Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Improve Content Extraction in Web Pages for Browser Reader modes

Jan Berg

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Prof. Dr. Marco Aiello
Commenced:	March 16, 2020
Completed:	September 16, 2020

Abstract

Web content extraction is the process of extracting specific information on websites with the help of an algorithm. It is used for a variety of different applications. Search engines use it to find the relevant information on a website to help index the website. Browser read modes improve the user experience by only showing the main content of the website to the user and removing all the noise like advertisements and navigational elements. The problem with main content extraction is that there is no perfect solution to it. Algorithms try to guess the important content of a website and not always succeed with that. The most used main content extraction algorithms today work by analyzing the underlying HTML structure of the website based on hand tuned heuristics such as word count and the used HTML tags. They do not consider other aspects such as position and size of elements. In this work we try to improve the accuracy of main content extraction algorithms currently used with the help of visual features such as position and size of elements. To evaluate the results we implemented two versions of a main content extraction algorithm as a plugin for the Chromium web browser. The first version only used heuristics based on features from the website that can be read directly from the HTML source file. The second algorithm additionally takes the styling of the website into account which requires parsing the HTML and CSS files of the website. Based on our measurements the visual based algorithm had a higher accuracy than the normal algorithm (80,1% instead of 73,2%).

Contents

List of Figures

List of Listings

Acronyms

ARIA Accessible Rich Internet Applications. 28, 31, 39

CSS Cascading Style Sheets. 3, 7, 15, 16, 24, 25, 26, 28, 39

DOM Document Object Model. 18, 22, 31, 39

HTML Hypertext Markup Language. 3, 7, 15, 16, 17, 18, 19, 21, 22, 24, 25, 26, 27, 28, 31, 36, 37, 39

NCE News Content Extractor. 21

1 Introduction

In the last decade the web became an integral part of our everyday life. It contains a lot of information which is displayed in various forms, be it through a news article describing a recent event or a forum where people share information about specific topics [1]. These websites all have different designs and can contain content the user does not want to see. This includes navigational elements or advertisements. Browser vendors implemented a mode called Browser read mode to help keep the focus on the content that matters to the user and avoid distractions on the website.

Read modes in browsers work by analyzing the content of the page, find the block which contains the interesting content and format it consistently to show it to the user. The difficult part about that is to find the useful content on the page. The process of finding this information is called content extraction. An algorithm looks over the website and returns all blocks of information which are defined as the main content of the website. In an article website the main content is the article. The algorithm tries to find the article and at the same time filter out unnecessary information like the pages navigation or advertisements. Content extraction algorithms are not perfect, because the definition of main content can vary between different people and the way websites can be structured.

Most of the research in content extraction focuses only on HTML structure without taking the styling into account. Lin et al. (2002) use TABLE tags to distinguish content [2]. Laber et al. (2009) focuses on performance for content extraction on news pages which relies on semantic HTML tags [3]. Gupta et al. (2003) looks at the DOM instead of the plain HTML, so styling information is parsed, but the paper focuses on removing unnecessary content and keeping the appearance similar to the original web pages, instead of basing the heuristics of content removal on the styling [4]. It works fairly good with articles and news pages, but has its problems on other pages. Because of the way HTML and CSS works it is possible for every element on the page to look like something completely different so it is not enough to rely on the semantic HTML elements to distinguish content. Another important aspect to consider are advertisements. Website developer usually have an interest in making their page accessible to more people and try to properly use semantic HTML elements. Even if they are not paying attention to that, they are not actively trying to make the HTML unreadable. Advertisements on the other hand try to mask their presence in the page and make it difficult for automated tools to remove them.

In this work, we propose another strategy to improve the overall accuracy of content extraction in websites by also using the styling information like size and position of elements on the page and compare the result with existing methods in terms of accuracy and efficiency in the context of browser read modes.

2 Background

2.1 Structure of Websites

Websites are described with a markup language called Hypertext Markup Language (HTML). HTML files consist of various HTML tags which can directly show content on the page, e.g. the tags `<img>` (image), `<input>` (text input field) or `<button>` (button). Other tags can also be used to wrap text or other elements such as `<p>` (paragraph), `<div>` (container) or `<span>` (inline container). Wrapping tags can include text or any number of other tags which provides information about all the sub elements inside it. To close a wrapping tag it is necessary to include a "/" before the tag name `<p>content</p>`.

A simple website can be seen in Listing 2.1. It has `<!DOCTYPE html>` at the beginning to tell the browser that this file is a HTML file and should be rendered accordingly. After that comes the `<html>` tag defines the container for all HTML elements. The `<body>` tag defines the main content of the page which is visible to the user. Inside the body starts the content of the page. The `<h1>` tag displays a heading. Heading tags are numbered based on the importance of the heading with the numbers 1 to 6 where 1 is the most important heading and 1 the least important. The `<p>` tag defines a paragraph and adds space around it so if multiple paragraphs are directly after each other they display a bit of space to distinguish them.

HTML tags can also contain attributes to provide additional information about them. An example for that is the link tag `<a>`. To define where the link should take the user on click, it requires a `href` attribute: `<a href="https://www.google.com">click me!</a>`.

To describe the presentation of the website a style sheet language called Cascading Style Sheets (CSS) is used. With CSS it is possible to change the appearance of HTML elements, e.g. the color, position, size or font used. Separating the content (HTML) from the presentation (CSS) makes it easier to reuse and control the appearance for different devices.

```
<!DOCTYPE html>
<html>
<body>

<h1>Headline</h1>
<p>Hello World</p>

</body>
</html>
```

Listing 2.1: Simple HTML Page

```
<!DOCTYPE html>
<html>
<head>
<style>
.blue {background: blue;}
p     {background: red;}
</style>
</head>
<body>

<div class="blue">I am blue</div>
<p>I am red</p>

</body>
</html>
```

Listing 2.2: Website with simple styling

CSS files contain a list of styling rules. Each rule defines the appearance of the elements like color and size. To know which element is affected by the rules, a selector is used. It can be as simple as the tag name or a complex expression which takes the parent and child elements into account. A simple example of CSS rules can be seen in Listing 2.2.

There are two containers defined inside the body. The first container has the class `blue` assigned to it. The class `blue` is also defined inside the `<style>` tag with a dot before the name. This dot is a selector and defines that this rule is only used for tags which have the class `blue` assigned to it. The second rule with the selector `p` doesn't have a dot, so it means it is applied to the tag name and not the class name. In this case all `<p>` tags have this rule applied to them.

If multiple rules are applied to the same element, the rule with the more specific selector has a higher priority and therefore overrides the other styles. It is also possible to add the `!important` annotation behind the declaration to always force this attribute over the others: `.blue {background: blue !important;}`

Another aspect of CSS is the inheritance. If a rule matches an element, it is not only applied to that element, but also to every child of it. Inherited styles have a low priority and are only used if the child element has no rules applied to itself.

With CSS it is possible to make every element look like something different. A web developer could change the font size of a `<div>` container and make it appear as if it was a headline or even a button. This is a problem for automated tools such as search engines, browser read modes or screen readers. To avoid building everything from container tags that have nothing to do with the actual content, HTML5 introduced new semantic HTML elements that not only function as a container, but also give a hint on what that content inside it is, see Figure 2.1 for an example. This example could also be build with only `<div>` tags, which would make it difficult for automated tools to find the main content, in this case the article. Semantic elements are not mandatory and should not be relied on too much.

