A Quantitative Comparison of Semantic Web Page Segmentation Approaches

Robert Kreuzer

Jurriaan Hage

Ad Feelders

Technical Report UU-CS-2014-018
June 2014

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

A Quantitative Comparison of Semantic Web Page Segmentation Approaches

Robert Kreuzer Jurriaan Hage Ad Feelders

Department of Computing and Information Sciences
Utrecht University
robert@site2mobile.com,J.Hage@uu.nl, A.Feelders@uu.nl

Abstract

This paper explores the effectiveness of different semantic web page segmentation algorithms on modern websites. We compare three known algorithms each serving as an example of a particular approach to the problem, and one self-developed algorithm, WebTerrain, that combines two of the approaches. With our testing framework we have compared the performance of four algorithms for a large benchmark we have constructed. We have examined each algorithm for a total of eight different configurations (varying datasets, evaluation metric and the type of the input HTML documents). We found that all algorithms performed better on random pages on average than on popular pages, and results are better when running the algorithms on the HTML obtained from the DOM rather than on the plain HTML. Overall there is much room for improvement as we find the best average F-score to be 0.49, indicating that for modern websites currently available algorithms are not yet of practical use.

Categories and Subject Descriptors H.3.3 [*Information Systems*]: Information Search and Retrieval; D.2.2 [*Software Engineering*]: Computer-aided software engineering (CASE)

General Terms Experimentation, Algorithms, Measurement

Keywords web page segmentation, algorithm, benchmark construction, empirical comparison, repeated study

1. Introduction

Web page segmentation is the process of taking a web page, and partitioning it into so-called *semantic blocks* (or segments), that we define as

> A contiguous HTML fragment which renders as a graphically consistent block and whose content belongs together semantically.

Note that semantic blocks in principle can be hierarchically nested, although in practice people rarely consider nesting more than two or three levels. In this paper we do not consider the process of *labeling*, which takes a partitioning of a webpage into semantic blocks, and then assign labels to them taken from some ontology.

Human beings are very good at partitioning: even if a website is in a language we are not familiar with, it is clear to us what is an advertisement, what is a menu, and so on. Web page segmentation algorithms, for one reason or another, seek to automate this process: among the applications for automated segmentation we find mobile web, voice web, web page phishing detection, duplicate deletion, information retrieval, image retrieval, information extraction, user interest detection, visual quality evaluation, web page clustering, caching, archiving, semantic annotation and web accessibility (Yesilada 2011). Our own motivation for looking at these algorithms is to employ them in a start-up company for engineering mobile applications from existing browser applications. An important step in transforming the latter to the former is the process of web page segmentation that currently needs to be performed manually, either by the owners of the website, or by people that work at the start-up.

In this paper we perform an empirical study of the usefulness of three web page segmentation *approaches*: the DOM-based approach, the visual approach and the text-based approach. We also consider whether combining two of these approaches gives superior results. We compare the approaches by taking a representative algorithm for each, and evaluate their effectiveness in web page segmentation by comparing their outputs to a manually constructed ground truth. This should answer our first research question, which reads

RQ1: How well do different web page segmentation approaches perform on a large collection of modern websites? In particular, can we observe that the algorithms indeed fare worse on modern websites?

An additional, related research question we considered is the following:

RQ2: Can the results be improved by combining two of these approaches?

As we shall see in this paper, the answer to both these questions is affirmative. And although the algorithm that combines the two approaches often outperforms the other algorithms, its effectiveness is still much below what we would like to see. A conclusion of this paper is then also that new, better algorithms should be sought to more effectively perform web page segmentation.

We have constructed two datasets: one with a number of popular websites, that we expected to be complex and therefore the hard to deal with, and a second set with randomly selected websites to avoid having our results biased too much towards complicated websites. Our dataset are open for anyone to use (Kreuzer et al. 2013). We have already been contacted by researchers to ask for our datasets, showing that there is indeed a demand for such a benchmark. Another motivation for doing this study is that authors of web page segmentation algorithms each have their own dataset, and their own measuring methodology, which makes it impossible

to compare existing algorithms based on what they report in their papers.

The paper is structured as follows: in Section 2 we discuss the various approaches to web page segmentation, and our choice of algorithms for the comparison. We had to construct our own dataset of marked up web sites, and we discuss the process of doing so in Section 3. It also includes a description of the tools we employed both in constructing the datasets, and the testing framework we developed to execute the actual comparison. In Section 4 we provide the outcomes of our comparison. As we reflect upon our study in Section 5, we discuss our general observations, but also highlight possible threats to validity. Section 6 discusses related work, and Section 7 concludes.

2. Approaches and algorithms

There are many algorithms for web page segmentation, and we cannot hope to consider them all. Instead, we opted to see which of the *approaches* to web page segmentation works well on modern websites. For each approach we chose a representative algorithm (typically the one that is generally most cited or compared against), obtained an implementation from others, or, in the absence of such an implementation, implemented the algorithm ourselves. The three approaches we consider are the DOM-based approach, the visual approach and the text-based approach.

2.1 The DOM-based approach (PageSegmenter)

In a DOM-based approach one simply looks at the DOM, the tree built by parsing the HTML, for cues on how to segment a page. (This tree does not include the properties added by external CSS files; these will only be present in the render tree.) The idea is that the HTML structure should reflect the semantics of the page. The quality of these approaches thus depends on how strongly this is the case. To do the segmentation they rely on detecting recurring patterns, such as lists and tables, and on heuristics, like headline tags working as separators, and links being part of the surrounding text. The approach is easy to implement and efficient to run, since one only needs to parse the HTML and not render the page. Complications are that there are many different ways to build a HTML document structure for the same content, styling and layout information is disregarded by design, and will not work immediately in the presence of Javascript (unless you serialize the DOM first).

The PageSegmenter algorithm is an example of a DOM-based segmentation algorithm (Vadrevu et al. 2005). The main idea is that the root-to-leaf paths of leaf nodes can be analyzed for similarities to find nodes which likely belong to the same semantic block. An example of such a path is `/html/body/p/ul/li`. Now, if multiple adjacent siblings have the same such path it is a pretty safe assumption that they also semantically belong together, as they are structurally part of the same list.

The authors formalized this notion of similarity of paths of leaf nodes as the *path entropy*: the path entropy $H_P(N)$ of a node N in the DOM tree is defined as

$$H_P(N) = -\sum_{i=1}^{k} p(i) \log p(i)$$

where $p(i)$ is the probability of path P_i appearing under the node N and k is the number of root-to-leaf paths under N.

