Visual Data Record Extraction From Query Result Pages

A Dissertation submitted in partial fulfillment of
the requirements for the degree of
MASTERS OF ENGINEERING in Computer Science
in
The Queen's University of Belfast
by

Mario Paolucci
7th May 2013

Supervisor: Dr Jun Hong

ABSTRCT

Data-aware search is becoming more and more popular on the World Wide Web. Query result pages, generated dynamically from web databases, are now ubiquitous. Developing a universal and generalized method for extracting structured data records from such pages is a very challenging task as the structure of the page, and therefore of the data, is not standardized and cannot be predicted. The advancement of web technologies and programming languages, which can dynamically change the structure of a web page, has rendered certain approaches to this issue, such as analysing the DOM or tag tree of a page, unsuitable for the task at hand. In this paper a new approach is proposed, one that tries to mimic human intuition in understanding the structure of a page and identifying structured data results. Features such as structural regularity of a page and content similarity are used to identify data records individually and discard the noise on the page. The experimental results carried out show that this approach produces a high level of accuracy.

Table of Contents

1. Introduction

The problem area that my research is focused on is Visual Data Record Extraction from query result pages. The World Wide Web has millions of web databases that can be accessed through various search engines and structured queries. All of the most prominent search engine providers such as Google, Yahoo or Bing, are focusing their efforts more and more on data-aware search as well as their more traditional document centric approach. They all realise that it is more beneficial and productive to the user if, in response to a search query, they are able to get structured data back rather than just a list of web links to pages that need further exploration and that might contain what they are looking for. For instance, a user might be looking for a book online and queries for its title through a search engine. It is obvious to see that a list of links to websites that sell books is far less beneficial to the user than a list of book objects each showing specific related information about that item such as its price, availability, thumbnail etc. Most of this structured content on the Internet is stored in large web databases and it is commonly referred to as Deep Web. This data is usually accessed through HTML query forms that get translated into database queries and then returned and rendered by the web browser in so called query result pages. Common examples of such pages are most e-commerce websites such as Amazon, eBay, Game etc. Once a query is submitted in one of these websites, a list of structured data items, which we will call data records, is presented to the user who can in turn easily gain access to the information they were looking for. Figure 1 shows an example of such a result page from Amazon.co.uk.

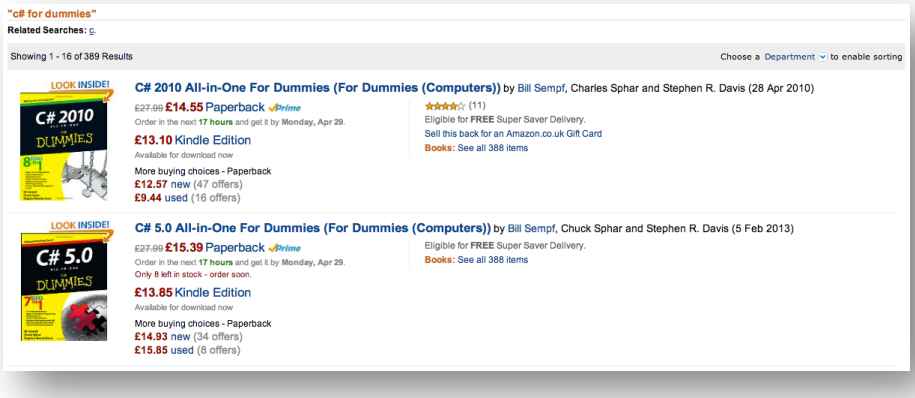


Figure 1 – Query Result Page from Amazon.co.uk

In the page above the query results are presented in the form of structured data records. Each of them represents a book object and contains relevant and related information such as the author of the

book, the price, the availability, the reviews etc. In order to make their consumption by users easier, structured data records are mainly displayed in web browsers in the manner shown by the example above. To automate the processing of these data records, which is needed for a variety of purposes such as Deep Web crawling, comparing and integrating data from different web databases etc., one of the crucial tasks to perform is the extraction of such records form their query result pages. This is a very challenging task as the way in which data records are structured in their respective query result page can differ greatly. The focus of this paper is the extraction of the data records from a query result page, that is, to identify the groups of data items that make up each data record within that page. The first step in the extraction process will be that of building a Visual Block Model (VBM) for the page by retrieving everything rendered on the page in the form of a visual block. Figure 2 below shows the same page from amazon.co.uk with all the visual blocks highlighted.

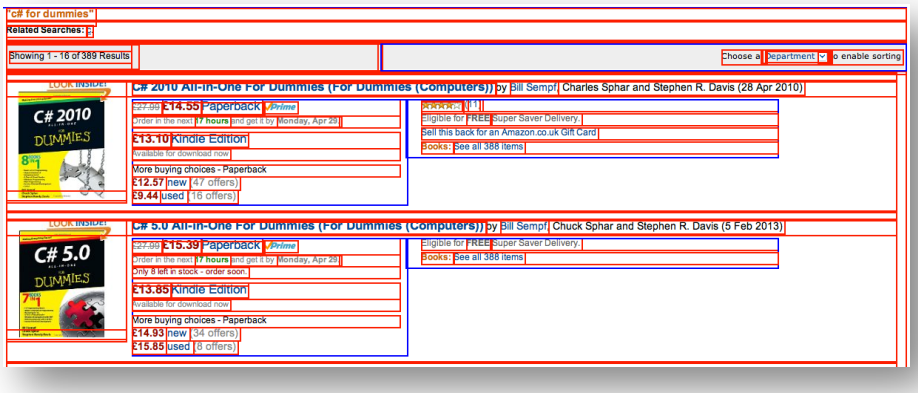


Figure 2 – Query Result Page From Amazon.co.uk with blocks highlighted

The second step is the clustering of the blocks based on several visual properties. Through various clustering techniques, based on the identification of repeated patterns in block structural and content similarity, candidate groups are selected and the data records extracted.

2. Related Work

In recent years the problem of automatic data record extraction from query result pages has received more and more attention from researchers around the world. The variety of uses that such a solution could have, make this a very relevant and popular issue. Over the years a number of different approaches have been taken to try and achieve the data record extraction.

2.1 Manual Approaches

2.1.1 Minerva

The very first approaches taken in order to achieve data records extraction were manual approaches in which programmers would construct wrappers in order to identify and extract all relevant data records and data fields. The primary example of a tool that did this is probably Minerva [3]. It is evident that such approaches are not suitable to today's needs due to their poor scalability. Today's efforts are much more focused on automating this process and finding solutions that are portable and scalable across data domains.

