Bachelor thesis
Computer Science

Radboud University

Comparing Web Page Layouts using Tree Edit 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:
Dr. Fabian Gieseke
fgieseke@cs.ru.nl

July 20, 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 web page layouts does not yet exist.
Thus we have designed a program that translates web page layouts into data trees, and uses those to calculate the tree edit distance between them, so that we are able to perform 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. Most website analysis is done by looking at the content of a website, but not at the web page layout itself. However being able to analyze and compare web page 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 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 are able to calculate the distance between different web pages we can see which web pages are similar and which ones differ the most.
A simple solution would be retrieving certain attributes about the web page's design and then using vector distance as a metric. However the outcome is entirely dependent on the attributes that where chosen, and no clear guidelines on constructing such guidelines exist. Unfortunately, when quantifying distance over large and complex data elements, it becomes more difficult to find a purely objective metric. Because, while we would still be able to say whether two web pages differ, being able to say what web page is more different becomes more of a subjective decision.
In our research we use an adapted version of the tree edit distance algorithm, while it is not known if this objective approach of measuring the distance between layouts is consistent with human judgment, we assume the original algorithm is sufficiently close to be meaningful, and we will evaluate our own adaptation.
However to do this, we have to find a way of translating web pages into data trees, so that we can then use our tree edit distance to analyze our data.

1.1 Focus of this research

Our research focuses on the following two questions:

- Is it possible to gain useful information about websites, while only analyzing the web page layout?
- Is tree edit distance a good proxy for comparing web page layouts?

By answering these, we hope to be able to show that analyzing web page layouts is possible and worth the effort.

Chapter 2

Related Work

In A Robust Algorithm for the Tree Edit Distance [3], Pawlik and Augsten have shown that the Tree Edit Distance can be computed with a time complexity of $)(n^3)$, with $n = max(|F|, |G|)$. However because we want to use a varying substitution cost, we will not be implementing this algorithm.
The volume and evolution of web page templates [2] has shown that 40 to 50% of the content on the web was template content. When we cluster our data-set with the distance algorithm, we hope that the clusters correlate with templates used.
In A segmentation method for web page analysis using shrinking and dividing [1] web page analysis is done by using shrinking and dividing on the web page. This research shows that it is possible to retrieve useful information from web pages by only looking at the representation of the web page.

2.1 Preliminary knowledge

Before we dive deeper into our research, it might be useful to explain certain aspects of the information-domains we are using.
A web page's structure 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. When the web page is loaded into a browser, the page's `div`'s and its content are structured into a Document Object Model (DOM). Since the DOM is a data tree we can easily translate it into a tree, on which we can use our edit tree distance algorithm.
The elements of the websites we will 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 will be using to calculate the attributes we want to be stored.

Chapter 3

Method

3.1 Transformation

The DOM model contains a lot of information, however a large part of this information is not needed. The DOM also does not directly contain all the information we do need. Because of this, we need to be able to transform the DOM into a data tree that contains only the information we need.
The first component is the part of the program that is responsible for transforming web pages into data elements. This is done by opening a virtual browser window using the Ghost.py library for python. The window we have use, has a resolution of $1920x1080$; while different resolutions can be used when needed, we have chosen to use this resolution, because it is one of the most used resolutions on newer computers.

3.1.1 Style sheet handling

Once the window is generated, a web page can be loaded in. On every web page 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 because the Same-origin policy does not not allow scripts to acces data from a domain different from the one it is executed at.
However it is worth noting that, this is 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.
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 do not have to look through all the css-rules every-time we want to look up one of the style properties of an element.

3.1.2 Tree representation

When all of the style attributes have been made in-line, we only have to concern ourselves with the inline style attributes of each element of the DOM tree.
For every element we want to 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 do this by recursively calculating the width and height of every element, and their placement relative to their parent 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.
When calculating these values another problem arises. Auto generated values that are created when the 'auto' keyword is used, are not always correct. Since the specific implementation of this is browser specific, the exact result depends on the browser that is used. Because we are 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 do not use the *'Window.getComputedStyle()'* method, but we calculate the values ourselves.
Once calculated, the elements and their values are saved to our local database, so that they can be used for analysis.

3.2 The analyzer

The second component of the program can then be used to analyze the new data elements using our own adaption of the tree edit distance metric.[1]
When we look at an element in our tree, we can simplify 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 is by looking at these two properties. We can project the four values to two vectors, and then calculate the normalized 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

[1]The adaption of the tree edit distance algorithm can be found in the `LooseEditComparer.java` class.

