Automatic Recognition of Blog Entries

Caspar Ahrensberg

Revised Version 1.2

Kongens Lyngby 2012
IMM-B.Sc.-2012-0023

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.Sc.-2012-0023

Summary (English)

The goal of the thesis is to determine if a page can be recognized as a specific type of page based on the structure of its HTML elements. It will try to do so by using Tree Edit Distance to generate a matching structure from said pages structures which then in turn can be used to test against when an arbitrary page is presented, thus answering if the page is a Wordpress blog or not. The algorithm used is the Restricted Top Down Mapping which imposes restrictions enforcing the DocType of HTML while mapping from one tree to another. A series of test will be run on the algorithm to determine its precision when answering if a site is a blog or not.

Summary (Danish)

Målet for denne afhandling er at verificere om en hjemmeside kan blive genkendt som en bestemt type hjemmeside baseret på dennes HTML elementers struktur. Dette vil blive prøvet gjort ved at bruge Træ-Ændrings-Distance til at genere en mønster struktur fra de omtalte sider som så igen kan bruges til at bestemme om en arbitrær side er en Wordpress blog eller ej. Den specifikke algoritme er Top-Ned-Kortlægning som pålægger regler som håndhæver DocTypen af HTML mens der kortlægges fra et træ til et andet. En serie af tests vil blive kørt på algoritmen for at determinere hvor præcis den er når den besvarer om en side er en blog eller ej.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfilment of the requirements for acquiring an B.Sc. in Informatics.

The thesis deals with determining the type of a web page based on the structure of its HTML elements. In its limited scope it will focus only on Wordpress blogs.

The thesis consists of a report and a program contained in a zip-file, accessible from http://www.student.dtu.dk/~s093258/.

Lyngby, 01-Juli-2012

[illegible]

Caspar Ahrensberg

Acknowledgements

I would like to thank my advisers Jørgen Villadsen and Patrick Hagge Cording for support and Kapow Technologies for access to Henrik Pilegaard.

Contents

Chapter 1

Introduction

Information gathering on the internet is a widely spread discipline used in many different regions of the corporate world, ranging from site indexing for search engines to shopping habits for shopping sites to general trends for ad companies, the one not excluding the others. When gathering the information knowing what type of site you are visiting is a great help to determining what and where to extract data. With this as motivation this thesis will look into the possibilities of using web pages, represented by their visual structure, to determine if they resemble a certain type of site. This will be done by building a program that can be fed an amount of pages that share a common denominator, be it that they are Search Engines, File Sharing or as in this project Blogs.
Blogs are what can best be described as public journals where people can express their opinions and smaller companies can describe their progress towards projects. Being that there are no restrictions on what blogs might contain the visual layout might differ drasticly from page to page, but often this does not affect the overall structure of the page itself, as these changes lies mostly in the color scheme, the font and the pictures used. Therefore the assumption is made that the overall structure, code-wise, will be the same for most blogs. To further enhance this assumption the project will limits its scope to focus only on blogs from Wordpress.com which are built from the same template. See figure **??**[1] and **??**[2] for examples on blogs. It is clear from the two pages that there are

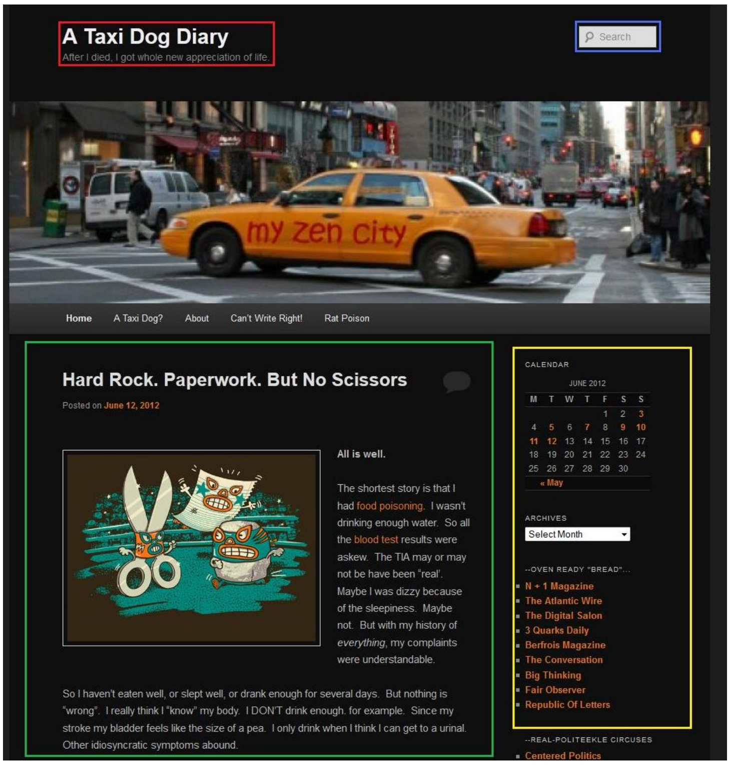
[1]Screen shot taken from `http://taxi-dog.com/`
[2]Screen shot taken from `http://dailytfarp.wordpress.com/`

some similarities, which is what the program should focus on, for example the large picture headlining both pages.

This project will be based on the algorithm developed by Reis et al. presented as pseudo code in [?] as they have in their test of the performance reached 88% positive results when using it. It should be noted that they only tested their program against pages that were of the same type as the matching structure was created from, News Pages, as they were focusing on how to extract data from those same pages. Thus the new content added by this thesis will be to determine how well the algorithm fares when presented with both pages belonging to, and outside the scope of the matching structure. The algorithm will be analysed and the overall efficiency for separating pages into different groups based on their structure will be evaluated.
Their algorithm builds on Tree Edit Distance which describes how far two trees, structurally, are from each other and which can in addition to this return a series of operations needed to transform the first tree into the second, which allows for both creation and evaluation of the matching structure through the same algorithm.

Other approaches have been made to determine and extract data from web pages. Most notable in regards to using a different approach is [?] that utilizes keyword matching on the address of the site, and on the content of the site in combination with a light structure analysis to find headlines, titles and the main content of articles. They too have a very high rate of success on extracting data from the news pages, 96%, but their keywords are the product of a partially manual analysis which means that running their results against any pages that are not news pages will result in gathering of false data.

Contrary to the previous article, if the algorithm proves to be able to separate pages of one type from another, the program will after some simple changes be able to answer what the type of page is being tested, instead of just returning if it is a blog or not. This could be done by creating a database of different matching structures, for different types of pages, which the program would then be able to run the page in question against.



A Taxi Dog Diary

After I died, I got whole new appreciation of life.

Hard Rock. Paperwork. But No Scissors

Posted on June 12, 2012

All is well.

The shortest story is that I had food poisoning. I wasn't drinking enough water. So all the blood test results were askew. The TIA may or may not be have been "real'. Maybe I was dizzy because of the sleepiness. Maybe not. But with my history of *everything*, my complaints were understandable.

So I haven't eaten well, or slept well, or drank enough for several days. But nothing is "wrong". I really think I "know" my body. I DON'T drink enough. for example. Since my stroke my bladder feels like the size of a pea. I only drink when I think I can get to a urinal. Other idiosyncratic symptoms abound.

CALENDAR

JUNE 2012

M	T	W	T	F	S	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

« May

ARCHIVES

Select Month

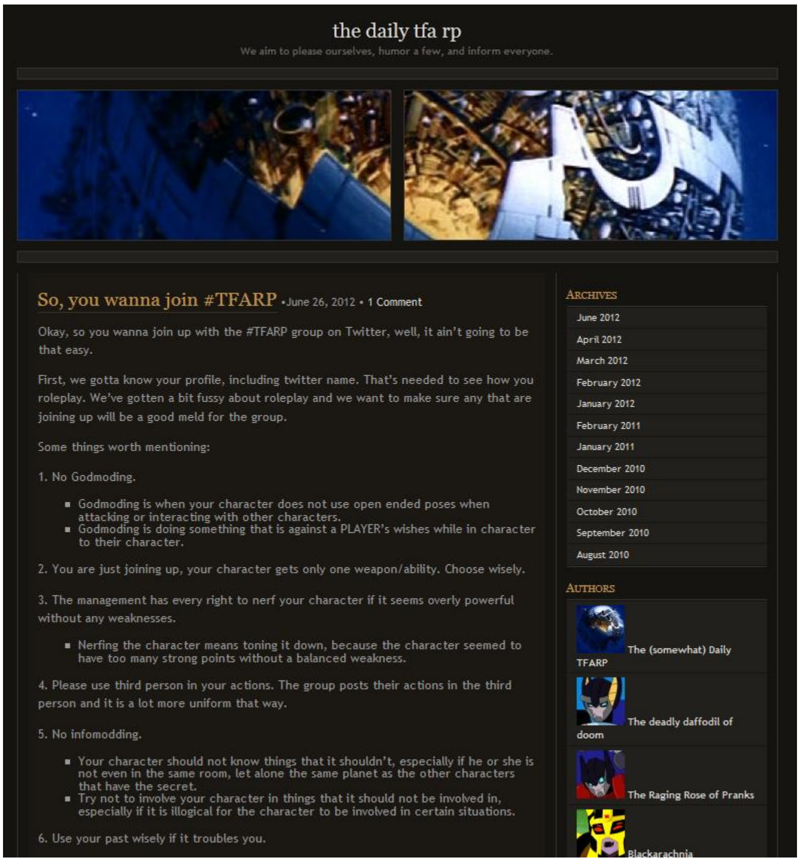
--OVEN READY "BREAD"...