Thus one needs to first build a dictionary D which maps all root-to-leaf paths in the tree to their probability of occurrence. For a given node N the path entropy $H_P(N)$ can then be computed by first getting all of the root-to-leaf paths below that node, looking up their probabilities in D and plugging that into H_P. In pseudo code the algorithm can be formulated as follows:

```
for Each Subset S of Nodes[] do
    H(S) := Average Path Entropy of all nodes in S
    if H(S) <= MedianEntropy then
        Output all the leaf nodes under S as
            a new segment PS
    else
        PageSegmenter(Children(S))
```

Unfortunately, we found the algorithm to be unclear in two points. Point one is the formulation "for each subset S of Nodes[]", which is problematic in two ways: For one, if literally each subset is meant, then the algorithm becomes non-deterministic as the order in a set is not defined. Second, it makes little sense to consider each subset, as that also includes non-contiguous sets of nodes, and these cannot be segments. The second point is related, as it also pertains to the question of which subsets are meant. It is the recursion into the children of S which is unclear, since literally taking all of the children of the different nodes in S and then running the PageSegmenter on that set could lead to the detection of blocks, consisting of non-contiguous nodes (when their parents are different for example).

We therefore took the freedom to interpret the algorithm as we think the authors probably meant it, but could not confirm this with them. Essentially, instead of each subset we consider only ranges of contiguous nodes, and we exclude the full range of children and the empty range from consideration. We start with the largest ranges since we want to find the main blocks first and then later recurse into them to find the sub-blocks. If a block is found (i.e. when the average entropy of the nodes in a range is smaller or equal to the median entropy), it is added to the result list and we recursively invoke the algorithm on the siblings that precede it, and on the siblings that follow it, in order to make sure that we find all contiguous blocks of nodes. Implementation is straightforward, but a few things must be kept in mind: the tree should have explicit text nodes (which are then the leaves in the tree), because the assumption is that all content lies at the leaves. In their paper the authors mention that they exclude text nodes from the tree where the text contains *modal verbs* (such as could, should, would...) in order to decrease noise. Our implementation does not do so, because this would make the algorithm language-specific, which is something we want to avoid.

2.2 The visual approach (VIPS)

Visual approaches most resemble how a human segments a page, i.e. they operate on the rendered page itself as seen in a browser. They thus have the most information available, but are also computationally the most expensive because pages must be rendered. They often divide the page into separators, such as lines, white-space and images, and content and build a content-structure out of this information. They can take visual features such as background color, styling, layout, font size and type and location on the page into account. To render the page we need access to a browser engine, which complicates the implementation of an algorithm. And clearly it requires external resources such as CSS files and images in order to work correctly.

For this approach we elected VIPS (Vision-based Page Segmentation) algorithm (Cai et al. 2003a), which appears to be the most popular web page segmentation algorithm. As indicated by the name this algorithm is based on the rendered representation of a page. It analyzes the DOM after all the styling information from CSS rules have been applied and after Javascript files were executed (and potentially modified the tree). It is tightly integrated with a browser rendering engine since it needs to query for information such as the dimensions on screen of a given element. One thus has to decide on a fixed viewport size in advance on which the page should be rendered. Concretely, the algorithm builds a vision-based

content structure, independent of the underlying HTML document, deciding during a top-down traversal whether something represents a visual block, or whether it should be subdivided further by using a number of heuristics, such as "if a sub-tree contains separators like the `<hr>` tag, subdivide". We used an existing implementation from Tomas Popela (Popela 2012); the original implementation is not available anymore.

2.3 The text based approach (BlockFusion)

The *text-based* approach differs from the other two in that it does not take the tree structure of the HTML into account at all. Algorithms only look at the (textual) content and analyze certain textual features like e.g. the text-density or the link-density of parts of a page. These techniques are grounded in results from quantitative linguistics which indicate that, statistically, text blocks with similar features are likely to belong together and can thus be merged in a single block. The optimal similarity threshold depends on the wanted granularity and needs to be determined experimentally. The algorithms tend to be fast and easy to implement since they work independently from the DOM, but like the DOM-based approach will not work with JavaScript (unless you serialize the DOM first), do not not take structural and visual clues into account, and the extraction of sub-blocks requires local changes to the text-density threshold (since we can't employ document structure).

The representative chosen for the text-based approach is the BlockFusion algorithm (Kohlschütter and Nejdl 2008). The algorithm is grounded in the observation, coming from the field of quantitative linguistics, that the so-called *token density* can be a valuable heuristic to segment text documents. The token density of a text can simply be 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, by splitting on so-called separating gaps, which are HTML tags other than the `<a>` tag. For each atomic block the token density can then be computed. A merge strategy is then employed to merge blocks into progressively larger ones, if the difference between token densities of two adjacent blocks is below a certain threshold value. This is done repeatedly until no more blocks can be merged. Due to this design the algorithm does not support multiple levels of blocks by default, but by an extension in which we locally introduce a second smaller threshold value, and then call the BlockFusion algorithm on each (already merged) block, we can achieve a two-level hierarchy.

While there is no reference implementation of the BlockFusion algorithm as it is given in (Kohlschütter and Nejdl 2008), there is a related open source library from the same author, called boilerpipe, which is described in (Kohlschütter et al. 2010). We implemented the BlockFusion algorithm (specifically the BF-plain variant as described in (Kohlschütter et al. 2010)) on top of this library, since this allowed us to stay as close to the original implementation as possible because we could reuse many functions needed for the algorithm. For example the function that generates the atomic text blocks, which are later merged, is taken unmodified from the boilerpipe library as well as the block merging function and the function to calculate the text density of a block. We used a text density threshold value of $\vartheta_{max} = 0.38$, which the authors found to be the optimal value in their experimental evaluation.

2.4 A combined approach (WebTerrain)

The WebTerrain algorithm was developed as our own contribution to the segmentation problem. The main idea was to see if we can combine the different approaches from the other algorithms in order to improve upon the end result. The algorithm is based on a novel heuristic which inspired the name of the algorithm: Firefox has a little known feature which allows the user to see a 3D-rendered version of any website (choose Inspect Element after a right-click on any given page, and then click on the cube icon). The result looks similar to a geographic terrain map. This feature works by assigning an additional depth-value to each visible element on top of the common width- and height-values, which are already used in the normal 2D-representation of the page. The depth-value is simply the tree-level of the element. Experiments with this feature revealed that the elevation profile corresponds pretty well to what we would consider the semantic blocks of a web page. The heuristic that this observation leads to has the interesting property that it combines a plain structural approach with a rendering-based approach into one, since it not only takes the DOM tree into account but also the visibility and dimensions of each element. It is not possible to tell by simply looking at the original HTML document how it will ultimately be rendered. One does not know how much space each child of an element will take up on the screen, or if it will be visible at all. For this, one needs to actually render the page (although it does not need to be painted to the screen, of course) using a layout engine like e.g. the WebKit, which we employed (pyt 2013).

Further details about the various algorithms, implementations, complications, and how we extracted the necessary information from the outputs of the implementations are omitted for reasons of space, but can be found in (Kreuzer 2013).

3. The datasets

Since web page segmentation is a fairly well-studied problem, many authors have done an empirical evaluation of their algorithms. The datasets and methods used for this evaluation vary widely. There appears to be no standard dataset for this problem, instead everyone seems to create their own dataset by first randomly sampling web pages (sometimes taken from a directory site such as http://dmoz.org) and then manually marking up the semantic blocks and often also labeling the blocks according to a predefined ontology. To further illustrate this, we consider how the three chosen known algorithms were validated by the authors that proposed them.