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 is smaller than the shortest distance. The distance between two branches equals 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.

function GETDISTANCE($node_0$,$node_1$)
    $subCost \leftarrow$ GETVECTORDISTANCE($node_0.vector, node_1.vector$)

    C_{large} := the set of children of the node with the most children.
    C_{small} := the set of children of the node with the least children.
    $leavesCost \leftarrow 0$
    for $\forall c_0 \in C_{large}$ **do**
        $min \leftarrow getCost(c_0)$
        for $\forall c_1 \in C_{small}$ **do**
            $min \leftarrow min(min,$ GETDISTANCE$((c_0, c_1)))$
        end for
        $leavesCost \leftarrow leavesCost + min$
    end for
return $subCost + leavesCost$
end function

function GETVECTORDISTANCE(v_0,v_1)
    $vectorDistance \leftarrow |(v_0) - (v_1)|$
    $l \leftarrow min(|v_0|, |v_1|)$
return $(vectorDistanc/l)$
end function

function GETCOST(node)
    $cost \leftarrow$ GETVECTORDISTANCE($node.vector, node.parent.vector$)
    for $\forall c \in node.children$ **do**
        $cost \leftarrow cost +$ GETCOST(c)
    end for
return cost
end function

Chapter 4

Discussion

In this chapter, we evaluate to what extent the comparison of web page layout structure using weighted tree edit distance, captures the right information.

4.1 Evaluation of used data

During the development-process, we have used the top 25 most-visited websites in the US, taken from Alexa.com, 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 inline css and refers to an external domain for style declaration.
We have left out two websites: Twitter.com and Reddit.com. Because of the infinite scrolling capabilities of these pages, they were too 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.
Our current parser, also does not handle every css-attribute, however the current parser is able to retrieve the css-attributes, thus a future version would be able to handle these. This means that while our solution is incomplete, it is still a valid proof of concept, since it still shows us that there is information inside the websites layout.
This leaves us with the following data set.

Parsed websites (the websites where parsed on the 20^{th} of May 2016)				
google.com	Facebook.com	Amazon.com	Wikipedia.org	Ebay.com
Netflix.com	Linkedin.com	Craigslist.org	Pinterest.com	Live.com
Imgur.com	Go.com	Bing.com	Chase.com	Instagram.com
Paypal.com	Tumblr.com	Diply.com	cnn.com	Espn.go.com
Msn.com				

Before we start assessing the usefulness of analyzing our data, we have to take into account that due to the size of this research we are only using a small set of data. Thus some properties and correlations might not be apparent with our current set and some might not be there once a larger set is used.

4.2 Evaluation by clustering

The second component of our program is able to do nearest-neighbor clustering using the tree edit distance as distance metric. Using this on the 23 websites we have, results in the following clusters:

- Cluster 0 Pinterest.com, Facebook.com and Paypal.com
- Cluster 1 Instagram.com, Live.com, google.com, Amazon.com, Netflix.com, Craigslist.org, Bing.com and Chase.com
- Cluster 2 cnn.com and Go.com
- Cluster 3 Msn.com , Linkedin.com , Wikipedia.org, Imgur.com, Espn.go.com, Tumblr.com and Ebay.com

If the layout structure can convey the function of a website, clustering should group similar sites together. Unfortunately the clusters do not seem to correlate with the purposes of the parsed websites; socialmedia and newssites are divided over different clusters. While only online advertisement platforms (Craigslist and Ebay) are in the same cluster.

4.3 Evaluation by similarity

An alternative analysis evaluates the result of the similarity measure per site. This is done by generating a directed graph, in which every vertex represents one of the web pages. From every vertex a directed edge is created towards their nearest neighbor. The resulting graph is able to show us which sites are generic. A generic site is a site with the a large amount of edges going towards it.
When the similarity graph of our 23 websites is generated[1], we notice that Google.com and Live.com have the 'most generic' web pages. when we take a look at the design of these two websites, we notice that these websites consist of mostly empty space and very few displayed content.
When we look at their neighbors (which are the other members of Cluster 1), we notice that except for Amazon.com, all of these share a similar grade of complexity. This is because the similarity graph has the property that

[1] a figure of this graph can be found in the Appendix

the complexity of a web pages design correlates with the amount of edges going toward its vertex.
We can count the amount of vertexes leading (either directly or indirectly) to a web page, and rank the pages according to this sum. The result is a ranking showing how generic each web page is. We hope that the highest ranking pages could serve as design templates, because they are the most generic according to our ranking.
According to this measure, Google.com (with 4 direct neighbors) and Live.com (with 5 direct neighbors) are the most generic web pages in our study. Google.com is also similar to Live.com.

4.4 The value of tree edit distance as a proxy

Our current adaptation of tree edit distance seems to be a good basis for design distance metrics for websites. Because its weighting prefers less complex websites, we can use it to rank websites on their simplicity in layout and possibly to generate templates.
However the original tree edit distance algorithm is only able to asses whether a node in the tree is equal to another node or not. Because of this we have to come up with our own way of calculating the difference between different nodes, bringing us back to our original problem, but only smaller.
And while it is possible, calculating the distance between different nodes using our algorithm can be time consuming. Thus our adaptation of the tree edit distance may need to be improved before it can be applied to a larger data set.

