page segmentation model. It is an hybrid approach, and it follows the bottom-up strategy [19].

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 IV-A.

Figure 2 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}.
2) **Composite block**. Detects the composite 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}.

B. 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. 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 4 shows which content category includes other content categories.

The process begins from the leaves of the DOM tree, towards the $W.root$. If an element is found that meets the condition C above defined, the process stops for this branch. Figure 3 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 IV-A) and a first version of the block graph are built (cf. section IV-A).

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 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
Blocks = {Page};
E = {};
W'_BoM = (Blocks, E);
GM_BoM = {};
Terminal ← getTerminalElements(W);
foreach element ∈ Terminal do
    while element ≠ W.root and ¬C(element) do
        element ← element.parentElement;
    end
    if element ≠ W.root then
        create block b;
        b.elements ← element;
        b.label = element.category;
        rect = createRectangleFromElement(element);
        add rectangle rect to GM_BoM;
        b.rect = rect;
        b.weight = normalized_area(b);
        add vertex b to W'_BoM;
        add edge (Page, b) to E;
    end
end
```

Algorithm 1: Fine-grained Segmentation Construction

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

C. Composite Block

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

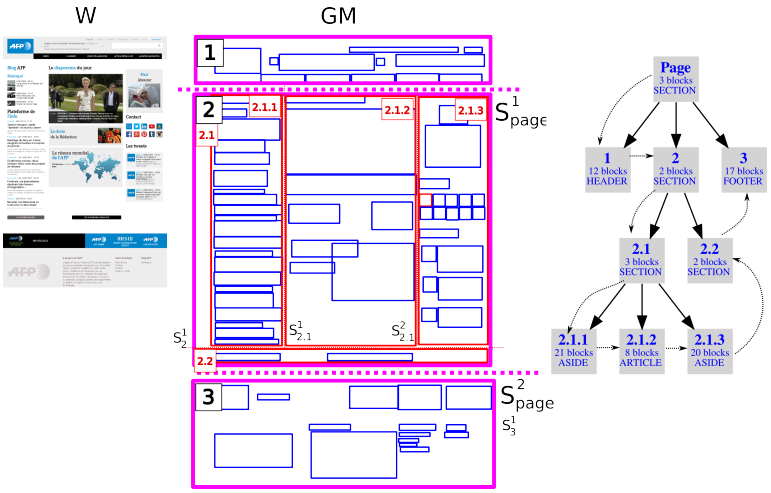

Figure 2: Segmentation Model Example

Figure 3: Block Detection Based on Content Categories

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 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.

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 *$s \in Separators$* **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

D. 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*.

The rules are checked in the given order for efficiency purpose:

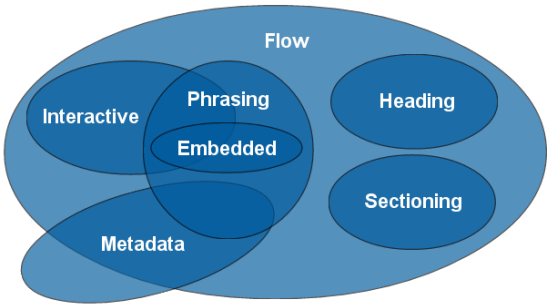

Figure 4: HTML5 Content Models. Source: http://www.w3.org

the first rules are most discriminant.

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 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 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 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 5c show the merged blocks. Block *f* is contained into block *d*, so they are merged and the label *flow* is assigned. Figure 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

VI. Segmentation Evaluation Model

In this section we present 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 work.

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[4] and the JavaScript version for the open source community[5].

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.

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.