For the VIPS algorithm the authors did not first create a dataset with a ground truth. Instead they ran VIPS on their sample pages and then manually graded whether the segmentation of that page was "perfect", "satisfactory" or "failed". This approach is problematic on two levels: First, there is the obvious conflict of interest, since the authors themselves are grading the results of their own algorithm. Second, whether a segmentation is "perfect" or "satisfactory" is rather subjective and can thus not be repeated by others.

For the BlockFusion algorithm the authors did not use precision and recall, but instead they used two cluster correlation metrics, namely *Adjusted Rand Index* and *Normalized Mutual Information* to quantify the accuracy of the segmentation. They did first create a ground truth manually, but it is unclear whether this was done by the authors themselves or by volunteers.

For the PageSegmenter algorithm the authors do use precision, recall and F-Score in their evaluation. But differently from us they did not do this for all blocks in general on a page, but they divided the blocks into three classes first (which they call Concept, Attribute and Value) and applied the metrics to each of these classes. This again prevents a direct comparison as this division into three classes is specific to their algorithm and not applicable to other segmentation algorithms.

Before building our own dataset we investigated the datasets used by other authors to find out how they chose their sample pages, sample sizes and whether they downloaded only the HTML documents themselves, or the referenced external resources as well. Furthermore we wanted to see whether any of these datasets would be suitable for our study as well.

We found five datasets; they are shown in Table 1. The manually labeled ones vary in size from 105 to 515, with the exception of the TAP knowledge base (Guha and McCool 2003) at a size of 9,068 which was a semantically labeled database that was used as a test-bed for the Semantic Web but is unfortunately not available anymore. The Web pages are sampled completely at random in (Chakrabarti et al. 2008), in (Kohlschütter and Nejdl 2008) they are taken from the Webspam UK-2007 dataset(Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/) comprising over 100 million pages, which is focused on labeling hosts into spam/nonspam, in (Kovacevic et al. 2002) they first downloaded 16,000 random pages from the directory site www.dmoz.org and randomly chose the sample pages from there. In (Vadrevu et al. 2005) they make a distinction between template-driven and non-template-driven Web pages (i.e. pages generated by a web page authoring system and hand-coded ones) which is not made by the others.

The block assignment was sometimes done by the authors and sometimes by volunteers, the latter being preferable to avoid bias. It is not always mentioned what granularity of blocks was used (i.e. whether only top-level blocks were marked up or sub-blocks as well), but no one specifically mentioned marking up sub-blocks which leads us to the assumption that no sub-blocks were highlighted. Since none of these datasets are available online or from the authors directly we were unable to confirm this though.

One other notable observation is that all datasets seem to consist only of HTML documents without any of the external resources referenced from the pages. While this is certainly partly due to the datasets being already up to 11 years old, when web pages on average were still a lot less complex than they are now, this is not realistic anymore for websites that are currently on-line. We discuss why later on in this section.

We were ultimately unsuccessful in finding a suitable dataset: either the authors of the papers never replied to our inquiries, and in the one case that we did find the dataset it was not suitable for our purposes since it was focused on news websites, and only included the original HTML sources and no external resources. The latter is a problem in our case, because all algorithms that depend on a rendered representation of the page will deliver very different results for a page with all external resources and one without (we want to lay out this point in a bit more detail, since it is relevant for building a suitable dataset.) In conclusion, we decided to construct our own benchmark set, which we have made publicly available (Kreuzer et al. 2013).

3.1 External resources in web pages

Web pages are not only made up of HTML documents but they can reference a number of external resources that the browser will download in order to render the page properly. The most common ones are images, videos, CSS files (which describe how a page should be styled) and Javascript files (often adding interactivity and animations to a page) but there are also lesser known ones like font files, JSON or XML files (providing raw data), favicons, and vector graphic files.

A browser will first download the HTML document itself, parse it into the DOM tree (i.e. an object representation of the document) and then find all the referenced resources and download those as well. If there are one or more external style sheets they will be parsed as well, and the style rules will be applied to the DOM. Finally if there were Javascript files found they will be interpreted by the Javascript engine built into the browser and they may apply arbitrary transformations to the DOM. Finally a render tree can be built from the DOM which is then painted on the screen.

So it is clear that if you only download the HTML document itself then its rendered representation will be vastly different from the page including all resources. For this reason we decided to build a dataset consisting of HTML documents together with all their external resources (and all links rewritten accordingly so that they point to the right files). Javascript poses a real challenge for any kind of Web page analysis. Since a Javascript program can modify the DOM arbitrarily and furthermore load in more data or other Javascript programs from external sources it is possible that the original HTML document and the rendered page have virtually nothing in common.

In effect, if we want be able to validate algorithms that employ the visual approach, we must include in our dataset all the external resources that a given web page needs to be visualized. To retrieve, for a given web page, all such external resources is not a trivial exercise. For example, CSS files have the ability to refer to other CSS files, and these may again refer to ever more such files. Javascript is really hard to deal with since it is a Turing complete language (unlike HTML and CSS). In practice we solve the problem by simply running the Javascript programs, but whether we have in fact retrieved all resources that we shall ever need is in fact undecidable in general. The best practical solution we could find is the `wget` utility, using finely tuned parameters[1]. It handles all of the difficulties mentioned above except references from within Javascript programs, and it also rewrites the links so that they all point to the right locations. We found that the downloaded pages rendered identical or nearly identical to the online version in most cases. The pages that used Javascript references to an extent that they could not be properly rendered offline were excluded from the dataset (18 out of 100 for the random dataset and 30 out of 100 for the popular dataset).

Having retrieved the web pages and the associated resources, they had to be marked up so that we would have a ground truth to compare the algorithms against. We implemented a small program, called Builder, in Javascript that, given the url of the web page, allows one of our volunteers to easily mark up the web page and store the results. It is run by clicking a bookmarklet which will load and run the Javascript program from our server. It then works by highlighting the DOM node the user is hovering over with the mouse, allowing her to select that block and showing her a menu where she can choose what type of block it is. The possible choices to classify a block were:

High-level-blocks Header, Footer, Content, Sidebar

Sub-level-blocks Logo, Menu, Title, Link-list, Table, Comments, Ad, Image, Video, Article, Searchbar

This block ontology was chosen with the goal of being comprehensive and it was divided into High-level blocks and Sub-level blocks (or level 1 and level 2 blocks) since Web pages can be segmented on different levels of granularity. E.g. a content-block can have a title, an image and an article as sub-blocks. While in principle there is no upper limit to how many levels of granularity you can have on a page, we found two levels to be sufficient in the majority of cases and have thus restricted ourselves to that.