- N + 1 Magazine
- The Atlantic Wire
- The Digital Salon
- 3 Quarks Daily
- Berfrois Magazine
- The Conversation
- Big Thinking
- Fair Observer
- Republic Of Letters

--REAL-POLITEEKLE CIRCUSES

- Centered Politics

Figure 1.1: Blog example



the daily tfa rp

We aim to please ourselves, humor a few, and inform everyone.

So, you wanna join #TFARP •June 26, 2012 • 1 Comment

Okay, so you wanna join up with the #TFARP group on Twitter, well, it ain't going to be that easy.

First, we gotta know your profile, including twitter name. That's needed to see how you roleplay. We've gotten a bit fussy about roleplay and we want to make sure any that are joining up will be a good meld for the group.

Some things worth mentioning:

1. No Godmoding.

- Godmoding is when your character does not use open ended poses when attacking or interacting with other characters.
- Godmoding is doing something that is against a PLAYER's wishes while in character to their character.

2. You are just joining up, your character gets only one weapon/ability. Choose wisely.

3. The management has every right to nerf your character if it seems overly powerful without any weaknesses.

- Nerfing the character means toning it down, because the character seemed to have too many strong points without a balanced weakness.

4. Please use third person in your actions. The group posts their actions in the third person and it is a lot more uniform that way.

5. No infomodding.

- Your character should not know things that it shouldn't, especially if he or she is not even in the same room, let alone the same planet as the other characters that have the secret.
- Try not to involve your character in things that it should not be involved in, especially if it is illogical for the character to be involved in certain situations.

6. Use your past wisely if it troubles you.

ARCHIVES

June 2012
April 2012
March 2012
February 2012
January 2012
February 2011
January 2011
December 2010
November 2010
October 2010
September 2010
August 2010

AUTHORS

The (somewhat) Daily TFARP
The deadly daffodil of doom
The Raging Rose of Pranks
Blackarachnia

Figure 1.2: Blog example

CHAPTER 2

Foundation

This chapter will look into the foundation of the project, by exploring the basics of the structure of Wordpress Blogs, how to convert a HTMLpage into a tree and the HTMLUnit library for Java.

2.1 Wordpress

In limiting the scope of the project the Wordpress format of blogs have been chosen. The Wordpress blogs use a simple CMS (Content Management System) to provide users with an easy way to set up their own blogs, either on the Wordpress domain (*xxx*.wordpress.com) or as a package that can be placed on own server space, providing the same benefits. This provides the project with the benefit of a similar structure to most of the blogs, which then in turn should yield better results when running the algorithm.

The frontpage of a common Wordpress blog will in most cases have the following structure, presented in figure **??**.

- *Headline*: The name of the blog, often with a subtitle.
- *Search*: A text field for searching the blog.

- *Picture*: A picture framing the top of the blog.
- *Blog post*: The front page can contain several posts that are shown in order from newest to oldest, these can either be the entire post or just teasers.
- *Add. Func.* A navigation menu with additional functions, often date-based searches for blogs will be here.

A basic post is presented in figure **??** where the same basic format with picture, headline and search function still persists, but now only the selected post is presented (*Blog post*), along with a field for posting/reading comments regarding the text (*Comments*).

This format is subject to change, but almost every blog will have a top title followed by a picture. The navigation will either be placed north, south, west or east and some might present whole blog posts on the front page while others might just give a couple of lines.

2.2 HTMLto Tree

Web pages are built in the language HTML, which is element based. This means that every element in a page is affected by the element that contains it, in HTML represented through the use of start- and end-tags. Elements that cannot contain other elements (for example images) will just be represented by a start tag. The nesting of elements can now be interpreted as a parent-child relationship, which is conveniently represented as a tree structure through the use of the Document Object Model (DOM) for the page.

In figure **??** a sample page (T1) is presented in its HTML form. It is a very basic page containing only a title, a picture and some text.
In figure **??** the tree structure, of the DOM, of T1 is displayed.

2.3 HtmlUnit

HtmlUnit is a headless browser for Java, able to process JavaScript and Ajax, in addition to filling out forms and clicking buttons. Script processing is not a must-have for this project, but it is important that sites are executed similarly when being processed, as some pages might use scripts for content control, how

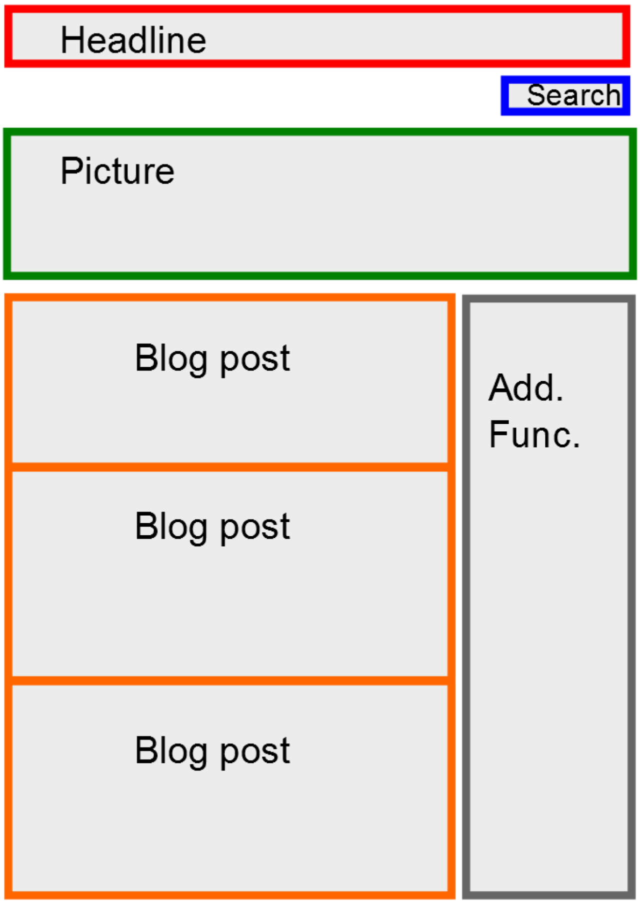


Figure 2.1: Mock-up of a blog front page

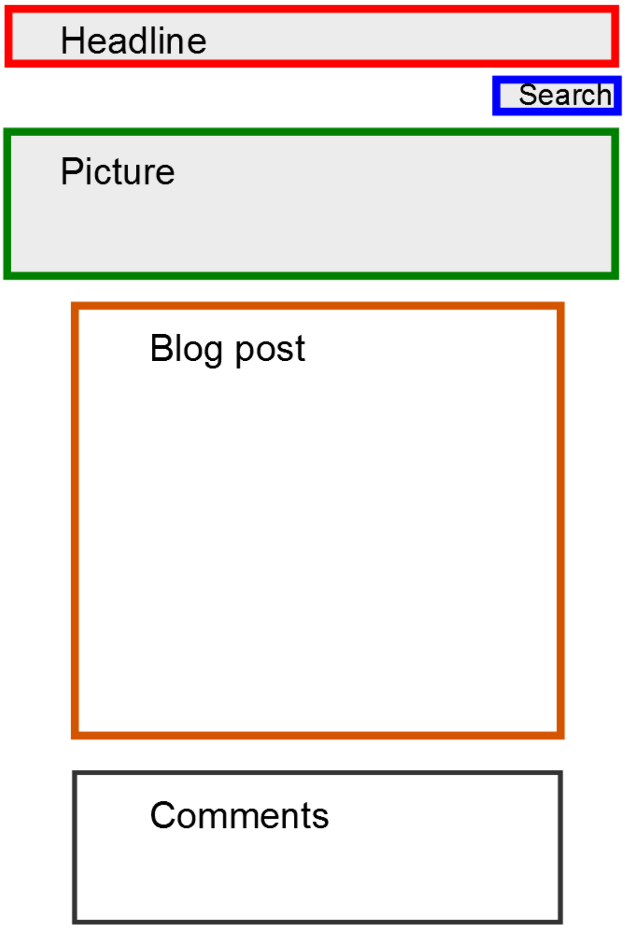


Figure 2.2: Mock-up of a blog post

```
<html>
    <head>
        <title>T1</title>
    </head>
    <body>
        <div id="t1">
            <p>
                T1
            </p>
            <img src="img-src">
        </div>
    </body>
</html>
```

Figure 2.3: Simple page (T1) presented in HTML

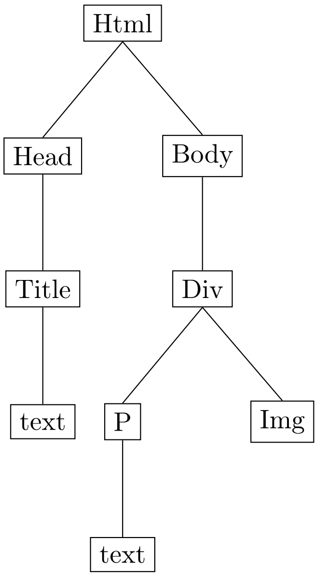


Figure 2.4: The tree structure of T1

to display the menu, auto-updating list of items, and so forth - all elements that can change the structure of the page. But most importantly, HtmlUnit can return a DOM-representation of the page it is visiting, allowing for easy access to a tree-representation of a site.
HtmlUnit is mainly used for test automation on web pages, but can also be used for web scraping or content extraction.