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

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

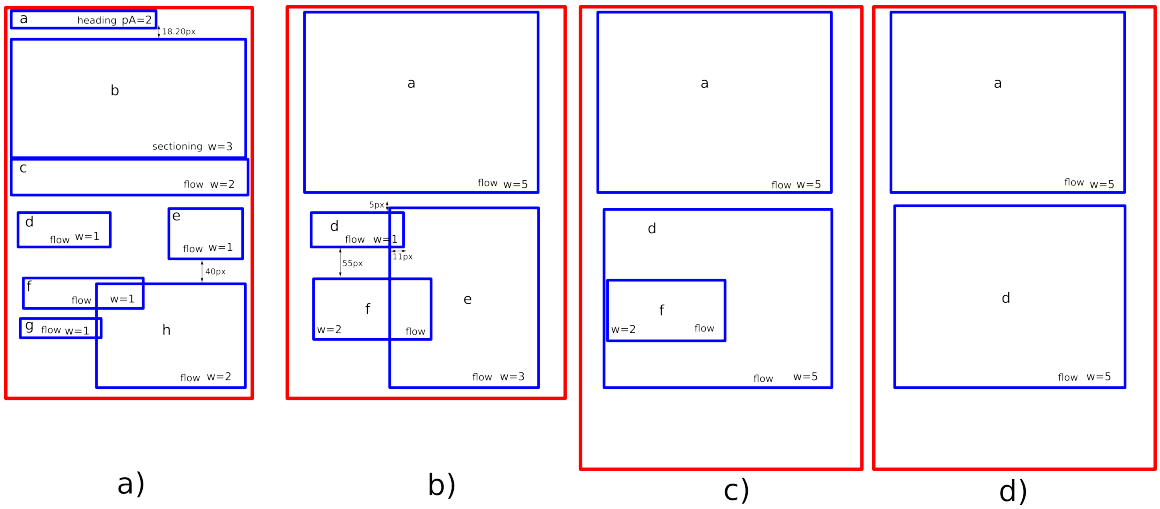

Figure 5: Merging Blocks and Labeling

This section 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 section, 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, in other words, they are flat segmentations. Our evaluation model is an adaptation to web pages of the model presented by [20] for scanned page segmentation evaluation (see Section VI-A). The quality of a segmentation is evaluated by using the block correspondence. The block correspondence measures allows knowing to what extent the generated blocks match those of the ground truth.

We present the evaluation model adaptation (VI-A), the representation of a segmentation (VI-B) and the representation of the evaluation (VI-D).

A. Model Adaptation

In order to adapt to web pages the model presented by Shafait et al. [20] 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 behaviour 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.

B. 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 (VI-B1 and VI-B2), as well as the importance of blocks and how it is computed (VI-C).

We present the concepts used along the section. We use the notation described in Section IV-B. We use the concepts of page, block and block graph based on the concepts described in the same section.

1) Absolute Representation of a Segmentation: Each block B is associated with its rectangle ($B.rect$), its label ($B.label$) and its weight ($B.weight$). 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 VI-C 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:

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

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.

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:

$$N\Phi_A(W, SC) \longrightarrow (NW'_A, NGM_A) \tag{2}$$

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.

C. 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 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. [21], 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:

$$computed\_importance(B) = \sum_{ij} im_{i,j} \tag{3}$$

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:

$$B.importance = \frac{computed\_importance(B)}{\sum_{b \in W'_A.Blocks} computed\_importance(b)} \tag{4}$$

D. 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 block correspondence (VI-E).

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 VI-B 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 VI-B1 and VI-B2. 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. Equation 5 shows the function.

The quality of a segmentation is measured by block correspondence. It measures how well the blocks of W'_P match with the ones of W'_G.

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.

E. 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 6(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.

$$evaluate(\Phi_G,\, \Phi_P,\, t_r,\, t_i,\, \text{ND},\, \text{NP}) = (\text{text coverage metric}, \text{correspondence metrics}) \quad (5)$$

As seen on Figure 6(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

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

in P fits perfectly with the 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. 6, 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. 6, 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. 6, 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. 6 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. 6 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.*

$$evaluate(\Phi_G,\, \Phi_P,\, t_r,\, t_i,\, \text{ND},\, \text{NP}) = (TC,\, C_x,\, IC_x\,) \quad (6)$$

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 $C_q + Cm = |G|$.

VII. Proposed Solution for Migration

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.