For robustness we implemented the following solution to marking up the web pages in which the client (for marking up the web pages) and the server (serving the local version of the web pages) reside on the same machine: we first make the full page available offline using `wget`, then open that page in a browser, load the Builder, and add all the blocks and finally serialize the changed DOM to disk again. If one subsequently wants to get out all the blocks of a page one can do so using a simple Xpath query[2].

[1] The magic incantation is: wget -U user_agent -E -H -k -K -p -x -P folder_name -e robots=off the_url

[2] Xpath query to get all blocks: '//*[@data-block]'

Paper	Sample taken from	Size	Granularity	HTML only?	Type	Created by	Available
(Vadrevu et al. 2005)	TAP knowledge base(Guha and McCool 2003)	9, 068	?	yes	template-driven	external	no
	CS department websites	240	?	yes	non-template-driven	8 volunteers	no
(Kovacevic et al. 2002)	Random sites from dmoz.org	515	1	yes	all	the authors	no
(Kohlschütter and Nejdl 2008)	Webspam UK-2007(Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/)	111	1	yes	all	external	no
(Chakrabarti et al. 2008)	Random web pages	105	?	yes	all	the authors	no

Table 1. Datasets used in the literature

We built two different datasets, one containing only popular pages and one containing randomly selected pages. This was done to see if the algorithms performed differently on random and on popular pages on average. For the popular dataset we took the top 10 pages from the 10 top-level categories from the directory site `http://dir.yahoo.com/`. The chosen categories were Arts & Humanities, Business & Economy, Computer & Internet, Entertainment, Government, Health, News & Media, Science and Social Science. We believe this gives us a representative sample of popular websites, although not of websites in general. We manually checked all websites whether they still rendered properly after having been downloaded and removed the ones that were broken, which left us with a total of seventy popular pages in the dataset.

For the random websites we made use of the web service from `http://www.whatsmyip.org/random-website-machine/`. to generate a hundred links, which we then downloaded. The service boasts over four million pages in its database and the only filtering done is for adult content, which makes it sufficiently representative for the Internet as a whole. After removing pages that did not render properly offline we ended up with a random dataset consisting of 82 pages.

The marking up of the semantic blocks on these pages was done by three volunteers. They were instructed to first mark up all the level-1 blocks they could find and subsequently all the level-2 blocks within each level-1 block, according to the generic ontology we gave earlier in this section.

When establishing the ground truth with the help of outsiders, we must in some way be reasonably sure that this "truth" is objective enough. In other words, can we expect the intuitive understanding of what is a semantic block among our test persons to be aligned, and, moreover, in line with what the average person surfing the Web would say? The study of (Vadrevu et al. 2005) indicates that this is indeed the case. They report an overlap of 87.5% between eight test subjects who they asked to mark up the same web pages. Although the sample is small, it corresponds to our own anecdotal experience in this study. However, we did find that we had to be specific about the level of granularity (e.g. "the most high-level (i.e. biggest) blocks and their most high-level sub-blocks"), since there can be many levels.

3.2 The testing framework

To ease our comparison, we implemented a tool that executes the evaluation of a given algorithm for a given dataset, and returns to us the statistics. One design goal was that the framework should be generic enough to deal with the various algorithms, and not assume that these would all be implemented in the same language, or follow a particular API. All algorithms are given as input the HTML of the page including any resources the page might need to be rendered. The outcome of each algorithm is a marked up HTML page, that can then be compared against the ground truth for the page.

The testing framework uses a pipeline as its main design pattern. As inputs it takes the algorithm and the dataset specified by the user and it generates a statistical evaluation of the performance of the algorithm on that dataset as output.

The pipeline has four distinct components which have clearly defined interfaces. The first of these is the *DatasetGenerator* which is simply a function that knows how to retrieve the original HTML documents and the HTML documents with the manually highlighted blocks (the ground truth). In our dataset this information is e.g. provided by a *mapping.txt* file which simply provides a *url : filepath* mapping.

The original HTML document is then fed to the *AlgorithmDriver*, which is a small function unique to each algorithm, that knows how to apply the specified algorithm to the given document. This function needs to be specific to each algorithm since algorithms can be implemented as libraries, executables or even Web services. The driver interfaces with the algorithm by some means like e.g. a sub-process or an HTTP request, and returns the extracted blocks.

Since there is no unified format for semantic blocks and different algorithms return different formats it is necessary to normalize the data representation of the blocks. The *BlockMapper* component takes care of this. It takes the raw blocks, which can, for example, be only the textual content that has been extracted, or fragments of the HTML, and maps them back onto the original HTML document to produce our standard format. For this standard we decided to use the original HTML where the semantic blocks have been marked up with the two additional attributes `data-block` and `data-block-type`. These attributes are either added to already existing elements in the document or if multiple elements need to be grouped together we wrap them in an additional `div` element and add the attributes there. Attributes with a "`data-`" prefix have the advantage of being valid HTML 5 and were added as a means to attach private data to elements, which does not affect the layout or presentation. Furthermore, storing the algorithm results as HTML has the advantage that you can still render the page and see the highlighted blocks (with an appropriate CSS rule added) and they are also easy to query via Xpath or CSS selectors. Each individual algorithm has its own implementation of BlockMapper, because the formats of the returned semantic blocks differ widely.

Finally there is the *Evaluator* component that takes the normalized Block HTML and the HTML ground truth as inputs and does the statistical evaluation of the algorithm results. It calculates precision, recall and F-score, and returns the number of blocks retrieved by an algorithm, the number of correctly found blocks (the hits) and the total number of relevant blocks on a page. The results for each page are written to a CSV file so that they can be analyzed and visualized with common spreadsheet software.

The equality of blocks is tested with two different metrics: an exact match metric and a fuzzy match metric. For both metrics the

blocks are first serialized to text-only (i.e. all tags are ignored) and white-space and newline characters are removed to reduce noise. The exact match metric then does a string equality check to find matching blocks (the intersection of the set of found blocks and the ground truth is taken), the fuzzy match metric considers strings that differ at most 20 percent as equal.

Most of our framework is implemented in Python. A good reason for doing so is the availability of a Python library (pyt 2013) providing complete DOM bindings, thus making Python a full peer of Javascript, which was a necessary requirement for implementing segmentation algorithms that need to query the DOM. Also there are robust libraries for parsing (often invalid) HTML documents and querying them via Xpath. Furthermore Python programs, which are (typically) interpreted, can easily be modified and extended, making Python a good choice for prototyping and problem exploration.

4. Results

In this section we present the results of our evaluation of the four different segmentation algorithms. We tested all algorithms in a number of different configurations using the testing framework presented in Section 3.2. First, we tested them on the two different datasets which we created for this purpose: the randomly selected dataset and the popular dataset. The first one consists of 82 random pages and the second one of 70 popular pages, all marked up by our assessors. We chose these two types of datasets to test whether the algorithms perform differently on random and on popular pages on average.