Using HtmlUnit is very simple. First a webclient is instantiated and then a page is called with

HtmlPage page = getPage(<address>)

This function returns an object of the type HtmlPage. This object represents the same tree as displayed in figure **??**, so to get the top node a text based search on the tags is performed, which is expressed in the following line of code,

page.getElementsByTagName("html").get(0)

The above line returns a list of all matches, but since a page should never contain more than a single instance of the HTML-tag, the first element will suffice.

HtmlUnit allows for settings to be configured on the webclient before accessing a page. Several parameters can speed up the process, for example the execution of both JavaScript and CSS is optional, both considerably speeding up the process if turned off (note that turning off CSS does not change the structure of the page, where JavaScript might). In addition, redirects, popups, timeouts and other parameters are controllable at this step.

In figure **??** an example on how to use HtmlUnit is presented. First the webclient is instantiated and configured. CSS will not be executed, scripts must complete in less than 2 seconds and redirects to other pages will happen automatically. Then a page is opened, *google.com* in this case, and the page is then traversed returning all button elements, using XPath (notice that XPath always returns a list of HtmlElements so a cast is necessary). Finally every button is clicked, in this case leading to an empty search with Google. If a search was to be conducted, the search bar would have to be located and then filled with information through the use of *setValueAttribute(<query>).*

```java
import com.gargoylesoftware.htmlunit.WebClient;
import com.gargoylesoftware.htmlunit.html.HtmlButton;
import com.gargoylesoftware.htmlunit.html.HtmlPage;

import java.io.IOException;
import java.util.List;

public class HtmlUnitExample {

    public static void main(String[] args) {
        try {
            WebClient webClient = new WebClient();
            webClient.setCssEnabled(false);
            webClient.setJavaScriptTimeout(2000);
            webClient.setRedirectEnabled(true);

            HtmlPage myPage = webClient.getPage("http://
                www.google.com");
            List<HtmlButton> buttons = (List<HtmlButton>)
                 myPage.getByXPath("//button");

            for(HtmlButton button : buttons) {
                button.click();
            }

        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

Figure 2.5: An example of how HtmlUnit can be utilized

Chapter 3

Tree Edit Distance

When determining how close two trees are to each other the return value can be expressed as a sum in this context called *distance*. This distance is based on the amount of operations it takes to transform the first tree into the second. Each operation is assigned a cost, based on the specific project, which will be added to the distance, each time said operation is "run". The distance will therefore become the total number of operations required times each operations cost.
This project will only concern itself with the three basic operations: Relabel, Delete and Insert.

- *Relabel*, is the act of changing either the type or the content of a node, into the corresponding type or content of its destined node.

- *Delete*, is the act of removing a node from a tree. If the deleted node has any children all of these will be removed via their own delete operation.

- *Insert*, is the act of inserting a node into a tree.

3.1 Notation

In the chapters to come the following notation will be used. A tree will be denoted by a T_x, where x is the trees number in a sequence. The children of the root of T_x are denoted t_y, where y is the position of the child. v_x and w_x refers to pairs of nodes where v is from T_1 and w is from T_2. $parent(v_1)$ points to the parent of v_1. A *matching tree* is the final product of combining several different trees into a generalized tree through a sequence of mappings, and is what will be used to try and determine if a page is a blog or not based on the distance. The functions *delete*, *insert* and *relabel* works the following way. $delete(c)$ returns the cost of deleting all nodes in c, where c is a list of descendants. $delete(v)$ returns the cost of deleting the node v. The same goes for *insert*. $relabel(v, w)$ returns the cost of changing the label of v to w.

3.2 Mappings

To keep track of what changes a tree must undergo to become another tree a set of pairs, called a mapping, is created. Look for example at figure **??** where two tress are shown. An arrow from one node to another means that those two nodes map to each other, if their labels do not match a *relabel* operation must be used. A node with no outgoing arrows in the first tree (T1) is to be *deleted* and a corresponding node in the second tree (T2) is to be *inserted*. To transform T1 into T2 a mapping is defined in figure **??** and a total of 3 operations are used.

The above example is small, and therefore the possibilities are equally so, but when the trees grow in size, so does the number of mappings and some of these might express combinations that are not useful. To avoid this restrictions can be placed upon the mappings, limiting which nodes are allowed to be mapped to other nodes. This project will focus on the Top-Down-Mapping and the extended Restricted-Top-Down-Mapping. But before looking at these, a few ground rules must be set for the mapping.

1. A node can never appear more than once in the mapping, that is to say that if a pair consists of (v_1, w_1) in M then there will never be a pair of (v_1, w_2) in M.

$$v_1 = v_2 \text{ iff } w_1 = w_2$$

2. Any number of children of a node will always appear in the same order as they were before the mapping, after the mapping.

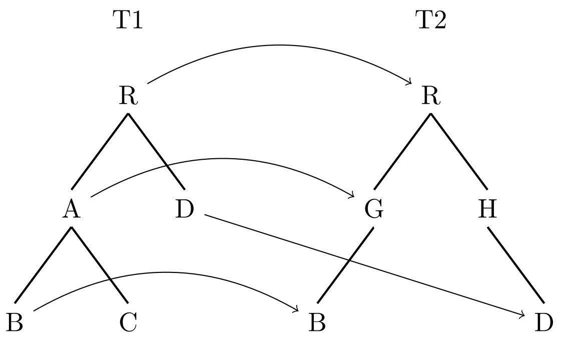


Figure 3.1: Graphic example of a mapping

Pair	Operation
(R,R)	-
(A,G)	Relabel
(B,B)	-
(C,∅)	Delete
(D,D)	-
(∅,H)	Insert

Figure 3.2: Mapping corresponding to figure **??**

$$v_1 \text{ is to the left of } v_2 \text{ iff } w_1 \text{ is to the left of } w_2$$

3. A node will never take the position of its own parent nor its own child.

$$v_1 \text{ is an ancestor of } v_2 \text{ iff } w_1 \text{ is an ancestor of } w_2$$

3.2.1 Top-Down-Mapping

In a top down mapping the following restriction is imposed on the mapping:

- If a node is in the mapping, so is its parent. This prevents nodes from being inserted between a parent and a child.

$$v = w \text{ if } parent(v) = parent(w)$$

In figure **??** and **??** the restriction of the TDM is imposed on the same two trees that was discussed in the previous section. Notice that the restriction causes

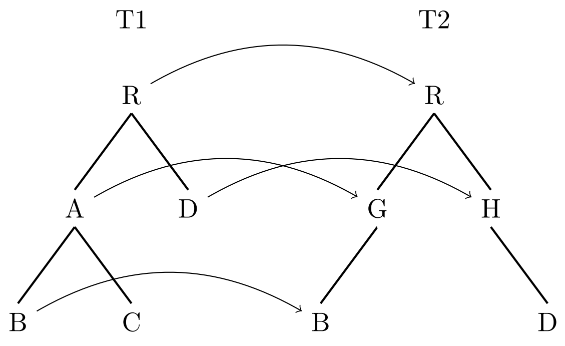


Figure 3.3: Graphic example of a top down mapping

Pair	Operation
(R,R)	-
(A,G)	Relabel
(B,B)	-
(C,∅)	Delete
(D,H)	Relabel
(∅,D)	Insert

Figure 3.4: Mapping corresponding to figure **??**

the distance to increase from 3 to 4. Note that this restriction not only limits the search space, but increases the overall cost of the mapping too, therefore mappings should not be compared in terms of cost

3.2.2 Restricted-Top-Down-Mapping

In the restricted top down mapping the following restriction is imposed:

- A pair of children cannot be in the mapping unless their parents have the same label. This means that a node that has been renamed cannot have any children in the mapping.

$$v = w \text{ iff } parent(v).val = parent(w).val$$

In figure **??** and **??** the effects of the RTDM is shown on the two trees used in the previous two sections. Again the total distance from T1 to T2 increases, this time by 2 as both B's will use an operation each on *delete* and *insert* respectively,

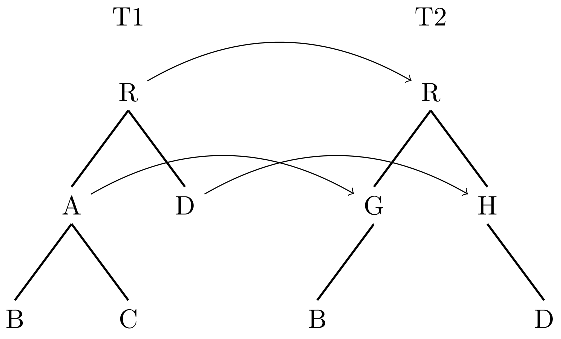


Figure 3.5: Graphic example of a restricted top down mapping

Pair	Operation
(R,R)	-
(A,G)	Relabel
(B,∅)	Delete
(C,∅)	Delete
(D,H)	Relabel
(∅,B)	Insert
(∅,D)	Insert

Figure 3.6: Mapping corresponding to figure **??**

as they both have the same parent but with different labels. The total cost in this case ends at 6.

3.3 Original

Using the mapping concept, a distance can be calculated. Various algorithms for this has been proposed, but this thesis only works with the one described in [?]. This algorithm, known as RTDM Algorithm, uses the Restricted Top Down Mapping and utilizes its more strict form to reduce the search space thus obtaining a faster computation time.

Before the algorithm is run, all equal sub trees at the same level are merged. This will be further explained in section **??**.

The algorithm this project is built on can be seen in figure **??**. The algorithm takes two trees and an error threshold as input. An array M, the size of the

number of children on each root, is then initialized holding a position for each pairing. Each position is set to the value 0.
Three different costs are now computed.

1. *del* : This cost is the sum of the previous node pair of the first tree. It assumes that the node on the second tree, it is trying to match to, is better paired with the previous node and therefore instead deletes itself and all its descendants.
2. *ins* : The theory of this cost is the same as *del* but instead of looking at the first tree it looks at the second, and inserts the node and all its descendants.
3. *sum* : This cost is the cost of mapping the two current nodes to each other. For this there are 5 cases.
   (a) The previous node pairs cost was above the threshold and therefore it does not make sense to continue down this path. The cost is set to infinite.
   (b) The two nodes and their sub trees are identical, the cost is set to 0.
   (c) The first node is a leaf. A relabel is made between the two nodes, and all descendants of the first node is deleted.
   (d) The second node is a leaf. Same procedure as the previous point, but with insertion instead.
   (e) Neither of the nodes are leafs. Thus they both have children and they can be viewed as trees, the optimal solution to their mapping is found through another call to the algorithm.

Finally the minimum of the costs are found and placed on the pairs position in the array.
The final value produced by the algorithm is the total cost of mapping Tree 1 to Tree 2.

The functions used in the algorithm is:

- $delete(t : tree)$: Computes the total cost of deleting all descendants of t.
- $insert(t : tree)$: Computes the total cost of inserting all descendants of t.
- $relabel(t_1 : node, t_2 : node)$: Computes the cost of changing the label of t_1 to the label of t_2. If they are equal this is done for free.

Algorithm 1 RTDM(T_1, T_2, e)