2.2 Semi-Automatic Approaches

2.2.1 WEIN and W4F

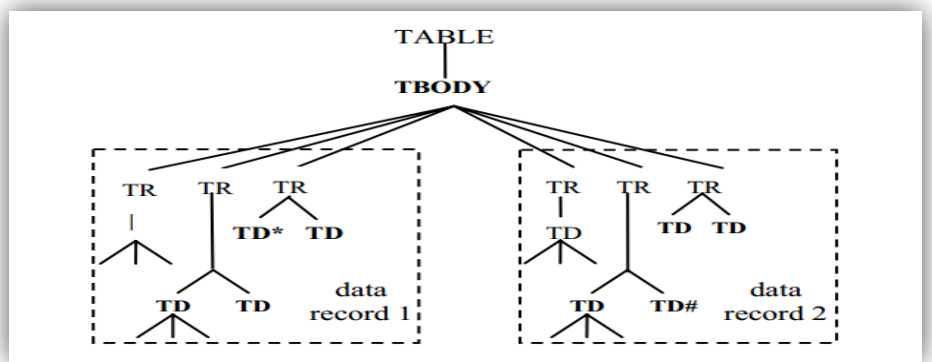
In later years some semi-automatic approaches were developed such as WEIN [4] and W4F [5]. The former uses a sequence-based approach in which the document is represented as a sequence of characters. The extraction process is achieved using a set of extraction rules generated from a set of training examples. The latter on the other hand uses a tree-based approach. The web page's hierarchical tree called the DOM tree (document object model tree) is parsed and the extraction performed based on the structure of such tree. These approaches, though a big improvement over the manual ones examined above, still require some manual efforts like labelling sample pages etc. and therefore have proven both impractical and time-consuming.

2.3 Automatic Approaches

2.3.1 IEPAD and DEPTA

More recently researchers have focused their efforts in developing automatic tools for data record extraction from query result pages. The very first attempts at this were based heavily upon the analysis of the result page's HTML structure and on its tag tree. Researchers tried to find similarities in terms of code structure between the different data records in order to group them and extract their content. This approach has inherent and clear limitations. Firstly they are completely HTML dependent. At first glance this might not seem like a big issue since most web pages are written in HTML and should, in theory, present similar tag trees and code structure. However HTML is an evolving mark-up language; this means that at every iteration of its lifecycle, as new tags are introduced, old ones become obsolete and are therefore removed. Data records extraction techniques that rely on a page's HTML structure would have to constantly be amended and modified to cater for such changes and to ensure that they are

compatible and work correctly with all different versions of HTML as they are released. In addition to this, the introduction of new mark-up languages for defining web pages such as XHTML, XSLT combined with CSS mean that such systems' field of action is further restricted and their domain further reduced. One more issue that such solutions have to face is the ever-increasing complexity of the HTML structure of modern web pages. If to this we add the fact that scripting languages such as JavaScript keep modifying the page's HTML dynamically, it becomes evident that the task of identifying data records similarities in query result pages based solely on the page's code and tag tree, is one that is only going to get more difficult with the advancement of web development technologies and therefore one that is not ideal. Good examples of this approach are IEPAD [1], or DEPTA [2]. These automatic approaches do not use any derived



extraction rules, they simply perform extraction for each Web page directly. Both of them are mainly based on analysing the source code

Figure 3 – An example tag tree used by DEPTA

of the Web page and therefore are constraint by the limitations described above. In particular DEPTA operates on the HTML tag tree structure of the page by first aligning data items in pairs of records that can be matched with certainty and then all the remaining data items are added incrementally. Figure 3 shows an example tag tree generated from a segment of a page that DEPTA would build and use to first align and then extract the data.

However this approach is still very much limited by the fact that it does not utilizes any visual information to cluster and extract data records and instead it's heavily reliant on the tag tree structure of the page and therefore to the HTML code of the document.

2.3.2 Visual based approaches

There is very limited amount of works that try to utilise some visual information about the web page in order to extract the data records. Some examples include ViNTS [6], ViPERS [7], HCRF [8] and ViDE [9]. In order to avoid some of the above discussed limitation of

previous approaches, ViNTS makes use of a combination of HTML tags and visual features to create wrappers for any given search engine and to help in selecting the most promising wrapper from those generated by the wrapper builder module. Figure 4 shows the main components of this wrapper builder module.

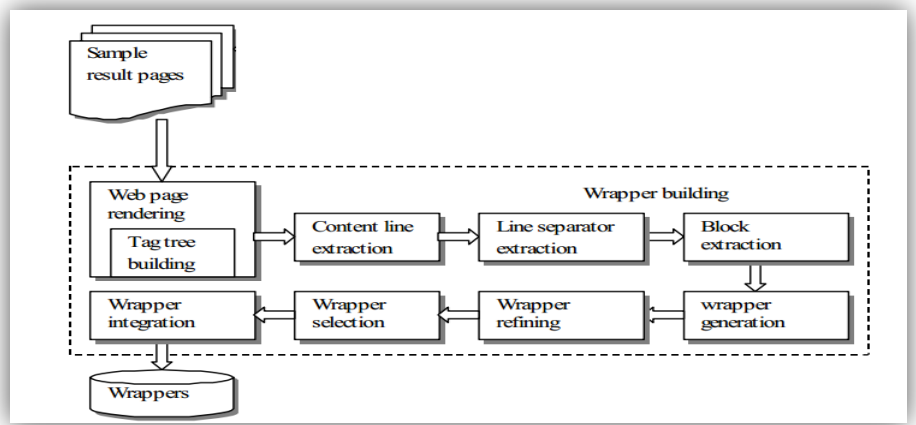


Figure 4 – The ViNTS wrapper generator

While this approach attempts to make use of the page's visual features during the extraction process, it is still evident from the diagram above that the wrapper generation is still heavily dependent on the tag tree and the HTML structure of the page making this approach also limited and language dependent. ViPERS also tries to use some basic visual information found on the query result pages to extract the data records by using some quite sophisticated alignment techniques. The tool first carries out some pre-processing steps to enhance the recognition robustness, and then it uses some metrics based on typical HTML structures to perform the alignment. This technique however not only makes some fundamental assumptions about query result pages (for example that a data record is either formed by a single coherent sub-tree or ranges over multiple adjacent siblings, which isn't necessarily always the case if we for instance think of a scripting language acting on the page), but it is also still very heavily reliant on the tag tree structure of the page. To this end it suffers the same limitations as the approaches described earlier. The last two solutions mentioned, HCRF [8] and ViDE [9], make use of some visual features as well but are reliant on the output of the so-called VIPS algorithm [10]. In this algorithm, the vision-based content structure of a page is deduced by combining the DOM structure and the visual cues. Firstly some visual information, such as position, background colour, font size etc., and the DOM structure are obtained from a web browser. Then from the root each node is analysed and checked based on visual cues to judge whether it forms a single visual block or not and to decide if it should be extracted or not. After further refining of the visual blocks the final vision-based content structure for the web page is outputted. Figure 5 below illustrates the visual block extraction process for the VIPS algorithm.