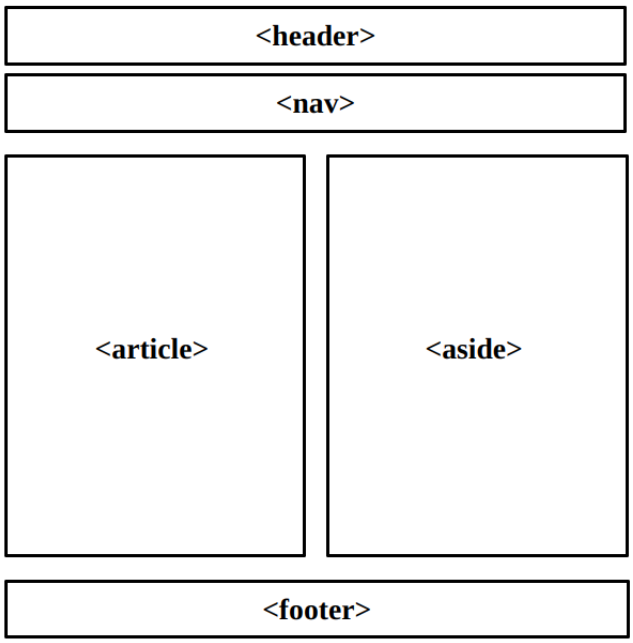

Figure 2.1: Semantic HTML elements

2.2 Content extraction methods

There are various ways to extract content from a website. For example extracting all images from a website is as easy as finding all links in the HTML that point to a file with a known image file format like PNG or JPG. If the goal is to extract an article or other text and filtering out the unimportant information like advertisements or navigational elements it can get difficult.

2.2.1 Extraction from a known website

The easiest solution to extracting any content from a website is if the structure of the website is known beforehand. In this case the algorithm always finds the desired content by looking at the predefined position in the HTML file. This form of content extraction is often used for web scraping to gather data to make a comparison portal where prices from different websites are compared or to get specific information that only this website provides. An example for that can be seen in Figure 2.2. This is a website which shows data for all football players. If the user wants to know the contract expiration date he can track down where this information is located in the HTML file and use it in an automated program.

The benefit of this approach is that it is really easy to implement and as long as the website doesn't change, the relevant information will always be there. The downside is that this only works for a specific website and cannot be extended to work on every other website. An advanced method for this are content extraction algorithms that are optimized to work on multiple websites that share a common structure like news websites. They have the advantage of knowing the general structure of the website but at the same time work on various different websites instead of only one.

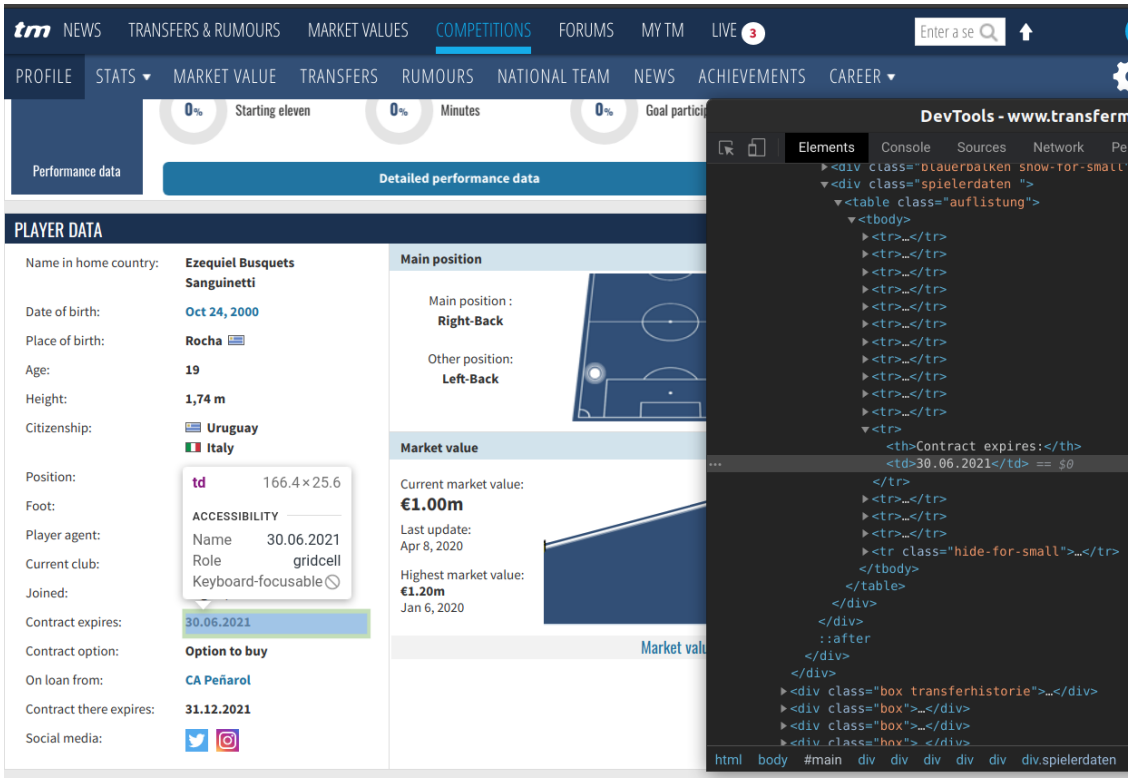

Figure 2.2: transfermarkt.com - Details of a football player

Laber et al. used this approach to improve the content extraction on news websites [3]. The algorithm only works on news websites and extracts the title and body of the news article without comments or images. The algorithm has 3 phases.

- Phase 1: In the first phase the algorithm searches for a block that probably contains all the main content. It achieves this by taking into account the link density, number of paragraph tags and general text. If no block satisfies the conditions, the root is returned.
- Phase 2: The second phase consists of removing comments. Comments usually have a date, name and other repeatable patterns associated with them, so the algorithm searches for the beginning of these repeatable patterns and removes them.
- Phase 3: In the last phase the algorithm searches for the title. This is achieved by a set of heuristics: the title comes before the main content, is usually also the title of the page and does not contain many words.

2.2.2 Extraction based on DOM

Another approach to extract main content from a website is by analyzing the Document Object Model (DOM). The DOM is a language independent representation of XML or HTML as a tree structure. Every node represents a part of the original file and can contain child nodes [5]. A representation of a DOM tree can be seen in Figure 2.3

This approach requires the HTML file to be parsed before analyzing the content, but it helps to analyze the structure of the website.