```
1:  $m \leftarrow T_1$ number of children
2:  $n \leftarrow T_2$ number of children
3:  $M[m, n] \leftarrow 0$ at all positions
4:  for $i = 1 \rightarrow m$ do
5:      for $j = 1 \rightarrow n$ do
6:          $t_1 \leftarrow$ child of $T_1$ at pos $i$
7:          $t_2 \leftarrow$ child of $T_2$ at pos $j$
8:          $C_i \leftarrow$ all descendants of $t_1$
9:          $C_j \leftarrow$ all descendants of $t_2$
10:         $del \leftarrow M[i-1, j] + delete(C_i)$
11:         $ins \leftarrow M[i, j-1] + insert(C_j)$
12:         if $M[i-1, j-1] > e$ then
13:             $sum \leftarrow \infty$
14:         else if $t_1$ is equal to $t_2$ then
15:             $sum \leftarrow 0$
16:         else
17:             if $t_1$ is a leaf then
18:                 $sum \leftarrow relabel(t_1, t_2)$
19:                 $sum \leftarrow sum + insert(C_j)$
20:             else if $t_2$ is a leaf then
21:                 $sum \leftarrow relabel(t_1, t_2)$
22:                 $sum \leftarrow sum + delete(C_i)$
23:             else
24:                 $sum \leftarrow$ RTDM$(t_1, t_2, e)$
25:             end if
26:         end if
27:         $M[i, j] \leftarrow min(del, ins, sum)$
28:     end for
29: end for
30: return $M[m, n]$
```

3.4 Wild Cards

In the mapping, the original algorithm use wild cards to describe the leafs of the tree. Four wild cards are used:

- **.** (*Single*): This node must appear once, and once only.
- **?** (*Option*): This node can appear once, but no more.
- $+$ (*Plus*): This node must appear atleast one time.
- ***** (*Star*): This node can appear any number of times.

Every node in a tree starts out as a *Single*. When the sub trees are compressed in the step before mapping the tree, the wild cards have been extended to apply to all nodes in the sub tree. These will have their type changed to a $+$. When nodes are inserted and deleted in the mapping, if they are part of compressed sub tree or a leaf, their types will be changed accordingly.

$$\begin{aligned} \textbf{.} &\longrightarrow \textbf{?} \\ + &\longrightarrow \textbf{*} \end{aligned}$$

The reason for allowing the wild cards to move from the trees and into the compressed sub trees is that, as will be further explained in **??**, the sub trees often contain large repeated structures (tables and/or menus) which would actually harm the matching tree more if they were required, than if they appear as possibilities which can consume those rare cases.

3.5 RTDM and the Internet

The restrictions and preparations to the RTDM is especially useful when working with trees based on the HTML structure and the internet in general. We will see that not only does the restrictions reduce the search space, they also do so while upholding and further enforcing the rules of HTML.

3.5.1 Preparation

In the preparation part all equal sub tress on the same level is merged. This step is especially useful at reducing search space on the internet as for example menus

and tables that often create a much larger search space, without providing any real value to the mapping cost can be reduced to a single sub tree with 3 or 4 levels of children as opposed to several sub trees with the same amount, or more, descendants.
An example[1] on the effectiveness of the compression can be seen from figure **??** to **??** where a total of 27 nodes are compressed down to 3, drastically reducing the search space for this sub tree path. The same counts for other similar structures: Tables filled with data, large amounts of text parted only by paragraph tags, and so on.
The compression is run bottom up, so that smaller sub trees are first compressed, which allows for the larger, higher up, sub trees to have a higher chance of being equal. For example would a table with one row with a single paragraph not match another node with two paragraphs if the comparison was made top down.

3.5.2 Top-Down-Mapping

Section **??**, restricts the mapping to use only node pairs where both parents are present in the mapping. This restriction makes sense when working with HTML, assuming the page is well formed, as HTML has a number of restrictions on what tags can be contained in what tags. For an explanation please refer to figure **??** that shows a tree of HTML dependency where each child can be contained in its parent and often also in its grand(+)parent. See the following color guide:

- *Blue*: These show a separation, that means that the following tags can be contained in the above, but their children cannot. An example of this would be the *table* tag.
- *Black*: These tags and their children can be contained in the above, and they can themselves contain text or other tags of the same kind.
- *Red*: These tags are end tags. They can be contained in all the above, but they cannot contain any tags or text themselves, think images.

The top down restriction upholds the separation of tags, preventing cases where for example image tags would be mapped directly to the HTML root tag.

[1]Menu taken from `http://sm.pf.dk/dtulan/index.php?readme`

```
[li] - Single
    [a] - Single
        Welcome - Single
[li] - Single
    [a] - Single
        FAQ - Single
[li] - Single
    [a] - Single
        Food - Single
[li] - Single
    [a] - Single
        Setup - Single
[li] - Single
    [a] - Single
        Participants - Single
[li] - Single
    [a] - Single
        Readme - Single
[li] - Single
    [a] - Single
        Schedule - Single
[li] - Single
    [a] - Single
        Login - Single
[li] - Single
    [a] - Single
        Register - Single
```

Figure 3.7: A menu of a page before compression

```
[li] - Single
    [a] - Single
        #text - Single
```

Figure 3.8: Figure **??** after a compression of the sub trees

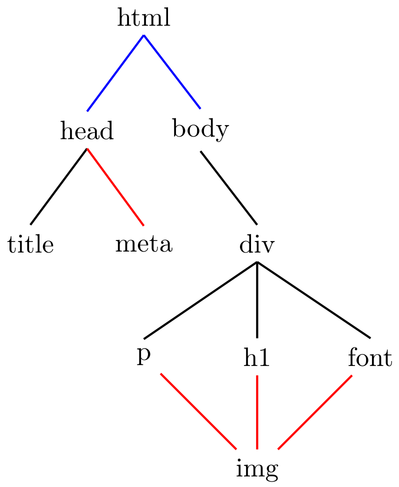


Figure 3.9: Tree depicting the dependency of HTML tags

3.5.3 Restricted-Top-Down-Mapping

Continuing with figure **??** the restricted top down mapping further helps upholding the separation of tags, by now making sure that for example children of body and head does not map to each other. The same can be said for children of tables, defining rows, that would be able to map to font changing tags.

Both the restrictions are very important when keeping a consistent mapping, as the mapping would quickly become corrupted with tags that were out of context, which then in turn would cause a very diverse mapping able to match a much broader spectrum of pages, thus defeating the purpose of itself.

3.6 Extended

In my work with the algorithm I have made some changes which improves the effectiveness of the algorithm, these I will be explaining here. The final version of the original algorithm used in the program can be seen in figure **??**, which is called eRTDM, for extended RTDM.

Before looking at the algorithm it should be noted that the label of each node does not correspond to a single tag, instead using a list of tags. This means that when the mapping is created, whenever a relabel operation is called, instead

that nodes tag is added to the list of tags for the matched node. This reduces the amount of nodes added to the tree while maintaining the same amount of possibilities, thus reducing the search space. Of course this means that a constant is added to the running time, as each time a node is compared to another node, their tags are compared to check for a common value.
Assuming that we never match a matching tree to another matching tree, only the first trees node will be able to contain more than one tag. In addition the total amount of tags in HTML is 93[2]. Of these not all of them can be at the same nodes, therefore no more than a few different tags should show up per node and the constant added to the running time in worst case is less than 100.

Now stepping into the algorithm the first noticeable change is the assignment that *sum* gets directly after *del* and *ins* have been set. A case that would not happen very often in the previous algorithm, but which proved a problem was that the algorithm would return a total value of 0 if the last two nodes being compared were equal. Now instead the sum always builds on the total cost of the previous set of nodes, thus keeping in style with the algorithm being dynamic. This also affects all previous statements using the *sum* as can be seen in all 5 cases of the calculation of it.

Next to notice is the added 6th state in the calculation of *sum* where the restriction imposed from **??** is used to short cut away from nodes that do not have the same value. Regarding the HTML aspect the algorithm now allows a table root node to be mapped to a font node, but the children of both are deleted/inserted respectively, further adding to the total cost.

3.7 Examples

To show the effects of extending the eRTDM this section will provide a very small sample on how a distance calculation looks with and without the extension.

The example will look at figure **??** and the two corresponding matrices **??** and **??**. Position R(1,1) is calculated in figure **??**. Comparing the two it is clear that they are equal except for the last position, where the original algorithm returns 0, and the extended returns 1. This is due to the fact that the right hand side of T1 and T2 are equal sub trees of the root and both are considered last since they are to the right.

Notice that the final distance calculated is different from what would have been expected, see figure **??**, as the algorithm ends with distance 1 where the number

[2] http://www.quackit.com/html/tags/

Algorithm 2 eRTDM(T_1, T_2, e)