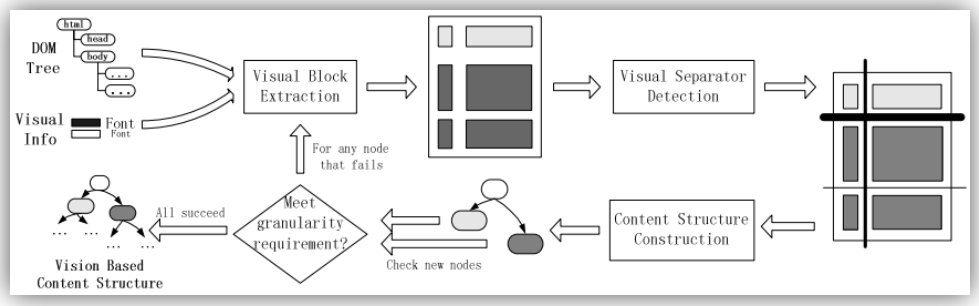


Figure 5 – The VIPS algorithm.

While both HCRF [8] and ViDE [9] make use of the above-described algorithm, there is some difference between the two. Firstly in HCRF the tag information is still an important feature. Secondly it starts from the assumption that every record corresponds to one block in the visual block tree, which is not always the case. In addition ViDE makes use of some more sophisticated cluster grouping algorithms shown in Figure 6 below.

Algorithm block regrouping

Input: $C_1,C_2,\ldots,C_m$: a group of clusters generated by blocks clustering from a given sample deep web page P

Output: $G_1,G_2,\ldots,G_n$: each of them corresponds to a data record on P

```
Begin
//Step 1. sort the blocks in Ci according to their positions in the page
(from top to bottom and then from left to right)
1 for each cluster Ci do
2    for any two blocks bi,j and bi,k in Ci   //1≤j<k≤|Ci|
3      if bi,j and bi,k are in different lines on P, and bi,k is above bi,j
4            bi,j↔bi,k;      //exchange their orders in Ci;
5      else if bi,j and bi,k are in the same line on P, and bi,k is in front of bi,j
6            bi,j↔bi,k;
7  end until no exchange occurs;
8  form the minimum-bounding rectangle Reci for Ci;
//Step 2. initialize n groups, and n is the number of data records on P
9  Cmax={Ci | |Ci|=max{|C1|,|C2|,……, |Cm|}};   // n=| Cmax |
10 for each block bmax,i in Cmax
11    Initialize group Gi;
12    put bmax,i into Gi;
//Step 3. put the blocks into the right groups, and each group
corresponds to a data record
13 for each cluster Ci
14    if Reci overlaps with Recmax on P
15       if Reci is ahead of (behind) Recmax
16          for each block bi,j in Ci
17             find the nearest block bmax,k in Cmax that is behind (ahead
                  of) bi,j on the web page;
18             place bi,j into group Gk;
End
```

Figure 6 – Clustering Grouping algorithm used by VIDE

Both these approaches have a higher reliance on the visual features of the page which makes them the closest to what we will be trying to achieve in this paper, however there is one fundamental difference: their work is completely dependent on the output of the VIPS algorithm and therefore there is no control over the clustering and the creation of the visual blocks. Our work will use a rendering engine that will simply return all of the pages' features in the form of separate ungrouped blocks. We will then develop our own clustering and grouping techniques, based solely on visual observations that we can make regarding the data records in order to extract them. This is what makes our approach innovative and unique. The measure of success of this new approach will be measured in terms of the accuracy of the results obtained. We will run our prototype against a series of datasets made up of pages taken from some of the most popular websites that offer structured data result pages.

3. Investigation Method

Before the main techniques of my approach to extract the data records from a query result page are presented I will describe the basic concepts and components that are needed to best understand them.

3.1 The Visual Block Model

The Visual Block Model of a query result page is a product of the pages' tag tree and visual features obtained through JQuery from the Cascading Style Sheets. Such model is composed of every item on the page and obtaining it is the very first step in my methodology. Making use of a rendering engine, the tag tree of the pages is traversed and for each item over 20 visual CSS properties are acquired through JQuery injection. Using the SWT Java library a browser object is firstly instantiated, then pointed to the URL of the query result page to be analysed and finally it has a JQuery script injected in order to retrieve the visual properties that I require. Throughout the rest of the paper I will refer to each of the items obtained as *visual blocks*. After retrieving all visual blocks I proceed to render them: this process of rendering draws a rectangular box around each element on the page as shown by Figure 7:

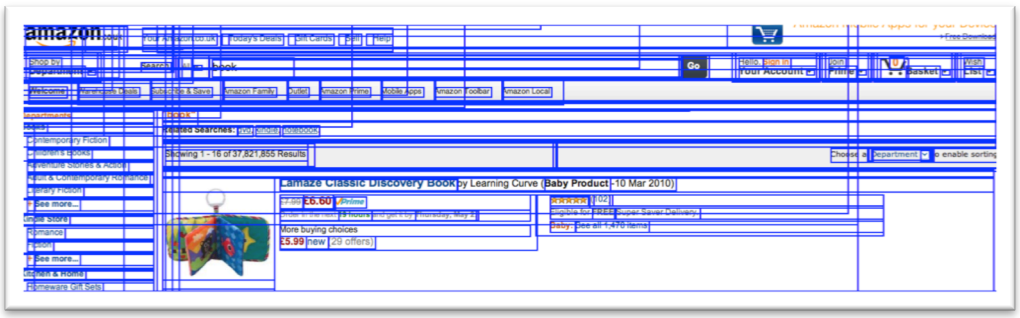


Figure 7 – Visual Block Model rendered on the page.

As shown quite clearly in Figure 7, the Visual Block Model initially includes a lot of noise items gathered from the web page. It is quite common in fact for web designers to include items in the code of a web page that don't represent any visible element but rather are there either to ensure that the page displays correctly, or to facilitate the dynamic manipulation of the page at a later stage. For this reason after every visual block is acquired I perform some extensive clean up work using the CSS properties obtained. The properties chosen are those that have a good cross browser representation and are most commonly used by web designers such as *fontFamily, fontSize* and *fontWeight.* An example of this clean-up work mentioned above is that of going through the VBM and eliminating any block whose visibility property isn't set to "visible". Any such block would not be representing anything concrete on the page and is therefore irrelevant to this research. Some of the visual blocks in the Visual Block Model are larger than the rest and represent the main structure of the query result page. These particular blocks I will refer to as *container blocks*, each of which contains one or more *child blocks*. The remaining visual blocks, that is those that don't contain any other block, represent the individual data items displayed on the page and will be referred to as *basic blocks*. The four borders of each block represent its position on the page. These values are derived from the distance of each border to the origin of the two-dimensional plane, which is located in the top left corner of the browser with the x-axis running from left to right and the y-axis running from top to bottom.