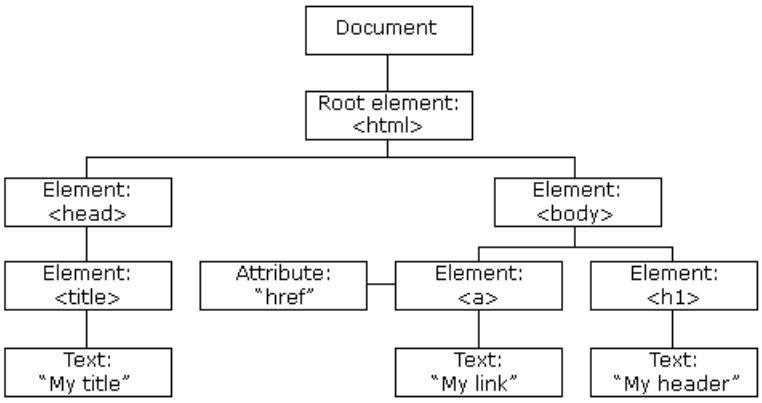

Figure 2.3: HTML DOM Tree

2.2.3 Visual content extraction

It is also possible to extracting content based on the visual representation of the website. This gives the algorithm a similar representation how a person viewing this website would have. The website first needs to be rendered. After that the algorithm can access the information of elements such as position, color and size. This helps to understand where the important content blocks are. In combination with the other approaches the visual based extraction can provide additional information for the algorithm to work with.

3 Related Work

3.1 Extracting from News Webpages

Laber et al. introduced a fast way to extract the main content on news web pages [3] called News Content Extractor (NCE). They define the main content as the title and the body section which contains the text, but excludes comments and other visual elements like images. To verify the results they used a training dataset of 324 news web pages from 22 sites. The algorithm achieved 90.7% accuracy across the training set which is comparable to other algorithms using the same dataset. One advantage comes from the fact, that it only relies on the same model for all pages, comparable algorithms rely on multiple different models for different sites [6]. Another point they tried to achieve was performance. Because of that the algorithm only relies on the HTML layout and ignores all styling information to not have to parse the web page before analyzing it.

The proposed algorithm has a good identification rate, but only solves the problem for news article websites which cannot be applied to the rest of the web.

3.2 DOM based extraction

Gupta et al. [4] proposed a solution to content extraction which relies on the DOM tree instead of the raw HTML file. The algorithm goes through the tree and applies filters to it. A simple one that removes specific HTML tags such as images and links. After that the algorithm tries to remove advertisements by comparing the hyperlinks to that of a list of known advertisement providers. This is similar to how most ad blockers operate. When a web page gets loaded, the ad blocker checks all connections of the browser. If one of those connections is a known advertisement provider, the ad blocker blocks the browser from accessing the link. One of such lists used for ad blocking is EasyList [7]. They also included other heuristics to remove lists which mostly contain links and tables without substantial information. Substantial information is defined as a list of HTML tags selected by the user. Examples include buttons, input fields, styles or scripts.

Instead of trying to find the main content directly, the algorithm removes everything that is considered not useful for the user. The algorithm parses the DOM tree, but completely ignores the styling and visual layout of the webpage. It also relies on tables for content extraction. In the early days of the web tables were used to define the layout of web pages, but with the advancements in the web standards they became unnecessary and usually not used anymore for layouts, only for their original purpose, displaying tables [8].

3.3 Tag Ratios

Extracting main content through tag ratios is another approach first described by Weninger et al [9]. Their solution is called Content Extraction via Tag Ratios (CETR). This solution does not depend on specific HTML Elements, but instead focuses on the ratio between HTML tags and non HTML tag characters on a line by line basis.

First the algorithm removes all script and style tags because it only depends on HTML tags and the text inside them. The second step is to calculate tag ratios for all lines which then can be represented in a histogram. A smoothing algorithm is applied to the histogram in order to also catch shorter lines of the main content like the news article title. This model is further extended by using a 2 dimensional histogram. The 2 dimensional model is an ordered sequence of values instead of an unordered set of values.

The benefits of this approach is that it doesn't depend on hand tuned heuristics which are based on specific HTML tags. Another benefit is the use of a variable γ to define the threshold for the main content. For example by setting $\gamma = 0$ the algorithm would return every line. This value can be adjusted to make sure to not miss part of the main content but also reduce the precision of filtering out noisy content.

3.4 Visual based content extraction

Cai et al. used the visual representation of websites for their content extraction algorithm [10]. Their algorithm only depends on the visual representation and is independent of the underlying structure such as HTML. The solution achieved comparable results to other algorithms.

3.5 Summary

This chapter showed various different approaches to content extraction. Section 3.1 showed an extraction method that only works for news webpages. It uses the knowledge that the page is a news page to optimize the heuristics. Section 3.2 introduced an algorithm that works on the DOM tree in order to remove noisy content from the page. Section 3.3 showed an approach by using the ratio between HTML tags and text within a line in the HTML document to decide where the main content is.

Comparing these approaches shows that everyone has another definition of the main content. Laber et al. defines the main content as the title and the body of the article without images or comments [3]. Gupta et al. defines the main content as everything that is not an advertisement or a list of links [4].

A similarity between most of these approaches is that the execution time is generally really fast, because they do not need to parse the complete web page before applying the algorithm to it. This has the drawback that the styling information of the web page is not used for the main content extraction.

4 Methodology

4.1 Main Content

As seen in Chapter 3 the definition of the main content varies between works. In this chapter we try to define what the main content is in our case.

Most websites have more content than just the article or some text. They also usually contain a header or a side-nav which allows navigation trough the website and shows other related content, a Footer with legal information such as the privacy policy or the website owner and many other things like different blocks with advertisements.

There are other websites where the main content doesn't consist of text. The main content in a photo sharing website are the pictures. Other websites might allow the user to download certain programs. In that case the actual program is the main content and not the text which contains meta information about the program to download.

The main content also depends on the user viewing the website. Every user has his own view of what information is relevant and what not. An example for that are news articles which allow other users to commentate below them. Some users might consider the discussion in the comments also part of the main content, others don't want to see it.

For our definition of the main content we have to take the context into account. We try to improve the content extraction especially for browser read modes. That means the content we focus on is mostly text based and is not a video portal or a gallery of images. For the definition of the content and later the evaluation of the algorithm we categorized the content into three groups.

1. Main Content. The content of the website which is the most important one for the majority of the user and should be included in the output of the algorithm.
2. Optional Content. This content could be considered part of the main content because it is useful for some users, but is usually not the main focus of the web page. An Example for that is the comment section of news websites. It doesn't matter for the evaluation if the algorithm outputs this content or not.
3. Noisy Content. This contains all the other blocks which are not the main focus of the website. This includes advertisements, navigation links, privacy policy and other content which is not directly related to the main content. It is considered an error if the algorithm outputs this content.

4.2 Datasets

To evaluate the results of our implementation it is necessary to have a dataset of various different web pages.

4.2.1 L3S-GN1 Dataset

The L3S-GN1 dataset was created by Kohlschütter et al. [11] for their main content extraction algorithm. The dataset is a collection of news web pages where the main content is manually labeled. It contains a total of 612 different web pages which was randomly sampled from a larger set of 254.000 articles. The dataset stores two versions of the web page, the original HTML file and the human assessed one. The annotations are in the form of `<span>` tags with specific CSS classes which indicate the type of the following content.

Although this dataset is quite large and has detailed annotations, we cannot use it for our tests. Not only is the dataset limited to news articles, it also doesn't contain any CSS which makes it possible to guess the exact layout of the website by only accessing the HTML file. In this case the visual based approach would have no additional information and would return the same result.