```
1: $m \leftarrow T_1$ number of children
2: $n \leftarrow T_2$ number of children
3: $M[m, n] \leftarrow 0$ at all positions
4: for $i = 1 \rightarrow m$ do
5:     for $j = 1 \rightarrow n$ do
6:         $t_1 \leftarrow$ child of $T_1$ at pos $i$
7:         $t_2 \leftarrow$ child of $T_2$ at pos $j$
8:         $C_i \leftarrow$ all descendants of $t_1$
9:         $C_j \leftarrow$ all descendants of $t_2$
10:        $del \leftarrow M[i-1, j] + delete(C_i)$
11:        $ins \leftarrow M[i, j-1] + insert(C_j)$
12:        $sum \leftarrow M[i-1][j-1]$
13:        if $M[i-1, j-1] > e$ then
14:            $sum \leftarrow \infty$
15:        else if $t_1$ is equal to $t_2$ then
16:            $sum \leftarrow sum + 0$
17:        else
18:            if $t_1$ is a leaf then
19:                $sum \leftarrow sum + relabel(t_1, t_2)$
20:                $sum \leftarrow sum + insert(C_j)$
21:            else if $t_2$ is a leaf then
22:                $sum \leftarrow sum + relabel(t_1, t_2)$
23:                $sum \leftarrow sum + delete(C_i)$
24:            else if $t_1.val$ equals $t_2.val$ then
25:                $sum \leftarrow sum+$ eRTDM($t_1, t_2, e$)
26:            else
27:                $sum \leftarrow sum + relabel(t_1, t_2)$
28:                $sum \leftarrow sum + delete(C_i)$
29:                $sum \leftarrow sum + insert(C_j)$
30:            end if
31:        end if
32:        $M[i, j] \leftarrow min(del, ins, sum)$
33:    end for
34: end for
35: return $M[m, n]$
```

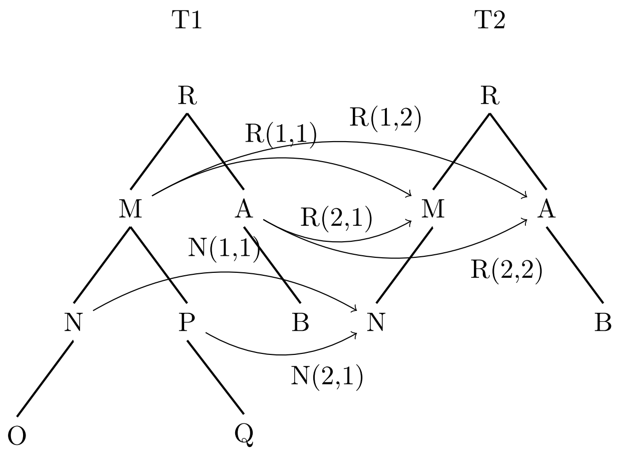


Figure 3.10: The RTDM matrix comparison shown with arrows. The positions are (i,j)

R	i=0	i=1	i=2
j=0	0	0	0
j=1	0	1	2
j=2	0	1	0

Figure 3.11: Matrix corresponding to the distance calculation on **??** using RTDM

of operations needed is 3.

3.8 Altered

As seen in **??** the extended version of the algorithm fixes a problem, but a correct distance is not yet reached. The problem seems to stem from node $T_{1,2}(M)$ where the algorithm correctly maps the branch $T_1(N-O)$ to $T_2(N)$ with a delete cost of 1, but then forgets to delete the other branch, $T_1(P-Q)$. To fix this I present the Altered RTDM (aRTDM).

This algorithm works with RTDM as the base, but takes the approach of playing devils advocate. Where RTDM and eRTDM starts by initializing the matrix M to 0 everywhere, which roughly corresponds to assuming that every node will be mapped, the aRTDM starts by initializing the matrix to the combined cost

eR	j=0	j=1	j=2
i=0	0	0	0
i=1	0	1	2
i=2	0	1	1

Figure 3.12: Matrix corresponding to the distance calculation on **??** using eRTDM

N	j=0	j=1
i=0	0	0
i=1	0	1
i=2	0	1

Figure 3.13: Matrix corresponding to the sub-distance calculation on **??** using eRTDM

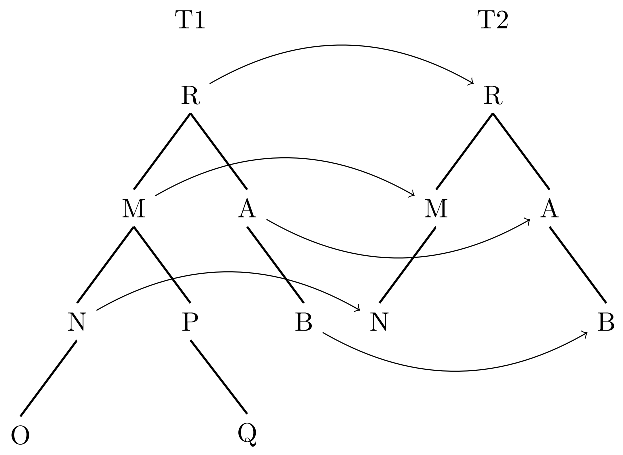


Figure 3.14: A mapping of example **??** with a total distance of 3 (from deleting nodes O,P,Q)

of deleting and inserting every descendant from both trees, effectively assuming no nodes will be mapped. From here on the algorithm subtracts the reward of mapping two nodes to each other from the total sum.

Here follows the introduction to aRTDM which can be seen in algorithm **??**. The algorithm takes two trees as input. First an array M, the size of the number of children on each root, is initialized holding a position for each pairing. Each position is set to the value cost of deleting and inserting all descendants of T_1 and T_2 respectively.
Then three different costs are now computed.

1. *del* : This cost is the sum of the previous node pair of the first tree. It assumes that the node on the second tree it is trying to match to, is better paired with the previous node and therefore instead deletes itself and all its descendants, since this cost is already assumed from the start, no change to the value is made.
2. *ins* : The theory of this cost is the same as *del* but instead of looking at the first tree it looks at the second, and inserts the node and all its descendants. Again the cost remains the same.
3. *sum* : This cost is the total cost of mapping the two current nodes to each other. It initializes its value to the cost of the previous node pair and then subtracts the cost of the two nodes being matched, as they will be mapped no matter what. For this there are 4 cases.
   (a) The two nodes and their sub trees are identical, the cost is reduced by the combined cost of inserting and deleting the two sub trees.
   (b) The first or the second node is a leaf. A relabel is made between the two nodes.
   (c) Neither of the nodes are leafs and share the same label. The cost of the two sub trees is calculated by aRTDM and the inverse value is then returned from *invertCost.*
   (d) Neither of the nodes are leafs but do not share the same label. The cost is increased by a relabel operation.

Finally the minimal of the costs are found and placed on the pairs position in the array.
The final value produced by the algorithm is the total cost of mapping T_1 to T_2.

invertCost can be found in algorithm **??**. Since the algorithm returns the cost of the mapping, which roughly corresponds to the amount of nodes not mapped, an inversion is required to get the sum of nodes that were mapped. This is obtained by computing the cost of deleting/inserting all descendants of the two trees and then subtracting the distance of the two from this value.

Algorithm 3 aRTDM(T_1, T_2)

```
1:  $m \leftarrow T_1$ number of children
2:  $n \leftarrow T_2$ number of children
3:  $m \leftarrow C_1 \leftarrow$ all descendants of $T_1$
4:  $n \leftarrow C_2 \leftarrow$ all descendants of $T_2$
5:  $M[m,n] \leftarrow delete(C_1) + insert(C_2)$ at all positions
6:  for $i = 1 \rightarrow m$ do
7:      for $j = 1 \rightarrow n$ do
8:          $t_1 \leftarrow$ child of $T_1$ at pos $i$
9:          $t_2 \leftarrow$ child of $T_2$ at pos $j$
10:         $c_i \leftarrow$ all descendants of $t_1$
11:         $c_j \leftarrow$ all descendants of $t_2$
12:         $del \leftarrow M[i-1, j]$
13:         $ins \leftarrow M[i, j-1]$
14:         $sum \leftarrow M[i-1][j-1]$
15:         $sum \leftarrow sum - delete(t_1)$
16:         $sum \leftarrow sum - insert(t_2)$
17:         if $t_1$ is equal to $t_2$ then
18:             $sum \leftarrow sum - delete(c_i)$
19:             $sum \leftarrow sum - insert(c_j)$
20:         else
21:             if $t_1$ is a leaf $\|$ $t_2$ is a leaf then
22:                 $sum \leftarrow sum + relabel(t_1, t_2)$
23:             else if $t_1.val$ equals $t_2.val$ then
24:                 $sum \leftarrow sum-$ invertCost$(t_1, t_2)$
25:             else
26:                 $sum \leftarrow sum + relabel(t_1, t_2)$
27:             end if
28:         end if
29:         $M[i, j] \leftarrow min(del, ins, sum)$
30:     end for
31: end for
32: return $M[m, n]$
```

3.9 Examples with aRTDM

This example will again look at figure **??** and its corresponding mapping **??**. This time the matrices will have changed, which can be seen in a figure **??** and

Algorithm 4 invertCost(T_1, T_2)