3.2 Tag Tree and VBM

Although the tag tree of a web page and the Visual Block Model that we create are related, they are by no means the same thing. It is crucial to keep in mind while reading this paper that its main aim is to attempts to mimic human intuition in recognising the data records by adopting an all-visual approach. To achieve this, a visual representation of the page is needed, one that can facilitate the recognition of blocks that are spatially close together and visually resemble each other. The tag tree is a complex representation of the HTML code structure of the page in which, nodes that are found to be

close together, might be visually rendered at opposite ends of the page. Considering only the visual blocks in the VBM allows me to visualise the page in much the same way a person would. Furthermore the VBM model is a representation of the page that is much closer to the way it was designed: related items are spatially grouped together, which makes it easier to identify the data records. In addition to this, taking a visual approach and relying on a rendering engine to produce our model, allows for this solution to be shielded from changes to coding practices and standards. This means that it would not need to be adapted in the event of new coding developments.

3.3 Visual Blocks relationships

Now I will define some relationships between visual blocks both spatial and in terms of similarity. Firstly we will look at spatial relationships, which are crucial to recognise structural regularity of data records on a query result page.

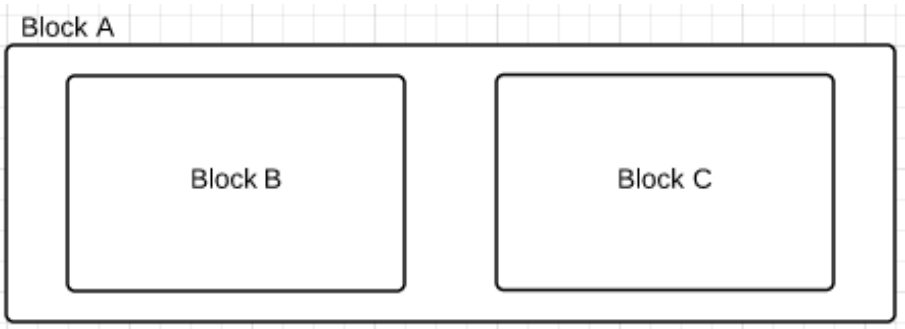


Figure 8 - Example of visual blocks contained within each other

Definition 1. ***Contains:*** *A visual block contains another visual block if at least one border is found inside the former and the remaining ones either overlap or are themselves found inside it.*

At the start of my research I assumed, much like some of the previous approaches, that, for a block to be contained inside another one, all its borders had to be found inside the former. However as I performed various experiments to come up with the above definition I discovered some fringe scenarios like the one showed in Figure 9 below.

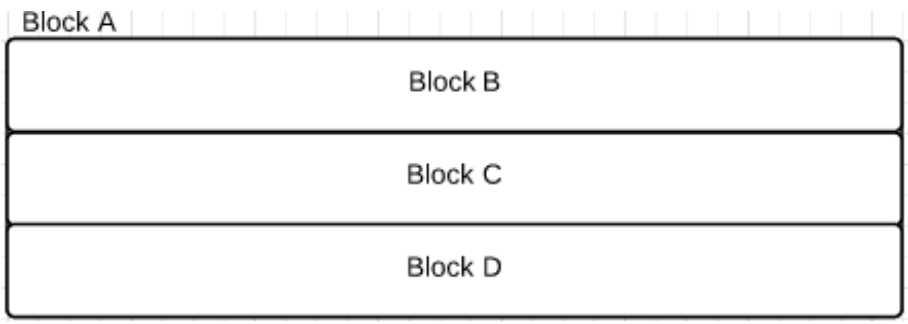


Figure 9 - Fringe scenario for contains

It is clear to see that in the case like the one above, an approach that would consider a block contained within another only if all of its four borders are found inside the former, would not be suitable. In the scenario showed in Figure 9, Blocks B, C and D are contained inside Block A while sharing some borders with it. For cases like these, the algorithm for finding a block that is contained within another needs to be more robust. Figure 10 shows the algorithm I developed for determining such spatial similarity.

```
contains(blockA, blockB)
{
   if all borders overlap
       return false

   if either all borders of blockB are found inside blockA or all borders of blockB except one overlap blockA
      and the remaing one is found inside blockA
      return true
}
```

Figure 10 - Contains algorithm

Definition 2. ***Visual Similarity:*** *Two visual bocks, A and B, are visually similar if all of the visual properties of both blocks are the same.*

The approach taken in this paper decides on visual similarity between two blocks by their visual properties obtained in the VBM creation stage. For instance, as shown in Figure 2, the visual blocks that contain the price of the book have the same visual properties and are therefore considered visually similar.

Definition 3. ***Width Similarity:*** *two blocks have similar widths if the width properties for both blocks are within a threshold of 3 pixels of each other.*

For instance, in Figure 3, the block containing all of the data items for the first book on the page has the same with as the one containing the data items for the second book. These blocks are therefore considered to have similar widths.

Definition 4. ***Container Block Content Similarity:*** *Two container blocks have similar block contents if they have a similarity threshold above a pre set threshold.*

In my approach I also need to determine whether or not two container blocks have similar content, that is if they have similar sets of child blocks. In determining this, there are a few observations to be made. First of all in a query result page data records are not guaranteed to contain the exact same number of data items. This means that, while it is a fair assumption to make that every data record will contain certain necessary items (such as the tile of a book or the price for it), there are also a number of data items that are optional and are found only in some of the data records (Figure 2 has an example of this with the

"new" and "used" price labels). From this follows that it would be incorrect to consider two container blocks to have similar contents if all of the data items found inside the visually matched. For this reason in my approach a similarity threshold is calculated based on the relationship between the number of child items matched and the total number of child items, which is used to determine container blocks content similarity. The threshold varies between 0 and 1, where one means the two container blocks are identical and 0 that they have nothing in common.

3.4 Data Record Extraction

Each data record on the page is represented by a visual block, which contains all the data items related to that data record and nothing else. I have completed a survey over 150 query result pages and observed that in 97% of them it is the case that there is a singe block in the page's structure that contains the data record information and nothing else. The goal of my approach is to identify this visual block for each data record on a query result page. I will refer to this type of block as *data record block*. Following this first task, the next tasks that follow from the work done in this paper are to align and then to annotate the data items from each record into a structured representation. My algorithm will attempt to achieve the data extraction by applying a number of similarity metrics in separate stages of visual block clustering: first it goes through every block and creates a Map of blocks with children; in the second stage it performs a one pass clustering of the visual blocks with children based on their width similarity; in the third stage it goes through each group of blocks obtained in stage two and measures, for each block in each group, its content similarity to the rest of the visual blocks in that group in order to calculate a similarity coefficient for the group; finally it adds groups with a similarity coefficient higher than a pre-set threshold to a list of what I will refer to as *candidate groups* and, out of those candidates, the group that has the highest number of recurring similar blocks and is the widest will be selected as the *final candidate*.