As a second variable we ran the algorithms on both the original HTML, i.e. the HTML document downloaded from the source URL via a single GET request, and the DOM HTML, i.e. the HTML document obtained by waiting for all external resources to load and then serializing the DOM. As there appears to be a trend to build pages dynamically on the client-side using Javascript, we were interested to see whether our results would reflect this. It is also of note that our tool to mark up blocks manually was browser-based and thus operated on the DOM, making the DOM HTML the true basis of our ground truth. We believe this is a more sensible basis than the original HTML, since it is what the user ultimately sees when viewing a page, and it also is what the creator of the page intended as the final result.

Finally we used two metrics to compare the generated results to the ground truth, the exact match metric and the fuzzy match metric. Both of them compare the string contents of the blocks to each other. Each block is serialized to only text with all HTML tags removed and white-space and newlines removed as well. For the exact match metric it then simply checks for string equality. This is of course a very strong criterion, as a minimally different string would be counted as false, while for most applications it would likely be perfectly sufficient. For this reason we also do a fuzzy string comparison to check for a similarity ratio of better than 0.8 between strings.

So all together there are four testing variables: algorithms, datasets, HTML-type and metrics. This yielded 32 test runs in total, the results of which are presented in the 8 tables below (each algorithm is in each table to facilitate direct comparisons). For each algorithm we show the average Precision, Recall and F-Score values. Precision is a measure of quality that is defined as the number of relevant results out of all retrieved results. Recall is a measure of quantity that is defined as the number of retrieved results out of all relevant results. The F-Score is a combination of the two, defined as $F = 2 * \frac{P*R}{P+R}$. Additionally we also show the average number of retrieved blocks, valid hits (i.e. the number of relevant results returned by the algorithm) and the total number of relevant results (determined by the ground truth). The latter is interesting as it shows the difference in the average number of retrieved blocks and it also shows differences between the two datasets.

Random dataset, original HTML input

We first present the results of running the four different algorithms on the dataset consisting of 82 randomly selected pages. On average we have 12.24 relevant blocks on a random page. BlockFusion returns on average about twice as many blocks as there are relevant blocks. PageSegmenter returns about four times as many blocks as there are relevant blocks. VIPS returns too few blocks on average. Finally, WebTerrain is the closest in the number of retrieved results to relevant results. As we expected, results are considerably better for the fuzzy match metric as compared to the exact match.

Exact match metric

Under the exact match metric, Precision and Recall are generally very low. BlockFusion and VIPS recognize hardly anything. Precision is highest for WebTerrain and Recall is highest for PageSegmenter.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.03	0.06	0.04	25.99	0.77	12.24
PageSegmenter	0.11	0.27	0.14	46.96	2.97	12.24
VIPS	0.07	0.06	0.06	7.42	0.91	12.24
WebTerrain	0.25	0.22	0.21	10.9	2.23	12.24

Table 2. Results for Random-HTML-Exact. Note that the reported numbers are averages over 82 web pages. For example, the reported F-score was computed by first computing the F-score for each web page, and then taking the average of these 82 F-scores. Therefore the reported values for Precision, Recall and F-score may not satisfy the formula for the F-score as given in the text.

Fuzzy match metric

Precision and Recall are clearly better for the fuzzy match metric with the number of hits roughly doubling. Especially VIPS improves substantially, indicating that a number of its blocks were only slightly off from the ground truth. The best F-Score (0.42, WebTerrain) is still rather low.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.06	0.11	0.07	25.99	1.51	12.24
PageSegmenter	0.19	0.45	0.24	46.96	5.24	12.24
VIPS	0.28	0.16	0.17	7.42	1.99	12.24
WebTerrain	0.48	0.43	0.42	10.9	4.5	12.24

Table 3. Random-HTML-Fuzzy

Random dataset, DOM HTML input

For the DOM HTML input, we again observe a notable improvement when comparing the exact to the fuzzy match metric, but not quite as dramatic as for the original HTML. The number of retrieved blocks is generally higher (WebTerrain is minimally lower), reflecting the observation that the DOM HTML is typically more complex (as mostly things are added, rather than removed).

Exact match metric

BlockFusion is performing poorly, but better than on the original HTML. PageSegmenter again exhibits low precision and high recall. VIPS has the best precision and lower recall, while WebTerrain does similarly on both, giving it the best F-Score.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.08	0.14	0.09	30.96	1.79	12.24
PageSegmenter	0.11	0.39	0.16	65.04	4.47	12.24
VIPS	0.36	0.21	0.24	9.24	2.73	12.24
WebTerrain	0.34	0.29	0.29	10.58	3.04	12.24

Table 4. Random-DOM-Exact

Fuzzy match metric

We see about a 50% improvement compared to the exact match metric. WebTerrain and VIPS have the highest precision, and PageSegmenter and WebTerrain have the highest recall. Compared to the original HTML we see some improvements as well, especially for the VIPS algorithm. Overall we see the highest scores here out of all benchmarks.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.1	0.17	0.12	30.96	2.35	12.24
PageSegmenter	0.15	0.51	0.2	65.04	6.12	12.24
VIPS	0.51	0.26	0.3	9.24	3.33	12.24
WebTerrain	0.57	0.49	0.49	10.58	5.33	12.24

Table 5. Random-DOM-Fuzzy

The popular dataset

Here we present the results of running the four different algorithms on the dataset consisting of 70 popular pages. On average we have 16.1 relevant blocks on a page. The slight variation in relevant blocks is because we had to exclude a few (no more than four) pages for some of the algorithms, as they would not be handled properly due to issues in their implementation (e.g. a GTK window would simply keep hanging).

Between the original HTML and the DOM HTML one can see that the number of retrieved blocks universally goes up, giving another sign that the DOM HTML generally contains more content. Overall the results are again better for the DOM HTML, questioning the use of the original HTML in web page segmentation algorithms.

Popular dataset, original HTML input

The pattern seen in the random dataset repeats: results for the fuzzy match metric are about twice better than for the exact match metric. Both BlockFusion and PageSegmenter return decidedly too many blocks on average, but only PageSegmenter can translate this into high recall scores. VIPS and WebTerrain are fairly close to the relevant number of blocks.

Exact match metric

The results are generally poor with WebTerrain having the best precision and PageSegmenter having the best recall.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.03	0.06	0.03	72.85	1.07	16.15
PageSegmenter	0.05	0.24	0.08	124.43	4.05	16.22
VIPS	0.07	0.09	0.07	16.72	1.17	16.11
WebTerrain	0.18	0.17	0.16	13.86	2.19	16.09

Table 6. Popular-HTML-Exact

Fuzzy match metric

The results are better than for the exact match metric, but overall still not convincing. Again WebTerrain and PageSegmenter are the best for precision and recall respectively.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.05	0.12	0.06	72.85	2.07	16.15
PageSegmenter	0.09	0.42	0.13	124.43	6.74	16.22
VIPS	0.13	0.15	0.12	16.72	2.23	16.11
WebTerrain	0.37	0.35	0.33	13.86	4.81	16.09

Table 7. Popular-HTML-Fuzzy

Popular dataset, DOM HTML input

Similar to what we saw in the random dataset the improvement from exact to fuzzy matches is smaller than it was for the original HTML, but still substantial.

