Bachelor thesis
Computer Science

Radboud University

Comparing Website-layouts using Edit Tree Distance

Author:
Spaendonck, P.H.M. van
S4343123
`p.h.m.spaendonck@student.ru.nl`

First supervisor/assessor:
Prof. dr. ir., Arjen P. de Vries
`A.deVries@cs.ru.nl`

Second assessor:
title, name
`e-mail adress`

June 6, 2016

Abstract

The rapidly increasing size of the internet demands an automated solution for retrieving data about websites. While we are already able to analyze websites on their contents, an automated solution for analyzing and comparing layouts does not yet exist.
Thus we've designed a program that translates website layouts into data trees, and uses those to calculate the edit-tree-distance between them, so that we are able to do layout-analysis.

Contents

Chapter 1

Introduction

The internet is a growing source of revenue and information. Being able to analyze this collection of data is very useful, currently most website analysis is done by looking at the content of a website, but not at the website layout itself. However being able to analyze and compare website layouts might be more worthwhile then we think. We could for example remove redundant backups of websites by looking if their layout has changed or use clustering to group up different websites with similar designs.
To be able to use modern analysis tools, we have to be able to quantify distance between two elements. If we're able to calculate the distance between different websites we can see which websites are similar and which ones differ the most. A simple solution would be retrieving metrics about the websites design, for example most used color, however this solution can be very subjective as its outcome is entirely dependent on which metrics where chosen. Thus we have to look at a more objective and more mathematical solution. Unfortunately, when quantifying distance over large and complex data elements, it becomes more difficult to find a purely objective metric. Because, while we'll usually still be able to say whether two websites differ, being able to say what website is more different becomes more of a subjective thing.
Because of this, we will be looking at adapting an already existing technique instead of coming up and testing a new one. In our research we will be using the edit tree distance. A technique that can be used to calculate distance between two data trees.
To this we will have to find a way of translating websites into data trees, so that we can then use our edit-tree-distance to analyze our data.

Chapter 2

Preliminaries

Before we start diving deeper into our research, it might be useful to explain certain aspects of the information-domains we're using. A websites layout is defined by div's. A div is basically a box in which content (like text or an image) can be displayed. A div can also contain another div which again can contain more content. This structure is very similar to a tree; every node (div) of the tree might have one or more leaves (pieces of content) attached to it or might have another node attached to itself. For this research we will convert websites into data-trees so we can calculate the edit distance between them.
The elements of the websites we'll be studying, have style properties assigned to them, either using a style-sheet or using inline attributes. These rules explain how they should be displayed, and this is what we'll be using to calculate the attributes we want to be stored.

Chapter 3

method

The first part of the final product is able to change websites into data. While it currently does not take every css attribute in consideration when calculating the corresponding values, it does for most of them, and shows it is possible to turn website layouts into data elements.
The other part of the product is able to calculate distances between the generated elements, and is also able to generate averages between data elements, which allows us to use analysis tools such as clustering.
The total product shows us that it is also possible to analyze website on their layout alone. And while it could certainly be improved on certain aspects, as will be explained later in this chapter, it does give a working step-by-step process of converting website-layouts into data-elements and analyzing these.
It is worth noting that, during the development process, we've used the top 25 most-visited sites in the US to test our software. These websites have been chosen, because not only are they widely used, they also use varying design principles. For example, google.com uses mostly inline css declaration, while live.com uses almost no inlinecss and refers to an external domain for style declarations.
However during the development-process, we've left out three websites: Twitter.com, Reddit.com and Tumblr.com. These were left out because they were to big to be handled by our software. However if the html-parser were to be made more efficient, then these websites would also be able to be parsed.

3.1 design

As explained above, the created piece of software is combination of two pieces.

3.1.1 the translator

The first piece is the part of the program that is responsible for turning websites into data elements. This is done by opening a virtual browser window using the Ghost.py library for python. The window that we've used for this research was $1920x1080$, however different resolutions can be used when needed.
Once the window is generated a website can be loaded in. On every website we use two pieces of JavaScript code. The first, "pre.js" is used to make XMLHttpRequests to load in style-sheets from external domains, and turning them into style-nodes. This is done because it is no longer possible to use data from a different domain than the domain at which the JavaScript is executed.
When all external style-sheets have been converted "core.js", the second piece of JavaScript, is executed. First all the internal style-sheet rules are assigned to their corresponding elements, by making them inline. This is done so that we don't have to look through all the css-rules every-time we want to look up one of the style properties of an element. Once we've done this, we start recursively calculating the width and height of every element, and their placement relative to their parent element.
Thus for every element we keep the following information:

- the elements width
- the elements height
- the elements horizontal offset, relative to its parent
- the elements vertical offset, relative to its parent
- the children (and their values) belonging to this element

We also have an extra slot we use to save information that we can use for debugging. This value is however never used later.
Once calculated, the elements and their values are saved to our local database, so that they can be used (by the second piece of our program) for analysis.

3.1.2 the analyzer

The second part of the program can then be used to analyze the new data elements using our own adaption of the edit-tree distance metric. [1] When we look at an element in our tree, we can simplyfy it to having two properties, the four values (width, height, vertical offset and horizontal offset) and its possible children. The way we calculate the distance between two branches

[1]The adaption of the Edit-Tree-Distance algorithm can be found in the "LooseEdit-Comparer.java" class.

is by looking at these two properties. We can project the four values to two vectors, and then calculate the rational distance between the original vector and the vector of the other branch, this gives us the cost of substituting the original element with the other one.
Then we calculate the distance between all the children of our element and all of the children of the other element. We use the shortest distance found, or use the cost of creating the specific child if it's smaller than the shortest distance. The distance between two branches is than the total of these two costs.
For calculating the average we do a similar thing. Since the four values are actually vectors we can calculate these averages via conventional mathematical means. For the children, we try to find the closest pairs again, and then calculate the averages of those pairs.

3.2 difficulties

During the design and creation process, we've stumbled on certain problems that are worth mentioning.
As stated before, one big problem with JavaScript is that it is impossible to retrieve data from an external domain. We solved this by using CORS to make the external data local. This is however only possible if the external domain has CORS enabled. Thus the problem could arise that it is no longer possible to correctly retrieve the css-information via the current method. Another problem is auto generated values that are created when the 'auto' keyword is used. Since the specific implementation is browser specific, the exact result depends on the browser that is used. Since we're using a custom browser, these values are not always computed as we want. For example *'margin auto'* does not center elements in our browser, while it would do so in others. Because of this we don't use the *'Window.getComputedStyle()'* method, but we calculate the values ourselves.

3.3 discussion

Hier zal ik mijn eigen werk aan de kaak stellen, en discussieren wat duidelijke verbeterpunten zijn voor het onderzoek.

Chapter 4

Related Work

Hier zal ik kijken naar vergelijkbaar onderzoek, en kijken naar mogelijk vervolg onderzoek of toepassingen van dit onderzoek.

Chapter 5

Conclusions

Our current implementation is able to create an objective representation of the websites we're using. We've shown that it is possible to retrieve information from a website, while only looking at the layout. However our current implementation is very crude and could definitely be improved on certain aspects.

Appendix A

Appendix

The written code can be found at `https://github.com/VizuTheShaman/Parsie`