4.2.2 Custom Dataset

Because of the mentioned problems with the other dataset we need to create our own to fit the requirements. The dataset needs to have styling information for every web page available additionally to the HTML. Also, the web pages HTML should be annotated to show the Main Content and the Optional Content. Everything else is considered noisy content.

The annotations are defined similar to the L3S-GN1 Dataset as CSS classes. This makes it easy to visualize them. The CSS classes start with the prefix `qwe-` to avoid possible naming conflicts with existing classes on the web page.

- `qwe-main` defines the Main Content of the page
- `qwe-optional` defines the Optional Content of the page

An example of how such annotations look like can be seen in Figure 4.1. This is an example from `stackexchange.com`, a questions and answers website. The solid outline indicates the main content. The dashed outline shows the optional content. The most important content on the page is the question and the answers to that. The comments on the other hand are annotated as optional content because they are usually not the most relevant information on the website.

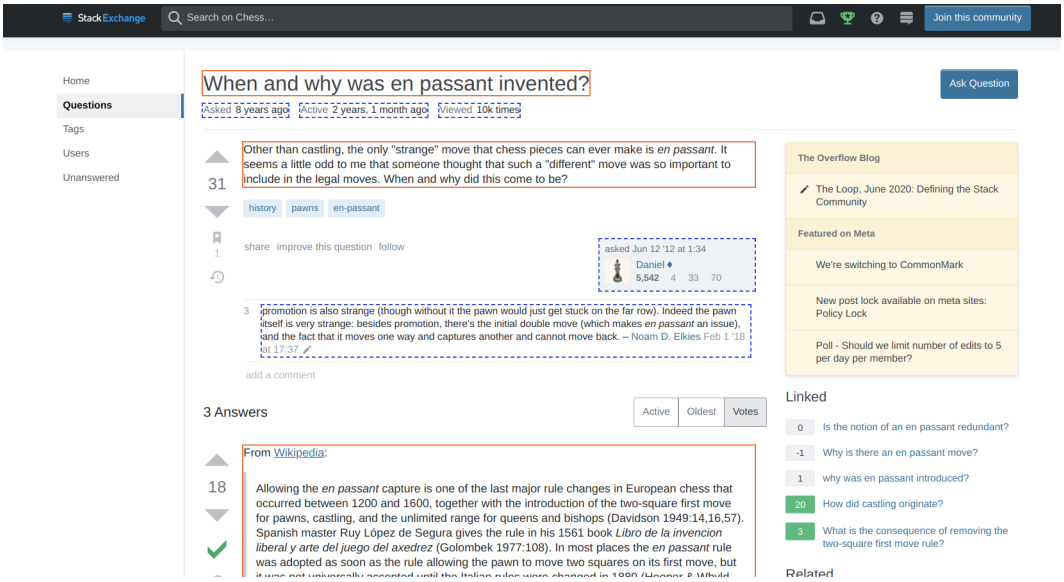

Figure 4.1: Example of dataset annotations. Solid outline indicates main content. Dashed outline indicates optional content

The custom dataset consists of 21 websites from the List of most popular websites from Wikipedia.org [12] and alexa.com [13]. Some websites were not included in the dataset for various reasons:

- They require the user to register before being able to see the content
- The main content consist mostly of images and/or videos, this makes it not useful for browser read modes

All websites from the dataset contain their HTML and CSS files. Images and JavaScript files were removed to reduce the size of the dataset and decrease browser load times. More details about the dataset can be seen in Figure 4.2.

Website title	content blocks	contains ads	size in MiB
A Poem for Peter_ A Lyrical Illustrated...	35	yes	0.30
Arzt über Covid-19-Station_ Wir muessen...	13	yes	3.23
Belarus protesters defy crackdown to...	25	yes	4.87
Booking.com_ Hotels in Hamburg...	13	no	2.68
Corona-Maßnahmen_ Zurück zur Normalität...	36	yes	2.90
Corona-Protest und Rechtsextreme_ Keine...	33	yes	0.53
'Deeply reckless'_ Critics slam leaked...	27	no	1.34
Der Sport-Tag am Sonntag, 30. August 2020...	25	yes	1.40
Filmmaker Michael Moore warns of 2016...	28	yes	1.98
Flexikum _ Deutsche Telekom	18	no	1.13
Google News - Coronavirus - Letzte Meldungen	24	no	2.21
Here is what you need to know about...	20	no	0.40
history - When and why was en passant...	24	no	0.78
How I Write and Learn – The Writing...	27	no	0.36
jon-stewart-interview	29	no	0.19
Move 3D obj File with XYZ co-ordinates...	26	no	0.64
Serie A_ Trotz geplatztem Milan-Deal...	31	yes	3.63
telekom.com Competitive Analysis...	40	yes	2.96
The Heart of the Uprising in Minnea...	35	yes	1.71
Wetter Berlin _ wetter.com...	33	yes	3.59
Lionel Messi - Barcelona; The dilemma...	24	yes	0.64

Figure 4.2: Details of the custom dataset which shows the title of the website, number of main content blocks it contains, if the website shows advertisements and the size of the HTML and CSS files combined in Mebibyte (1 mebybyte = 1024 kibibytes = 1.048.576 bytes)

5 Design

To compare the results of main content extraction with and without visual guidelines we need to have two versions of it. It is not easy to use existing solutions because of the different views on how the main content is defined, but we can still apply the same ideas to our solution.

The general approach we take is to look at every Element in the web page and assign it a score to how likely it is part of the main content. For this we use several weighted heuristics to form the final score, this is also called a static evaluation function which is often used to evaluate the boards in chess programs [14]. The evaluation function looks like this:

$$w_1 f_1 + w_2 f_2 + ... + w_n f_n$$

where w_i are weights and f_i are features.

Example for features include the word density, HTML tag used or font size. The weight indicates the importance of the feature and how much it contributes to the overall score. This makes it easy to adjust and fine tune the weights after implementation is done to achieve the best possible accuracy.

After a score is assigned to every element the algorithm adds all elements where the score is greater than a predefined threshold to a list. This list then contains all elements that the algorithm thinks are part of the main content. In our test we added a border around the elements to make it visible on the website. An example for another use case is making a browser read mode. In this case the algorithm would remove every element on the website which is not part of the list and display the remaining elements with the same font.

The first solution has no access to visual information like the position of the elements, font colors or size. It solely depends on the HTML structure and it's contents. This is the usual approach most main content extraction algorithms take. The advantages of it are that it doesn't require any parsing of the original HTML file which makes the algorithm faster in general.

For the second solution we also add visual based information to the heuristics and try to improve the overall accuracy that way. This approach has the advantage of getting more information to use for the heuristics but at the same time also increasing the complexity of the implementation. This approach needs access to a fully featured HTML parser in order to work.

5.1 First solution: plain HTML

For the first solution we focus only on information available within the HTML structure. No visuals are taken into account.

Word count. One important feature of determining the main content is to take the word count of the current element into account. Large bodies of text are a good indication of main content.

HTML tags. HTML tags can also help identify the content inside of them. The `<p>` tag indicates a paragraph which is mostly used for blocks of text. Another example is the `<h1>` tag which displays a headline and is usually only used once for the general headline of the main content. There are also semantic HTML tags which behave identically to general block tags, but have a different name to provide a meaning for automated tools like screen readers, search engines or read modes. Semantic HTML tags like `<section>` and `<article>` hint at the main content, other tags like `<footer>` and `<nav>` are clearly not part of the main content.