3.5 Stage I: Blocks with children

In this first stage I iterate through the visual block model, which in my Java representation is an ArrayList of VisualBlock objects, and, for each visual block, I retrieve its children, if any. If the block at hand has more than one child block (I use this check as a mean of removing items that aren't relevant since in my survey of over 150 query result pages I found that in 100% of the cases the data record block had more than one data item) then I add that bock as a container block to the data structure representing all blocks with children. Figure 11 shows the algorithm that I use to traverse the VBM and create the "blocks with children" list.

```
foreach currentBlock in allBlocks
{
    ListOfChildren = all children for currentBlock

    if ListOfChildren contains more than one block
    {
        add the currentBlock to the BlocksWithChildren map as the key and
        the ListOfChildren as the values

        mark currentBlock's basic block flag to false
    }
    if the ListOfChildren doesn't contain any block
    {
        mark currentBlock's basic block flag to true
    }
}
```

Figure 11 - BlocksWithChildren Algorithm

The Java construct that I make use of for storing the blocks with children is a HashMap in which the key is the parent block and the value is an ArrayList of VisualBlock objects representing the children. Figure 12 shows a graphic representation of this HashMap.

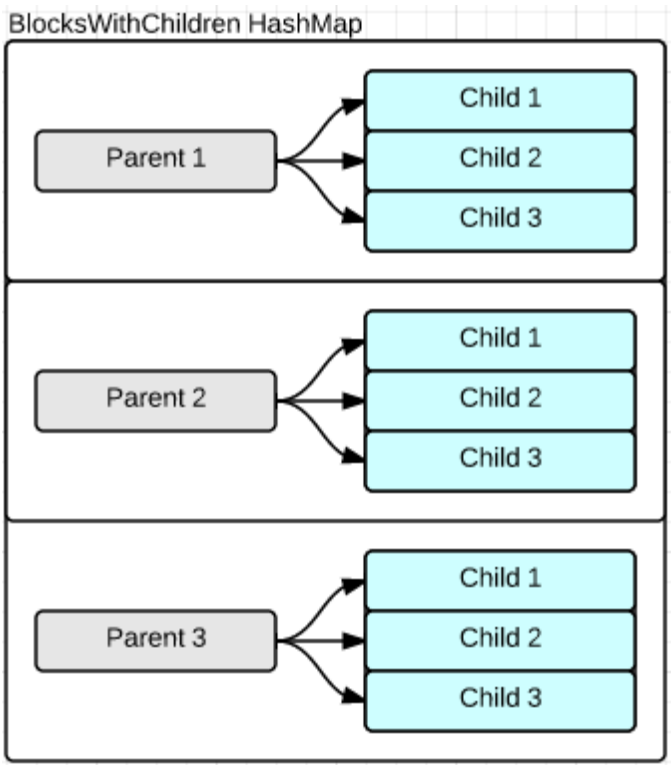


Figure 12 - blocksWithChildren map representation

This type of data structure allows me to easily access visual blocks and their children and it facilitates the next stages of my clustering.

3.6 Stage II: Width Clustering

In the second stage of the data record extraction process I iterate through the "blocks with children" HashMap and perform a one-pass clustering of the container blocks in terms of their width similarity. Figure 13 shows the algorithm I use to perform the clustering.

```
foreach block1 in the blocksWithChildren HashMap
{
    save block1 in a local variable.

    foreach block2 in the blocksWithChildren HashMap
    {
        if block1 and block2 have similar widths
        {
            if there are no groups yet or it doesn't belong to any existing group
            {
                create a new group and add block1, block2 and their children to it.
                add the new group to the allBlocksByWidth HashMap.
                remove block2 from the blocksWithChildren HashMap.
            }

            else
            {
                get the current group and add block1, block2 and their children to it.
                remove block2 from the blocksWithChildren HashMap.
            }
        }
    }
}
```

Figure 13 - Cluster by width algorithm

I add all the groups to a new HashMap containing all groups by width. I then use this in the next stage to select my candidate groups. I also discard any group that contains too few items. The conclusion I have come to, while surveying query result pages and testing the clustering algorithm is that, when clustering by width, many of the groups are noise groups, that is visual blocks that happen to be of similar width but are totally irrelevant. I was able to get rid of such groups by calculating a threshold to be used to discard groups that are too small. In the current version of my algorithm, any group that contains less than four items is discarded. This value can be considered very safe considering that each of the surveyed 150 query result pages presents an average of 15.6 data records. In the future some machine learning techniques can be employed to calculate a more accurate value. To test that my cluster by width behaved as expected I draw all groups and colour code them as to make it very clear to which group each visual block belongs. Figure 14 shows the VBM for a different query result page after the first two stages of clustering have been carried out.