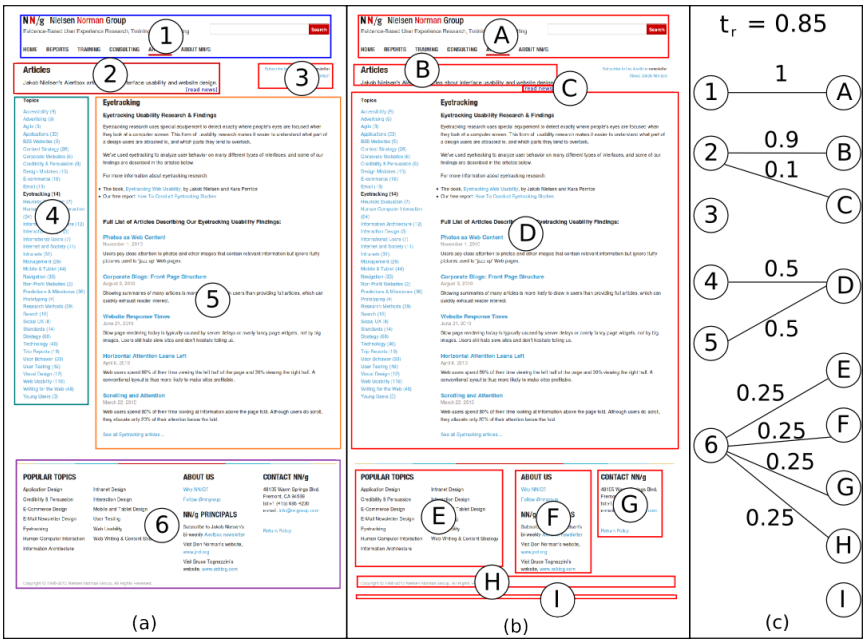

Figure 6: (a) Ground-truth Segmentation. (b) Computed Segmentation. (c) BCG

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 Section VI-D.

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

VIII. Experiments

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

A. Experimentation Design

1) GOSH Collection: 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.

Within a collection, each page 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.

Web pages are taken from the GOSH (GOogle SearcH) collection that we built. It is described as follows.

Web pages in this collection are selected with respect to their category. This selection is based in the categorization made by Brian Solis [22], "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 [6]. The GOSH collection contains 125 pages.

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

3) Experiments: 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 VI-E, 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.

[6] https://github.com/asanoja/web-segmentation-evaluation/tree/master/dataset

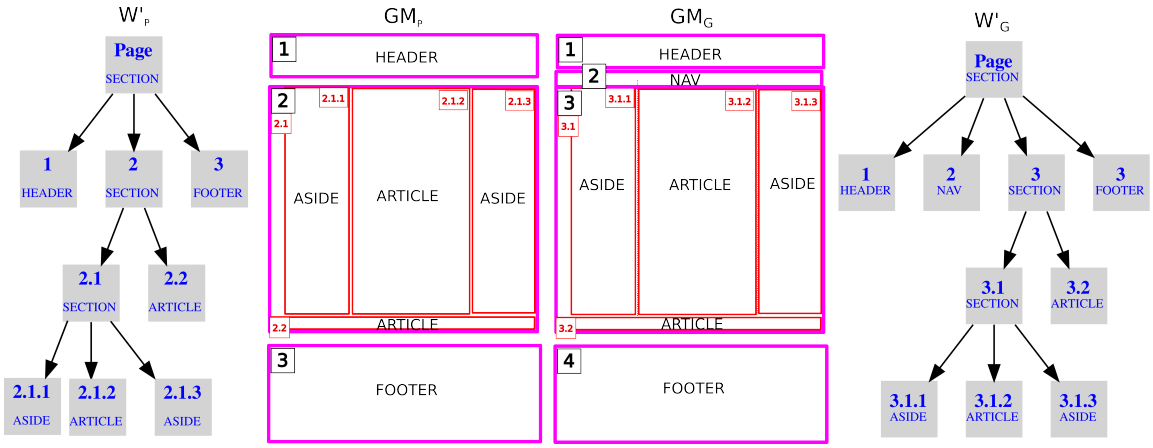

Figure 7: Labels for the Manual and Computed Segmentation

Table I: MIG5 Pages by Categories

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

B. Manual-design-Of-Blocks Tool

In order to build a Web page segmentation grountruth we develop the tool MoB (Manual-design-Of-Blocks). It is conceived as a browser extension and expose functionalities to expert users for creating a manual segmentations [7].