ARIA attributes. Accessible Rich Internet Applications (ARIA) are attributes that were included into the HTML 5 standard to provide a way to make the web more accessible to people with disabilities. They help systems like screen readers to identify sections of the website and provide useful information to the user. These attributes also help automated algorithms extract the main content like in our case. The ARIA attributes are similar to the semantic HTML elements by providing the information, but they don't require an extra HTML tag. They are defined as the role attribute on HTML tags: `<div role="main">`

It is also possible to combine these features for better results. HTML tags on their own are sometimes not enough. It makes also sense to look at the parent element or the other siblings before deciding whether the element is part of the main content or not. If a headline is surrounded by navigational elements it is probably not part of the main content, but part of the navigation on the website.

There are more properties on HTML tags that could be taken into account. For example the class property which is a list of CSS classes or the id, which is a unique identifier across the website. If the class or id contains the name "footer", it might be an indication that this element is part of the footer and not the main body. The problem with this is that both these properties are not limited to a subset of valid names, but instead can contain anything. This makes them difficult to properly analyze.

5.2 Second solution: visually based

By taking the styling information of the website into account we can also add heuristics based on the appearance of the elements. The visual heuristics are added on top of the already existing ones.

Position and size. The position and size of each element can indicate their relevance on the website. Centered elements are probably more relevant than the ones at the side. Often times it is not as simple as that, websites can have different layouts and look completely different to each other. One approach is to group the different elements into visually distinct blocks similar to Figure 5.1. This makes it possible to score each element based on their parent container position and how it relates to other blocks on the page.

Font Using the font size and color can also help in identifying the main content. Websites often use different font sizes and colors to emphasize certain aspects of the page. Smaller text or text with less contrast is more likely to be additional content instead of being part of the main content. The font style can be another indicator. Font changes usually only occur if the context changes.

There are more information that can be considered like the background color or animations, but they can get very difficult to properly evaluate because of the difference in all websites.

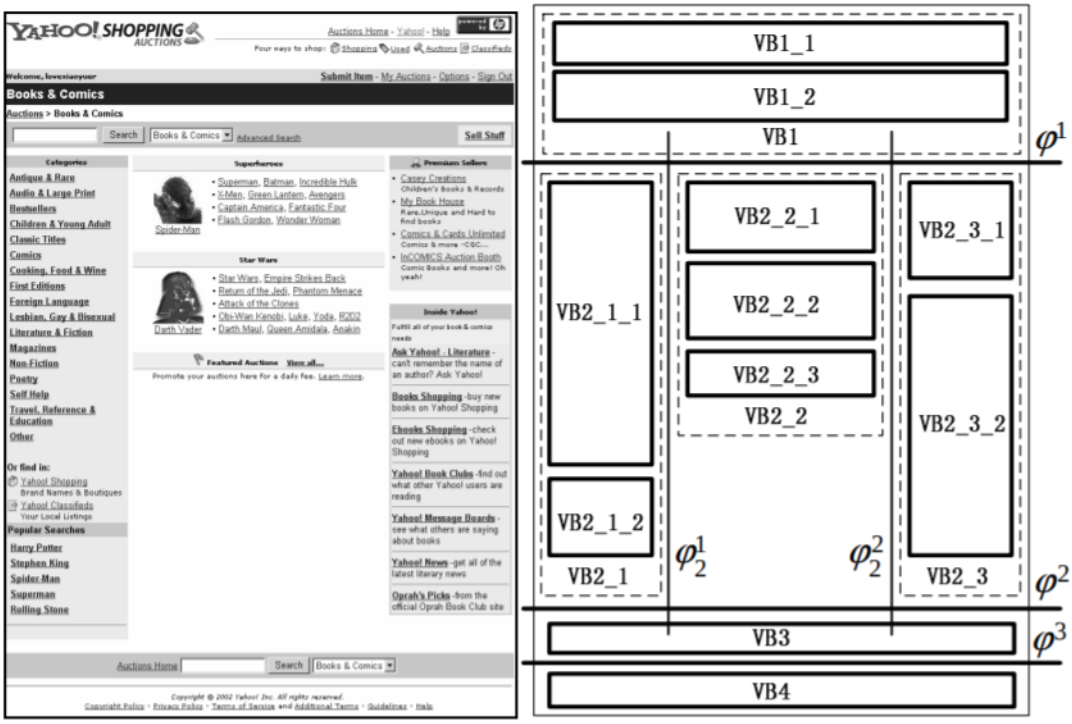

Figure 5.1: Vision based content blocks [10]

6 Implementation

For the implementation of the two solutions we need to read the content of the page and output the relevant main content. Because the visually based approach requires the website to be parsed before having access to all the visual information we decided to implement the solution as a plugin for the existing Chromium Browser. This avoids the need to write a custom HTML render engine and also allows us to edit the website which helps to visualize the result.

The first step of the algorithm is to pre-process the document. The heuristics are based on the DOM, but not everything there is relevant to us. The algorithm only takes the `<body>` tag into account and also removes `<script>` and `<style>` tags from the DOM because they are not visible to the user and therefore cannot be part of the main content. It also replaces the content of some HTML tags to make it easier for the algorithm to evaluate it. An example for that is the `<code>` tag which is used to show a code snippet. If the website adds different colors to the words similar to how code editors show them, they have to add a lot of different HTML tags to achieve this. The pre-process step in the algorithm removes them and replaces the content with just the text.

After the pre-processing is done the algorithm finds all text nodes within the body with at least 2 characters. This helps to remove single letter nodes which are usually only symbols like arrows. In HTML text nodes are different from HTML tags. They have fewer properties, cannot have styles applied to them and don't have a position or any other relevant information attached to them. This is why we don't want to use the text node directly, but instead the HTML tag which contains the text node.

Now that the elements are prepared the evaluation function can start. First is the evaluation without visual information. The algorithm uses the heuristics described in Chapter 5.1. The biggest influence to the final evaluation score comes from the ARIA attributes and semantic HTML tags because they are specifically designed for automated tools to understand the HTML. If the role property with the value `main` is present it greatly reduces the search space because the developer explicitly tells with this property that this element contains the main content of the website. Other evaluation features include a score based on the HTML tag used and the number of words in a given element.

After the score is calculated a second evaluation step is executed. The evaluation function takes the neighbors of the element into account and adds part of their score to it. This increases the evaluation score for elements which have a low score but are surrounded with elements that have a high score. It also helps in grouping elements because all of the main content is usually in the same block and not spread over the entire website.

Features that indicate the element is part of the main content increase the evaluation score and features that indicate the opposite decrease the score. This means that with a perfect heuristic every element with a score bigger than 0 would be considered part of the main content. An example of the evaluation scores can be seen in Figure 6.1.