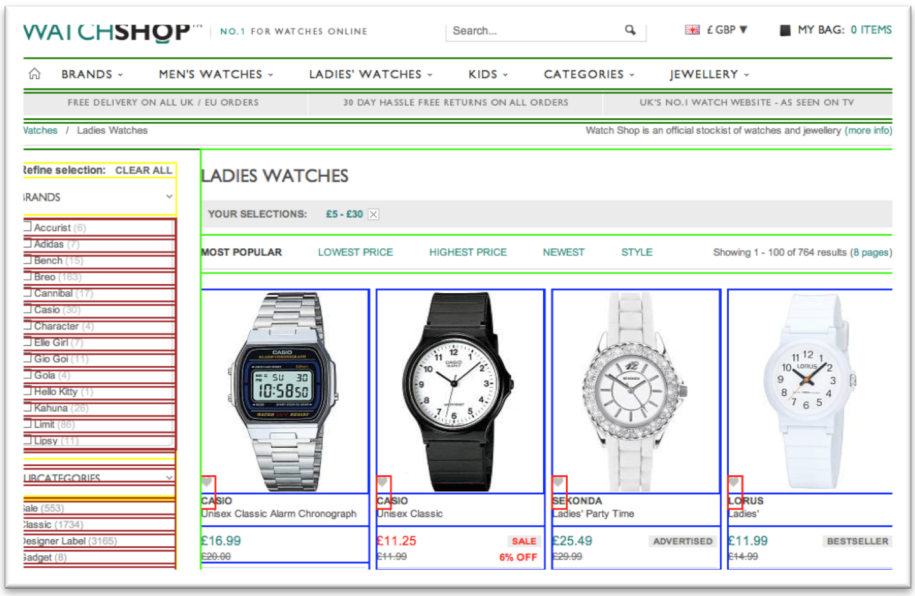


Figure 14 - VBM after first two stages

3.7 Stage III: Content Similarity and Candidates Selection

In the third stage of the extraction process I iterate through the HashMap containing the groups of visual blocks clustered by width in the previous stage and determine if the group members are visually similar in terms of their content and in terms of their children block's visual similarity in order to select accurately which blocks to add to my final list of candidates for the extraction. To achieve this I define a similarity function that takes two container blocks as parameters and determines whether or not their children are visually similar by comparing for each child block in the first container all of its CSS properties obtained in the VBM creation stage to the children of the second container to find a match. As mentioned beforehand, it is not possible, due to the number of optional items that a data record could present, to only match two container blocks if all of their children match. For this reason my algorithm first calculates a similarity threshold using the number of children that have been matched and the overall number of children of the container block, and then, if this threshold is higher than a pre set value, it will consider the two container blocks to be a match. The threshold value is currently set to just over a half. This value was obtained through the survey I carried out of the query result pages and empirical observation; it is not a stretch to consider two container blocks as visually similar if most of the basic blocks they contain are visually identical. Figure 15 below describes the algorithm used for determining container blocks content similarity:

```
for each group in allGroupsByWidth
{
    for each block1 in group
    {
        if block1 has already been looked at
            break

        for each block2 in group
        {
            if block1 and block2 overlap
                remove the outermost one from group

            if block1 and block2 visually are similar
            {
                keep count of how many blocks block1 is similar too
                keep count of how many blocks overall are similar in group
            }
        }

        calculate block1 similarity threshold
        decide if it should be removed from currentGroup based on threshold

        record that block1 has been looked at.
    }

    calculate currentGroup similarity threshold
    decide if it can be one of the candidates based on threshold

}
```

Figure 15 - Algorithm for Candidates selection

Once the above algorithm is run it will have determined which groups of blocks out of those that had been clustered together in the previous stage present similarities that go beyond width and can be added to the shortlist of groups that might be the record blocks.

3.8 Stage IV: Final Candidate Selection

In the fourth and final stage of the extraction process I use the output of the previous phase, which is an ArrayList of HashMap objects, which contains the groups that have met the minimum visual similarity threshold requirement, to select the final group that will be most likely to contain the record blocks. At this particular stage all candidate groups will contain visual blocks that resemble each other. The observation that I have made regarding data records on a query result page is that, in 98% of the cases I surveyed, a record block could be identified as the block on the page that had the highest number of "similar" blocks to it. From this I can obtain the first and main metric for selecting the final candidate: I will select the group that contains the highest number of recurring similar blocks. Using this metric alone however is not sufficient to guarantee that the group selected is the record blocks' one. This is due to the fact that, often, the basic data items that make up the data record are themselves sub-grouped in sub-container blocks. As a direct consequence such sub-containers will be all both of similar widths and visually similar in terms of block content. These will therefore be picked up as one of the final candidate group. For this reason simply selecting the group that contains the highest number of recurring similar blocks is not accurate enough as this property is not unique to the record blocks in all of the cases. If one used only this metric the results would be inconsistent

and would be completely dependant upon the order in which the candidate groups are examined. To resolve this issue I add a secondary selection metric: I will select the group that contains the highest number of similar blocks and is the widest. Both these metrics on their own are quite weak and not suitable to selecting the final candidate; combined however they provide a strong and reliable way to select the group that is most likely to contain the data records. Figure 16 helps to visualise this.

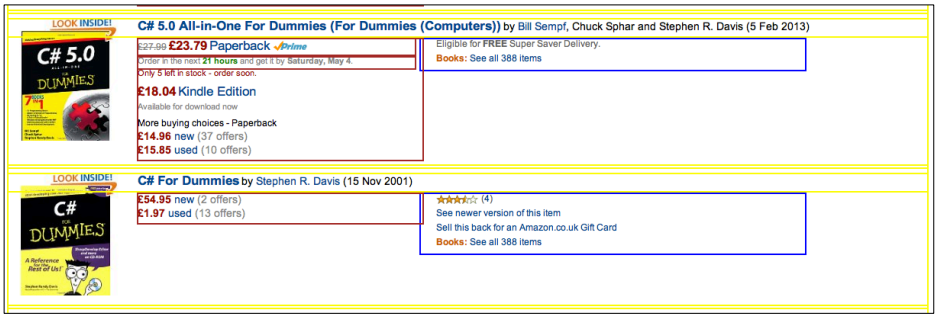


Figure 16 - Query Result Page with candidates colour coded

As seen above the list of candidates contains similar blocks. It is clear to see that using only one of those metrics would be unsuitable and how the use of both of them provides a reliable criterion for the final selection. Figure 17 shows the algorithm used to perform this final selection.

```
for each candidateGroup in Candidates
{
     if the size of candidateGroup is equal to the current biggest size
    {
        if the width of the candidateGroup is greater than the current biggest
            {
                reset the the current biggest width to zero as to make sure that
                this group's width will be bigger than the current biggest in the
                final check
            }
    }

    if (the width of candidateGroup is greater than the current biggest width AND
       the size of candidateGroup is greater than the current biggest size)  OR
       the size of candidateGroup is greater than the current biggest size
       {
        candidateGroup's width becomes the current biggest width
        candidateGroup's size becomes the current biggest size
        candidateGroup becomes the current group to be returned
       }
}
```

Figure 17 - Final Candidate Selection Algorithm

Once selected the final candidate group will be rendered on screen. Blue borders will be drawn around container blocks and red borders around basic blocks.

4. Experimental Results

I have implemented my algorithm for data record extraction from query result pages in a software prototype using Java. In this section I will first of all describe the dataset that I have used to test the prototype and then determine how accurate it is. Finally I will explain some of the metrics used to interpret the results obtained. As a point

of reference I compared my solution to ViNTs [6], a highly regarded extraction system available online. ViNTs [6] uses some visual features in the extraction but it mainly relies on HTML and tag tree structure to identify the data records. My results show that the new approach proposed in this paper, which relies solely on visual similarities and is completely decoupled from the query result page's code structure, achieves significantly higher accuracy than approaches such as ViNTs [6].

4.1 The Dataset

The first thing to consider when considering data record extraction experiments is that there is no standard dataset for experimental analysis. For this reason the dataset I use comprises of websites taken from third part search engines (such as Google, Yahoo and Bing) and were selected from the list of websites I compiled while surveying the 150 query result pages. Due to time constraints and to the fact that for some quite complex pages the computation time is quite high, I was only able to run my tests on a third of the surveyed pages. The websites used include the most popular e-commerce sites online at this particular time (such as Amazon and eBay). The dataset also presents a lot of variety in terms of the style and structure in which the data records are displayed. I was careful to select the websites that presented these variation in order to really put to the test my "all visual" approach and to make sure that the solution I built was not domain or data-record-layout specific. The web sites in our datasets are drawn from a number of domains, including Books, Movies, Shopping and Properties. In total the dataset used contains over 2000 data records.