4.5 Future work

An unfortunate issue of trying to analyze data-sets that have not been analyzed before, is the lack of a frame of reference for measuring its validity, because of this most of the validity comes from further usage of the methods, that we have designed.
One of the most prominent things of continuing on this research will be analyzing a bigger collection of websites. Due to time constraints and the large time-complexity of our algorithm, we were only able to use a small set of websites. However, now that we are capable of transforming web pages, we should be able to use a larger set in following research.
In our current research we have not looked at human interpretation. It would be an interesting extension of our work to use surveys to test the validity of our current implementation choices.
A possible usage of the similarity graph, would be creating a directed graph using a lot more websites, one edge coming from every node and going into their respective nearest neighbor. If the bigger graph maintains the same

property, we can see how complex a website is, by counting how few edges go into the website's node.
Further research could then be conducted to whether this complexity correlates with the usability of a website.

Bibliography

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

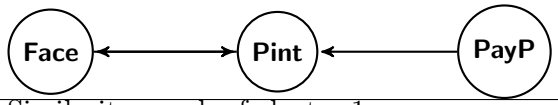
[2] David Gibson, Kunal Punera, and Andrew Tomkins. The volume and evolution of web page templates. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, WWW '05, pages 830–839, New York, NY, USA, 2005. ACM.

[3] Mateusz Pawlik and Nikolaus Augsten. Rted: A robust algorithm for the tree edit distance. *Proc. VLDB Endow.*, 5(4):334–345, December 2011.

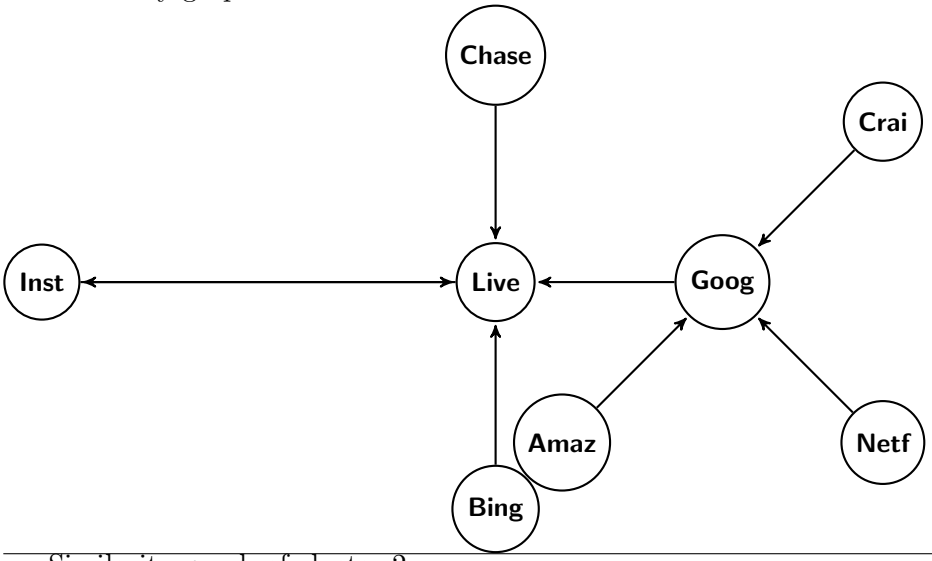
Appendix A

Appendix

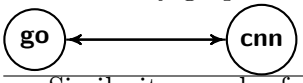
The written code can be found at `https://github.com/VizuTheShaman/Parsie`



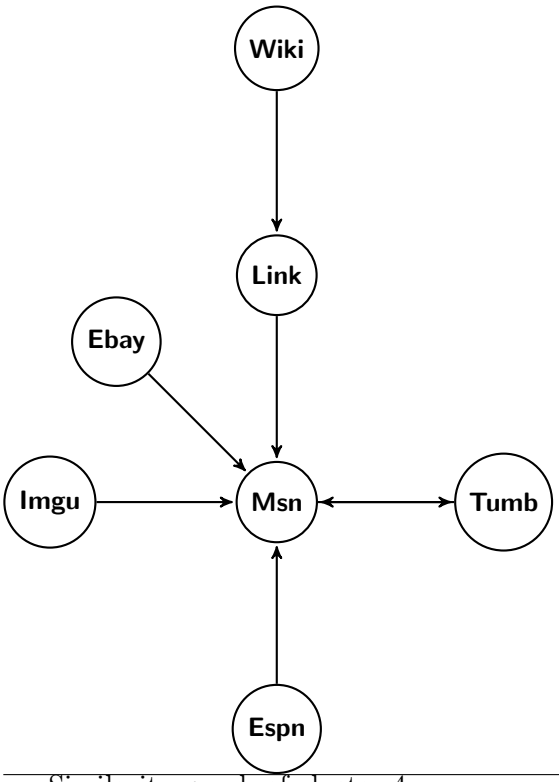
Similarity graph of cluster 1.



Similarity graph of cluster 2.



Similarity graph of cluster 3.



Similarity graph of cluster 4.