Exact match metric

The results are overall better than for original HTML with the biggest gains for VIPS and WebTerrain. WebTerrain has both the highest precision and the highest recall in this test.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.03	0.08	0.04	81.75	1.34	16.15
PageSegmenter	0.05	0.27	0.07	163.71	4.56	16.3
VIPS	0.13	0.14	0.12	19.51	2.25	16.11
WebTerrain	0.27	0.28	0.26	14.75	3.77	16.09

Table 8. Popular-DOM-Exact

Fuzzy match metric

The results for the popular dataset are the best again, as in the random dataset, when running on the DOM HTML and using the fuzzy match metric. The results for BlockFusion are again the worst. PageSegmenter has again low precision and high recall. Noticeably different is VIPS, as it does not exhibit a high precision, as it did for the random dataset. Recall is similar, though slightly lower. WebTerrain exhibits the highest precision and recall, but precision is 0.1 points lower and recall 0.03 points lower than for the random dataset.

Algorithm	Prec.	Recall	F-Score	Retr.	Hits	Rel.
BlockFusion	0.04	0.12	0.06	81.75	2.13	16.15
PageSegmenter	0.07	0.41	0.11	164	6.68	16.22
VIPS	0.19	0.21	0.17	19.51	3.29	16.11
WebTerrain	0.47	0.46	0.42	14.74	6.43	16.09

Table 9. Popular-DOM-Fuzzy

5. Reflection

In this section, we discuss the results of the previous section in some detail. We first consider what the observed effects were of the various testing variables.

Random vs. popular datasets We created one dataset consisting of random pages and one consisting of popular pages to see whether the segmentation algorithms perform differently on them. As can be seen from our results all algorithms perform virtually always better on the random pages than on the popular pages. We believe this is due to the increased complexity of popular pages, which can be seen from the fact that they on average had 32% more blocks than a random page. Furthermore we also found that a popular page on average consists of 196.2 files in total (this number includes all the external resources referenced from a page), while a random page only consists of 79.4 files on average. The number of retrieved blocks are also universally higher for all algorithms on the popular pages. But while the number of blocks in the ground truth was only 32% higher, the numbers for the algorithms increased

by (much) more than that: BlockFusion 164.1%, PageSegmenter 152.1%, VIPS 111.1%, WebTerrain 39.3%. It thus seems that the algorithms do not scale well with increasing complexity. This could also partly explain why our results are generally less favorable than what has been found in earlier publications, as they are up to 10 years old, and the Web has become much more complex since then. It also shows the need for new techniques that deal well with this increased complexity.

Exact vs. fuzzy match metric We found that the number of recognized blocks improved significantly when using the fuzzy match metric as opposed to the exact match metric, as was to be expected. We believe that the results from the fuzzy match metric are generally more valuable since the quality of blocks will still be sufficient for most applications. Furthermore it can easily be adjusted to find more or less precise matches by adjusting the matching ratio.

Original HTML vs. DOM HTML Comparing the original and the DOM HTML we found that the results of the segmentation for the DOM HTML are virtually always better, which is true for all algorithms on both datasets. This is due to the fact that the DOM HTML is what the user ultimately sees in the browser, it is thus the final result of the rendering process. While in the past it might have been sufficient to analyze only the original HTML, this is not true any more. As the Web becomes more dynamic and the use of Javascript to manipulate the page becomes more prevalent, there is not necessarily a link between original and DOM HTML any more. This also implies that one cited advantage of text-based segmentation algorithms, namely that they do not require the DOM to be built and are thus very fast, is not true any longer, as even for these algorithms it is necessary to obtain the final HTML for optimal results.

The four segmentation algorithms

The four algorithms differ widely in their performance, and none of them performed well enough to be universally applicable, as the highest average F-Score was 0.49 (WebTerrain). Our comments here pertain to the test runs using the fuzzy match metric and the DOM HTML because we consider those the most relevant. But the general conclusions hold for the other testing combinations as well.

BlockFusion This algorithm showed the worst performance on both datasets. Both precision and recall are very low (< 0.1 and < 0.2 respectively). It also returns too many blocks on average (2.5x too many for the random dataset and 5.1x too many for the popular dataset). We could thus not repeat the results from (Kohlschütter and Nejdl 2008). We conclude that a solely text-based metric is not sufficient for a good segmentation, but that it can be used to augment other approaches.

PageSegmenter This algorithm exhibits low precision and (relatively) high recall (< 0.2 and > 0.4 respectively). This is due to the fact that it retrieves by far the most blocks from all algorithms (5.3x too many for the random dataset and 10.1x too many for the popular dataset). The number of false positives is thus very high. It would thus be interesting to see if this algorithm could be optimized to return fewer blocks while retaining the good recall rates.

VIPS This algorithm showed the biggest difference between the random and the popular dataset. Precision was high and recall mediocre for the random dataset (0.51 and 0.26 respectively), while both were low for the popular dataset (0.19 and 0.21 respectively). It is not clear why there is such a substantial difference. The number of retrieved results is slightly too low for the random dataset, while it is slightly too high for the popular dataset (25% too low and 21% too high respectively). In terms of the F-Score the VIPS algorithm had the second-best result.

WebTerrain This algorithm showed relatively high precision and recall for both datasets (both > 0.4). It retrieved slightly too few blocks for both datasets (14% too few for the random dataset and 8% too few for the popular dataset). We thus find that a combination of structural and rendering-based approaches enhances overall results. Furthermore the terrain heuristic seems promising. Future work could therefore likely improve upon these results by using more sophisticated combinations of different approaches and heuristics.

Analysis of variance

We performed an analysis of variance (ANOVA) of our raw results to test the impact of the four factors `algorithm (with levels: blockfusion, pagesegmenter, VIPS, WebTerrain)`, `html (levels: dom, html)`, `dataset (levels: popular, random)` and `metric (levels: exact, fuzzy)` on the F-score. The results of the analysis are shown in Table 10.

	Df	Sum Sq	Mean Sq	F stat.	p-value
algorithm	3	20.59	6.864	319.416	$<$ 2e-16
html	1	1.58	1.581	73.569	$<$ 2e-16
dataset	1	2.28	2.280	106.119	$<$ 2e-16
metric	1	4.44	4.438	206.536	$<$ 2e-16
algorithm:html	3	1.26	0.419	19.505	1.74e-12
algorithm:dataset	3	0.24	0.080	3.731	0.0108
algorithm:metric	3	2.19	0.730	33.983	$<$ 2e-16
html:dataset	1	0.17	0.172	8.006	0.0047
html:metric	1	0.05	0.052	2.401	0.1214
dataset:metric	1	0.10	0.096	4.468	0.0346
Residuals	2298	49.38	0.021		

Table 10. ANOVA summary

We can read from the column labeled *p-value* that for all terms except `html:metric` the null hypothesis of 'no effect' will be rejected at the conventional significance level of $\alpha = 0.05$. To avoid confusion we note that the Analysis of Variance was performed on the F *scores* obtained for different combinations of factor levels, which is an entirely different quantity then the F *statistic* which is used to test the significance of different (combinations of) factors in explaining the variation in observed F *scores*.