```
1: $C_1 \leftarrow$ all descendants of $T_1$
2: $C_2 \leftarrow$ all descendants of $T_2$
3: $sum \leftarrow delete(C_1) + insert(C_2)$
4: $sum \leftarrow sum -$ aRTDM($T_1, T_2$)
5: return $sum$
```

aR	j=0	j=1	j=2
i=0	11	11	11
i=1	11	7	7
i=2	11	7	3

Figure 3.15: Matrix corresponding to the distance calculation on **??** using aRTDM

??.

Notice that this time the algorithm reaches the expected distance of 3 for mapping T_1 to T_2.

N	j=0	j=1
i=0	5	5
i=1	5	3
i=2	5	3

Figure 3.16: Matrix corresponding to the sub-distance calculation on **??** using aRTDM

Chapter 4

The Program

The *.java* files along with the HTMLUnit library can be collected, as a *.zip* file, from the following location http://www.student.dtu.dk/~s093258/.

4.1 HTree

The representation of a tree used in this project. Contains functions for calculating size and height of the tree in addition to a *toString()* method. Is a serializable class so it can be saved.

4.2 HNode

The bread and butter in the trees. Each node contains, in addition to a list of children and a link to its parent, fields for keeping track of its type of wildcard, if it is a tag or a piece of text and a list of allowed tags.
This class got a number of functions, the most noticeable being *equals(HNode)* and *equalsSubTree(HNode)*. The first checks whether two nodes have a label

in common and the second checks whether two nodes and their respective sub trees are equal. Is also serializable as they are a part of the tree.

4.3 RTDM

The implementation of the algorithm. Split up in two major functions, *distance(HTree,HTree)* and *mapping(HTree,HTree)*. The first calculates the distance from one tree to another and returns it as an integer. The second computes a mapping from two trees and returns the root node of the corresponding matching tree. Helper functions for calculating the total cost of deleting/inserting a nodes descendants are also found in this class, along with functions for inserting and merging nodes to form the mapping.

4.4 Generalizer

Contains all utility that is not directly related to the algorithm. This includes compression of sub trees: *compressSubtrees(HNode)*, creating a tree from a DOM representation of a page: *createTree(DomNode, HTree)*, save and load functionality, a random blog address generator and the Web Client used for connecting to the internet.

Chapter 5

Solution

In chapter **??**, two different views on blogs were introduced, the front page and the actual article. In my solution I have decided to focus only on the front page, as these where the easiest to access by using the built in randomizing function in Wordpress[1], which redirects to the front page of a random blog on Wordpress domain. Therefore, from after this passage the word *blog* will refer to the front page of a blog.

On the subject of JavaScript mappings were created and tests were run against pages with JavaScript enabled, as quite a lot of content today is managed by JavaScript and if a page would run with it disabled a lot of the structure might be lost, leading to a less descriptive matching tree.

After the algorithm had been implemented the next step was to find a threshold for the distance which would separate blogs from arbitrary pages. First several matching trees was created using a varying number of blogs (10, 50, 100) and two sets of costs ([re:0,del:1,ins:1],[re:1,del:1,ins:1]). The first cost set was based on [**?**] where the second was created for experimental purposes.
Then a measuring of the distance from each of these matching trees against two sets of pages began. One set containing only blogs, and one containing other types of pages with a varying degree in closeness to the looks of blogs, called non-blogs. Within each set of costs there would be a maximum distance

[1] `http://en.wordpress.com/next/`

measured to a blog, and a minimum distance measured to a non-blog page which would give two cases for determining the threshold.

1. $max(blogs) < min(\neg blogs)$: If this was to happen, the answer would be quite simple, as the threshold could then be placed within the *max* and the *min*. A more precise value could be found through further experimentation.

2. $min(\neg blogs) <= max(blogs)$: If this was to happen, the answer is no longer straight forward. Depending on the amount of non-blog pages that had a lower cost than the actual blogs two possibilities were open.

   (a) A low amount of pages resulted in false-positives: Run further test with more pages of the same kind, seeing if it is a general trend. If not, the actual impact on the precision of the program could be revealed through testing.

   (b) A large amount of pages resulted in false-positives: Would indicate that the matching tree was either too large, a lot of nodes not getting matched and therefore deleted, too small, a lot of nodes needed to be inserted, both of these resulting in a high cost for every site, or the matching tree could be too open, resulting in each node mapping to too many different tags, effectively matching almost anything.

Due to the abnormalities found in **??** it became clear that the project could take one of two paths.
One path would keep focusing on the RTDM algorithm and continue to work on this to solve the abnormalities discovered. The first step would then be to find and correct the erroneous behaviour that led it to return a total distance of 1 where 3 was expected.
The other path was to return to the basic Tree-Edit-Distance algorithm, for example the one proposed in [**?**] and then expand on this algorithm with both the Top-Down restriction and the Extended-Top-Down restriction thus obtaining an algorithm which should return the same solution as the RTDM algorithm. Note that the time and memory complexity would not be directly comparable. The search space reduction should still be the same, as these were obtained through the mapping restrictions and the sub tree merging.

The choice fell on the first path which lead to the aRTDM, since a lot of time had already been invested in analysing the RTDM algorithm. The idea for this alteration came from the realization that the original algorithm returned values with too low a value. By forcing it to assume the worst case possible it might still have some of the same erroneous behaviour, which it did not, but the differences between the trees would be more emphasized as the overall distance grew.

The program is based on the assumption that all the visited pages were well-formed, that is, no tags violated the HTML dependency. This assumption should be safe as Wordpress blogs are automatically created when using the CMS system.

Chapter 6

Results

In this section the results from testing the program will be presented. Matching trees were constructed based on 10, 50 and 100 different blogs, as seen in **??**. These where then tested on two sets of 25 different pages.
The first set contains legit blogs, as described in chapter **??**. The other set is a combination of different types of pages. These range from movie based, youtube.com, to picture based, amazon.com, to large amounts of data, dr.dk, to blog look-a-like, thedailywtf.com.

6.1 eRTDM

The following two figures, **??** and **??**, shows the correlation between distance and matching tree obtained by running the program with said specifications and costs: relabel 0, delete 1, insert 1.

When first looking at figure **??** one might notice that the 3 matching trees does not return the same values. This is because each mapping is made from different blogs and thus they have a differe nt structure from one another, depending on how the pages used to create them looked.

From these results it is not clear what is, and what is not a blog. When the label

cost is 0 the values seem arbitrary when comparing the same site by different matchings, counting both the blogs and the non-blogs.
When the label cost is 1 again the distance seem very dependent on what pages it has been mapped from. From 10 to 50 pages, the distance seems to increase, while it from 50 to 100 takes a steep step down to only calculating distances of in the range of [0;11]. This could be an indicator that the matching is close to its optimal size when the number of pages used are 50, providing a good number of possible matchings without allowing each node to map to too many different tags.
When observing both sets of costs, the blogs seem to perform a little bit better than the non-blogs, in terms of returning values in a more stable interval. This could allow for a upper limit, rougly 40, on what would be considered a blog, but other measures would then have to be taken to sort out the pages that fell in the same range of distances.
From chapter **??** the algorithm ends in the very last option where the distances of blogs and false positives are almost the same, and it is not just a minority which means that no threshold can be determined to separate blogs from other pages. From this I conclude that the RTDM algorithm in its current form is not fitted to provide a specific enough structure for blogs to be recognized.

6.2 aRTDM

Now looking to figure **??** through **??**. The values now show a general tendency to be lower for the blogs and higher for the non-blogs. Especially to matching trees stand out:

- m100r0: This matching tree achieves good results, for most of the blogs. The *min/max* between the two types of sites, does not have a great interleaving, but what is found is that a "grey zone" can be established in the interval [2600;2650]. This is chosen based on that it gives a minimum off pages in the grey zone, while obtaining as high a value as possible for the blogs. When this is chosen 5 blogs and 5 pages are in the grey zone, while 2 blogs falls outside of it.
- m50r1: This matching tree too yield good results. The grey zone is again the interval [2600;2650], gaining 9 blogs and 4 non-blogs in the grey zone and 3 blogs outside not recognized.

All in all does the matching tree created with a relabel cost of 0 perform better than its opposition, both in terms of blogs recognized, but also in terms of

size	M10	M50	M100
r0	1249	6636	7944
r1	1185	7021	8474

Figure 6.1: Sizes of trees constructed using aRTDM

creating a more precise matching tree, the higher the amount of blogs used, as opposed to relabel cost 1 which seems to be getting too big when it hits 100 blogs used in its creation. This is further enforced by the fact that a larger tree is created when a relabel cost is introduced as seen in figure **??**, which means that the matching tree with no relabel cost is better at consuming new blogs into its structure.

6.3 Data

All matching trees created for the aRTDM, their corresponding link files and the link files for testing against, blogs and non-blogs, can be found in the *data* file contained in the *.zip* file containing the program which can be found at `http://www.student.dtu.dk/~s093258/`.

eR0	**M10**	**M50**	**M100**
1	38	26	33
2	24	15	19
3	10	6	11
4	38	27	30
5	37	37	40
6	26	15	19
7	39	26	36
8	24	15	19
9	32	47	39
10	43	35	31
11	9	7	8
12	36	1	31
13	24	16	21
14	29	39	26
15	31	37	1
16	10	6	11
17	32	47	39
18	34	24	31
19	34	24	31
20	37	38	40
21	37	37	40
22	31	23	30
23	35	22	29
24	24	6	19
25	24	15	18

Figure 6.2: A table for the e-distances of 25 random blogs matched against the 3 matching trees with re-label cost 0

eR0	**M10**	**M50**	**M100**
1	10	7	10
2	12	13	12
3	18	21	18
4	13	14	13
5	8	9	8
6	14	13	18
7	76	101	30
8	35	30	29
9	4	3	8
10	TO	17	23
11	31	31	25
12	221	245	109
13	TO	37	33
14	43	31	23
15	27	18	23
16	8	5	9
17	13	9	14
18	14	22	17
19	55	106	23
20	106	94	22
21	245	245	109
22	125	125	109
23	51	47	27
24	15	22	14
25	20	12	12

Figure 6.3: A table for the e-distances of 25 random non-blog pages matched against the 3 matching trees with re-label cost 0

eR1	**M10**	**M50**	**M100**
1	6	20	2
2	8	20	3
3	17	20	3
4	3	19	2
5	8	21	3
6	18	11	2
7	8	21	3
8	8	20	3
9	5	21	4
10	4	20	2
11	14	22	3
12	12	10	1
13	17	11	2
14	3	19	3
15	21	1	1
16	17	20	3
17	5	21	3
18	8	17	3
19	8	17	3
20	8	21	2
21	8	21	3
22	4	22	3
23	20	11	2
24	8	21	3
25	7	20	3

Figure 6.4: A table for the e-distances of 25 random blogs matched against the 3 matching trees with re-label cost 1

eR1	**M10**	**M50**	**M100**
1	2	6	3
2	18	27	T/O
3	8	9	2
4	13	13	2
5	8	8	2
6	7	8	3
7	23	18	2
8	19	28	3
9	14	3	2
10	20	T/O	1
11	24	18	11
12	63	92	1
13	TO	13	T/O
14	9	13	2
15	10	11	1
16	15	21	2
17	T/O	8	3
18	16	8	2
19	T/O	9	3
20	2	9	3
21	63	91	2
22	63	91	3
23	19	T/O	3
24	1	8	2
25	16	10	2

Figure 6.5: A table for the e-distances of 25 random non-blog pages matched against the 3 matching trees with re-label cost 1

aR0	**M10**	**M50**	**M100**
1	467	2392	2620
2	515	2402	2575
3	844	2603	2616
4	612	t/o	2576
5	476	2360	2579
6	535	2406	2605
7	632	2435	2439
8	801	2415	2471
9	1516	2296	2593
10	520	2343	2572
11	853	2248	2494
12	915	2729	2470
13	491	2346	2551
14	581	2382	2532
15	834	2449	2501
16	847	2617	2622
17	1516	2295	2592
18	2108	2401	3359
19	2108	2401	3359
20	467	2392	2620
21	558	2377	2520
22	569	2335	2586
23	513	2404	2581
24	614	2442	2562
25	655	2403	2593

Figure 6.6: A table for the a-distances of 25 random blogs matched against the 3 matching trees with re-label cost 0

aR0	**M10**	**M50**	**M100**
1	650	2438	2657
2	718	t/o	t/o
3	574	t/o	2648
4	545	2459	2680
5	538	2451	2672
6	515	2412	2609
7	747	2697	2915
8	541	2445	2657
9	528	2416	2619
10	t/o	2956	3052
11	1362	3099	3290
12	1006	2918	3118
13	t/o	t/o	t/o
14	709	2445	2674
15	559	2448	2648
16	1158	3039	3189
17	t/o	2669	t/o
18	626	2418	2607
19	t/o	2784	2928
20	1322	t/o	3362
21	774	2690	2905
22	649	2551	2759
23	4783	6120	6556
24	582	2479	2691
25	689	2584	2796

Figure 6.7: A table for the a-distances of 25 random non-blog pages matched against the 3 matching trees with re-label cost 0

aR1	**M10**	**M50**	**M100**
1	t/o	2611	2799
2	469	2574	2762
3	780	2873	2840
4	538	2581	2700
5	432	2552	2764
6	485	2630	2799
7	591	2647	2648
8	760	2698	2690
9	1467	2574	2856
10	484	2576	2742
11	834	2437	2577
12	907	2643	2609
13	451	2575	2721
14	526	2608	2723
15	798	2906	2832
16	778	2873	2842
17	1467	2574	2856
18	2031	2561	3617
19	2031	2561	3617
20	410	2611	2799
21	501	2587	2718
22	534	2568	2786
23	494	2619	2780
24	577	2636	2803
25	607	2650	2830

Figure 6.8: A table for the a-distances of 25 random blogs matched against the 3 matching trees with re-label cost 1

aR1	**M10**	**M50**	**M100**
1	592	2663	2880
2	t/o	2747	2975
3	535	t/o	2856
4	494	2654	2844
5	488	2648	2832
6	503	2623	2806
7	709	2868	3087
8	496	2649	2835
9	477	2615	2800
10	1451	3272	3401
11	130	3362	3491
12	963	3141	3311
13	t/o	3236	3414
14	671	2708	2884
15	571	2691	2850
16	117	2837	3357
17	776	2881	3031
18	573	2659	2795
19	937	3018	3138
20	1300	3373	3578
21	t/o	2885	3069
22	610	2762	2949
23	4787	6441	6750
24	553	2702	2885
25	644	2787	2973

Figure 6.9: A table for the a-distances of 25 random non-blog pages matched against the 3 matching trees with re-label cost 1

Chapter 7

Conclusion

In this thesis the RTDM algorithm has been analysed, implemented and evaluated, which has first led to an extended and subsequently to an altered version of said algorithm. Note that the current implementation and analyses have been created from [?] which did not contain a section for definitions which caused the algorithm to be interpreted on the basis of the article and on the work the algorithm is based on ([?] and [?]).
While the extended version did not solve the problem of recognizing blogs, the altered version has shown potential as to defining a grey zone in which all pages below its mininum value are blogs whilst all pages above its maximum values are non-blogs. The research does not directly give an answer to whether a page is a blog or not, but it does show promise that with experimentation into the field of Wild Cards and possibly applying more restrictions a definitive answer might be found.
As explained in **??** the mapping restrictions have been validated as useful when working with trees built from HTML structures. The same goes for the Wild Cards and the merging of sub trees as these, too, are valid and efficient ways to reduce the search space and improve the receptiveness of the matching tree.

7.1 Future Work

In chapter ??, ??, it was shown that blogs could be determined to a certain point whereupon a grey zone of interleaving values appeared. This could possibly be improved by creating a more specific algorithm to produce a better, more precise matching tree, which might eliminate this zone. Another solution might be to expand on the types of pages matched. If a large amount of page types are matched, a page in a blog grey zone might either fall perfectly into another category or outside all other categories, causing the page to be a non-blog or a blog, respectively.
Further research into the effects of Wild Cards and Operation costs should also be explored as possible ways to enhance the precision of the algorithm.

Appendix A

Addresses for matching trees

A.1 Matching Tree 10 Blogs