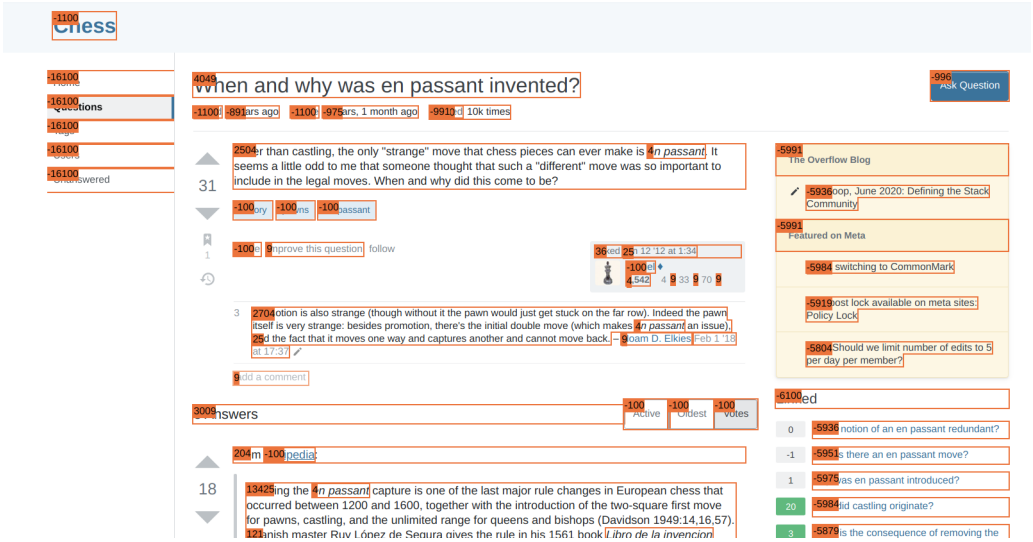

Figure 6.1: Annotation of a website with the basic heuristic evaluation scores

Elements on the sides have scores between -5000 and -16000 which strongly indicate that they are not part of the main content. The headline and body of the question have scores over 2000 with some smaller elements also having a negative score. If we now remove every element that doesn't have a score bigger than 10 the result is the main content of the website, see Figure 6.2.

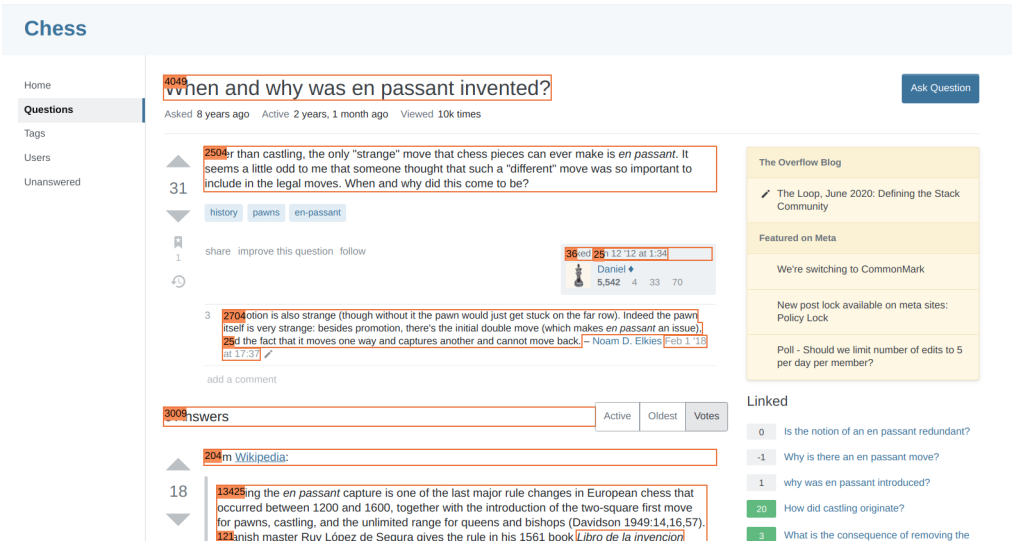

Figure 6.2: Annotation of a website with the basic heuristic evaluation scores

The best threshold of the evaluation score can vary between websites. In general, we found the optimal values to be between 10 and 200.

6.1 Visually based solution

The visually based solution reuses the same base heuristic features, but adds a few new ones on top of them. One of the new features is using the position and size of each element to determine the boundary of the main content. The algorithm calculates the size and position of every element and uses this information together with the individual score from the base heuristic to determine the boundary of main content. Elements are weighted according to their evaluation score. The algorithm takes the minimum and maximum x position of all the elements and sorts them in a list. The top and bottom 10% are removed from the list to avoid outliers. Finally, the algorithm chooses the minimum and maximum position from the remaining elements in the list and uses these positions as the boundary for the main content. Every element outside of the boundary gets a penalty to the evaluation score. An example can be seen in Figure 6.3. The green lines show the boundary where the algorithm calculated the main content to be. The blue blocks with the numbers show the visual based evaluation score. Elements outside of the boundary get a penalty of -1000 to their score.

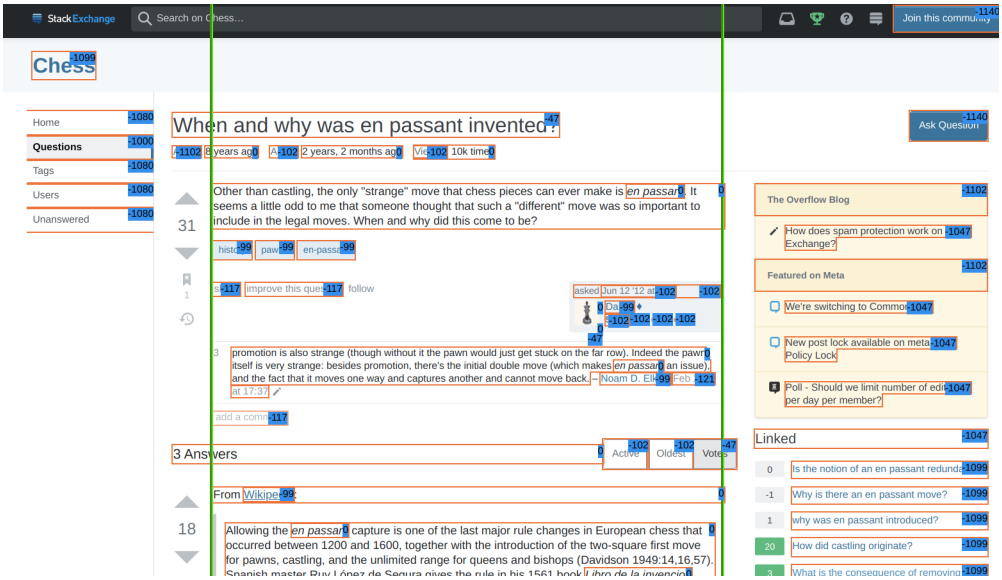

Figure 6.3: Visual based evaluation scores and main content boundary

There are also some smaller features that add to the evaluation like the contrast of the text. The algorithm first goes through every text node on the websites and determines the contrast between the text and the background. The luminance is used to better simulate how the human eye perceives light [15] [16]. After the calculation is done the most common contrast ratio is set as the default for the website. Every element that now has a lower contrast ratio than the default will get a penalty to the evaluation score.