We see in Table 10 that the four factors `algorithm`, `html`, `dataset` and `metric` are all highly significant (p-value $<$ 2e-16) in explaining observed variation in F-scores. The analysis thus confirms our intuition that these factors are relevant for an analysis of web page segmentation algorithms.

We also included *interaction terms* in the analysis (all the colon-separated variables, such as `algorithm:html`). An interaction term is the product of two variables that can in itself be a significant predictor of the outcome. A high significance for an interaction term means that the two variables interact, i.e. the effect on the outcome (in our case the F-score) for a given variable x is different for different values of a variable y (for a given interaction term $x : y$).

In Table 10 we see that all interaction terms are significant at $\alpha = 0.05$, except for the term `html:metric`. The term `html:metric` not being significant means that the influence of `metric` on the F-score is typically similar for different values of `html`. We can explain this by looking at the results reported in Section 4, where we found that the fuzzy metric always returns a higher F-score than the exact metric, regardless of the type of HTML used.

Encountered problems

We ran into a number of complications and inconveniences during our work. The first is the discovery was that there is no de-facto "standard" dataset on which everybody bases the evaluation of their

segmentation algorithm, as is common in other fields such as spam detection. We have open-sourced our datasets and hope that they will be of use to others (Kreuzer et al. 2013).

The second problem was that we could not obtain the original implementation of any of the three algorithms in our comparison (the implementation of VIPS is not from the original authors). This again leads to duplication of work, as we had to re-implement these algorithms, and it makes the results more fragile as it is impossible to prove that they were implemented exactly according to their specification. This is true as often the descriptions of algorithms are not specific enough and required interpretation.

While people in general have a shared understanding of what constitutes a semantic block on a particular level (i.e. top-level, sub-block, sub-sub-block), there can still be a difference in the granularity that a specific algorithm is targeting. This needs to be taken into account when comparing different segmentation algorithms.

5.1 Threats to validity

As in any empirical study there are various threats to validity. Unless noted otherwise, all discussed threats are to external validity.

A threat to construct validity concerns the implementations and our interpretations of the algorithm. Because we had to make these interpretations, and we did not obtain answers to our e-mails about these interpretations from the authors, we run the risk that we are not exactly measuring their algorithm, but a variation thereof. We have made clear in the paper what our interpretations are, and why we judge them to be reasonable.

In our study, we compare approaches to web page segmentation by looking at a particular instance of each approach. In our selection we have chosen well-known, often-cited representatives of each approach. To compensate, we have made our framework and datasets open for everyone to use, so that others can easily extend upon our work, by implementing other instances of the paradigms testing these against the ones that we have implemented.

When it comes to the experimental data, we used two datasets: one with popular web pages, mainly because doing well on pages that are often read by people is something an algorithm should be rewarded for, but also to serve as a "worst-case" since we expected these websites to be more complex than the average website. Since we also did not want to bias too much towards such pages, we also included a large sample of random pages, with the aim of increasing external validity. However, we did have to drop 48 out of 200 web pages, because when rendered locally they differed from the original web page. This means that our results may not generalize to websites that essentially exploit JavaScript references in a complicated way.

The mark up was performed according to a particular ontology (Section 3), which may hurt external validity. We do believe our ontology to be generic enough to be applicable to most existing websites. A second issue at this point is construct validity: did our three test subjects understand the ontology, and what was expected of them? To make sure that that was the case, the first author first explained the ontology to them, and checked five segmentation results for each test subject to see whether they had understood him well enough. It was also verified for a few samples pages that the volunteers agreed on the web page segmentation for those pages. Another study confirms this finding (Vadrevu et al. 2005). We should note that our test subjects all have an IT background, which may decrease external validity.

An issue in our study is that one of the algorithms we consider has been of our own devising. We note, however, that having confirmed that our implementation was correct, i.e., it behaved as we designed it to do on a few web pages, we did not make any modifications to it during or after we ran our experiments. This to avoid the danger of overfitting our algorithm to the chosen datasets.

6. Related Work

The research on structuring information on web pages into semantic units goes back at least to 1997 (Hammer et al. 1997), in which an extraction tool is described where the user can declaratively specify where the data of interest is located and how the extracted data should be interpreted. Subsequent authors tried to automate the process of locating where the information of interest resided. The process can be broken into two distinct steps: segmentation (what belongs together in blocks) and labeling (what is the best description for the block, in terms of some chosen ontology).

In (Vadrevu et al. 2005), we find the PageSegmenter algorithm that uses the structural and presentation regularities of web pages to transform them into hierarchical content structures (i.e., "semantic blocks"). They then proceed to tag the blocks automatically using an abstract ontology consisting of *Concepts, Attributes, Values* and *None*. They tested their work experimentally against the TAP knowledge base (Guha and McCool 2003) (which was not not available anymore for our study) and on a home-made dataset consisting of CS department websites. In (Vadrevu and Velipasaoglu 2011), the authors also rate the individual blocks by learning a statistical predictor of segment content quality and use those ratings to improve search results.

In (Kovacevic et al. 2002), the approach is based on heuristics that take visual information into account. They built their own basic browser engine to accomplish this, but do not take style sheets into account, and they avoid calculating rendering information for every node in the HTML tree. They then define a number of heuristics on the rendered tree assuming that the areas of interest on a page are header, footer, left menu, right menu and center of the page, and where they should be located, e.g. header on top. An issue is that this assumption does not hold for web pages that are more similar to desktop applications. The authors test their algorithm experimentally by first building a dataset where they manually label areas on 515 different pages, then run their algorithm on the dataset and subsequently compare the manual and algorithmic results. Their overall accuracy in recognizing targeted areas is 73%.

In (Chakrabarti et al. 2008), the authors turn the DOM tree into a complete graph, where every DOM node is a vertex in the graph. Each edge is then assigned a weight that denotes the cost of putting these two vertices into the same segment. The weights are learned from a dataset regarded as the ground truth by looking at predefined visual- and context-based features. Finally they group the nodes into segments by using either a correlation clustering algorithm, or an algorithm based on energy-minimizing cuts; the latter performs considerably better in their empirical evaluation. Their evaluation is based on manually labeled data (1088 segments on 105 different web pages).

Another approach is to consider text-density as the driving heuristic (Kohlschütter and Nejdl 2008). Instead of analyzing the DOM tree, like many other authors do, the focus is on discovering patterns in the displayed text itself. Their key observation is that the density of tokens in a text fragment is a valuable cue for deciding where to separate the fragment. More details were given earlier in the paper. They evaluate their approach experimentally using a dataset consisting of 111 pages. They achieve a better precision and recall than (Chakrabarti et al. 2008).