Users can create blocks based on Web page elements. They can merge blocks, navigate into the element hierarchy to produce a block graph [8] (*c.f* Section 2), or produce a flat segmentation (*i.e.* leaves in the block tree). These segmentations are stored in a repository [9] for the evaluation.

C. Ground Truth Building

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

The MoB tool (*cf.* Section VIII-B) 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). 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. The separation is set to $pD = 30$ because usually these regions can be very close one to each other.

[7] http://www-poleia.lip6.fr/ sanojaa/BOM
[8] Usually a tree
[9] http://www-poleia.lip6.fr/ sanojaa/BOM/inventory

D. 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.

E. Measuring Labels

The manual segmentation Φ_G and the computed segmentation Φ_P are formal defined in Section 3. 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 ($labels(W'_A)$).

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 7, shows the labels for the manual and computed segmentation. The list of labels from the manual segmentation

```
Data: Block: b
Result: B.label
if b.weight > pA then
    if b in the visible part of page then
        if b is the first block on top then
            if proportion of elements covered by b
            is greater than a constant then
                if b is composite then
                    B.label=SECTION;
                else if b has no siblings then
                    B.label=SECTION;
                else
                    B.label=HEADER;
                end
            else
                B.label=HEADER;
            end
        else if proportion of elements covered by b is greater
        than a constant then
            if b is composite then
                B.label=SECTION;
            else
                B.label=ARTICLE;
            end
        else if proportion of hyperlinks covered by b is
        greater than a constant then
            B.label=NAV;
        else if b is in the middle/center of the page then
            B.label=ARTICLE;
        else if b is the last block at bottom then
            B.label=FOOTER;
        else if b is at left/right of the page then
            B.label=ASIDE;
        else
            B.label=ARTICLE;
        end
    else if b is the last block at bottom then
        B.label=FOOTER;
    else
        B.label=ARTICLE;
    end
end
```

Algorithm 5: Label Assignment Algorithm

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:

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

The migration of Figure 7 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 [23].

$$error(W'_G, W'_P) = LD(labels(W'_G), labels(W'_P))$$

where LD is the Levenshtein distance. 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:

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

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

F. 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 VI-E. 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.

IX. 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 Section VI-D.

1) Measuring Labels: Table II shows the average values of the metrics defined in Section VIII-E for the MIG5 collection separated by categories. Column *CL1* represents the correct label measure ($correct\_labels(W'_G, W'_P)$). The *CL2* column represents the amount of labels in a segmentation ($|labels(W'_G)|$). The *CL3* column represents the rendering error ($error(W'_G, W'_P)$). The last two columns represents de precision and recall measures.

Table II: Average Values for Correct, Expected Labels and Error for the MIG5 Collection

category	*CL1*	*CL2*	*CL3*	*prec*	*rec*
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 III: Correspondence Metrics for the MIG5 Collection with $t_r = 0.1$ and $t_t = 1$

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

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 II shows the precision and recall metrics. Figure 8 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.

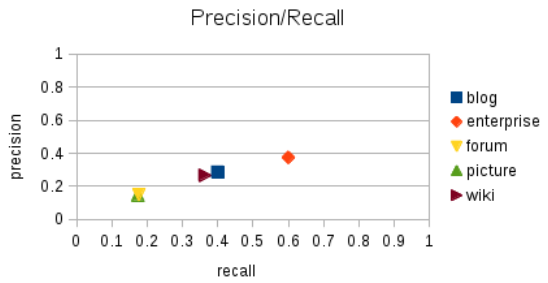

Figure 8: Precision and Recall for the MIG Collection

2) Measuring Rendering Errors: Table III 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.

X. 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 VI-E. 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 Section VI-D 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.

This work focus on the migration of the rendered version of a Web page, however as a future work it is interesting to include into the analysis other components of the Web pages such as Javascripts, CSS1 and CSS2. We need to assure that all dependencies of the migrated version and its accessibility are according to the new format.