4.2 Performance Metrics

I use a number of performance metrics for my experiments:

- ***Ground Truth***: The set of data records from all of the web pages collected per web site.
- ***True Positives***: These are data records extracted correctly from all of the web pages per web site. Ideally, the true positives are the same as the ground truth for each web site.
- ***False Positives***: These are data records extracted incorrectly from all of the web pages per web site. Ideally, the number of the false positives should be zero.
- ***Precision:*** This is the number of true positives divided by the number of both true and false positives per web site. The average precision across web sites is calculated by averaging the precisions of individual web sites.
- ***Recall:*** This is the number of true positives divided by the number of data records in the ground truth per web site. The average recall is calculated by averaging the recalls of individual web sites.

4.3 Experimental Results

The results below were gathered by firstly extracting manually for each query result page, all the data records; then I executed both the prototype I developed and ViNTs [6] on the dataset and recorded the number of correctly extracted data records and incorrectly extracted data records for each page. I consider a data record successfully extracted if it contains the all the visual blocks that are contained in the corresponding record on the query result page. Finally, I calculate the average precision and recall for all the websites in the dataset. The results are presented in Table 1 below.

Algorithm	Precision Average	Recall Average
myPrototype	93%	97.2%
ViNTs	45%	31.14%

Table 1 - Results comparisons with ViNts

*Note: The experiment were run on a MacBook Pro running OS X Mountain Lion 10.8.3 and using Eclipse version 4.2.1

4.4 Experimental Analysis

4.4.1 Considerations on ViNTs

As shown in Table 1 the performance of my prototype is considerably better than that of ViNTs. In analyzing the results produced by ViNTs for the dataset I noticed that ViNTs identified a large number of false positive data records. This contributed greatly to their low precision score. By inspecting these false positives, I discovered that ViNTs frequently selected the wrong sub-section of the page as the data-rich section (i.e. the region of the page which is most likely to contain the data records). For example, they often selected a page navigation menu or the side filters for a query as the data rich section, and then extracted each menu item or filter as a false positive record. In these cases, it was then impossible for their technique to extract the correct data records. Consequently, the number of true positive data records they identified was also small, which contributed to their low recall score. This is not a criticism of ViNTs. Their technique worked very well on the web pages that were available when ViNTs was developed; rather it serves to highlight that modern web pages are vastly different from older web pages and how, for these newer pages that present a much more complicated structure, approaches such as ViNTs [6] don't perform very well.

4.4.2 Considerations on Prototype

There are a few considerations to be made regarding the prototype I developed in order to correctly understand the results shown above. Firstly let us consider the positive aspects. My prototype was able to identify all data records across the vast majority of the query result pages tested and in those cases it extracted little to no blocks that were noise or false positives. This evidence supports the hypothesis and the claim made in the introduction that, in extracting data records from query result pages, the reliance on the code structure of the page is the wrong approach to take and it produces inconsistent and negative results especially when it comes to analyse more modern and dynamic web pages. It also highlights the fact that the visual approach taken in this paper by far provides more consistent and positive results. While the overall technique and methodology used for the data record extraction are sound and produce very good results, there are a few aspect of my prototype that need fine-tuned and that can generate some negative output. I have been aware of such imperfections as I developed and tested the algorithms and they are what I will perfect in the future. These have to do with various thresholds that are manually set in my prototype and whose value has been obtained solely from my own empirical analysis.

4.4.2.1 Width Considerations

The first aspect to consider is related to the concept of width similarity: two considerations have to be made when trying to determine what does it mean for two blocks to have similar widths:

1. By saying that two blocks have similar widths only if their respective widths are equal to each other is too strict. In my investigation I have found that in a number of query result pages, some data records present slight variations in width (usually 1-2 pixels). For this reason, on those pages the extraction process would exclude certain data records if it made use of the above approach to determining width similarity.
2. If we decide, as I did, to establish a threshold for width similarity at least two issues arise:
    a. How to determine the value for the threshold is the most obvious one. I currently have a value set which has been obtained from experimentation. Ideally however some machine learning techniques could be employed to calculate a more accurate value.
    b. The second issue is a subtler one. The thing to keep in mind about the way in which data records are structured on the page is this: web developers will, in the majority of the cases, group the data items that make up the data records in a number of sub-container

blocks. These can vary both in width and in terms of number of elements they contain. It is sometimes the case that two or more of these sub-containers will have widths values that fall within the width threshold discussed above. In such a scenario all these sub-containers will be grouped into their own "groupByWidth" during stage two of my clustering and consequently most likely make it to the final list of candidates since they all will look virtually identical to each other. At that stage, since my primary metric for picking the final candidate is to select the group with the highest number of recurring similar blocks, such the sub-containers group (being higher in number than the data record group), will be selected as the final candidate producing the wrong output.

The first possible solution to the above problem is, as mentioned above, to use some machine learning techniques to calculate a threshold value that is as accurate as possible for the highest possible number of pages.

4.4.2.2 Content Similarity Considerations

Another solution to the above issue, which also helps me to consider the next imperfection of my algorithm would be to perfect the content similarity function both in terms of which visual features it makes use of (by doing further investigation and selecting new visual properties), and by introducing machine learning techniques to set the content similarity threshold more accurately. As mentioned before, I consider two blocks to be visually similar if most of their children are visually identical. In this instance "most" means setting the content similarity threshold to a value that is just over half. While this produces good results in the majority of the cases it does produce some wrong results on certain specific pages in which, mostly due to an unusual web page structure, two data record will be erroneously considered visually similar. To improve the accuracy of the content similarity function would also help in eliminating noise blocks throughout the clustering process thus increasing the overall algorithm accuracy.