In (Cai et al. 2003a), we find an approach based on the visual representation of a web page. Instead of looking at the DOM tree representation of a web page, they developed a recursive vision-based content structure where they split every page into a set of sub-pages (visual blocks of a page), a set of separators and a function that describes the relationship between each pair of blocks of a page in terms of their shared separators. They deduce this content structure using the VIPS algorithm(Cai et al. 2003b) which goes top-down through the DOM and takes both the DOM structure

and the visual information (position, color, font size) into account. They test their algorithm experimentally by sampling 140 pages from different categories of the Yahoo directory and running their algorithm on it and then manually assessing whether the segmentation was "Perfect", "Satisfactory" or "Failed". Later work rates web page blocks according to their importance (Song et al. 2004).

(Baluja 2006) focuses on the application of optimizing existing web pages for mobile phones by, first, dividing the web page into a 3x3-grid. The user can then interactively arrange for the website to be optimized for mobile phone screen. The page segmentation algorithm is based on clues from the DOM combined with a number of computer vision algorithms. Specifically, they use an entropy measurement to construct a decision tree that determines how to segment the page. They test their approach on a number of popular websites where they achieve good results in most cases (they rarely cut through coherent texts). One artificial limitation of their approach is that it divides the page into at most 9 segments, but it seems possible to adapt it to other grid sizes.

In (Akpinar and Yesilada 2012), the authors improve upon the popular VIPS algorithm. They focus on improving the first phase of VIPS, the visual block extraction. They observe that the original algorithm has certain deficiencies due to its age (it is from 2003) and the evolving nature of the Web. They address these issues by dividing all HTML tags (including the ones introduced by HTML 5) not into three classes but into nine instead, and define new separation rules for these classes based on visual cues and tag properties of the nodes. Unfortunately they do not give an empirical evaluation of their updated algorithm.

We found only one paper that, like us, is focused on comparing existing approaches to web page segmentation and labeling: (Yesilada 2011). The author answers the five W's (Who, What, Where, When and Why) for about 80 papers. The classification of the approaches is largely qualitative including bottom-up vs. top-down, DOM-based vs. visual, how the evaluation of the approaches is measured (precision and recall, success rate, or execution time), and whether specific heuristics are made based on assumptions about the layout of web pages. The paper also lists the assumptions and limitations of the different algorithms.

7. Conclusion and Future Work

As to our first research question, *how well do existing web page segmentation algorithms work on modern websites,* we conclude that performance in general has gotten worse over time. While all three older algorithms, BlockFusion, PageSegmenter and VIPS, showed a strong performance in their original publications, this does not hold any more on our dataset using recent web pages. As our study shows, the main reason for this is the increasing complexity of websites and their ever more dynamic behavior due to the increasing prevalence of DOM manipulations via Javascript.

Regarding the second research question, whether the existing approaches can be improved, we showed that this is indeed possible by combining two of three approaches in the WebTerrain algorithm, which consistently had the highest F-scores in our benchmarks.

The systematic exploration and testing of the different algorithms was enabled by the testing framework we developed for our research. It allows to exchange datasets and algorithms and is also easily extensible with more page segmentation algorithms. It thus forms a solid basis for future work in this field. Researchers have already contacted us to use our datasets for their own validation. Promising-looking directions are more sophisticated combinations of different approaches and more directed segmentation algorithms that e.g. only focus on certain segments on a page or that target only specific domains of websites.

References

Python webkit DOM bindings. http://www.gnu.org/software/pythonwebkit/, 2013. URL `http://www.gnu.org/software/pythonwebkit/`.

E. Akpinar and Y. Yesilada. Vision based page segmentation: Extended and improved algorithm. http://cng.ncc.metu.edu.tr/content/emine.php, Jan. 2012. URL `http://cng.ncc.metu.edu.tr/content/emine.php`.

S. 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*, page 33–42, 2006.

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 5th Asia-Pacific web conference on Web technologies and applications*, page 406–417, 2003a.

D. Cai, S. Yu, J. R. Wen, and W. Y. Ma. VIPS: a visionbased page segmentation algorithm. Technical report, Microsoft Technical Report, MSR-TR-2003-79, 2003b. URL `ftp://ftp.research.microsoft.com/pub/tr/TR-2003-79.pdf`.

D. Chakrabarti, R. Kumar, and K. Punera. A graph-theoretic approach to webpage segmentation. In *Proceeding of the 17th international conference on World Wide Web*, page 377–386, 2008.

Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/. Yahoo! research: "Web spam collections". http://barcelona.research.yahoo.net/webspam/datasets//datasets/uk2007/. URL `http://barcelona.research.yahoo.net/webspam/datasets//datasets/uk2007/`.

R. Guha and R. McCool. TAP: a semantic web test-bed. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):81–87, Dec. 2003. ISSN 1570-8268. doi:10.1016/j.websem.2003.07.004.

J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. 1997. URL `http://ilpubs.stanford.edu:8090/250/`.

C. Kohlschütter and W. Nejdl. A densitometric approach to web page segmentation. In *Proceeding of the 17th ACM conference on Information and knowledge management*, page 1173–1182, 2008.

C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, page 441–450, New York, NY, USA, 2010. ACM.

M. Kovacevic, M. Diligenti, M. Gori, and V. Milutinovic. Recognition of common areas in a web page using visual information: a possible application in a page classification. In *IEEE International Conference on Data Mining*, pages 250 – 257, 2002. doi:10.1109/ICDM.2002.1183910.

R. Kreuzer. A quantitative comparison of semantic web page segmentation algorithms (msc thesis), 2013. `http://www.cs.uu.nl/wiki/Hage/SupervisedMScTheses`.

R. Kreuzer, M. El-Lari, R. van Nuenen, and S. Hospes. Random and popular datasets for validating web page segmentation algorithms, 2013. `https://github.com/rkrzr/dataset-random`,`https://github.com/rkrzr/dataset-popular`.

T. Popela. Implementation of algorithm for visual web page segmentation (msc thesis), 2012. URL `http://www.fit.vutbr.cz/study/DP/DP.php?id=14163&file=t`.

R. Song, H. Liu, J. R. Wen, and W. Y. Ma. Learning block importance models for web pages. In *Proceedings of the 13th international conference on World Wide Web*, page 203–211, 2004.

S. Vadrevu and E. Velipasaoglu. Identifying primary content from web pages and its application to web search ranking. In *Proceedings of the 20th international conference companion on World wide web*, page 135–136, 2011.

S. Vadrevu, F. Gelgi, and H. Davulcu. Semantic partitioning of web pages. In A. Ngu, M. Kitsuregawa, E. Neuhold, J.-Y. Chung, and Q. Sheng, editors, *Web Information Systems Engineering – WISE 2005*, volume 3806 of *Lecture Notes in Computer Science*, pages 107–118. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-30017-5.

Y. Yesilada. Web page segmentation: A review. 2011. URL `http://wel-eprints.cs.manchester.ac.uk/148/`.