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.

References

[1] B. Laws. *Seriously, Another Format? You Must Be Kidding.* CSE NEWS, vol. 36, no. 2, pp. 41, 2013.

[2] D. S. H. Rosenthal, T. Lipkis, T. Robertson, and S. Morabito. *Transparent Format Migration of Preserved Web Content.* D-Lib Magazine, vol. 11, no. 1, 2005.

[3] J. Van der Hoeven. *Emulation for Digital Preservation in Practice: The Results.* The International Journal of Digital Curation, vol. 2, no. 2, pp. 123-132, Decembre 2007.

[4] J. Garret. *Preserving Digital Information.* Technical report, Commission on Preservation and Access and the Research Libraries Group, 1996.

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

[6] S. H. Park, N. Lynberg, J. Racer, P. McElmurray, and E. A. Fox. *HTML5 ETDs.* In Proceedings of International Symposium on Electronic Thesis and Dissertations, Austin, TX, USA, 2010.

[7] A. N. Jackson. *Formats Over Time: Exploring UK Web History.* CoRR, abs/1210.1714, 2012.

[8] Y. Xiao, Y. Tao, and Q. Li. *Web Page Adaptation for Mobile Device.* In proceedings of the The 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2008), pp. 1-5, Dailan, China, October 2008.

[9] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. *Extracting Content Structure for Web Pages Based on Visual Representation.* In proceedings of the 4th 2008 International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM'08), APWeb'03, pp. 406–417, Xian, China, 2003. Springer-Verlag.

[10] M. B. Saad and S. Gançarski. *Using Visual Pages Analysis for Optimizing Web Archiving.* In Proceedings of the 2010 EDBT/ICDT Workshops, EDBT '10, vol. 7, no. 43, pp. 1-43, New York, NY, USA, 2010.

[11] J. U. Mahmud, Y. 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, pp. 31–40, New York, NY, USA, 2007. ACM.

[12] O. Wu, Y. Chen, B. Li, and W. 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, pp. 337-346, Hong Kong, China, 2011.

[13] Z. Pehlivan, M. Ben-Saad, and S. Gançarski. *Vi-diff: Understanding Web Pages Changes.* In Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part I, DEXA'10, pp. 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] Y. Y. Tang and C. Y. Suen. *Document Structures: A Survey.* International Journal of Pattern Recognition and Artificial Intelligence, vol. 8, no. 5, pp. 1081-1111, 1994.

[15] Z. Nie, J.-R. Wen, and W.-Y. Ma. *Webpage Understanding: Beyond Page-Level Search.* SIGMOD Rec., vol. 37, no. 4, pp. 48-54, March 2009.

[16] D. Chakrabarti, R. Kumar, and K. Punera. *A Graph-Theoretic Approach to Webpage Segmentation.* In Proceedings of the 17th ACM International Conference on World Wide Web, pp. 377–386, Beijing, China, 2008.

[17] C. Kohlschütter and W. Nejdl. *A Densitometric Approach to Web Page Segmentation.* In Proceedings of the 17th ACM Conference on Information and Knowledge Management, pp. 1173–1182, New York, NY, USA, 2008.

[18] X. Liu, H. Lin, and Y. Tian. *Segmenting Webpage with Gomory-Hu Tree Based Clustering.* Journal of Software, vol. 6, no. 12, pp. 2421–2425, Decembre 2011.

[19] A. S. Vargas. *Web Page Segmentation, Evaluation and Applications*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2015.

[20] F. Shafait, D. Keysers, and T. Breuel. *Performance Evaluation and Benchmarking of Six-Page Segmentation Algorithms.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 6, no. 30, pp. 941-954, 2008.

[21] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. *Learning Block Importance Models for Web Pages.* In Proceedings of the 13th ACM International Conference on World Wide Web, WWW '04, pp. 203–211, New York, NY, USA, 2004.

[22] B. Solis. *The Conversation Prism*. https://conversationprism.com.

[23] G. Navarro. *A Guided Tour to Approximate String Matching.* ACM Computing Surveys, vol. 33, no. 1, pp 31-88, March 2001.