4.4.2.3 Implementation Considerations

One final consideration I want to make is one regarding the specific programming language and data structure I used, how that affects certain aspects of my algorithm and how it could be improved. In the current version of the prototype all collections representing the relationship between containers and child blocks are stored using Java HashMap objects. This collection type allows me to build a tree-like structure and correctly represent the different correlations between

blocks. However it does present certain limitations. The main one I have encountered is one that is related to counting the items in the collection: in the final stage of the clustering process, I have to determine which group, from the list of candidate groups, contains the highest number of container blocks. However, the built in "size()" method for this Java collection, which returns every key-value pair in the HashMap, is not suited for my specific objective. This is because, as mentioned above, certain basic blocks are grouped within sub-containers. Figure 18 will help to better understand this:

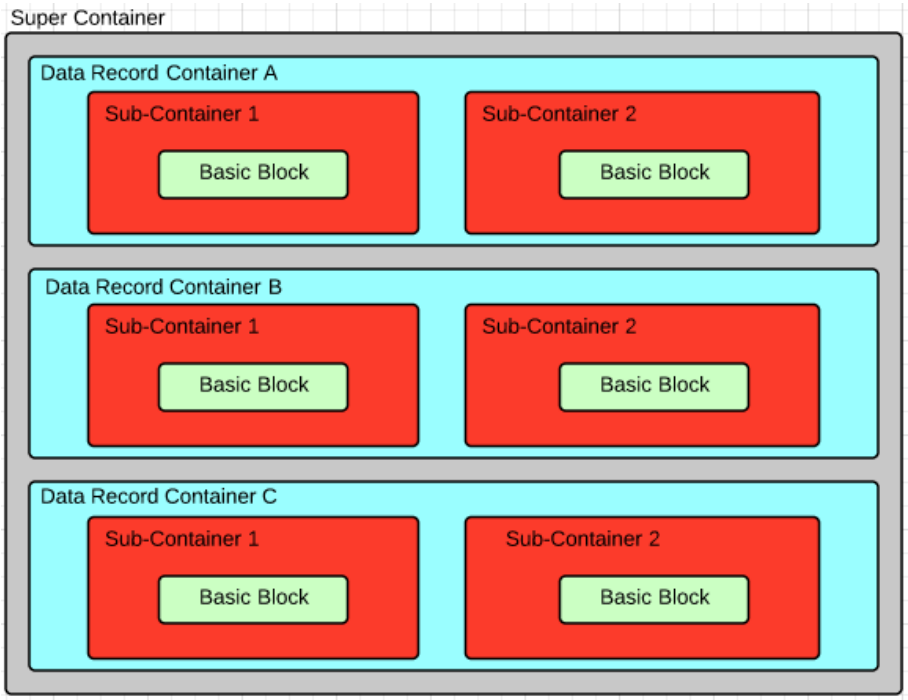


Figure 18 - Containers and Sub-Containers

The figure above shows a possible structure for the candidate group containing the data records. The important thing to keep in mind is that each data record block above is both a "value" and a "key" in the HashMap. It is a "value" as it is found inside the super container, and a "key" since it contains sub-containers. In the same way, the sub-containers are both "values", as they are located inside the data record, and "keys" since there are basic blocks contained in them. As a result of this, it is very clear to see that the HashMap built in "size()" method that returns a count of every key-value pair is unsuitable. My algorithm is interested in a count method that will return three (as there are three data records) for the above structure. If it were to use the built in method however it would be returned the value ten as it that is how many key-value pairs are present in Figure 18. For this reason I resorted to writing a custom function which would ignore any basic-blocks, super and sub-containers and return the correct value for the group's size. In order for the function to remove the super-container however an issue arises: a super-container is a container that has "too many" children. In the current iteration of the prototype that translates into a value derived from the survey I conducted and my own empirical findings. Again the accuracy of this could be improved

by adopting some machine learning techniques. One other possible solution is to consider possible alternatives to the HashMap. This new data structure could be a different built in Java tree like collection or one that is built from the ground up and completely tailored to the problem at hand.

5. Conclusions

In conclusion this paper presents a new approach to data record extraction from query results pages. My approach relies on an "all visual" methodology, which attempts to recognise the data records in terms of their visual similarity. The prototype developed first creates the visual block model and then, through various stages of visual blocks clustering, is able to isolate the data record blocks that are the most recurring visually similar and widest blocks. My results show that this approach is highly accurate and effective. It is in fact able to recognise the data record blocks on the vast majority of query result pages. The experimental results also highlight how an approach that is solely visual such as mine, is far more suited to the task of extracting data record from more recent, dynamic and complex query result pages than some of the more out-dated attempts that try to isolate the data records through analysing the tag tree and the HTML structure of the page. Extensive testing of the prototype has also highlighted certain imperfections that are all implementation specific. These are all related to the manual setting of certain thresholds and to the use of specific data structures. In future work I plan to address these issues by resorting to machine learning techniques to calculate the thresholds in a much more accurate manner and to further investigate which are the data structures that could eliminate some of the issues that arose during the development of the prototype.

6. References

[1] **C.H. Chang C.N. Hsu, and S.C. Lui,** *"Automatic Information Extraction from Semi-Structured Web Pages by Pattern Discover"* s.l. Decision Support Systems, 2003, Vol. 35, pp.129-147.

[2] **LiuY.Zhai and B. Liu,** " Web Data Extraction Based on Partial Tree Alignment". 2005. Proc. Int'l World Wide Web Conf. (WWW). Pp.76-85.

[3] **V.Crescenzi and G. Mecca** "*Grammars Have Exceptions",* s.l. Information Systems, 1998, Vol. 23, pp.539-565.

[4] **N. Kushmerick**, *"Wrapper Induction: Efficiency and Expressiveness"*. 1/2, s.l. Artificial Intelligence, 2000, Vol. 118, pp.15-68.

[5] **A.Sahuguet and F. Azavant** "*5. Building Intelligent Web Applications Using Lightweight Wrappers"* s.l. Data and Knowledge Eng, 2001, Vol. 36, pp.283-316.

[6] **H. ZhaoW.Meng, Z. Wu, V. Raghavan, and C.T. Yu** "*Fully Automatic Wrapper Generation for Search Engines"*. 2005. Proc. Int'l World Wide Web Conf. (WWW). Pp.66-75.

[7] **K.Simon and G. Lausen** *"ViPER: Augmenting Automatic Information Extraction with Visual Perceptions".* 2005. Proc. Conf. Information and Knowledge Management (CIKM). Pp.381-388.

[8] **J. ZhuZ.Nie, J. Wen, B. Zhang, and W. Ma,** *"Simultaneous Record Detection and Attribute Labelling in Web Data Extraction".* 2006. Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDD). Pp.494-503.

[9] **Wei LiuXiaofengMeng, Weiyi Meng,** *"ViDE: A Vision-Based Approach for Deep Web Data Extraction"*, s.l., IEEE Transactions on Knowledge and Data Engineering, 2010, Vol. 22.

[10] **D. CaiS.Yu, J. Wen, and W. Ma,** "Extracting Content Structure for Web Pages Based on Visual Representation". 2003. Proc. Asia Pacific Web Conf. (APWeb). Pp.406-417.

[11] **NielsenJakob,** *"Top Ten Web-Design Mistakes."* [Online][Cited: 6 November 2012] http://www.useit.com/alertbox/20021223.html.

7. Appendix