```
http://sitesbernhardp.wordpress.com/
http://psychiatrymcqs.wordpress.com/
http://lindaslinkstoliterature.wordpress.com/
http://scarlettcorbin.wordpress.com/
http://loansnoguarantor.wordpress.com/
http://geeksyndicate.wordpress.com/
http://twintotstoteens.wordpress.com/
http://yonkersrealestate.wordpress.com/
http://jessemacgregorjones.wordpress.com/
http://twintotstoteens.wordpress.com/
25/06/2012 23:00:07
```

A.2 Matching Tree 50 Blogs

```
1  http://qawaariyyah.wordpress.com/
2  http://driodroid.wordpress.com/
3  http://srvjournalabstracts.wordpress.com/
4  http://leeharam0211.wordpress.com/
5  http://loansnoguarantor.wordpress.com/
6  http://travelbib.wordpress.com/
7  http://sportonthebox.wordpress.com/
8  http://dawncompk.wordpress.com/
9  http://speciesdewittz.wordpress.com/
10 http://openletterfromparis.wordpress.com/
11 http://humnda7134.wordpress.com/
12 http://jctitushoratio.wordpress.com/
13 http://marantophotography.wordpress.com/
14 http://dailywalter.wordpress.com/
15 http://dawncompk.wordpress.com/
16 http://emyannie.wordpress.com/
17 http://pennockfloral.wordpress.com/
18 http://mnkristy.wordpress.com/
19 http://scarlettcorbin.wordpress.com/
20 http://bryologue.wordpress.com/
21 http://findingyourblessings.wordpress.com/
22 http://mikeeliasz.wordpress.com/
23 http://ndnmafia.wordpress.com/
24 http://healingmothersspirit.wordpress.com/
25 http://buzzhub.wordpress.com/
26 http://dawncompk.wordpress.com/
27 http://srvjournalabstracts.wordpress.com/
28 http://spencermakesgames.wordpress.com/
29 http://speciesdewittz.wordpress.com/
30 http://psychiatrymcqs.wordpress.com/
31 http://psychiatrymcqs.wordpress.com/
32 http://jessemacgregorjones.wordpress.com/
33 http://localtvwhnt.wordpress.com/
34 http://fastboyceyy.wordpress.com/
35 http://ourhumbleabowed.wordpress.com/
36 http://saharanwilderness.wordpress.com/
37 http://mnkristy.wordpress.com/
38 http://jessicabuzanko.wordpress.com/
39 http://healingmothersspirit.wordpress.com/
40 http://qawaariyyah.wordpress.com/
41 http://pennockfloral.wordpress.com/
```