7 Evaluation

To evaluate the different algorithms we use the dataset which was defined in Chapter 4.2. For every website in the dataset we run both algorithms and compare the annotated main content blocks with the actual result of the algorithm. The number of successfully found main content blocks is divided by the total number of main content blocks. For every wrongly annotated block a penalty is added to the score.

This gives us the following formula:

$$accuracy = \frac{s - p}{n}$$

where `s` is the number of successfully found main content blocks, `p` is the penalty for wrong annotations (number of errors) and `n` is the total number of main content blocks.

The average accuracy for the normal algorithm is **73,2%** and **80,1%** for the visual based algorithm. A detailed overview can be seen in Figure 7.1.

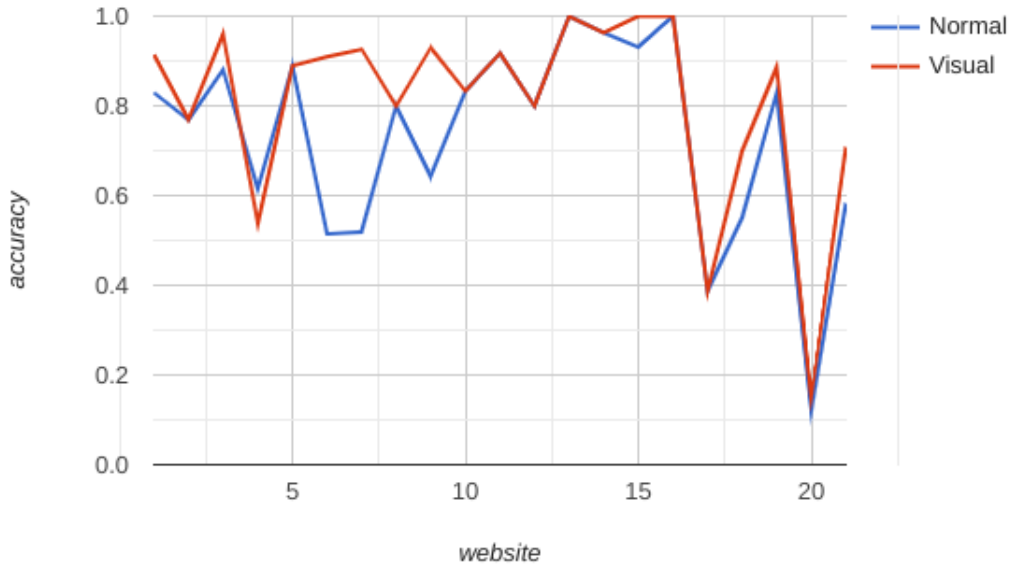

Figure 7.1: normal and visual based algorithm accuracy per website

The figure shows the accuracy for every individual website in the dataset. About half of the websites had no or very little difference in accuracy. Taking a closer look at the websites shows that all of them employed some sort of semantic HTML in their source code which made the visual based information redundant.

In the other half the visual based algorithm provided slightly better results with a few big improvements. The big improvements are on websites that show advertisements on the side and the main content in the middle. The visual based algorithm could determine the boundaries of the main content whereas the normal algorithm could not.

Overall the visual based algorithm performed better in all but one case, where the algorithm determined the main content boundaries wrong and identified some advertisements as main content.

Figure 7.2 shows the total number of wrongly annotated content blocks per website. In this case an

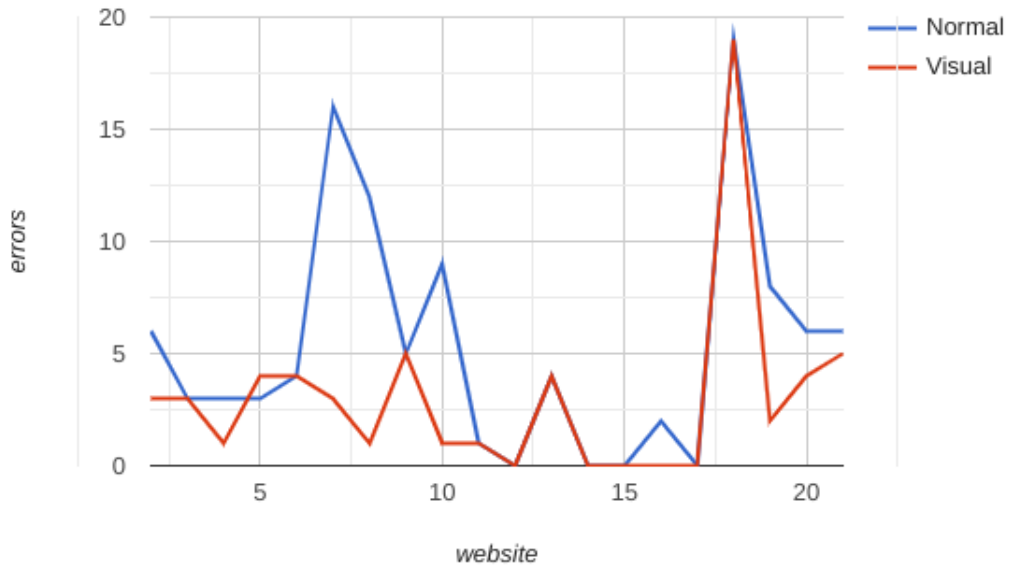

Figure 7.2: normal and visual based algorithm amount of errors per website

error is defined as labeling noisy content as the main content. For example if the algorithm returns a block that contains an advertisement it is considered an error, but not including part of the main content as the output is not considered an error here.

The visual based algorithm made fewer mistakes and has an overall consistent error rate. Looking closer at the tested websites shows that most of the errors from the normal algorithm are on elements where the score is close to the threshold which defines what elements are considered main content or noisy content. The visual based algorithm increased the confidence in these elements (adding or subtracting a large amount of the score) which helped reduce the errors.

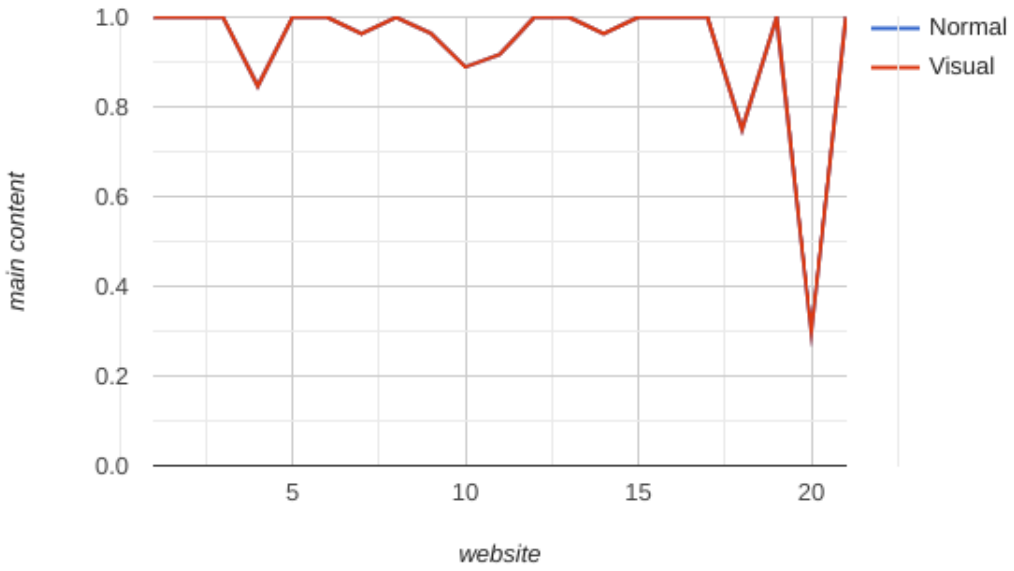

Figure 7.3: normal and visual based algorithm main content identification rate

Figure 7.3 shows the main content identification rate without taking the errors into account. Both algorithms performed the same, including the main content most of the time. They both also had problems identifying the title or headline of the website. Adjusting the scoring for HTML headline tags for the normal algorithm or the scoring for text size with the visual based algorithm increased the error rate more than they helped find the title. Advertisements often show bigger text than the rest of the website which made it difficult to find the best parameters.