```
http://atelierfestival.wordpress.com/
http://coldwellbankerbahamas.wordpress.com/
http://renytasari.wordpress.com/
http://broadyesl.wordpress.com/
http://marantophotography.wordpress.com/
http://paddypowerblog.wordpress.com/
26/06/2012 02:48:13
```

A.3 Matching Tree 100 Blogs

```
http://localtvwreg.wordpress.com/
http://denmarkusmgreentour.wordpress.com/
http://travelbib.wordpress.com/
http://dawncompk.wordpress.com/
http://renytasari.wordpress.com/
http://nhnaplesit.wordpress.com/
http://loopstagirl.wordpress.com/
http://localtvwreg.wordpress.com/
http://mnkristy.wordpress.com/
http://broadyesl.wordpress.com/
http://deviljazz.wordpress.com/
http://roneipowerrichmond.wordpress.com/
http://ikipsobm.wordpress.com/
http://dailywalter.wordpress.com/
http://marketingandprblog.wordpress.com/
http://thisearthnatureblog.wordpress.com/
http://healingmothersspirit.wordpress.com/
http://paddypowerblog.wordpress.com/
http://stfirmin.wordpress.com/
http://theweddinglark.wordpress.com/
http://weaponsbrennenb.wordpress.com/
http://jessicabuzanko.wordpress.com/
http://staugustinexperience.wordpress.com/
http://ionenewsone.wordpress.com/
http://travelbib.wordpress.com/
http://ronetherussparrmorningshow.wordpress.com/
http://rossdowsen.wordpress.com/
http://renytasari.wordpress.com/
http://cbschicago.wordpress.com/
http://ndnmafia.wordpress.com/
http://mnkristy.wordpress.com/
```

32 http://ronetherussparrmorningshow.wordpress.com/
33 http://theindustrycosign.wordpress.com/
34 http://rossdowsen.wordpress.com/
35 http://duflpress.wordpress.com/
36 http://ndnmafia.wordpress.com/
37 http://eatvidence.wordpress.com/
38 http://speciesdewittz.wordpress.com/
39 http://findingcomity.wordpress.com/
40 http://smalegalgroup.wordpress.com/
41 http://unquiettongue.wordpress.com/
42 http://grabbedme.wordpress.com/
43 http://howswedeislife.wordpress.com/
44 http://grabbedme.wordpress.com/
45 http://srvjournalabstracts.wordpress.com/
46 http://staugustinexperience.wordpress.com/
47 http://5kitchenfaucets.wordpress.com/
48 http://smalegalgroup.wordpress.com/
49 http://healingmothersspirit.wordpress.com/
50 http://westernseminaryblog.wordpress.com/
51 http://dailywalter.wordpress.com/
52 http://androidtabletlenovo.wordpress.com/
53 http://pennockfloral.wordpress.com/
54 http://travelbib.wordpress.com/
55 http://howswedeislife.wordpress.com/
56 http://buzzhub.wordpress.com/
57 http://thegesturehunter.wordpress.com/
58 http://denmarkusmgreentour.wordpress.com/
59 http://openletterfromparis.wordpress.com/
60 http://thisearthnatureblog.wordpress.com/
61 http://paddypowerblog.wordpress.com/
62 http://nationalpostnews.wordpress.com/
63 http://humnda7134.wordpress.com/
64 http://kimberlysellshomes.wordpress.com/
65 http://weaponsbrennenb.wordpress.com/
66 http://georgekreeger.wordpress.com/
67 http://denmarkusmgreentour.wordpress.com/
68 http://twintotstoteens.wordpress.com/
69 http://roneipowerrichmond.wordpress.com/
70 http://nhnaplesit.wordpress.com/
71 http://openletterfromparis.wordpress.com/
72 http://denmarkusmgreentour.wordpress.com/
73 http://howswedeislife.wordpress.com/
74 http://weaponsbrennenb.wordpress.com/
75 http://driodroid.wordpress.com/

```
http://artmodel.wordpress.com/
http://robintimweis.wordpress.com/
http://theweddinglark.wordpress.com/
http://cbschicago.wordpress.com/
http://ikipsobm.wordpress.com/
http://localtvwhnt.wordpress.com/
http://findingcomity.wordpress.com/
http://jessicabuzanko.wordpress.com/
http://wholezome.wordpress.com/
http://ionenewsone.wordpress.com/
http://geeksyndicate.wordpress.com/
http://ionenewsone.wordpress.com/
http://wholezome.wordpress.com/
http://sportonthebox.wordpress.com/
http://broadyesl.wordpress.com/
http://counselingreagencaj.wordpress.com/
http://buzzhub.wordpress.com/
http://svetomir.wordpress.com/
http://vanitymagazineblog.wordpress.com/
http://humnda7134.wordpress.com/
http://ronetherussparrmorningshow.wordpress.com/
http://seeallisoneat.wordpress.com/
http://yonkersrealestate.wordpress.com/
25/06/2012 23:26:06
```

Appendix B

Addresses for distance testing

B.1 Blogs

```
1  http://yakiratoya97a.wordpress.com/
2  http://terrilbaker.wordpress.com/
3  http://tbnranch.wordpress.com/
4  http://wendegarrison.wordpress.com/
5  http://billywbennight.wordpress.com/
6  http://walkofdarkness.wordpress.com/
7  http://wuxtian.wordpress.com/
8  http://mosedpress.wordpress.com/
9  http://kaminarianimereview.wordpress.com/
10 http://pinballtrading.wordpress.com/
11 http://davidgurrmusic.wordpress.com/
12 http://annebegg.wordpress.com/
13 http://abuhusamassalafi.wordpress.com/
14 http://emeraldcityedm.wordpress.com/
15 http://panasonictve.wordpress.com/
16 http://guilhaumegerard.wordpress.com/
17 http://billywbennight.wordpress.com/
```

```
http://terrilbaker.wordpress.com/
http://teachingnicholasak.wordpress.com/
http://stevepearce.wordpress.com/
http://vickynanjapa.wordpress.com/
http://soullightenment.wordpress.com/
http://contentprotection.wordpress.com/
http://pinballtrading.wordpress.com/
http://gtatickets.wordpress.com/
26/06/2012 02:50:28
```

B.2 Non-blogs

```
http://www.fmylife.com/
http://www.newgrounds.com/
http://grooveshark.com/#!/search?q=billy+talent
http://www.nerdtech.com/
http://www.alienworkshop.com/
http://loadrecords.com/
http://www.dreadcentral.com/
http://www.pastglory.nl/
http://www.agsfb.com/All_Girl_Summer_Fun_Band/AGSFB.html
http://www.brownstoner.com/
http://www.dr.dk/
http://www.dr.dk/Nyheder/Politik/2012/06/25/183431.htm
http://www.google.dk/search?q=daily+wtf&ie=utf-8&oe=utf-
        8&aq=t&rls=org.mozilla:da:official&client=firefox-a&channel=fflb
http://thedailywtf.com/
http://www.cad-comic.com/cad/
http://www.youtube.com/
http://www.youtube.com/user/CaptainSparklez?feature=g-logo-xit
http://stackoverflow.com/
http://www.amazon.com/
http://www.amazon.com/Breakfast-Foods-Grocery/b/
        ref=sv_gro_1?ie=UTF8&node=16310251
http://www.dtu.dk/default.aspx
http://maddox.xmission.com/
http://www.reddit.com/r/AdviceAnimals/comments/vlasp/
        there_can_only_be_two_babies/
http://imgur.com/
http://www.facebook.com/
26/06/2012 03:15:17
```

Bibliography

[Bil03] Philip Bille. Tree edit distance, alignment distance and inclusion. *IT Univ of Copenhagen TR20032*, (March), 2003.

[Cha99] Sudarshan S. Chawathe. Comparing hierarchical data in external memory. *Proceedings of the Twenty-fifth International Conference on Very Large Data Bases*, pages 90–101, 1999.

[DdCR04] Alberto H. F. Laender-Altigran S. da Silva Davi de Castro Reis, Paulo B. Golgher. Automatic web news extraction using tree edit distance. *Proceedings of the 13th international conference on World Wide Web*, pages 502–511, 2004.

[Tai79] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26:422–433, 1979.

[Val01] Gabriel Valiente. An efficient bottom-up distance between trees. *String Processing and Information Retrieval*, pages 212–219, 2001.

[Yan91] Wuu Yang. Identifying syntactic differences between two programs. *Software - Practice and Experience*, 21:739–755, 1991.

[ZZ08] Xindong Wu Xue-Gang Hu Fei-Yue Wang Zhu Zhu, Gong-Qing Wu. Automatic recognition of news web pages. *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 496–501, 2008.