8 Conclusion and Outlook

Overall the visual based algorithm provided a slightly better accuracy across the dataset (80,1% instead of 73,2%). There is no difference on websites which make extensive use of semantic HTML elements and ARIA attributes. On other websites the visual based approach provided the necessary information to eliminate some errors. Most of these errors had a small score difference to the defined threshold, so it might be possible to improve the heuristics in the normal algorithm to avoid those errors. On websites with advertisements placed on the sides the visual based algorithm performed the best. It was able to identify the advertisements and effectively remove them from the output. In general the visual based algorithm performed better and it is only a question if the additional cost of parsing the HTML and CSS is worth the effort. The normal algorithm only needs the DOM tree whereas the visual based algorithm also needs to parse the CSS of the website. This process is fast on modern browsers and is barely noticeable by the end user. The bigger difference is with the additional network bandwidth needed to also receive the CSS file. According to the archive the file size for CSS is about twice as large as the HTML file size at around 234 kilobytes [17]. This is not a large file and most internet connections should have no problems downloading this file without the user noticing any extra delays, but if the connection is slow this could add a noticeable delay. Another problem with the visual based algorithm is the extra parsing of the CSS file required. In our tests on a modern desktop computer with the most recent version of the Chromium web browser (85.0.4183.83) the algorithm usually took between 15ms and 25ms to run. In comparison, the parsing of the website took between 250ms and 530ms. It is also important to keep in mind that the algorithm was not optimized for performance. This is reasonably fast for a single website, but other tools such as web scraper from search engines need to parse millions or billions of websites for indexing purposes. For such tools this additional processing time can become very expensive. For personal use the visual based algorithm should be the preferred one provided the internet connection is reasonably fast. Other applications which use a lot of websites need to consider the extra cost of parsing CSS and evaluate if it is worth the increased accuracy or not.

Outlook

The developed algorithms could be further improved by tuning the specified weights for every heuristic. The best way to do this is to have a large dataset of annotated websites and a way to automatically evaluate the results with different parameters. Other improvements that could be made are adding website specific heuristics to the algorithm so it has a better understanding of the content and can more confidentially find the main content. This could help with the problems in identifying advertisements at the bottom after the main content where both algorithms had some problems.

The heuristics used in this work were quite simple and were kept at a more general level to use the for a broad range of different websites. More specialized heuristics can be used to increase the accuracy. For example instead of only counting the words, the algorithm could analyze the content of the text and make a prediction with this additional information. This would make the algorithm language specific but would provide more data for the final scoring.

The visual based algorithm could also be modified to change the weights of the heuristics based on the layout found on the website. Different layouts can indicate a certain type of website which would make it easier to find the main content and filter out noisy content.

Another approach to this problem is to use machine learning instead of heuristics. Main content extraction is a difficult problem without a clear answer. Machine learning could help in identifying the main content similar to how people do it. Machine learning requires a large number of training data which can take a lot of time to make because of the manual annotation required.

Bibliography

[1] M. Aiello, "The web was done by amateurs", in *The Web Was Done by Amateurs*, Springer, 2018 (cit. on p. 13).

[2] S.-H. Lin, J.-M. Ho, "Discovering informative content blocks from web documents", in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002 (cit. on p. 13).

[3] E. S. Laber, C. P. de Souza, I. V. Jabour, E. C. F. de Amorim, E. T. Cardoso, R. P. Rentería, L. C. Tinoco, C. D. Valentim, "A fast and simple method for extracting relevant content from news webpages", in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009 (cit. on pp. 13, 18, 21, 22).

[4] S. Gupta, G. Kaiser, D. Neistadt, P. Grimm, "Dom-based content extraction of html documents", in *Proceedings of the 12th international conference on World Wide Web*, 2003 (cit. on pp. 13, 21, 22).

[5] (). Dom standard, [Online]. Available: https://dom.spec.whatwg.org/#what (cit. on p. 18).

[6] D. d. C. Reis, P. B. Golgher, A. S. Silva, A. Laender, "Automatic web news extraction using tree edit distance", in *Proceedings of the 13th international conference on World Wide Web*, 2004 (cit. on p. 21).

[7] (). Easylist - overview, [Online]. Available: https://easylist.to/ (cit. on p. 21).

[8] E. Weinstock-Herman. (). The history of html table layouts, [Online]. Available: http://www.tiernok.com/posts/history-of-html-table-layouts.html (cit. on p. 21).

[9] T. Weninger, W. H. Hsu, J. Han, "Cetr: Content extraction via tag ratios", in *Proceedings of the 19th international conference on World wide web*, 2010 (cit. on p. 22).

[10] D. Cai, S. Yu, J.-R. Wen, W.-Y. Ma, "Extracting content structure for web pages based on visual representation", in *Asia-Pacific Web Conference*, Springer, 2003 (cit. on pp. 22, 29).

[11] C. Kohlschütter, P. Fankhauser, W. Nejdl, "Boilerplate detection using shallow text features", in *Proceedings of the third ACM international conference on Web search and data mining*, 2010 (cit. on p. 24).

[12] (). List of most popular websites - wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/List_of_most_popular_websites (cit. on p. 25).

[13] (). Alexa - top sites in germany - alexa, [Online]. Available: https://www.alexa.com/topsites/countries/DE (cit. on p. 25).

[14] (). Stockfish evaluation guide, [Online]. Available: https://hxim.github.io/Stockfish-Evaluation-Guide/ (cit. on p. 27).

[15] M. Anderson, R. Motta, S. Chandrasekar, M. Stokes, "Proposal for a standard default color space for the internet—srgb", in *Color and imaging conference*, Society for Imaging Science and Technology, vol. 1996, 1996 (cit. on p. 33).

[16] (). Web content accessibility guidelines (wcag) 2.0, [Online]. Available: https://www.w3.org/TR/2008/REC-WCAG20-20081211/#contrast-ratiodef (cit. on p. 33).

[17] (). Page weight | 2019 | the web almanac by http archive, [Online]. Available: https://almanac.httparchive.org/en/2019/page-weight (cit. on p. 39).

[18] (). Javascript html dom, [Online]. Available: https://www.w3schools.com/js/js_htmldom.asp.

[19] T. Gottron, "Evaluating content extraction on html documents", in *Proceedings of the 2nd International Conference on Internet Technologies and Applications*, Citeseer, 2007.

All links were last followed on August 31, 2020.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature