Fully automated web harvesting using a combination of new and existing heuristics

Date: July 6th, 2014
First Supervisor: Maurice van Keulen
Second Supervisor: Djoerd Hiemstra
Company Supervisor: Waldo Yntema

Niels Visser

Abstract

Several techniques exist for extracting useful content from web pages. However, the definition of ‘useful’ is very broad and context dependant. In this research, several techniques – existing ones and new ones – are evaluated and combined in order to extract object data in a fully automatic way. The data source used for this, are mostly web shops, sites that promote housing, and vacancy sites. The data to be extracted from these pages, are respectively items, houses and vacancies. Three kinds of approaches are combined and evaluated: clustering algorithms, algorithms that compare pages, and algorithms that look at the structure of single pages. Clustering is done in order to differentiate between pages that contain data and pages that do not. The algorithms that extract the actual data are then executed on the cluster that is expected to contain the most useful data. The quality measure used to assess the performance of the applied techniques are precision and recall per page. It can be seen that without proper clustering, the algorithms that extract the actual data perform very bad. Whether or not clustering performs acceptable heavily depends on the web site. For some sites, URL based clustering outstands (for example: nationalevacaturebank.nl and funda.nl) with precisions of around 33% and recalls of around 85%. URL based clustering is therefore the most promising clustering method reviewed by this research. Of the extraction methods, the existing methods perform better than the alterations proposed by this research. Algorithms that look at the structure (intra page document structure) perform best of all four methods that are compared with an average recall between 30% to 50%, and an average precision ranging from very low (around 2%) to quite low (around 33%). Template induction, an algorithm that compares between pages, performs relatively well as well, however, it is more dependent on the quality of the clusters. The conclusion of this research is that it is not possible yet using a combination of the methods that are discussed and proposed to fully automatically extract data from websites.

Table of Contents

1. Introduction

The Internet is an ever changing source of information. This information is mostly offered in human readable format, like rich formatted web pages. However, this information can also be used by computer programs, for example to aggregate, compare or evaluate different data sources. This would pave the way for a whole lot of practical and useful opportunities, for example creating a portal which indexes all second hand car sales within a certain radius of your residence.

The one problem here is that, in most cases, the information is structured very loosely, which makes it very hard for a computer program to discover a structure in a web page and to find the right information. And even if that would succeed, the structure of the web page might be updated after the structure discovery took place, forcing the program to redo the whole discovering process.

This research is commissioned by SQUAL. SQUAL is an innovative web development and internet marketing company located in Enschede, carrying out projects for companies of various sizes and locations throughout the Netherlands. One of the in-house projects of the company is to develop a mobile application and web site that harvests information from online shops, in order to extract price data on demand. This price data will be used to alert customers when the price of a certain product drops below a defined threshold. The outcome of this research will be used to develop an algorithm that is capable of such data extraction, in order to incorporate it into the aforementioned project.

1.1. What is Web Harvesting?

Web harvesting can mean a lot of things, all boiling down to the process of extracting information from the internet. In this research, we focus specifically on automated object extraction from semi-structured web pages containing a single object. Now, a lot of terms in that sentence need to be defined in order to know the exact focus of this research. With automated object extraction, the process of extracting object data without human intervention is meant. An “object” is defined as the most important collection of data on a web page. For example, on a web shop, that would be a product and on a vacancy site, that would be a vacancy. Only detail pages are considered holding objects, overview pages (such as search result pages) are not considered as object-containing pages.

With semi-structured, it is meant that the object is presented in a structured way, although this structure is not known beforehand, and might not be structured the same way between web sites, or even between web pages on the same web site. In some cases, only part of the data of the object is offered in a way (a structure) that is feasible for a computer to read or scrape, regarding the current state of technology. For example, plain text, in which the properties of an object are explained in human language, is not feasible to scrape in most cases. As opposed to semi-structured, fully structured data means that all of the data of the object that we want to scrape is offered in a way that is easily readable by a computer, such as JSON or XML.

So, summarized, the definition and scope of web harvesting in this research, is extracting objects from arbitrary web sites that offer a certain kind of objects, like vacancies, products or holidays.

During this research, the simplification will be made that all objects consist of label-value pairs, the labels being the properties of the object and the values the corresponding values of these properties. In the real world, this might be a more hierarchical construction (categories of properties, and properties having the same name, but belonging to different categories). This is done to limit the scope of this research.

1.2. Problems with Web Harvesting

Of course, a lot of effort is already put in (automated) web harvesting by research facilities like universities and big companies like Google. Also, a lot of literature can be found on different approaches to (automated) web harvesting. As can be expected, each approach addresses a different problem. For example, some literature addresses the problem of recognizing structures on a single web pages, and other literature addresses the recognition of important parts of web pages by trying to build a template from a set of web pages originating from the same web site. Nonetheless, most of the web sites that actually perform web harvesting, do this through pre-defined data structures. Even Google uses opengraph tags in order to extract product data from web pages. Of course, not all data that is displayed on a Google result page is delivered through pre-defined structures: data like opening hours is extracted using heuristics that do not require a pre-defined data structure. But what is the main problem with web harvesting? The answer is simple: not everyone offers their data in standard formats. And as if it is not hard enough to detect data-rich structures on web pages, data is often delivered in conveniently ambiguous formats (like an old and a new price for a product, or multiple prices on one page). This makes it really hard for a computer to distinguish between data that is actually of import, and data that is not part of the object that we want to scrape. While it is easy for a human to classify this data, a computer has a really hard time doing this. During this research, we will distinguish between the following types of problems:

- Structure problems: problems that have to do with how the website is structured, the placement of items, etc.
- Classification problems: problems that have to do with the ambiguity of data, for example: how easy it is to see that some label-value pair represents the price of the product.
- Aggregation problems: problems that have to do with splitting or combining different properties of objects in order to achieve something meaningful

Furthermore, another problem is that not all web pages contain an object according to the definition as stated earlier in this document. This can be seen as a pre-classification problem: the web pages need to be classified or clustered in order to determine which pages contain objects and which pages do not.

A more detailed analysis of problems that may occur during the automatic scraping of web pages, can be found in [1]. There, a set of web sites is analysed in order to find problems that stand in the way of recognizing information, extracting information, clustering web pages, scraping information from multiple pages and more. The main conclusion of the research is that there is only little research done on actual problems like ambiguity, the presentation of data and the amount of data that is shown on information-rich pages. Furthermore, the research proposes a method for further research, which will be the method used in this research.

1.3. Method and Structure

The method that will be followed in this research, is the method that is proposed by [1]. The first steps of this method are already completed in the aforementioned research. The steps that remain will be executed in this research. As the title of the research says, the research did a field study on the problems that could occur during automatic information harvesting on web pages, and the solutions that current literature and research offers. Therefore, it is already (partly) known what problems to expect when executing the steps that the research proposes. The research does suggest an implementation that covers the process from crawling to object integration (combining multiple objects in order to achieve an as complete as possible database). In this research, the following parts of the proposed method will be executed (marked in yellow):

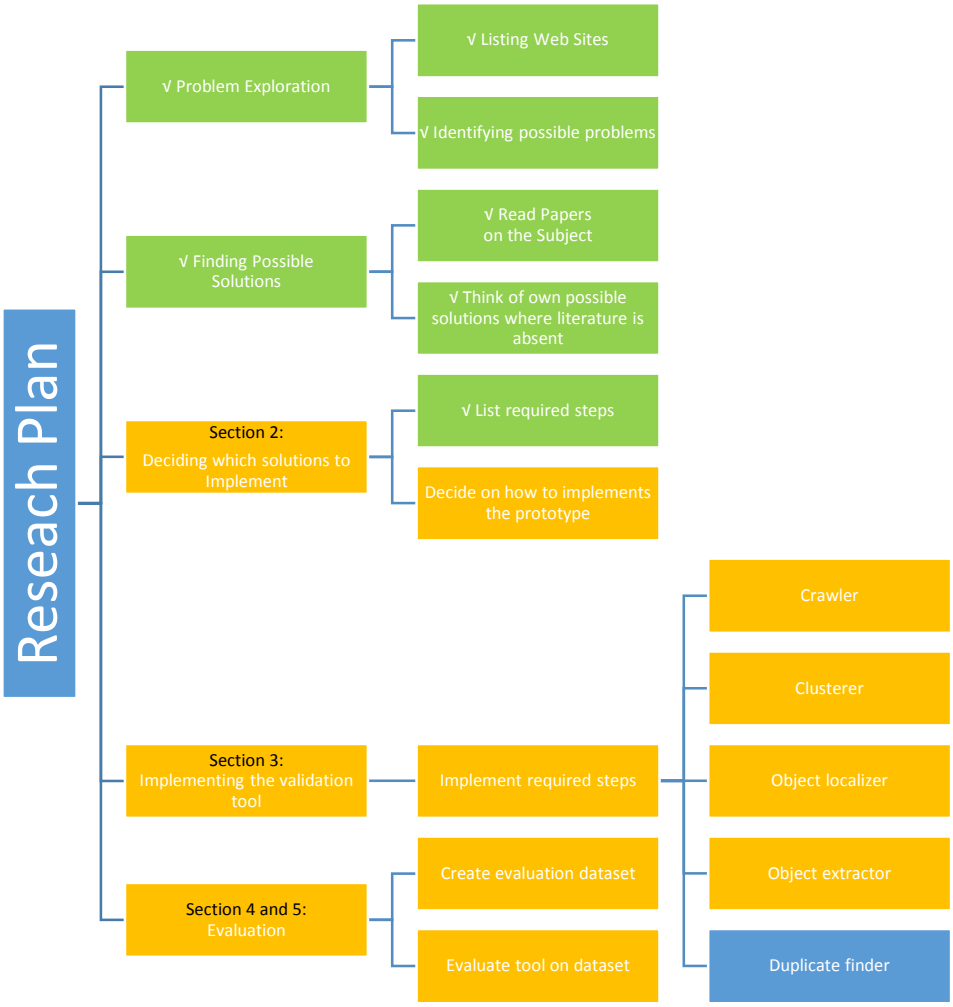

Figure 1: Research plan

The green parts are already executed in [1]. It should be noted that the most basic implementation will be used for the crawler, since the focus of this research lies on localizing and extracting objects.

The structure of the paper will be as follows (as is displayed in Figure 1):

First, a selection of approaches is made, based on existing research and intuition in chapter 2. This selection is made from [1], and is further extended with ideas originating from the context problem mentioned in the same document. Further, the different approaches are explained in detail. The goal is to give an overview of the methods that are used, and explain them in such detail that they can be reproduced in future research.

1.4. Research Questions

The following questions will be answered during this research:

- RQ1: What combination of existing and new information extraction methods can be used for fully automated object extraction from detail pages, originating from web sites that offer objects in a semi-structured way?
    - RQ1.1: What current methods exist for automatic extraction of objects from web sites that offer objects in a semi-structured way?
    - RQ1.2: What problems (partly) remain after applying existing methods for automatic extraction of objects from web sites that offer objects in a semi-structured way?
    - RQ1.3: What new methods can be thought of to solve the issues that could not be (wholly) covered by existing methods for automatic extraction of objects from web sites that offer objects in a semi-structured way?
    - RQ1.4: How sensitive are existing methods for automatic extraction of objects from web sites that offer objects in a semi-structured way to the structure of the content?
    - RQ1.5: How can existing and new methods for automatic extraction of objects from web sites that offer objects in a semi-structured way be combined best in order to achieve best accuracy?

1.5. Solutions to Implement

In order to answer the research questions, a prototype for dynamically crawling, clustering and scraping web pages will be developed and evaluated. The development of the prototype will embody most of the solutions as mentioned in [1].

2. Method

2.1. Solution Categorization

In order to achieve the goal of this research, existing research is used as well as ideas derived from existing research and brand new ideas. The problems that are faced when doing fully automated web harvesting, are categorized into three main categories, each divided into multiple sub-categories. This way, it is possible to categorize existing literature and new ideas easily, so that the solution to each of the main categories can be built from the ideas and research belonging to that category.

The three main categories were derived from the structure (as can be seen in Figure 19 later on). As can be seen, there are basically two approaches: inter-page and intra-page analysis. Since inter-page analysis is done between pages of the same type, the set of pages from a web site need to be categorized or clustered.

Below, the schematic representation of the structure of the categories is displayed. As can be seen, three sections and seven sub-sections are defined in order to categorize the ideas and solutions.

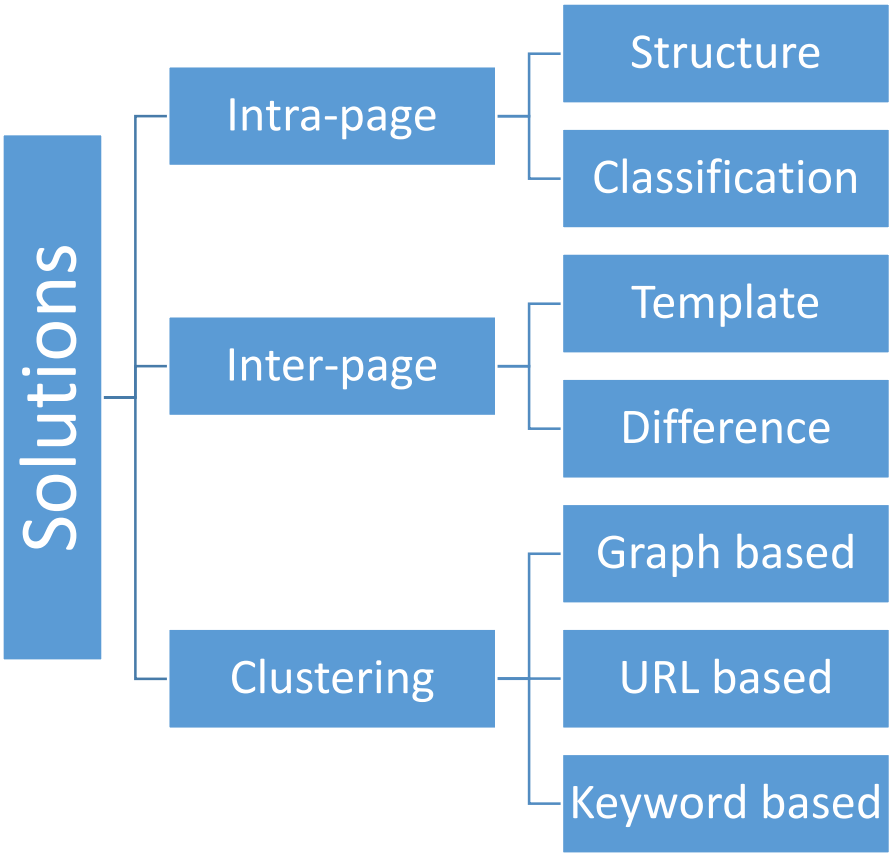

Figure 2: Overview of the categorization of existing literature and solutions

2.2. Implementation

In this section, the implementation of the theories that are used in the proposed method (the web harvesting method that this research proposes) will be explained. The structure of this section will be as follows: First, a global description of the implementation structure will be given. The design that is used to embody all modules of the solution will be explained. After that, the baseline implementation will be explained, which is the implementation to which all of the other

implementations will be compared. This implementation will be the most basic one, and all other implementations should logically be an improvement over the baseline implementation. Subsequently, the modules that are used to create the complete solution will be explained one by one.

2.2.1. Required Components

The prototype consists of six components:

- Crawlers
- Single-page extraction modules
- Clusterers
- Multi-page extraction modules
- Result combiners
- Evaluators

Each component generates output that can be used by another component, also known as 'pipe-and-filter'. However, in classic pipe-and-filter patterns, each output can only serve one input. In this implementation, a distinction is made between processors, splitters and mergers. Processors take an object as input, convert it into another object or alter the object in some way, and offer it to the output. Processors do not always require input, it is also possible for a processor to solely generate output, as is the case with crawlers. Splitters clone objects, so that one object can be offered to multiple processors. Mergers merge different objects into one object. How that is done, depends on the implementation.

In this way, a network of processors, splitters and mergers can be built. Throughout this document, it is referred to as a 'processing network'. Such a processing network usually starts with nodes (processors) that generate web pages (for example by crawling a web site).

The functionality of each component is as follows:

Crawler

The crawler is responsible for delivering the pages of a web site to the other components of the system. The following techniques will be distinguished during this research: **exhaustive crawling**, **structured crawling** and **other** techniques. Exhaustive crawling means following all links on a web site (that stay within the web site) until no new links can be found. Note that this may pose a problem when links on a web site are generated, because there may be an endless amount of links and web pages. Structured crawling is based on a certain structure of the web site that is known beforehand. In other words: when one knows where the links that are pointing to web pages that contain interesting data are located. These locations may be search result pages, or product category pages, for example. The scraper will only scrape the links on those pages, when using this technique. Other techniques may include link delivery through a database source or other user generated sources.

Single-page extraction modules

These components focus on recognizing structure on a single web page. This may be locating tables that adhere to certain rules, or finding locations in the DOM-tree that have certain properties. In the implementation used in this research, this will be a set of heuristics that is combined later on. Examples of these heuristics are: machine learning algorithms that are trained on recognizing objects that contain useful data, or heuristics such as: heuristics that try to find an xpath that occurs multiple times on a web page, thus likely pointing to an element in a repeating structure.

Clusterers

Clusterers are components that pre-process a set of web pages in order to enable multi-page extraction modules to compare pages of the same type. This process can be seen as categorization. The module tries to create categories (without knowing how many or what kind of categories exist beforehand). In this research, there will be distinguished between three different kinds of clusterers: **network-based** clusterers, **URL-based** clusterers and **structure-based** clusterers. Network-based clusterers use information based on the network graph of all web pages from a single web site. Since there is information about the type of web page to be gathered from the web pages it links to, and web pages that link to it (for example: search result pages will most often point to product detail pages), it may be used for clustering. URL-based clusterers are the most basic ones among the clusterers that are considered in this research: this technique uses the structure of the URL to cluster web pages. Since sub-categories, or product detail pages may be located after a slash that comes after the link to the parent category, all web pages that only differ in the last part of the url (the part that comes after the last slash, disregarding any trailing slashes) are likely to belong to the same category. Structure-based clustering focuses on the DOM structure of web pages. Since web pages of the same category usually look alike, the DOM structure for those pages should have a lot of overlap. Structure-based clustering uses this information to categorize pages based on overlapping DOM structures.

Multi-page extraction modules

Multi-page extraction techniques work a bit like structure-based clusterers: they exploit the fact that pages from the same category usually have a lot of overlap in the DOM structure. Usually, they assume that regions that change in a set of pages that belong to the same category, contains non-static content, and may contain object data.

Result combiners

Both methods, multi-page extraction modules and single-page extraction modules, deliver an object that is scraped from a web page. The goal of the result combiner is to integrate these objects in order to form a more complete object (for example: the single-page extraction module may find some properties that the multi-page extraction module does not find, and vice versa).

Evaluators

The evaluator will evaluate the quality of the scraped object, based on another method of scraping web pages or a pre-generated corpus. Basically, the method that is proposed in this research will be compared to the evaluation method.

2.3. Clustering

Clustering is the key to successful inter-page information extraction. If the clustering is not done right, the inter-page approaches will not yield any usable results. This is because of that the inter-page object extraction that are used in this research, have to compare between pages that are of the same type. When types of pages are mixed (for example: result pages and detail pages), no global structure may be found by the inter-page object extraction methods.

First of all, the following definition of clustering will be used throughout this chapter and the rest of the research:

> ***Definition 1****: Let S be the collection of all pages belonging to website W. Since not all pages on a web site may be crawled, the set of crawled pages S_v will be a subset of S. Each page $s \in S_v$ will be assigned to exactly one cluster C_x with $x \in 1..n$ and n being the number of clusters that the web site is divided into,*

with $1 \leq n < \infty$. n will depend on the solution that is used to cluster the web pages.

Second of all, after the pages are clustered, it is needed to identify the cluster that will act as input for the inter-page analysis. How this cluster is identified may vary from solution to solution.

The methods that are used for clustering the web pages, are chosen and implemented in a way that they assign each page to exactly one cluster. This way, there will be no overlap in clusters (e.g. overlap is when a page is assigned to two or more clusters). This is done because of the nature of the clusters that the web pages are being divided into: per definition, a page belonging to one cluster cannot belong to another. For example: a page will be either classified as a detail page or a search result page. Besides the fact that this behaviour of the clustering algorithms is desirable, most of the algorithms enforce this by applying a *one vs. many* structure. This means as much as having a chain of "*this page either belongs to this category, or to some other category"*-decisions, which always result in one category of cluster per object.

2.3.1. Solution 1: Keyword based

Clustering can be done by analysing the contents of the web pages you want to cluster. For example: web pages that have content that is alike, are likely to be related to each other in some way. The solution proposed in this section tries to cluster pages that have content that is alike. This is done using suffix tree clustering. Normally, a suffix tree is created for a single string, in this solution, a suffix tree will be created for the entire web page. Essentially, the suffix tree clustering technique places web pages that share a lot of sentences into the same cluster.

This idea is used in several researches to cluster documents. [2] Other research uses TF-IDF in combination with K-means clustering, in order to form clusters. However, in [2], the authors only take keywords from the URL, the title tag and meta tags into account. [3] This is not a feasible option for intra-site page clustering, since it is very likely that the meta tags of different web pages of the same web site contain the same content. Of course, the method could be based on just the title tag and the URL, but these two sources contain less words than the meta tags by far.

A word-based Suffix-Tree Clustering is used in order to cluster documents together, as proposed by [4]. As opposed to a normal suffix tree, a word based suffix tree will is built upon words rather than characters. (So words are treated like characters.) A normal Suffix Tree (of a string) is a compact trie in which all suffixes of that string are contained. Suffix trees were first proposed by [5]. Following every path from the root node to each of the leaves and concatenating the strings on the way results in a covering set of substrings of the string that the suffix tree is created for.

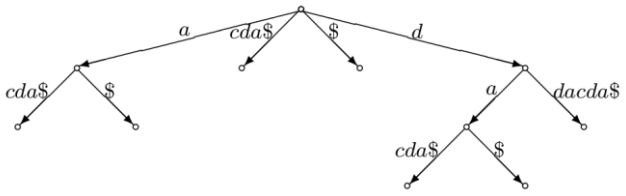

Figure 3: Suffix tree for ddacda$, as shown in [6]

For example: the suffix tree for ddacda$ is shown in Figure 3. As can be seen, all substrings can be created from following a path from the root node to one of the leaves (a, da, cda, acda, dacda and ddacda). In each leaf, the documents containing the sequence in question are present.

In [4], the whole web page is considered as one “string”, with the words on the page considered as letters. So the suffix tree algorithm will never split words across edges in the tree.

This process of creating a word-based suffix tree is done for each web page. After that, all suffix trees will be combined into one big suffix tree. When paths of two suffix trees overlap (meaning that they have share a path from the root node to an arbitrary node in the tree), both pages (documents) are added to all nodes on the shared path.

After creating this combined suffix tree, a score is calculated for each node in the tree. The score is equal to the number of documents in the node, times the length of the sentence (in number of words, so “This is a web page” will have a length of 5), with a maximum value of a parameter called c. (So if the length is > c, the value used in the calculation will be c.)

After creating this combined suffix tree, an undirected graph is constructed on top of it. Two nodes (vertices) are connected if two conditions hold:

- One of both nodes is ranked r or higher;
- The size of the set of pages that overlap between the nodes in question, divided by the biggest node, is higher than fraction f.

This leaves us with three parameters:

- c: the cap in the number of words
- r: the rank at which a node is included in a cluster
- f: the fraction of pages that should overlap within a node

Finally, two nodes (and thus the containing pages) belong to the same cluster, if the nodes are connected. (I.e.: there is a path from one node to the other.)

2.3.2. Solution 2: PageRank inspired

PageRank, as used by Google, is an algorithm that ranks web pages based on links that point to that web page, and the PageRank score that the pages that link to the page in question have [7] . The goal of PageRank is to rank pages by their significance, in a way similar to the significance ranking of scientific articles. PageRank is a graph-based, iterative ranking algorithm. The idea behind PageRank is that it takes two important metrics into account: the number of pages that link to the page in question (a metric that has existed even before PageRank was invented), and the importance of the pages that link to the page in question. In older information retrieval systems, only the number of pages that link to the page in question (hereafter referred to as “inlinks”) is taken into account, if anything graph-based was even done. This leaves out important information, since it may say more when, for example, the front page of yahoo.com is the only one that links to the page in question, than when hundreds of home.shadyprovider.com/users/~p.johnson/myfirstwebpage.html kind of sites link to it. Therefore, the research proposes a method that lets the “importance” of pages iteratively propagate though the graph that is the internet.

The idea behind incorporating the PageRank algorithm in this solution, is to cluster pages based on importance with regard to categories, and based on structure of a web site. This is done by letting the information that exists on web pages inside a web site propagate to other parts of the web site (web pages of that web site). In essence, this is a smoothing step, in order to reduce the number of clusters that result from the clustering step explained later on, and to increase the quality of the clusters. The information that is propagated to other web pages, consists of words and a certain score per word. The idea is that pages that link to another page, are related in some way. Think about breadcrumbs, category pages and search result pages, for example: the pages that link to the

same page in a breadcrumb, have a category in common (due to the reason breadcrumbs are used). Pages that are reachable from the same category page, most likely have a category in common as well. The clustering step uses overlapping information (words) on web pages to create clusters. This is the reason that propagating and sharing information between web pages result in a lesser number of clusters, due to an increased overlap in information. This may seem a self-fulfilling prophecy, because a certain overlap is created due to the information propagation step, but part of this overlap is discarded based on certain conditions.

Research suggests, similarity between textual content on a web page can be used for clustering purposes [8] . Besides that fact, it may be argued that sub-categories of web pages contain a subset of the content of their parent category. From this perspective, a PageRank like approach may be used in order to propagate the subjects of the web pages through the graph. The original (simplified) PageRank formula is as follows:

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

Formula 1: Simplified PageRank formula

$R(u)$ is the PageRank score of page u. c is an arbitrary constant, used as a damping factor (so it is smaller than 1). B_u is the set of pages that link to u (the inlinks). N_v is the number of outlinks (links that point from the page in question to other pages) of page $v \in B_u$. What this formula says in plain text is: the PageRank score of a certain page is evenly divided among the pages that it links to. This sentence implies that that certain page loses its score when that score is divided among the pages that it links to, but this is not the case. The score is kept and increases with the (partial) scores that it receives from pages that link to it.

In order to use a PageRank derivative for the purpose of this research, the following definition is stated:

***Definition 2**: Let L_P be all words from page p. Let $L_{p,20}$ be the top 20 words from page p. In this definition, it is assumed that $|L_p| \geq 20$.*

Normally, the score that is produced by the PageRank formula, is a score for the significance of a web site. In our case, we use the formula in order to let PageRank generate a score for the significance of a web *page* within a web site, *with respect to a certain word.* Normally, all sites that PageRank is executed on, are seeded with a certain score to start with. In our case, it is the TF-IDF score for the word that is investigated, on that web page.

The PageRank derivative is executed for each $l \in L_{p,20}$. This results in the propagation of top-20 words through all web pages of a web site. On each word, the Porter stemming algorithm for (in this case) Dutch words is used [9], in order to resolve words that have the same meaning (plurals, for example) to the same string. Eventually, the following set of word-page combinations is created, with each element having a certain score:

$$C = \left\{ (l, p) \mid l \in \bigcup_p L_{p,20} \wedge p \in S_v \right\}$$

Definition 3: Set of word-page combinations, every combination having a score

This comes down to: there exists a certain score for every combination of every (unique) word on the web site and every web page on the web site.

The PageRank formula (and thus the derivative as well) needs a certain seed: a score to begin with, assigned to a certain set of seed pages. Since the PageRank derivative is executed for each (unique) word on the web site, the TF-IDF score of the word in question will be used as a seed for each web page (of course, this score is different for every web page, due to the number of occurrences of that word on that web page). A certain threshold α is used in order to filter the word list: all words below that threshold will receive a score of 0. This is done in order to reduce the complexity of the algorithm. The result of this measure will be that words like “de” or “het” (the equivalent of “the”) will receive a score of 0, thus being left out of the algorithm. If, during the seed phase of the iteration, all pages are seeded with 0, the iteration is skipped and the word is not taken into account any further. At the end of all iterations, each page will have a score for each word from the web site. This score is used to create a ranked list of words for each web page of the web site.

This new ranked list will be referred to as $\mathrm{L}'_{p,20}$. This is the new ranked list (possibly containing other words than $L_{p,20}$) as generated by the altered PageRank algorithm. The format is the same as $L_{p,20}$, the only difference is that it contains new words. Then, for each element in the set of

$$R = \{\{p_a, p_b\} |\, p_a \in P \wedge\ p_b \in P \wedge p_a \neq p_b\}$$

Definition 4: Set of page - page pairs

a score is calculated, creating a graph with a vertex for every page, and an edge for every element $r \in R$. This score indicated the similarity between page p_a and p_b. Please note that the graph is undirected, since r is an unordered pair. Logically, the operation to calculate the score for two pages (hereafter referred to as $p_a \bowtie p_b$) is symmetrical. (See appendix A for proof.) Let $l_{p,n}$ be the n-th keyword in the list $\mathrm{L}'_{p,20}$, with $n = 1$ as the highest ranked word and $n = 20$ as the lowest ranked word from the top 20. The score $Q_{i,j}$ of the combination p_i and p_j (with $p_i \in P \wedge p_j \in P$) is calculated as follows:

$$Q_{i,j} = p_i \bowtie p_j = \sum_{q=1}^{20} \sum_{r=1}^{20} \begin{cases} (21-q)(21-r)\ if\ l_{p_i,q} = l_{p_j,r} \\ 0\ otherwise \end{cases}$$

Formula 2: Page score calculation

This score is used as weight for the edge between p_i and p_j. After creating this graph, two methods may be considered for dividing the graph into connected components that make up the clusters. First of all, the graph may contain a number of connected components, since there may be a lot of edges with a score of 0. As a filtering step, if the number of connected components in the graph is greater than a certain parameter m , the smallest connected component (measured in the number of vertices) is discarded until the graph consists of m or less connected components. m should be chosen in such a way, that only the components that are negligible are discarded (for example, connected components consisting of single pages, of which the algorithm is not capable of assigning to a cluster).

The two methods that are mentioned, consist of one method that takes a certain parameter q which is the maximum number of connected components that the algorithm divides the graph into. Of course, this number should be greater than m. The second method divides the graph into a number of connected components, based on a cost function.

The reason why first, components are discarded and then components are broken up, is that before discarding components, there are components in the graph that are not necessarily part of a cluster by definition of the method used (for example, pages that have very few information on them, or pages that have nothing to do with other pages, like a page describing the legal terms). The reason for breaking up the components again, is that q is the parameter for expecting that the site consists of q 'big' clusters. It may be the case that there is a certain weak link (edge) between two clusters, because a page just happens to have some similarity with another page from another cluster. Breaking up the clusters as described should fix this problem.

The first method is actually quite straightforward: the threshold for sustaining an edge between two vertices is raised until the number of connected components is equal to q. If it so happens that that is not possible (this is the case in which there are multiple edges which have the same score or weight), the threshold is lowered just below the weight of those edges.

The second method is a bit more complicated: it uses Kernel Density Estimation [10] [11]. This method is used in combination with the fact that similarity functions tend to either produce a high score or a low score, e.g. there are not many scores in between. The method as described finds the least dense point of a set of points spread along a (one-dimensional) axis. The value associated with this point will be the threshold for regarding two pages as similar enough.

Below, an example is given, with $s(p)$ defined as the weight of edge p. For every point, a Gaussian kernel is used.

For example, take five edges, one with weight 1 (so $s(p) = 1$), one with weight 2 and one with weight 3, one with weight 7 and one with weight 8. First, a Gaussian kernel is applied to the three edges, resulting in the following graph:

Figure 4: Visualization of a gaussian kernel being applied to each score

After that, the functions are summed, so the result (which is defined as $f(x)$) will look as follows:

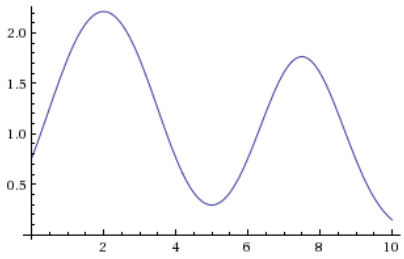

Figure 5: Combination of all gaussian kernels

Then, the minimum of $f(x)$ is determined, which is at x=5. Now, all edges with a score < 5 are discarded, resulting in a bigger number of connected components.

The scores of all pairs of vertices (the weight of the edges) are distributed along a one-dimensional axis. Each point on the axis is given a distribution (in this case a Gaussian kernel) and all these distributions are added. In more detail, the following is done:

$$f(x) = \sum_{p} ae^{-\frac{(x-s(p))^2}{2b^2}}$$

Formula 3: Gaussian formula as applied to the scores

In which the variables a and b are parameters to be chosen, $p \in P$ is the edge in question (so the summation is done for every edge) and $s(p)$ returns the weight of edge p. The resulting function $f(x)$ is minimized between $s(\min(P))$ and $s(\max(P))$. The resulting value for x is the threshold for the edge weight of two vertices being connected.

The result of this method can be used in the same way as the method that was proposed at first (guessing the number of clusters and discarding edged until the desired number of clusters is reached), but the threshold is dynamically chosen. This results in the generation of clusters from the graph that is constructed with the PageRank method.

2.3.3. Solution 3: URL based

The third clustering method used in this research is based on the hierarchical structure that is present in URLs. There are some sophisticated methods that use machine learning (Support Vector Machines) for this [12]. But this proposed method will focus on a form of simple pattern recognition in the URLs, based on [13]. The latter research states to have higher P/R measures. The proposed method will cluster pages based on the frequency of appearance of parts of the URL. So, parts of a URL that are present in a great number of URLs on a web site, will be used to generate a pattern, which in turn is used to cluster the web pages. For example, given three URLs, (shop.org/products/brick.html, shop.org/products/hammer.html and shop.org/products/saw.html) the method will look for parts of the URL that are present in a predefined fraction of the URLs (in this case the part shop.org/products/ and html) and will generate a pattern (in this case: shop.org/products/*.html, in which * is a wildcard). This pattern is executed on all URLs of the web site, and pages are assigned to a cluster accordingly (so each pattern will generate a cluster).

To achieve this, the URLs are tokenized at first. The research is unclear about what tokens are used [13], so the following tokens are used in this research (regular expressions):

- [^/]+ (this token splits a URL based on slashes, each slash may represent a level in the hierarchy that is presumably used in web sites)
- [^\?]+ (this part splits a previous token into a request part and query string)
- [^&]+ (this part splits the query string into query parameters)
- [^=]+ (this part splits query parameters into keys and values)

This list may be extended in future work, in order to use more fine grained tokens. The patterns are executed one after the other, breaking up tokens into smaller ones. The end result will be a list of lists of tokens. After that, lists of tokens will be compared to each other. This is done by counting overlapping sublists. So the tokens are counted in a hierarchical way: a path to a token may appear in other URLs as well. The number of appearances of similar paths is counted for that path, for example: in [www.shop.org]/[products]/[product.php]?[id]=[123] and [www.shop.org]/[products]/[product.php]?[id]=[124], the path [www.shop.org]/[products]/[product.php]?[id] will have a count of 2, the path [www.shop.org]/[products]/[product.php]?[id]=[123] will have a count of 1 and the path

[www.shop.org]/[products]/[product.php]?[id]=[124] will have a count of 1. After that, the longest path with a count that is greater than $\frac{n}{p}$ (in which n is the total number of URLs and p is the number of clusters that the web pages need to get divided into), a pattern is generated. So in this case, a pattern www.shop.org/products/product.php?id=* will be generated. This means that all pages of which the URL starts with www.shop.org/products/product.php?id= will be assigned to the same cluster.

2.4. Inter-Page Object Extraction

2.4.1. Solution 1: Using Template Induction

For extracting objects using information from a cluster of web pages, the research of [14] is used for the first implementation. The research describes a method for extracting data from web-sites, based on the assumption that most of the web sites use a template engine. The proposed method tries to distinguish between content that originates from templates and content that originates from the database (assuming an object is displayed on the pages in question), by comparing detail pages. The method also assumes that the pages that it is offered are part of the same category (like: result pages, front pages, detail pages, portfolio pages, etc).

The method consists broadly of two phases or stages. The first stage aims to find information that is not part of the template, and the second stage tries to deduce the template that was used for generating the pages. In this context, a stage or phase is regarded as an isolated step in the process of extracting the information from the aforementioned cluster.

Stage 1 of ExAlg (Constructing equivalence classes)

The first stage conceptually tries to group parts of web pages together that originate either from the database or from the template of the page. The input of this stage is a group of web pages of the same type (all detail pages, for example), and the output is a set of groups, each group containing either parts of the web pages originating from the database, or parts of the web pages originating from the template the web page was built from. A group cannot contain a mix of parts from both the database and template.

A group consists of ‘tokens’: a token is an elementary unit on a web page: a word or a tag. This means that the web page is split into parts like ‘<p>’, ‘’, ‘the’ and ‘car’. Therefore, even before the first stage is entered, the web pages must be tokenized, or to put it more blunt: chopped up into tokens. This stream of tokens is thereafter fed to the first stage of ExAlg.

For example, consider the following four dummy pages:

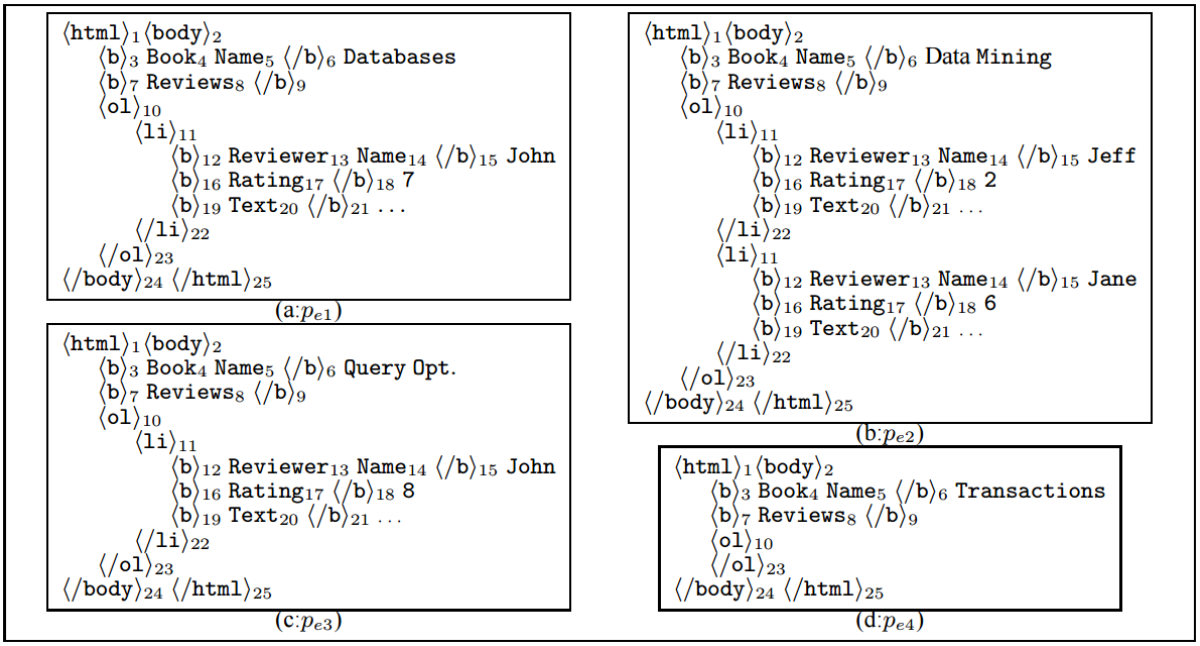

Figure 6: Example input pages

In Figure 6, all tokens are numbered, and corresponding tokens have the same number on all pages. Please note that this numbering is not part of the algorithm, and is done purely to make the example clearer. All tokens have a certain occurrence vector, which is a summary of the occurrences of that token on all pages. This can be seen in Figure 7: <body> has an occurrence vector of <1,1,1,1>, because its occurrence is 1 on all four pages.

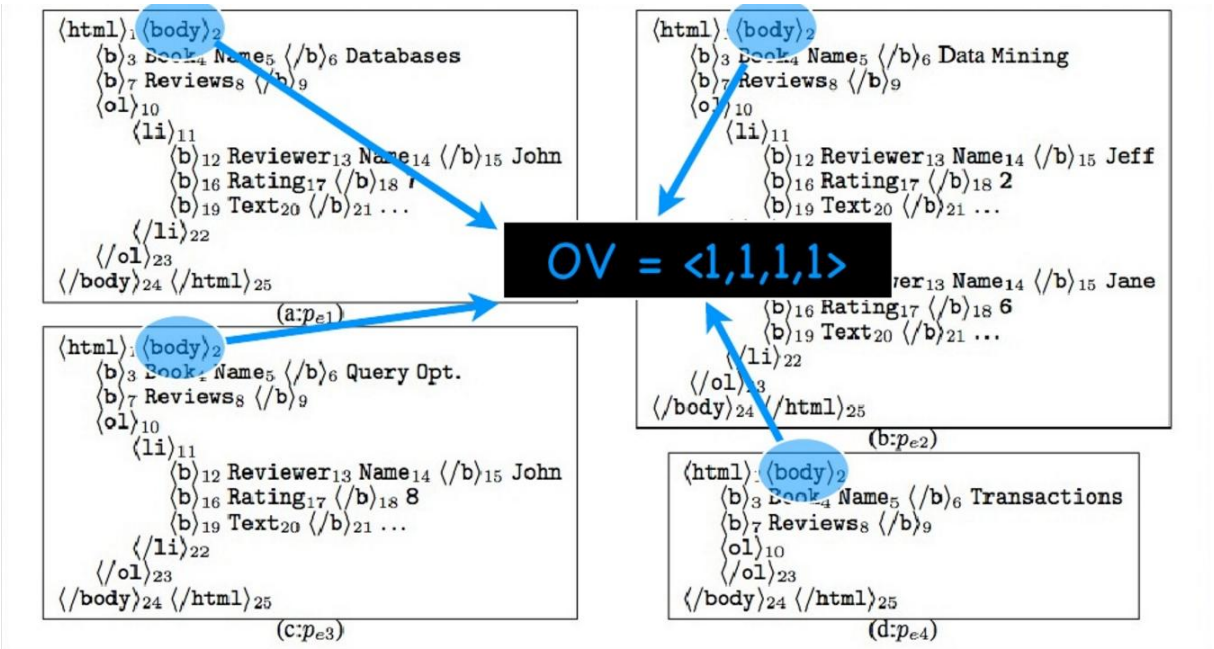

Figure 7: The occurrence vector of token <body>

The token “Reviewer” has a more interesting occurrence vector:

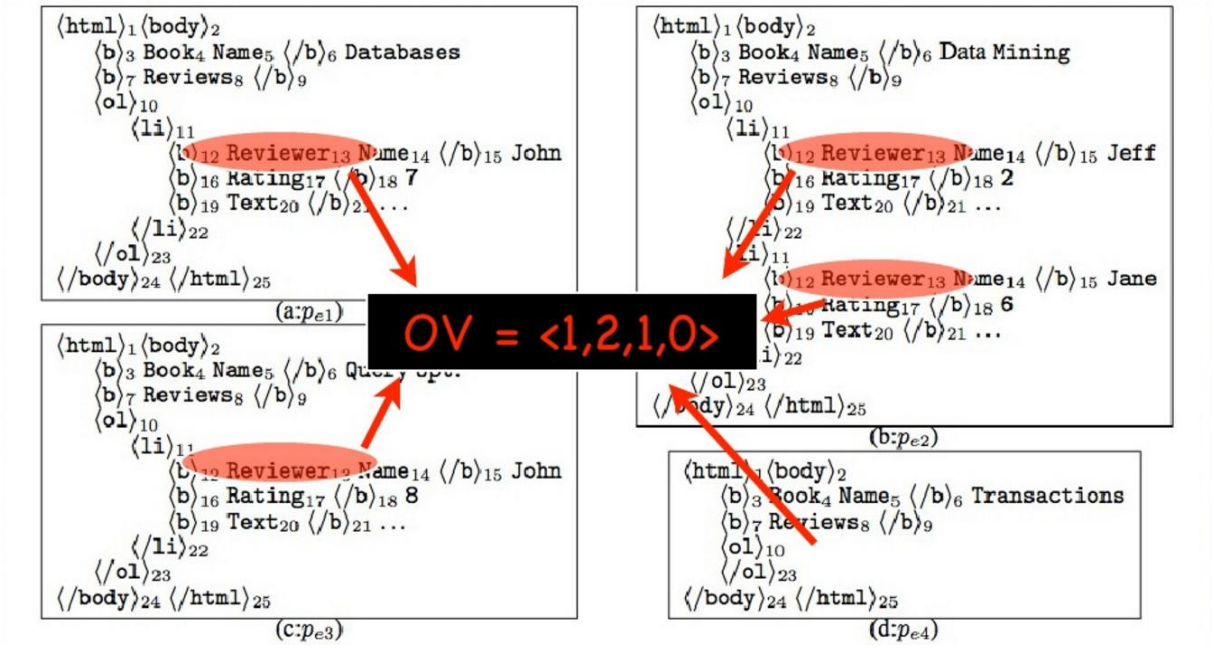

Figure 8: The occurrence vector of token "Reviewer"

During the first stage, the method (called ExAlg) determines equivalence classes. An equivalence class is a set of tokens (words, html-tags) that have the same frequency of occurrence for every page. Or, in other words, have the same occurrence vector. An example can be: `{<html>, </html>, <head>, </head>, <body>, </body>, ...}` *since all of these only appear once on every page*. Two dfferent tokens do not necessarily have to have the same occurrence between different pages, the only condition is that they have the same occurrence within a web page. If that is the case for multiple pages, the tokens are put into the same equivalence class. For example, the tags <table class="detail"> and <thead class="detail-header"> may occur twice on page A, and three times on page B. Since both tags *have the same occurrence on page A (e.g. 2 times)* and *have the same occurrence on page B (e.g. 3 times)* they are put into the same equivalence set. If that is the case for every page, the tag is likely to originate from a template (it just so happens that two tables are rendered on page A, and three tables are rendered on page B). As a counterexample, the tags and <p> may accidentally have the same occurrence on page A (let's say 5 times), but they do not have the same occurrence on page B (let's say occurs 4 times and <p> occurs 6 times on page B). Therefore, they are **not** put into the same equivalence class. This is most likely because these tags are used as markup for texts on the web page. And these texts most likely originate from a database. In our example, the elements that would be put into one equivalence class can be seen:

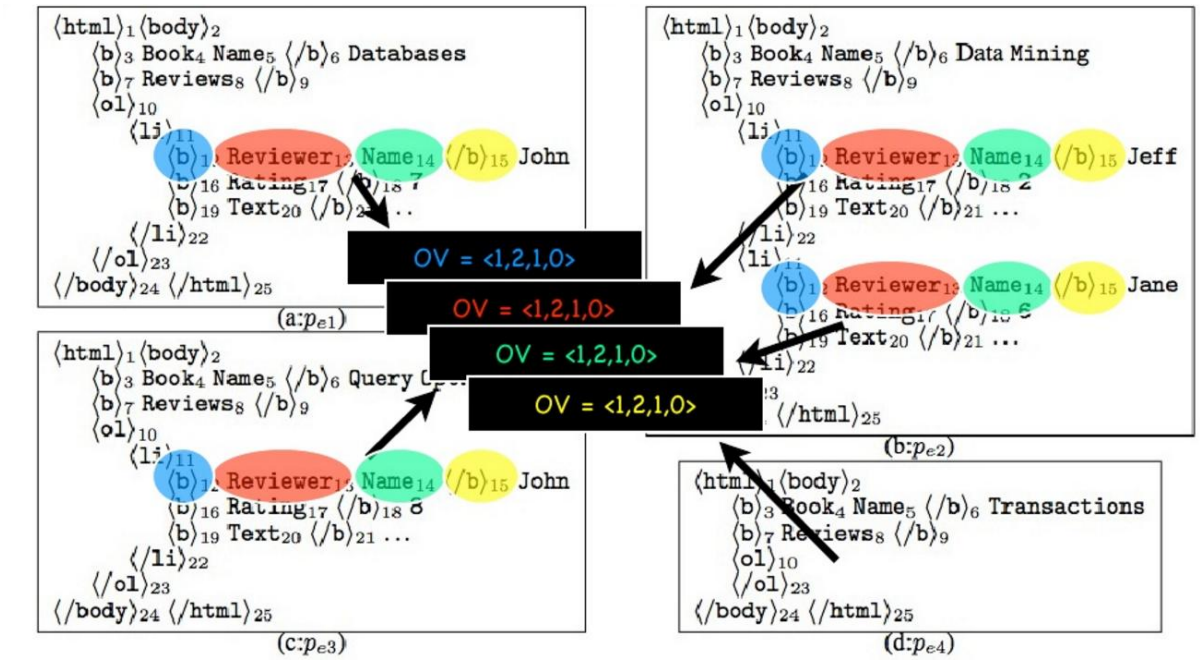

Figure 9: A set of tokens that will end up in the same equivalence class

So, given three different pages P1, P2 and P3, and tokens T1 and T2, T1 and T2 belong to the same equivalence class *iff* the following holds: both T1 and T2 occur x times in P1, y times in P2 and z times in P3, x, y and z being arbitrary natural numbers. Equivalence classes that do not meet a set of criteria, are discarded. This set of criteria is described later on.

The process of assigning tokens to equivalence classes is done in multiple iterations. One iteration consists of creating new equivalence classes and merging equivalence classes. The reason why this is done in multiple iterations, is because it has to be taken into account that the content of web pages is built hierarchically. For example, the token 'price' may occur multiple times on a page, once in a table and once in a description. The description may vary over pages, while the table exists on all pages. When differentiating between these 'price' tokens, it is possible to assign both to different equivalence classes. One iteration is done per level of the DOM tree, mimicking the hierarchical nature of the DOM structure.

The first stage ends, if no tokens are left to assign to a new or existing equivalence class.

Second Stage of ExAlg (Inducing the template)

The second stage of ExAlg aims to build the template. It starts with the root equivalence class: the equivalence class of which the elements occur exactly once on every page. In the aforementioned example, this is most likely the set of `{<html>, </html>, <head>, </head>, <body>, </body>, ...}`.

ExAlg then searches for neighbouring tokens: tokens within this class with no other tokens in between. For example: <html><head>. If there are no tokens in between, the neighbouring tokens are merged into a single string (this is part of the template building process). If there are tokens in between, and these tokens do not belong to an equivalence class, these tokens are considered to be originating from the database. If there are tokens in between belonging to an equivalence class, these tokens are recursively analysed as described above: tokens are merged if there are not tokens in between, and if there are tokens in between, it is decided based on the presence of an equivalence class whether they originate from a template or from a database.

During the first stage of ExAlg, equivalence classes that do not meet a certain set of criteria are discarded. This set consists of the following criteria:

- Equivalence classes must be ordered. “Ordered” meaning that all tokens from the same equivalence classes always occur in the same order in every page. Of course, when a page contains repetitions of the equivalence class, the tokens in each repetition must occur in the same order on every page. This is called the “ordered” property of an equivalence class. For example: given two elements A and B of a certain equivalence class, if the first occurrence of B is after the first occurrence of A, then the i-th occurrence of B should always be after the i-th occurrence of A, but before the i+1-th occurrence of A (except for the last occurrence of B). So the following sequence would be ordered: ABABAB, and the following sequence would not be: AABBABAB. Please note that a sequence of ABABABA is not subject to deciding whether or not it is ordered, since A has another frequency than B, and thus they would not end up in the same equivalence class.
- Each pair of equivalent classes must be nested. A pair of equivalence classes is nested if one of the two statements is true:
    - None of the instances from one class overlaps with the instances of the other class. Overlap is defined as that the first element of an instance of class A appears before the first element of an instance of class B, and the last element of the instance of A appears after the first element of the instance of class B. If this is true for A and B or for B and A, the pair of A and B is said to be overlapping.
    - Given instances of two classes, A and B, if an instance of A occurs in an instance of B, all elements of the instance of A must always be at the same position within B. A position within an equivalence class is defined as the space between two consecutive elements of an instance of an equivalence class. This position is denoted with a number, so position 1 is between the first and second element of an instance of a certain equivalence class. Position 2 is between the second and third element of an instance of a certain equivalence class.

In a few words, these conditions enforce that the tags or words of an equivalence class are always in the same order on every page, and that there is no overlap between equivalence classes in the sense that a tag of an equivalence class does not occur between tags of another equivalence class, but if two classes do overlap, then one should be enclosed in the other, and always be at the same position within the other.

To wrap up, after the first stage of ExAlg, a set of equivalence classes is constructed. These classes contain data that originates from a template. Based on these equivalence classes, a template will be constructed (reverse engineered) that is used to extract data in a structured way from pages that use that template.

The construction of the template is done recursively. There is one root equivalence class, that is the class which has an occurrence vector that is <1, 1, … , 1> (depending on the number of pages, the size of the vector varies, but the values are always 1). Since all classes are nested and ordered (see the aforementioned definitions), it is known where all children of the root equivalence class are positioned. These children are placed in the root at the corresponding positions. This step is repeated for the children of the children, and so on. In the constructed template, it is taken into account that

some elements of certain equivalence classes may occur zero or more times on a page. These are marked as repeating regions. In our example, this would yield the following template:

```
<html>
    <body>
        <b>Book Name</b> $content
        <b>Reviews</b>
        <ol>
            <li repeat>
                <b>Reviewer Name</b> $content
                <b>Rating</b> $content
                <b>Text</b> $content
            </li>
        </ol>
    </body>
</html>
```

Figure 10: Constructed template

In Figure 10, it can be seen that certain content originates from a database (market with $content), and certain regions are repeated (marked with the 'repeat' property). This constructed template is applied to any pages that data will be extracted from. It is applied in a template-greedy, content-lazy way: the template, including any repeating regions is applied as far as it goes to a web page. Once it hits a $content area, as little content as possible is matched before continuing to apply the template. This can be compared to regular expressions, in that case, the template would look as following:

```
<html>
    <body>
        <b>Book Name</b> .*?
        <b>Reviews</b>
        <ol>
            (<li>
                <b>Reviewer Name</b> .*?
                <b>Rating</b> .*?
                <b>Text</b> .*?
            </li>
)*      </ol>
    </body>
</html>
```

Figure 11: The template, expressed as a regular expression

The operators used in the regular expression are: "." (dot), which matches all characters, "*" (star) which stands for "zero or more of the preceding token", "?" (question mark) for lazy matching (regular expressions are greedy by default, trying to match as many characters as possible while still matching the whole expression, the lazy-modifier will complement that property, resulting in an expression that tries to match as little as possible). The last operators that are used, are the parentheses, "(…)", which group a set of characters together.

Now, extracting data from web pages based on this template becomes trivial: one only has to capture the parts denoted with ".*?".

2.4.2. Solution 2: Difference based

The second solution that is proposed for extracting objects from web pages using cross-page information, uses textual difference algorithms. Textual difference algorithms are often used to detect and display changes between documents, in such a way that it is best human readable

(meaning that a balance between instructions for replacing entire paragraphs and replacing single letters is reached). However, the texts that are usually compared, are flat (not hierarchical) whereas the data that is used in this research (web pages) is hierarchical.

The hierarchical nature of the content that is analysed, would imply using an edit distance such as the tree edit distance. This method compares two tree structures and yields a set of instructions to go from one tree structure to the other. This may seem like a feasible and existing solution for the goal that is described here, but the set of instructions would involve modifying the DOM tree, instead of just the text which leaves the structure of the document intact. (For example, instruction may include moving up a <tr/> tag several ancestors, until it is not a descendant of a <table/> anymore. This violates the purpose of using a difference algorithm (also known as 'edit distance') to do data extraction from DOM documents.)

The idea is, to use multiple iterations of the difference algorithm. One iteration per level of the DOM tree. This way, it is easily detectable which regions are repeated on a web page, and dynamic and static data can be revealed. Just like in the first inter-page solution, it is assumed that data that changes between pages originates from a database of some form, and has a high probability of containing data of interest (like product data).

One problem is that difference algorithms only compares two instances of something.

Difference Algorithm

The difference algorithm that is used in this solution, is Myers' diff algorithm [15]. This algorithm produces a minimal diff, meaning that the set of instructions to go from one version of something to the next, is as small as possible. In the case of this solution, two pages belonging to the same category are compared against each other. Instead of letters, words or lines (the tokens that are usually fed to the algorithm), tags are used, and the textual content within the tags. Tags are considered equal if all attributes are the same (same type and same value).

The difference algorithm works as follows: it takes two ordered sets of tokens (when comparing two pieces of text these tokens may be letters or words). For example: ABBC and ABCD. From these two sets, a graph is formed. This graph may look as follows:

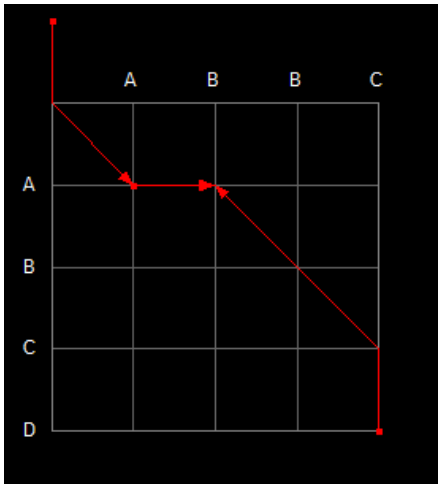

Figure 12: Example of the diff algorithm as used in this reseach

The shortest path (from the upper left corner to the lower right corner, in this case) represents the shortest set of instructions to go from one set to the other. The shortest path is calculated as follows: horizontal and vertical steps in the graph as shown above, have a cost of 1, and represent insertions

and deletions respectively (if the horizontal set is regarded as the new set, and the vertical set is regarded as the old set). Diagonal steps represent equality of the token at the position in question. Therefore, the cost of a diagonal step is equal to 0. How the shortest path is calculated, is not of import for the algorithm. There may be multiple implementation for that, which all yield an optimal result (there are multiple possibilities for an optimal result). In this case, the set of instructions to go from one set to the other will be:

- Accept(A)
- Insert(B)
- Accept(B)
- Accept(C)
- Delete(D)

To go from ABCD to ABBC.

Stages

The solution consists of three stages: the **pre-process stage**, the **comparison stage** and the **integration stage**. During the pre-process stage, the web pages are parsed, refined, compressed and made ready for the algorithm. During the comparison stage, the algorithm is ran over all pairs of pages, resulting in $\frac{1}{2}n^2 - \frac{1}{2}n$ combinations of pages, each pair having a set of differences and similarities. The goal of the integration stage is to combine these sets of differences into a single model which can extract data from a web page belonging to the same group as the compared web pages originate from.

Hierarchical Structure

As mentioned before, the algorithm is not applied the way it usually is. The algorithm itself remains untouched, but the data that is fed to the algorithm is not the document as a whole. Per level of the DOM structure (so at the lowest level, there is <html/>, and the level thereafter is <head/> and <body/>) the algorithm is applied. So for the first two levels, the algorithm will say that the documents are identical, since they both have the same tags with the same attributes (no attributes). In the third level, there may be some changes (insertions, deletions) from one document to the other. If a node is considered equal to the corresponding node in the other document, the contents of the node are compared recursively. This means that the comparison happens node by node when traversing DOM levels. It is not the case that entire DOM levels are compared to each other.

Note that the algorithm is originally designed for versioning, so one of both documents is considered as the “old” version, and the other document as the “new” version, resulting in terms like addition and deletion. On the third level, one may see that some script or meta tags may be missing or are added to the second document. This is where the algorithm will try to look for changes, and format these changes in such a way, that objects that are present in both pages, are considered being the same object. (For example, if a script tag with the src attribute “/scripts/jquery.js” is present in both pages, than it is assumed that it originates from the same line of code or html on the server.) If a tag is marked as addition or deletion, it is eliminated from the process and not inspected any further. This is needed, because the tags that are present or absent in one document, cannot be mapped to tags in the other document. Therefore, no comparison of the internal structure of these added or removed tags is impossible (where should it be compared against?).

There is, however, one exception: repetitions. If a tag that is considered the same is repeated multiple times, it is resolved to a single tag during the pre-process stage. This way, the algorithm does not have to deal with repetitions (which it will see as insertions or deletions, depending in

which file they are present or absent) since it cannot distinguish between a repetition and an insertion of a certain other tag type. These resolved, single tags need to be post-processed later on, of course. This post-processing is done in the integration stage.

Combining sets of differences

Ending with a huge amount of pairs and differences between pairs, one model is created from these pairs. This model consists of a set of xpaths, which (if the method works) point to the data that we are interested in. This set of xpaths is composed from the intersection of xpaths that are generated by comparing all possible combinations of two pages in the set. So for each pair of pages in the set, the content is compared as described, and a set of xpaths is generated, based on the differences and similarities between these two documents. During comparison, certain tags are eliminated, and the xpaths of the tags that are left over are returned. The tags that are eliminated, are tags that are present in one document but not the other (insertions and deletions), tags that only occur once and have the same structure in both documents and tags that occur multiple times, but have a different structure in one or both documents. So the tags that are left over, are the tags that occur multiple times in both documents, and have the same structure. The added value of Myers difference algorithm here, is that the algorithm sorts out which tag in one document maps to which tag in the other document. Eventually, all sets of xpaths of all pairs of pages are intersected, and the xpaths that emerge from this intersection are kept.

Example

Consider the following three web pages from our dummy web shop called “Dummy Webshop”, which sells all kinds of products:

```
<html>
    <head>
        <title>Dummy Webshop - Apple</title>
    </head>
    <body>
        <div class="header">
            <h1>Dummy Webshop</h1>
            <div class="menu">
                <ul>
                    <li><a href="/home">Home</a></li>
                    <li><a href="/shop">Shop</a></li>
                    <li><a href="/contact">Contact</a></li>
                    <li><a href="/about">About Us</a></li>
                </ul>
            </div>
        </div>
        <div class="content">
            <span class="product-title">Apple</span>
            <table>
                <tr>
                    <td>Calories / 100 grams</td>
                    <td>150</td>
                </tr>
                <tr>
                    <td>Price / 100 grams</td>
                    <td>&euro; 1,-</td>
                </tr>
            </table>
            <button>Order now</button>
        </div>
        <div class="footer">
            <p><img src="/img/logo.png" alt="logo" /></p>
        </div>
    </body>
</html>
```

Figure 13: Web page 1

```html
<html>
    <head>
        <title>Dummy Webshop - Computer Monitor</title>
    </head>
    <body>
        <div class="header">
            <h1>Dummy Webshop</h1>
            <div class="menu">
                <ul>
                    <li><a href="/home">Home</a></li>
                    <li><a href="/shop">Shop</a></li>
                    <li><a href="/contact">Contact</a></li>
                    <li><a href="/about">About Us</a></li>
                </ul>
            </div>
        </div>
        <div class="content">
            <span class="product-title">Computer Monitor</span>
            <table>
                <tr>
                    <td>Price</td>
                    <td>&euro; 150,-</td>
                </tr>
                <tr>
                    <td>Size</td>
                    <td>24 inch</td>
                </tr>
                <tr>
                    <td>Weight</td>
                    <td>2400 grams</td>
                </tr>
                <tr>
                    <td>Technology</td>
                    <td>TN</td>
                </tr>
            </table>
            <button>Order now</button>
        </div>
        <div class="footer">
            <p><img src="/img/logo.png" alt="logo" /></p>
        </div>
    </body>
</html>
```

Figure 14: Web page 2

```
<html>
    <head>
        <title>Dummy Webshop - Doorknob</title>
    </head>
    <body>
        <div class="header">
            <h1>Dummy Webshop</h1>
            <div class="menu">
                <ul>
                    <li><a href="/home">Home</a></li>
                    <li><a href="/shop">Shop</a></li>
                    <li><a href="/contact">Contact</a></li>
                    <li><a href="/about">About Us</a></li>
                </ul>
            </div>
        </div>
        <div class="content">
            <span class="product-title">Doorknob</span>
            <table>
                <tr>
                    <td>Price</td>
                    <td>&euro; 30,-</td>
                </tr>
                <tr>
                    <td>Colour</td>
                    <td>Bronze</td>
                </tr>
                <tr>
                    <td>Material</td>
                    <td>Plastic</td>
                </tr>
            </table>
            <button>Order now</button>
        </div>
        <div class="footer">
            <p><img src="/img/logo.png" alt="logo" /></p>
        </div>
    </body>
</html>
```

Figure 15: Web page 3

As can be seen, product data is present in the <table/> tag of the web site. This is the data we are primarily interested in. During the first step of the algorithm, the pages are formatted into well-formed XHTML. Since this step is beyond the scope of the algorithm, well-formed documents are assumed (as seen above). The second step is the compression step, which groups similar tags together. For the first document, this results in the following DOM tree: (the other documents are compressed in a similar fashion)

```
<html>
    <head>
        <title>Dummy Webshop - Apple</title>
    </head>
    <body>
        <div class="header">
            <h1>Dummy Webshop</h1>
            <div class="menu">
                <ul>
                    <li><!-- compressed data, count: 4 --></li>
                </ul>
            </div>
        </div>
        <div class="content">
            <span class="product-title">Apple</span>
            <table>
                <tr>
                    <!-- compressed data, count: 2 -->
                </tr>
            </table>
            <button>Order now</button>
        </div>
        <div class="footer">
            <p><img src="/img/logo.png" alt="logo" /></p>
        </div>
    </body>
</html>
```

Figure 16: Compressed version of page 1

Note that the /html/body/div elements are not grouped, because they have a different class (in order to be grouped, elements should have all attributes the same). This is done for all pages, after which the documents can be compared by the difference algorithm. This is done level by level. The first two levels are not that interesting (these are present on almost any web page, since they form part of the basic structure of web pages), so we skip directly to the third level in the body tag. Here, we see the following structure:

```
<div class="header">
<div class="content">
<div class="footer">
```

Figure 17: Substructure of all considered web pages

This structure is the same for all documents that are considered in the example. Therefore, the algorithm will look into each of the tags more closely. The div with class “header” is the same in all documents, and will therefore be skipped during this example. The div with class “content” is more interesting: first span.product-title will be compared. Since both nodes are considered equal and the node does not have any siblings that are considered equal, the algorithm looks one level deeper. The text nodes it encounters have a different value, so the xpath to the node is returned. The set of xpaths now consist of:

1. /html/body/div[@class='content']/span[@class='product-title']/text()

The algorithm proceeds to the next node, which is the table. Both nodes are considered equal, and does not have equal siblings. The tr node is inspected next. Just like the table, the tr node is considered equal in both documents and the children are inspected. The algorithm encounters a “compressed” node (the node that is inserted between a parent node and a subset of its children) Due to the compression algorithm, the children of this compressed or dummy node are all considered equal. If the number of children of these compressed nodes is different, which it is in this case, the xpath to this node is added to the list of xpaths. So the list of xpaths will look as follows, after visiting this node:

1. /html/body/div[@class='content']/span[@class='product-title']/text()
2. /html/body/div[@class='content']/table/tr

For the sake of simplicity, the head section of the documents is skipped in this example. If the head section were included, the list would look as follows:

1. /html/head/title/text()
2. /html/body/div[@class='content']/span[@class='product-title']/text()
3. /html/body/div[@class='content']/table/tr

In the proposed algorithm, nodes are considered equal if the type is the same, and all attributes of both nodes match (are present in both nodes, and have the same values).

This leaves us with three pairs of pages and resulting lists of xpaths (so we have three lists of xpaths). In this case, the lists are all the same. When executing the algorithm on a live corpus, some combinations of pages may result in different lists of xpaths. To combine these lists into one list, the intersection is taken. Since it may be possible that other page types contaminate the cluster (for example search result pages in a cluster of detail pages), the intersection may be taken with a softening parameter α. α takes a value between 0 and 1. 1 meaning that all lists should unanimously agree that an xpath should be in the result list, 0 basically means taking the union instead of the intersection of all lists. So if α = 0.5, only half of the lists have to agree before an xpath is added to the resulting list. In the algorithm, α is set to 0.9.

2.5. Intra-Page Object Extraction

2.5.1. Solution 1: Intra-Page Extraction using Document Structure

For the first set of heuristics, the research of [16] is used. The paper suggests a method for extracting multiple objects from a single web page, in this case search results. The intuition behind this research is based on the assumption that details of objects are most often displayed in repetitive structures like tables or lists, and the assumption that there is most often a lot of these details displayed on a detail page. The research tries to find these structures by looking for characteristics that are implied by the assumptions, like how many children a tag has (fanout) and the resemblance of the structure between these children.

This method consists of three different phases: re-formatting of the document, locating points of interest and the extraction of data.

Phase 1: Re-formatting of the document

This is the most straightforward phase of the three: a document may not be well formed (overlapping tags, for example). In this phase, the document is re-formatted, so that a well formed XML document is generated. Then, an XML library is able to generate a tag tree for further processing.

Phase 2: Locating points of interest

This phase consists of two steps: a step called “object-rich sub-tree extraction” and “object separator extraction”.

During object-rich sub-tree extraction, a minimal sub-tree is extracted from the document, which holds all objects. This is done by combining three characteristics of a sub-tree: the size of the fan-out, the size of the content (in characters) and the number of tags. Using these three characteristics, a score can be calculated for each element on the page. The element with the highest score is selected as the object that has the greatest potential of holding all data of interest.

During the object separator extraction, an object separator is generated, based on combining five heuristics: Repeating tag, Standard Deviation, Identified Path Separator, Sibling Tag and Partial Path. Each of these heuristics comes with its own ranked list of separators, which is then combined into one ranked list. An object separator is something that separates objects from each other. An example can be '
', since this separates list items, and list items may hold information that is of interest with regard to this research. How these heuristics work, is explained in more detail later on.

Phase 3: extracting objects of interest on a page

When an object separator is found, it is a trivial task to extract object from the object rich sub-tree using the found object separator. However, sometimes a separator breaks an object into two or more pieces. When this happens, a mechanism is needed to construct an object by merging those pieces together. Methods from existing research are used for this. [17]

After that, the list of extracted objects still may contain some noise. The list is refined by eliminating objects that do not conform a certain set of minimum criteria, which are derived by the object extraction process. For example, headers and footers may be extracted, and these need to be eliminated from the list. These are criteria such as structural checks, detecting the lack of a common set of tags, or detecting too many unique tags. Also, if the object is too large or too small, it is removed as well.

Heuristics for creating object separators

As mentioned, the object separators are created based on a set of heuristics that try to distinguish between parts that are part of the structure of a web site and parts that contain the object data. This is done within the context of the object rich sub-tree.

First, a list of candidate separator tags is generated. This is done by combining the following heuristics:

- **Standard Deviation:** First proposed in [17], the standard deviation heuristic measures the standard deviation in size (number of characters) of a list of objects (in HTML form), as separated by the corresponding candidate object separator. All candidate separators are sorted in ascending standard deviation.
- **Repeating Pattern:** First proposed in [17] as well, the repeating pattern heuristic measures the **absolute difference between pairs of tags and half of the sum of their individual count**. Pairs meaning "no text or other tags in between". For example: `<span><div/></span>` ([span,div] is considered as a pair. Please note that order matters, so this pair cannot be written as [div,span].) `<span/><div/>` ([span,div] is considered as a pair). Only children of a certain node are considered in [17], not all descendants. So, for example: if we have `<img/><br/><img/><br/>`, the absolute difference for [br,img] is 1, because the pair itself exists once, and the individual tags 2 each (2x img and 2x br). So $\left|c_{pair} - \frac{1}{2}(c_{tag1} + c_{tag2})\right| = \left|1 - \frac{1}{2}(2+2)\right| = 1$, which gives us an absolute difference of 1. The heuristic ranks the pairs by absolute difference first in ascending order, then by pair count in descending order.
- **IPS Tag Heuristic:** This is a fixed list of tags that are often used as separator tags. The list that is used by [17] is: *{tr, table, p, li, hr, dt, ul, pre, font, dl, div, dd, blockquote, b, a, span, td, br, h4, h3, h2, h1, strong, em, i}*. If a tag occurs in the object-rich subtree, it is ranked by its position of the ordered list of separator tags. Please note that the context (parent node) of the candidate separator tag is discarded. A suggestion could be designing a combined approach from Repeating Pattern and IPS Tags.

- **Sibling Tag Heuristic:** The sibling tag heuristic is based on the idea that sibling tags (closing and opening tags, on the same level in the DOM tree, without anything in between) that separate objects occur at least as often as the objects on the page. Sibling tag pairs are counted and ordered by their count descending, the topmost of the list being the most likely object separator, according to this heuristic.
- **Partial Path Heuristic:** The partial path heuristic lists all paths to all of the candidate separator tag descendants, and counts the number of occurrences of each path, after having done this for all candidate separator tags. Then, it ranks the paths (and corresponding candidate tags) by count, descending. This idea is based on that objects of the same type often have the same structure.

The right object separator tag is identified from the lists, by combining the lists using a probabilistic approach for combining of two or more independent observations. This means, that when given two heuristics A and B, and their corresponding probabilities P(A) and P(B) for tag T (P(A) representing the probability that tag T is the correct object separator, according to heuristic A), the probability that T is the correct object separator, is equal to P(A) + P(B) – P(A) * P(B). The tag with the highest aggregate probability is selected as the object separator tag.

When combining all the heuristics as described above, a precision of 100% and a recall of 98% can be achieved in some situations. [17]

There are a few comments on this method: This methods assumes that multiple objects are present on one page. This might be feasible for search result (one search result is considered an object, in that case), but for pages displaying the details of one object, this might not work as good as for search results. Of course, label-value relations that are present on the page and represent a property of the object can be considered as objects themselves, and the solution still holds. However, these objects are very small, and might not contain enough structure for the solution to work.

2.5.2. Solution 2: Classification of XPaths

Beside programmatically choosing XPaths that should extract relevant data by defining a set of rules [18] [19], it might also be possible to locate these objects using machine learning to classify XPaths into two classes: interesting or not interesting. The concept is as follows: a set of XPaths is extracted from a web page (this set is constructed in such a way, that all nodes of the web page are represented by exactly one XPath, and each XPaths may point to one or more nodes). After extracting the set of XPaths, features are extracted from the sets of nodes the XPaths point to. Then, machine learning algorithms are used to determine whether or not something is a label or a value.

The proposed solution consists of roughly three steps:

1. Extract a set of XPaths from a web page
2. Eliminate XPaths using a (list of) rule(s)
3. Use machine learning to select the XPaths that point to interesting nodes

How these steps work, is described in the sections below.

Extracting XPaths from a web page

The set of XPaths is constructed as follows: all nodes that directly contain text (so the node itself should contain text, not one of its descendants) are located. So, for the document as seen in Figure 18, this would be <title/>, <h1/>, <p/> and the two <a/>s.

Figure 18: Dummy structure of an HTML document

Then, a path from the root node (in this case, <html/>) to each of the text-containing nodes is calculated using only child steps. This is done without taking the indices into account, in order to create general XPaths that (hopefully) point to a set of nodes that contain the same class of values (labels or values, for example). XPaths that point to just one node are eliminated from the set, because the assumption is made that there must be multiple labels and multiple values present on the web page, and that these labels or values are located in nodes that can be reached from the same XPath.

In this case, after calculating the paths from the root to the text-containing nodes, the following list of XPaths is constructed:

/html/head/title
/html/body/h1
/html/body/p
/html/body/p/a

After elimination, only /html/body/p/a is left. Further elimination is done by machine learning, in order to select two XPaths that point to the label / value nodes. Basically, the method (rules + machine learning) should distinguish between label-value pairs and other content, like advertisements and menu items.

Feature Extraction

After listing the relevant XPaths of the web page, a set of features is extracted for each XPath. This is done by analysing the nodes that the XPath points to. The following features are extracted for each set of nodes:

- **The number of nodes**. The idea behind this is that often, there are numerous properties to one product. Therefore, the nodes that contain the interesting data should be abundantly present on the web page.
- **The number of ancestors**. This is used to determine the specificity of the XPath. The higher the number of ancestors, the more specific the XPath. (Considering all XPaths that just consist of child steps). A more specific XPath may mean that it points to a more specific set of nodes, thus limiting the noise from other nodes that do not contain interesting values.
- **The mean size of the content of the nodes (number of characters)**. Labels and values are not too big (like a description) and sometimes a bit small (like numeric values or yes/no values).
- **The standard deviation in size of the content of the nodes**. In this research, the assumption is made that labels are roughly of the same size (there will not be a label that is significantly bigger

than other labels). The same assumption is made for values. Therefore, this feature should be descriptive for whether or not something is a label or a value.
- **Whether or not the nodes have a CSS class in common**. Often, it is the case that labels share the same CSS class (like 'product-property' or something equivalent). The same assumption is made for values. Therefore, this feature may say something about the node containing a label of value.
- **Whether or not the structure within all nodes is the same (e.g. if all nodes contain the same subtree, disregarding text nodes)**. Content is often generated from a template, and label-value pairs are repeated by the template engine, replacing something in the template with the label or value. Therefore, the structure within the node containing the label or value will likely be the same as the structure within other nodes that contain labels or values.
- **Whether or not the nodes are part of a table (e.g. if they have an ancestor that is a <table/>)**. This feature is chosen because most label-value pairs are represented in a table. Of course, other structures can be used to present the label-value pairs, but tables form an important share of structures to represent label-value pairs.
- **The type of the nodes (should be the same for all nodes, since the last step of the used XPaths enforces that)**. It is assumed that some nodes are used more often to display labels or values than other nodes. For example, <meta/> is supposedly never used for this purpose, and <td/> is often used.
- **If applicable and possible: the mean width of the nodes**. It is assumed in this research that the properties of a product are displayed prominently on a web page, and thus having a significant width. For example: a label-value table will be bigger than a hundred pixels in width.
- **If applicable and possible: the mean height of the nodes**. Often, label-value pairs are just one line high. Therefore, the height of a node may say something about whether or not this may be a label-value pair.
- **If applicable and possible: the standard deviation of the width of the nodes**. Label value pairs are often presented in a table-like manner, thus all having the same width, which makes the standard deviation of the widths 0.
- **If applicable and possible: the standard deviation of the height of the nodes**. It is assumed in this research that all label-value pairs are just one or two lines high, making the standard deviation of the height of a set of label-value pairs low.
- **If applicable and possible**: whether or not the nodes have the same offset from the left of the document. This is derived from the assumption that label-value pairs are often presented as a table-like structure. Therefore, the offset to the left of the document should be the same for all nodes.

In the example above, this would yield the following list for /html/body/p/a :

Feature	Value
# of nodes	2
# of ancestors	3
Mean size	6
Std dev of size	0
Css class?	False
Structure?	True
Part of table?	False
Type	a

Table 1: Feature set of example XPath

Training Dataset

The nodes will be classified using supervised machine learning techniques. For that, a training set is needed. This training set is partly made by hand, partly generated (it is the same training set is used as the one for the inter-page object extraction).

Classification Algorithms

A set of classification algorithms is trained and evaluated on the set (of course, the training is done on a different part of the data set as the evaluation).

For classification, the following algorithms are used: an implementation of a neural network (multilayer perceptron), logistic regression, an implementation of a support vector machine (SMO based), a decision tree implementation and a random forest implementation.

The reasons for choosing these algorithms are as follows:

- Neural network: a neural network is capable of learning complex relations between attributes. One downside is that training may take a while with lots of features and instances, but none is the case in this research. Therefore, neural networks are considered good candidates for this research.
- Logistic regression: logistic regression is a fast and powerful algorithm when it comes to classifying instances by using a threshold for each feature value to separate instances. The features that are used in this research have such properties, such as the mean size saying something above and below a certain threshold.
- Support Vector Machine: SVMs are supposed to be able to learn complex relations between attributes, like neural networks. SVMs are a lot faster than neural networks. Since they are supposed to perform roughly the same, they have been chosen for the same reasons as neural networks.
- Decision trees: decision trees are able to re-use attributes, unlike logistic regression. Logistic regression is only able to learn relations like a > 3.4 && b < 1.0 && c > 77.9. Decision trees take this a step further: these algorithms may learn relations like a > 3.4 or (b < 1.0 and a < 2.0). Since the features that are used in this research may say only something when in a finite interval (finite meaning having an upper *and* lower boundary), this is a nice property to have in the classification algorithm. This is the reason that decision trees are considered.
- Random forest: due to good personal experiences with these algorithms, these algorithms are taken into account as well.

Conclusions

The algorithm accepts a web page and outputs a set of XPaths that should point to the nodes that contain the label-value pairs. The expectation is that this solution may work very well on some sites, but may fail miserably on certain other sites. In general, it is expected that the solution will work very well for the web sites that it is trained on (so one is able to train the solution for a specific web site). It is expected that sites that use the same structures (like tables) to represent the label-value pairs, reach a higher precision / recall when the algorithm is trained on one site and evaluated on another site using the same structures.

Suggestions for improvement

It could be lucrative to chain classifiers. The first classifier should further limit the space in which the second classifier tries to distinguish between interesting and not interesting. This may be done by letting the first classifier choose the most specific XPath that points to the node that contains all label / value pairs, such as /html/body/div/table .

This solves the problem that the classifier may choose too many XPaths (so we do not know which contains the right label / value pairs) or that the classifiers chooses two XPaths that are located on entirely different positions on the web page (and thus are no label value pairs). This means there will be less noise, since objects from outside the selected area are not taken into account.

The big disadvantage of this 'improvement' is that the solution will only be as good as the classifier that has to choose the label / value pair container. If the classifier fails to do so, the precision / recall will be 0.

3. Validation

The validation phase consists of # parts. First of all, an evaluation corpus or dataset is created in order to evaluate the results that the proposed method generates. To create the corpus, site specific scraping methods are designed, in order to scrape all information from a web site. The other part of the evaluation consists of running the solutions that are proposed in this thesis besides a baseline experiment. In the chapter about the implementation of the solutions, it is described how the baseline method operates. The results of the evaluation will be expressed using precision and recall. Precision is defined as the fraction of correct properties of an object, within the set of properties the method-to-be-evaluated returns. Recall is defined as the fraction of properties that is correct and found, compared to the total number of correct properties.

3.1. Evaluation Dataset

For the evaluation dataset, 12 web sites are crawled and indexed using manually created crawlers and extractors. Since the crawlers are created manually, it is a trivial task to recognize object pages and extract objects. Therefore, a situation is created in which pages without objects from the site are included. These pages are found by randomly following links on the web pages.

This process results in the following dataset that is used in this reseach:

Web site	Pages scraped	Object pages found	% of pages with object
Funda.nl	7971	877	11%
Jaap.nl	6845	412	6%
Nationalevacaturebank.nl	11483	1382	12%
Randstad	9590	2348	24%
Autoscout.nl	4625	1587	34%
Bijenkorf	6178	942	15%
Bol.com	9254	1247	13%
Esprit	4517	1134	25%
H&M	1382	244	18%
MediaMarkt	5412	812	15%
V&D	2551	478	19%
Wehkamp	1307	359	27%

Table 2: Dataset used in this research

In Table 2: Dataset used in this researchTable 2 an overview is given on the pages that are used in this research. The number of pages that are scraped (downloaded and processed) is given in the second column. The third column shows how many of these pages were classified as 'having an object' by a set of xpaths that are manually constructed for each web site. The last column shows the fraction of web pages that are considered containing an object. This is the second column divided by the third column.

3.1.1. Dataset construction

In order to evaluate the methods that are designed in this paper, an evaluation dataset is needed. This dataset needs to be of a certain quality and needs to be big enough. Because it needs to be big enough, certain parts of the dataset construction need to be automated. And because some parts of the dataset construction will be automated, a quality check is needed to say something about the quality of the dataset (how well the automated dataset builder performed). The idea of constructing the dataset is as follows: 12 sites are picked, and for each of these sites, a scraper is manually constructed. This requires the construction of 12 scrapers, and can generate a dataset of several thousands of pages per web site. The scrapers are sets of xpath expressions, which are created for every web site. There is one xpath expression that determines whether or not an object is available on the web page, and a set of xpath expressions that scrape the important information from the web page if an object is available. The web pages that are fed to the scraper (xpath rules), are retrieved by an exhaustive web site crawler. The crawler consists of two parts: one part that looks for non-object pages and one part that downloads object pages from the web site. The exhaustive part of the crawler starts at a certain point (usually the root of the web site, for example the page lying behind www.example.com) and visits web pages within the web site recursively (so it extracts all links from a web page, and adds these links to the web pages the crawler still needs to visit). In order to visit every web page on a web site once at most, the dataset constructor keeps track of web pages it already visited. Since web sites can contain a really large (possibly infinite, in case of generated pages) amount of web pages, it is not feasible to completely scrape a web site. Therefore, a web site is scraped, until the dataset constructor runs out of resources (memory). This will be between 1000 and 12.000 web pages per site, depending on how big web pages are and how long the urls on the web page are. A complete streaming solution with distributed hash sets is beyond the scope of this research.

3.1.2. Quality of the Evaluation Dataset

Since the construction of the evaluation dataset is partly automated, it is needed to check the quality of the evaluation dataset. This will be done using random sampling the dataset, and manually checking the samples. The size of the sample set will depend on the size of the dataset and how confident we want to be about the parameter that is estimated. The parameter that we want to estimate, is the chance that the automatic dataset builder is wrong about a page. “Wrong” or “incorrect” is defined as:

- The page contains an object, but the dataset builder does not extract object data
- The page contains no object, but the dataset builder extracts object data
- The page contains an object, and the dataset builder does not extract all object data (recall < 1)
- The page contains an object, and the dataset builder extracts wrong object data (precision < 1)

This will be done using picking 171 random pages and manually checking if the dataset builder scraped the page correctly. Using this sample set, a 97,5% confidence interval can be calculated, having a diversion of maximally 7.5% from the observed error rate. The explanation of these values will be given in the next section.

Method

P will be defined as $\frac{the\ number\ of\ incorrectly\ scraped\ pages}{total\ number\ of\ pages\ in\ the\ dataset}$. $\hat{P}$ will be the estimator of P. We will consider each process of scraping one page as a Bernoulli experiment. This means that the chance of success (or failure) is equal for each page. In practice, we can argue that there will be a greater

chance of success for scraping pages that do not contain an object versus pages that contain an object. Therefore, we will split the quality check into two parts, and calculate $\hat{P}$ for each part. This leads to the following two definitions:

$$P_{present} = P(incorrect \mid object\ on\ page)$$

$$P_{absent} = P(incorrect \mid no\ object\ on\ page)$$

Since both scraping an object from a page containing an object and trying to scrape an object from a page where no object is present are Bernoulli experiments, a sequence of scraping pages containing object or pages not containing objects is considered as a binomial distribution, with p = $P_{present}$ or p = P_{absent} respectively, and n being the number of pages that are scraped. So, this results in the following two expressions:

$$X_{present} \sim B\left(n, P_{present}\right)$$

$$X_{absent} \sim B(n, P_{absent})$$

With X being the number of *incorrectly* scraped pages. (And $X_{present}$ being the number of *incorrectly* scraped object-containing pages)

An estimator $\hat{P}$ for the parameter P of a binomially distributed stochastic variable, can be found through the following formula:

$$(1-\alpha)100\%\ BI(p) = \left(\hat{p} - c \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}, \hat{p} + c \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right)$$

With

$$\phi(c) = 1 - \frac{\alpha}{2}$$

So, we should take the following values for c, based on the confidentiality we want for the interval:

$$\alpha = 0.1 \rightarrow c = 1.645$$

$$\alpha = 0.05 \rightarrow c = 1.96$$

$$\alpha = 0.01 \rightarrow c = 2.575$$

$\hat{p}$ will be calculated as follows:

$$\hat{p} = \frac{X}{n}$$

In which X is the number of incorrectly scraped pages in a sample set of size n. And n is the size of the sample set. So, replacing $\hat{p}$ will result in a confidence interval of:

$$(1-\alpha)100\%\ BI(p) = \left(\frac{X - c \cdot \sqrt{X\left(1-\frac{X}{n}\right)}}{n}, \frac{X + c \cdot \sqrt{X\left(1-\frac{X}{n}\right)}}{n}\right)$$

As can be seen, n now depends on the number of incorrectly scraped pages X in the sample set, the confidence of the interval α and the width of the interval. So now, if we know three of the four

parameters, we can calculate the fourth. Also, it can be seen that the interval lies symmetrically around $\frac{X}{n}$.

The width of the interval can be calculated as follows:

$$w = \frac{X + c \cdot \sqrt{X\left(1 - \frac{X}{n}\right)}}{n} - \frac{X - c \cdot \sqrt{X\left(1 - \frac{X}{n}\right)}}{n}$$

$$= \frac{X + c \cdot \sqrt{X\left(1 - \frac{X}{n}\right)} - \left(X - c \cdot \sqrt{X\left(1 - \frac{X}{n}\right)}\right)}{n}$$

$$= 2\,c \cdot \frac{\sqrt{X\left(1 - \frac{X}{n}\right)}}{n}$$

Concluding, what the above says, is: the width of the confidence interval depends on $\hat{p}$ and on X (or: on n and on X, since $\hat{p} = \frac{X}{n}$). None of the parameters depend on the size of the *actual* dataset, which means that given a sample set with a certain n, X, c and w, the resulting confidence interval will be as confident for a dataset of 100 items, as for a dataset of 1 million items.

However, there is one problem: we do not know X (or $\hat{p}$) beforehand. We do know that w changes with respect to n and $\hat{p}$ (and c of course, but we are assuming a choice for c is made when calculating the rest). This is solved by finding the worst-case value for $\hat{p}$ (the value for $\hat{p}$ at which the width is at is highest). This can be done, because the quadratic width of the interval is parabolic with respect to $\hat{p}$:

$$w = 2\,c \cdot \frac{\sqrt{X\left(1 - \frac{X}{n}\right)}}{n} = 2\,c \cdot \sqrt{\frac{\frac{X}{n}\left(1 - \frac{X}{n}\right)}{n}} = 2\,c \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}} = 2\,c \cdot \sqrt{\frac{\hat{p} - \hat{p}^2}{n}}$$

$$w^2 = \frac{4c^2}{n}\hat{p} - \frac{4c^2}{n}\hat{p}^2$$

As can be seen, the width of the interval will be 0 for $\hat{p} = 0$ and $\hat{p} = 1$. Since a square root is a monotonically increasing function, and the maximum of $\frac{4c^2}{n}\hat{p} - \frac{4c^2}{n}\hat{p}^2$ lies between $\hat{p} = 0$ and $\hat{p} = 1$ (as is the case for all formulas that have the form $(x - x^2)$), the maximum of the square root of $\frac{4c^2}{n}\hat{p} - \frac{4c^2}{n}\hat{p}^2$ lies between $\hat{p} = 0$ and $\hat{p} = 1$ as well. Hence, the interval is at its widest at $\hat{p} = \frac{1}{2}$. So the formula for the worst case width, is:

$$w = 2\,c \cdot \sqrt{\frac{1}{4n}}$$

$$w^2 = 4\,c^2 \cdot \frac{1}{4n}$$

$$n = 4\,c^2 \cdot \frac{1}{4w^2} = \frac{c^2}{w^2}$$

So, if we allow a width of 0.15, and want a 95% confidence interval (which means: $\alpha = 0.05 \rightarrow c = 1.96$), the size of the sample set should be 171.

Sample Set Construction

The sample set is constructed by first dividing the dataset into two parts: a part where no objects have been scraped and a part where objects have been scraped. Since the web sites that have been scraped do not all have the same number of downloaded pages, pages are randomly discarded until each scraped web site has the same number of downloaded pages in the dataset. This is done for both parts. After that, 171 pages are randomly selected from both parts each, and are analysed in order to find incorrectly scraped pages. This process of random selection happens through the seeded random number generator of MySQL.

Results

Below, the realisation of sampling the dataset is shown. X is the number of pages within the sample set that is scraped incorrectly, $\hat{p}$ is the resulting fraction ($\frac{X}{171}$) and $\hat{p}_{max}$ is the upper limit of the 95%-confidence interval. The last characteristic basically translates to: "it can be said with 97.5% confidence, that the evaluation dataset does not contain more than $\hat{p}_{max} * 100\%$ incorrectly scraped pages".

Sample set	X	$\hat{p}$	$\hat{p}_{max}$
Object present	0	0	0
Object absent	11	0.0585	0.101

3.2. Prototype Design

The prototype that should eventually be implemented, is structured as displayed in Figure 19. In this figure, it can be seen how the aforementioned components relate to each other, and how they will be implemented in the prototype. During the validation of this research, only one path is evaluated at a time. A path consists of feeding the training set to the extraction methods that use learning techniques, optionally clustering pages, extracting data from the test data set, and evaluating the results that the extractors generate. This is evaluated by looking at how many properties that were present on a web page the algorithm actually did extract (in terms of precision and recall).

A few new aspects are introduced: the training set, heuristic combiner and evaluation set. The heuristic combiner is a module that combines the output of all single page object extraction heuristics into a single object. The single page extraction heuristics all deliver a ranked set of xpaths, of which each heuristic 'thinks' that that is the best order. These ranked lists are combined into one ranked list, and the top xpath is used in order to extract the object data from the web page. The step of combining the ranked lists, is done by the heuristic combiner.

The training set is used for methods that require training, such as machine learning or the initialization of an automatically generated template. This training set will consist of web pages that are not present in the evaluation set. The set of pages used for training and evaluation is divided randomly, so that part of the objects of which the properties are known will end up in the training set, and another part in the evaluation set. The evaluation set will be used to evaluate the quality of the objects that have been scraped. Therefore, the evaluation set will contain the correct object for each page containing an object. This way, the object that has been scraped by the method proposed in this research, can be compared to the object that is displayed on the page.

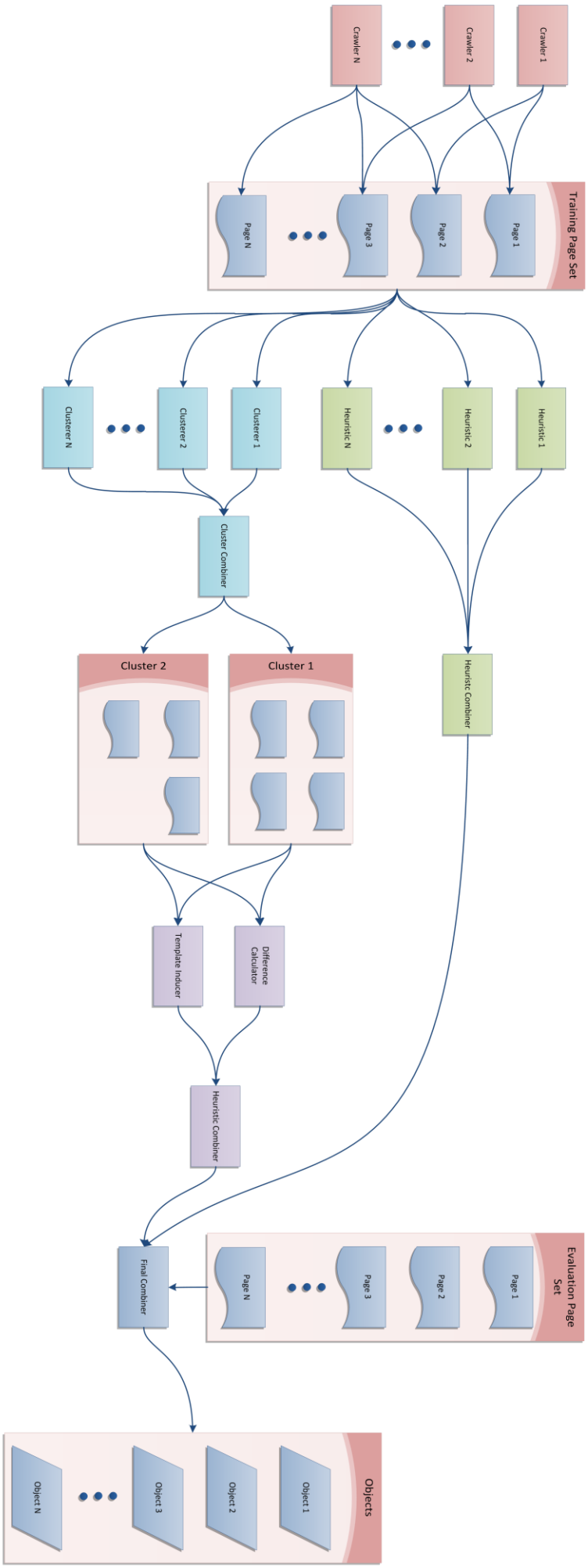

Figure 19: A schematic representation of the outline of the prototype implementation

4. Results

In order to clarify the results, the aggregates of all precisions and recalls for a web site are displayed as box plots. The box plots used in the visualizations have the following properties:

- The usual properties of box plots (min, first quartile, median, third quartile and max)
- The average (displayed as a black dot)
- The outliers (displayed as open circles)
- The far outliers (not displayed per outlier, but rather the presence of these far outliers is displayed as a single triangle on top. Although the triangle arguably implies that there are data points above it, it is not the case.) This can be seen clearly in the recall graph of De Bijenkorf: when looking at the inter page difference algorithm and keyword based clustering, it can be seen that the box plot is drawn until about 0.17, en there are outliers at 0.20. Besides the outliers, there are farouts. How many of those there are, is not visualized, only the presence of them. Their value is somewhere between 0.2 and 1.

Most of the web sites that have been analyzed, resulted in more or less the following distribution of precision and recall:

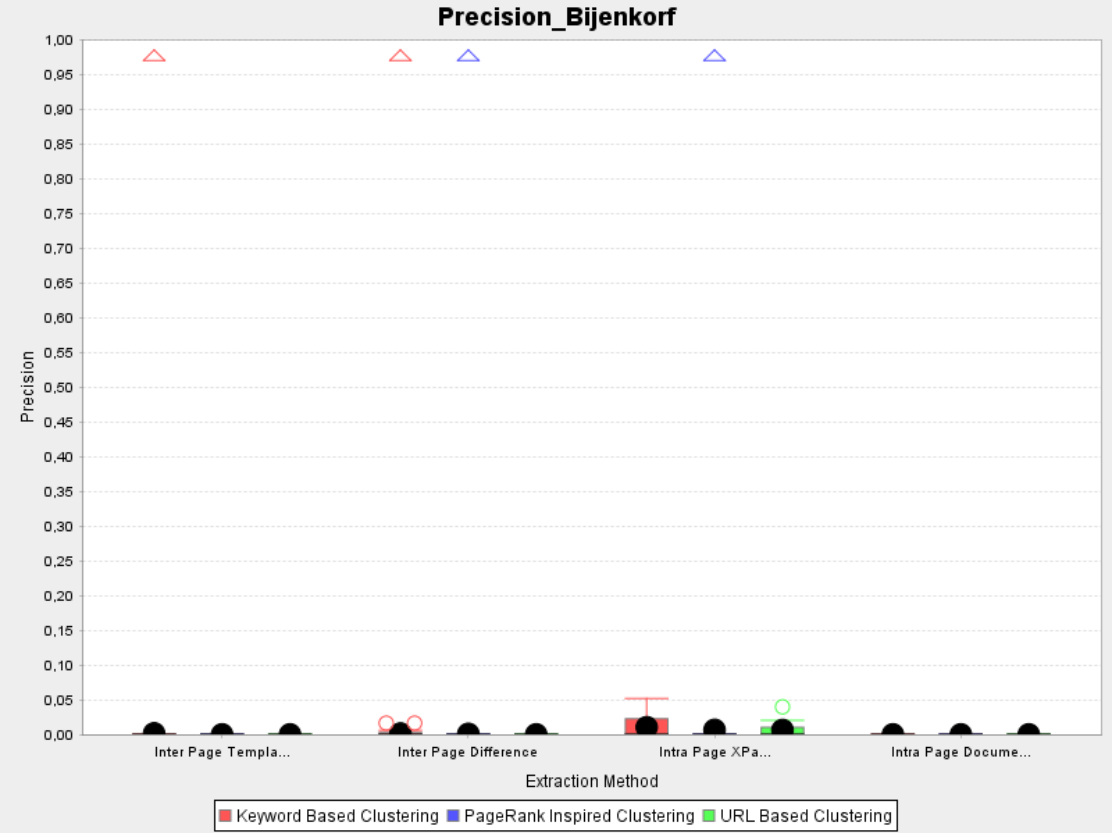

Figure 20: Bijenkorf precision, grouped by extraction method

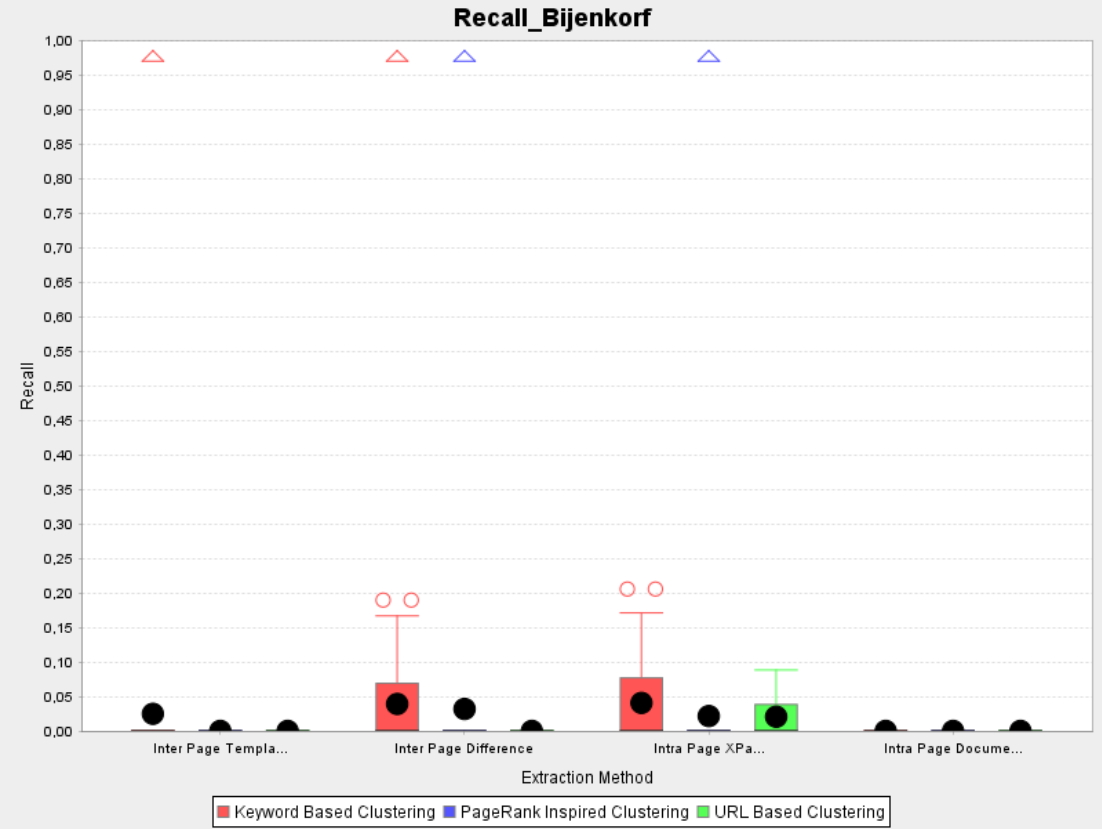

Figure 21: Bijenkorf recall, grouped by extraction method

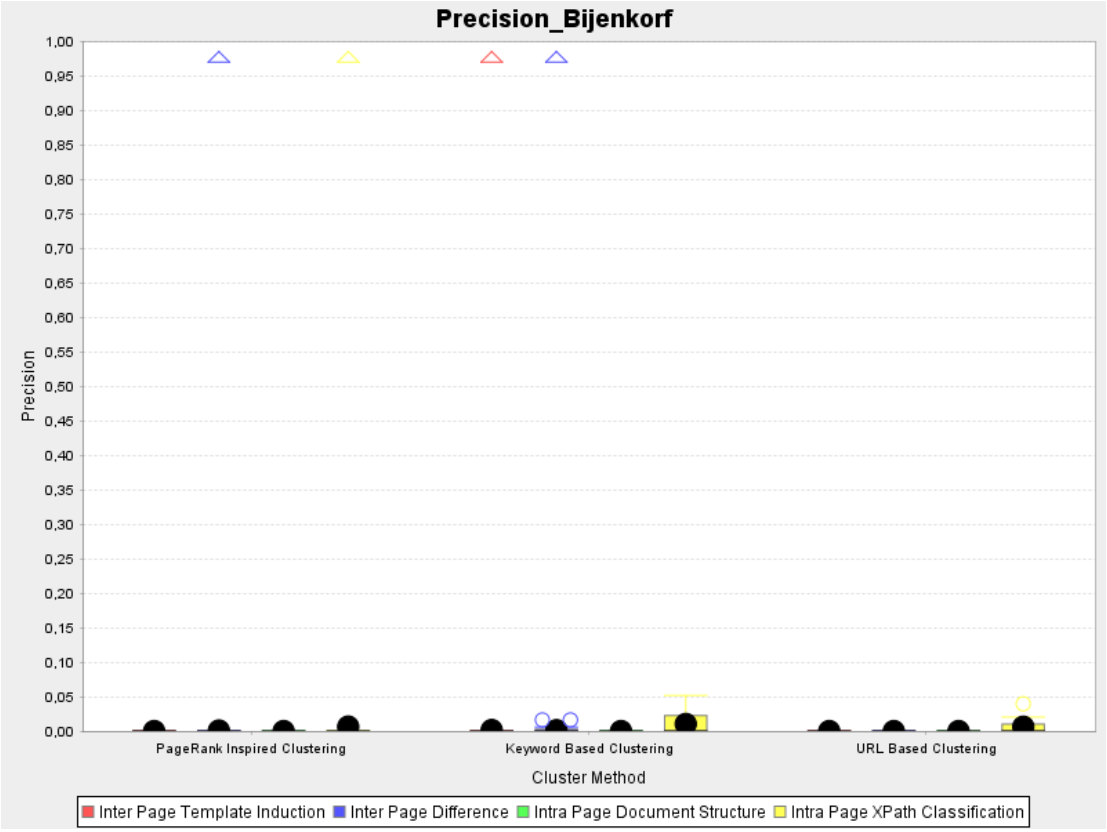

Figure 22: Bijenkorf precision, grouped by cluster method

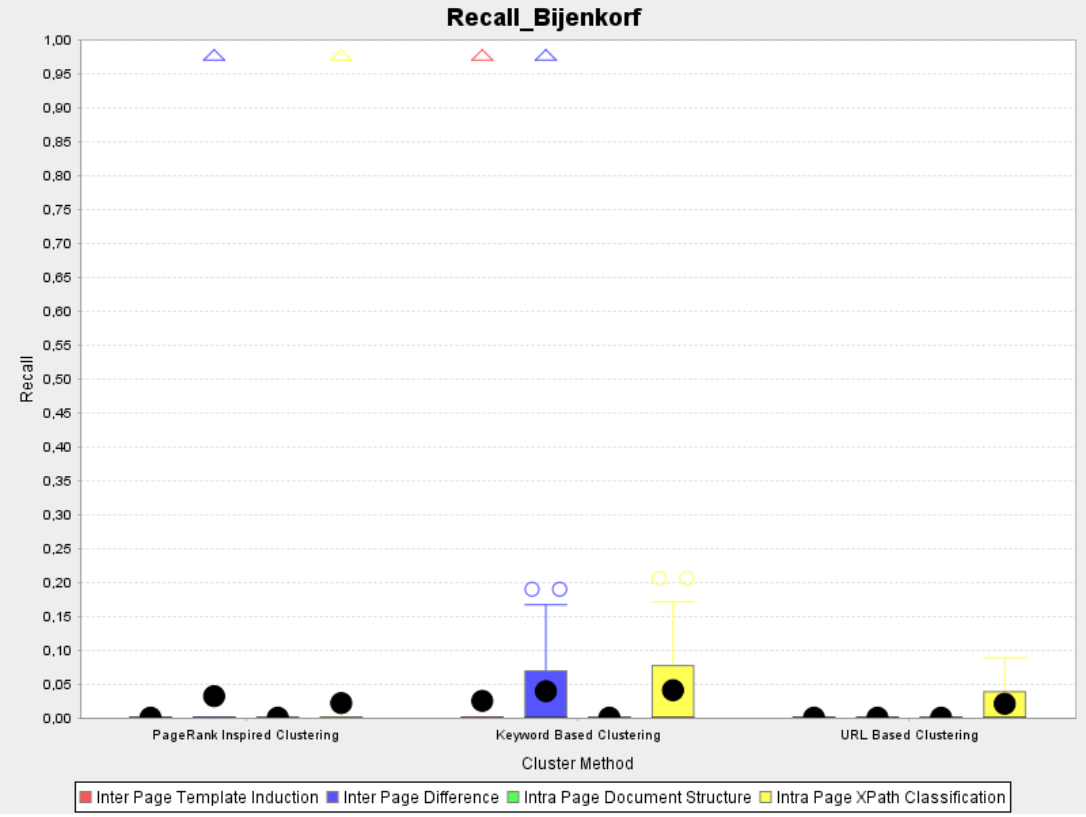

Figure 23: Bijenkorf recall, grouped by cluster method

Both the precision and recall are very low, the recall being marginally higher than the precision. Not much difference between inter and intra page methods can be observed. The corresponding precision and recall of the clustering is as follows:

Method	Precision	Recall
Pagerank Inspired Clustering	0.01479	0.02972
URL Based Clustering	0.00544	0.00849
Keyword Based Clustering	0.05086	0.08493

Table 3: Bijenkorf precision and recall per method

This means that the clusters are of very poor quality: a lot of pages within the cluster do not belong there, and a lot of pages that are outside of the cluster are the pages that contain data.

Please beware of the fact that the precision and recall as shown in the table have a totally different meaning than the precision and recall shown in the boxplots. Although they are related, it is possible that one of both has a high P/R and the other has a very low P/R.

Another note that should be kept in mind is that in these diagrams, the pages that do not belong to the cluster (data-less pages) are **not** taken into account. In the table, they are. The reason for this, is that there is no data present on these pages (or: no data that is of interest with regard to this research), so nothing can be said about the precision and recall of the algorithms that should extract the (nonexistent) data. Another option would have been to define the precision and recall as zero for these pages, which in will result in an artificially created centroid at the bottom of the box plot. As this may skew the observations, it is chosen to leave these pages out of the visualization.

However, there are a few web sites that stand out with regard to the precision and recall as shown above.

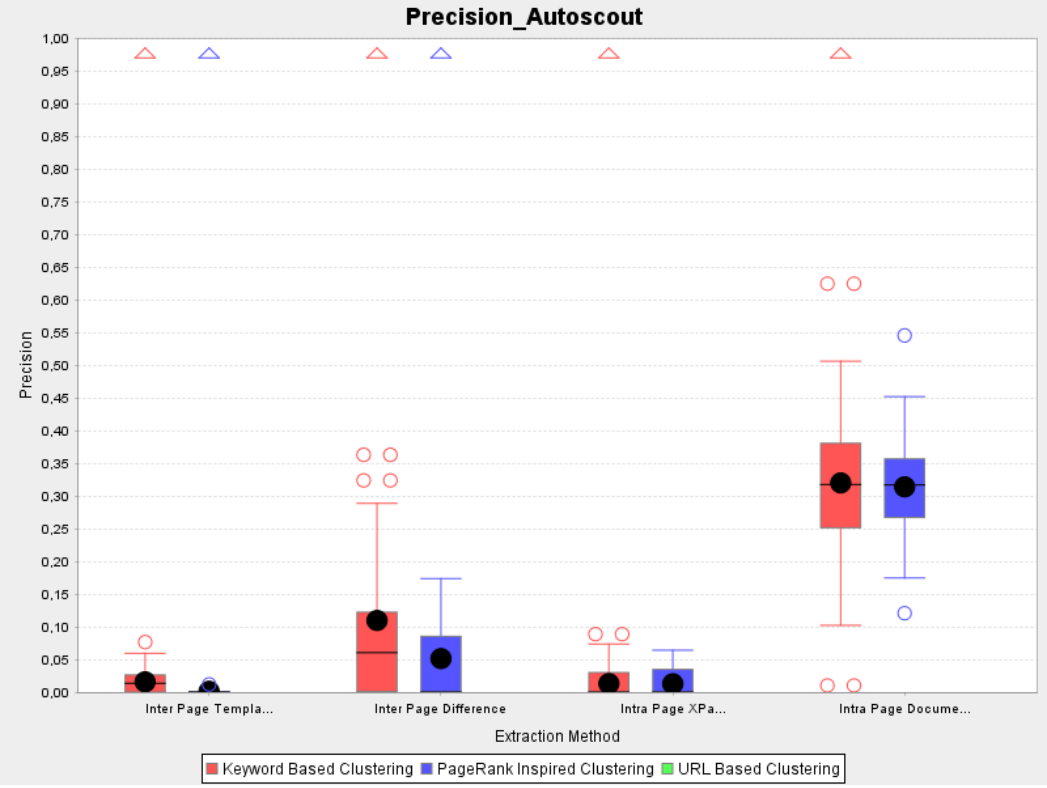

Figure 24: Autoscout precision, grouped by extractio method

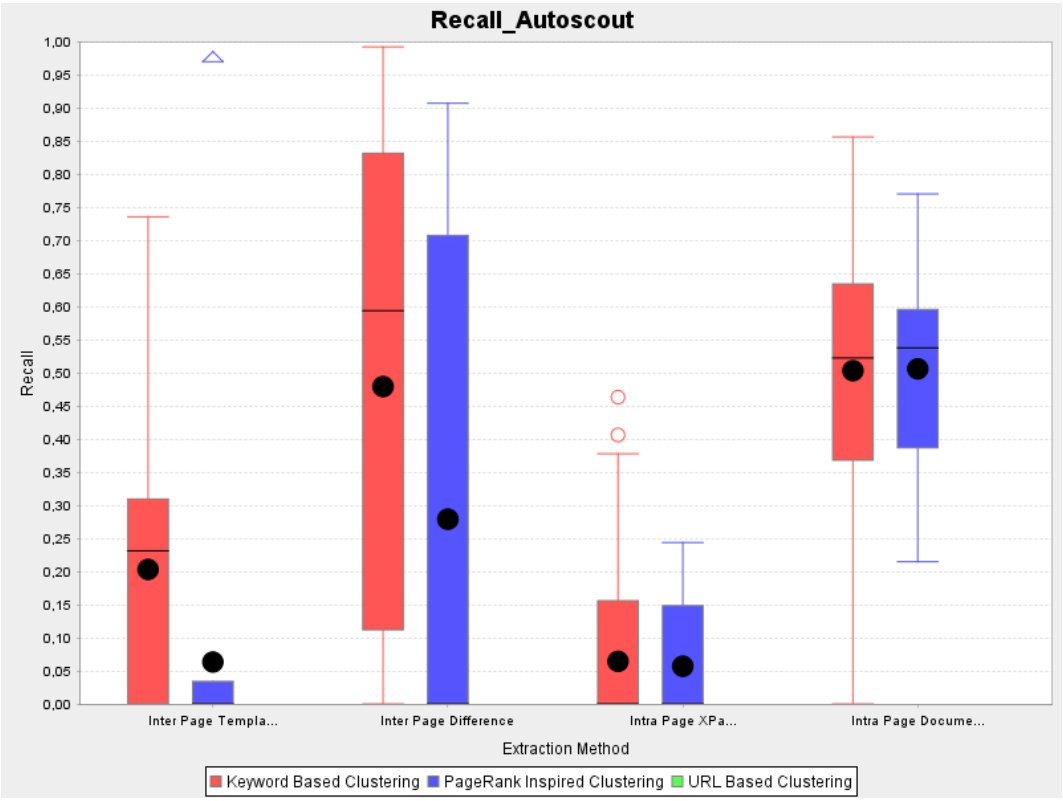

Figure 25: Autoscout recall, grouped by extraction method

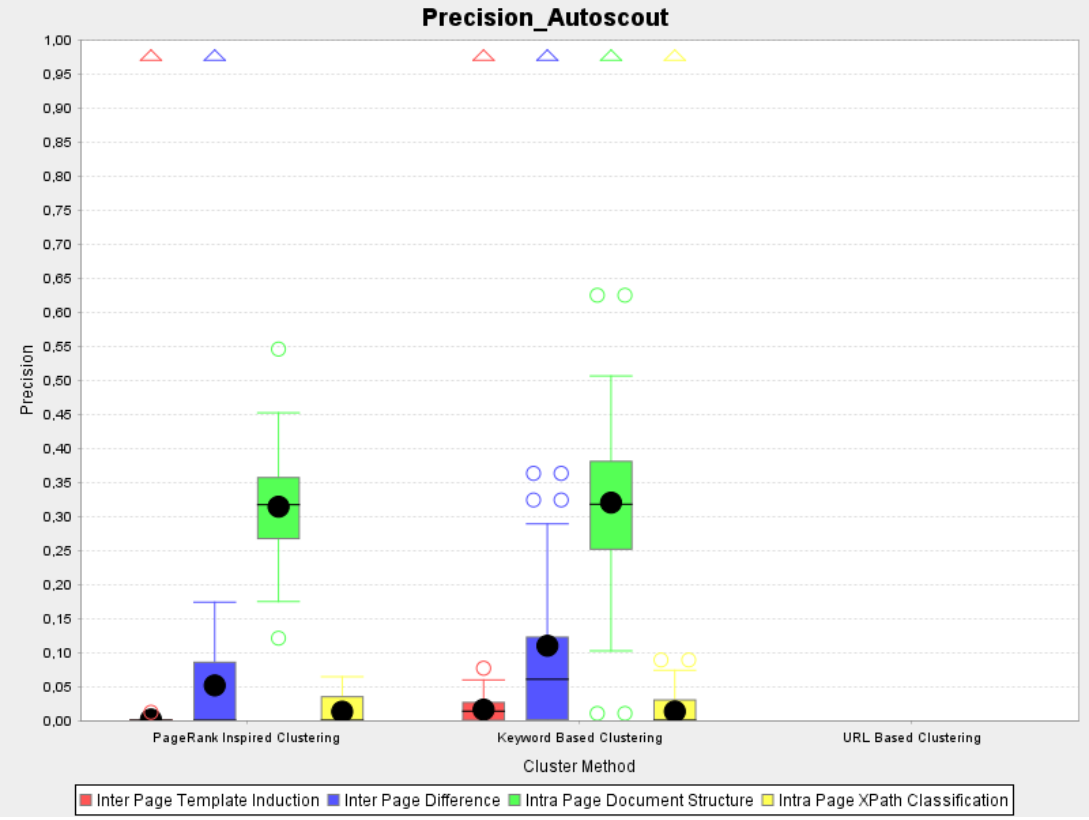

Figure 26: Autoscout precision, grouped by cluster method

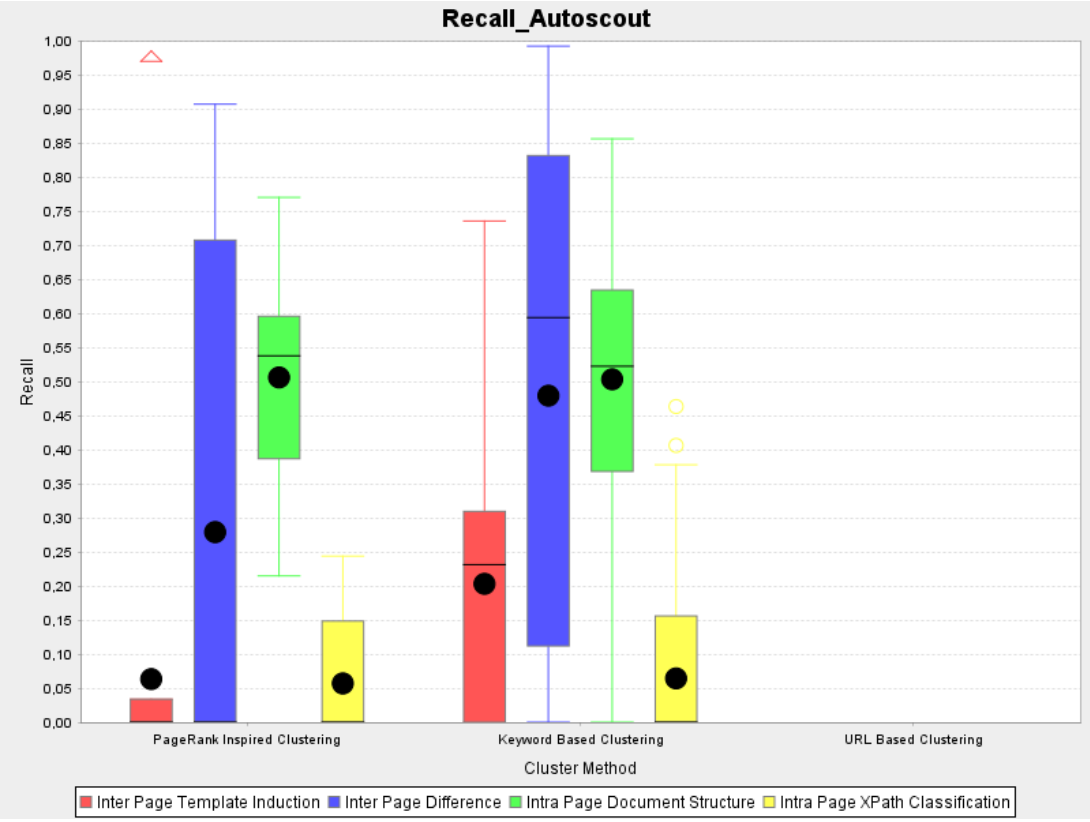

Figure 27: Autoscout recall, grouped by cluster method

The corresponding precision and recall for the clustering algorithms is:

Method	Precision	Recall
Pagerank Inspired Clustering	0.03299	0.02394
URL Based Clustering	0.00000	0.00000
Keyword Based Clustering	0.15214	0.11216

Table 4: Autoscout precision and recall of the clustering methods

As can be seen from the visualizations, URL based clustering does not work at all for this web site. This is a problem that has been encountered for a lot of web sites, especially clothing sites.

It is remarkable to see that the recall is much higher than the precision, meaning that the algorithms are too greedy: they extract a lot of content from a web page, among which a lot of correct content, but even more content that is not of import.

Both keyword based clustering and pagerank inspired clustering create clusters that contain a (relatively small amount of) correct web pages. Since both methods work partly in the same way, this was to be expected.

Again, there is no clear difference between intra- and inter-page methods: the two methods that excel here are one intra page and one inter page method.

The recall of the document structure extraction method is more reliable than the recall of the inter page difference algorithm (the latter one has a wider spread). Also, the precision is significantly higher.

The remarkable thing about the document structure extraction method, is that it either works relatively well, or does not work at all. An example of the latter case can be seen in the visualization for bol.com:

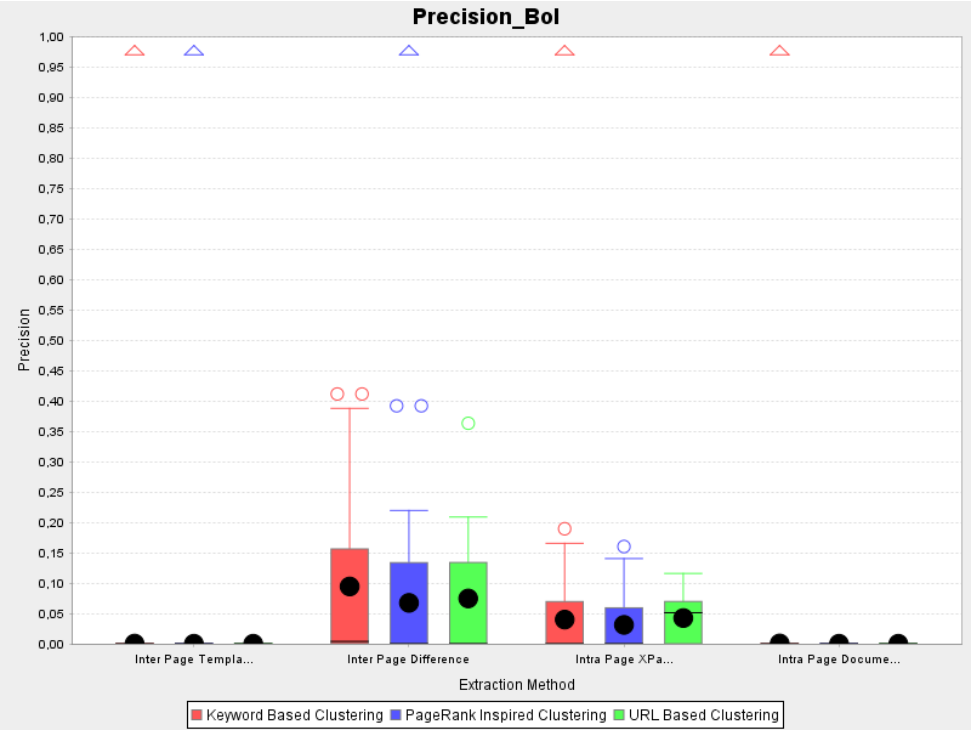

Figure 28: Bol.com precision, grouped by extraction method

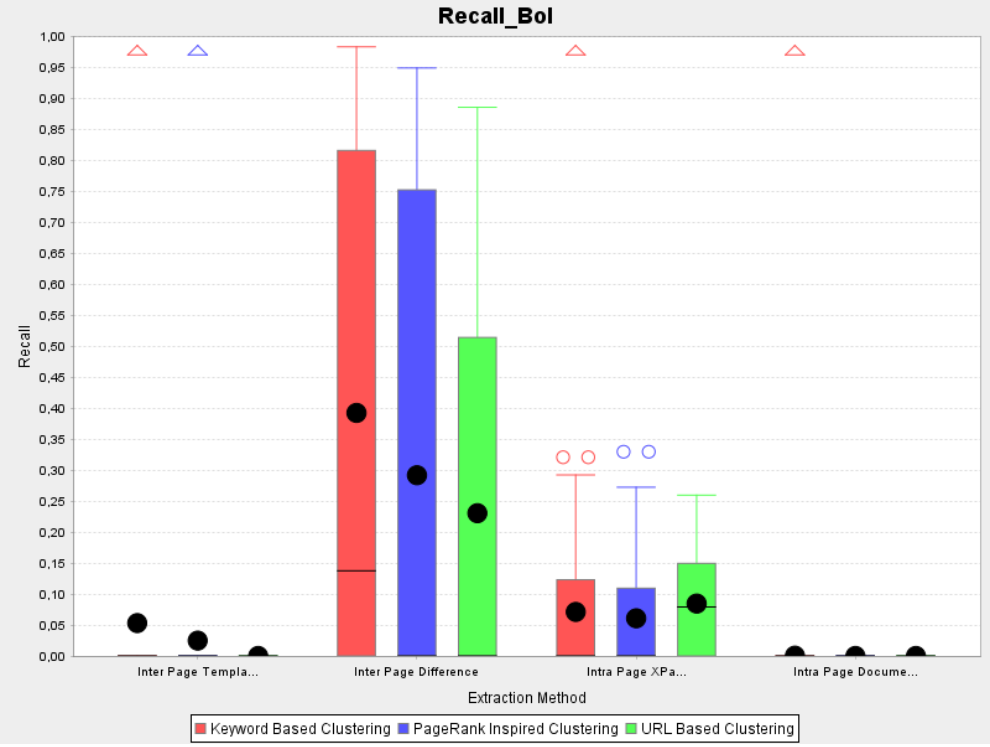

Figure 29: Bol.com recall, grouped by extraction method

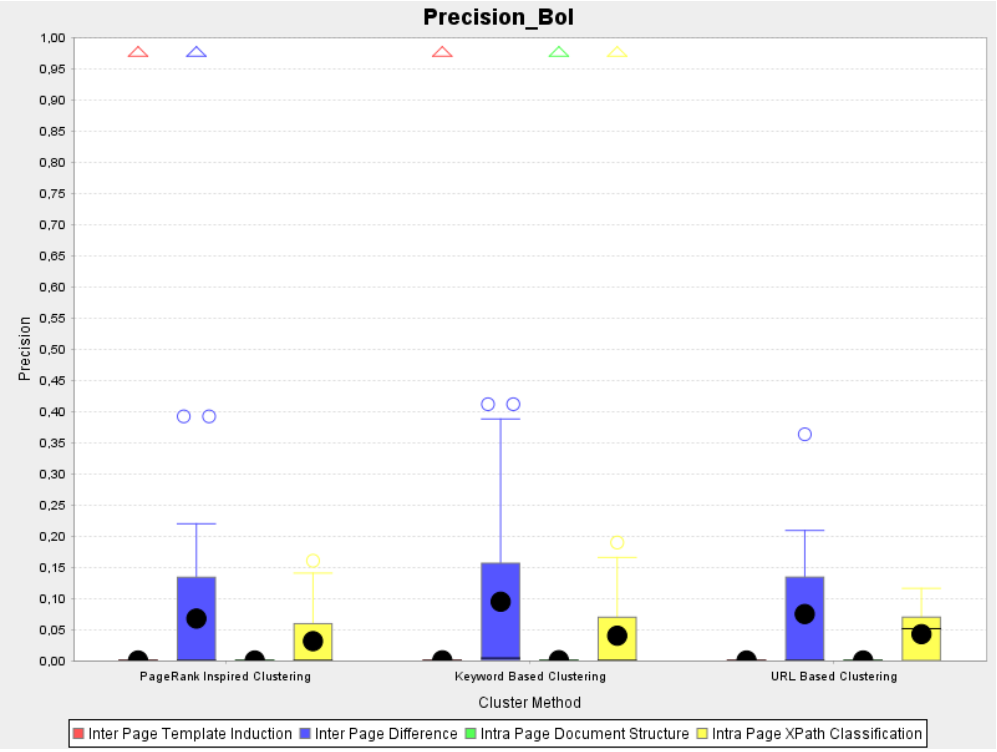

Figure 30: Bol.com precision, grouped by cluster method

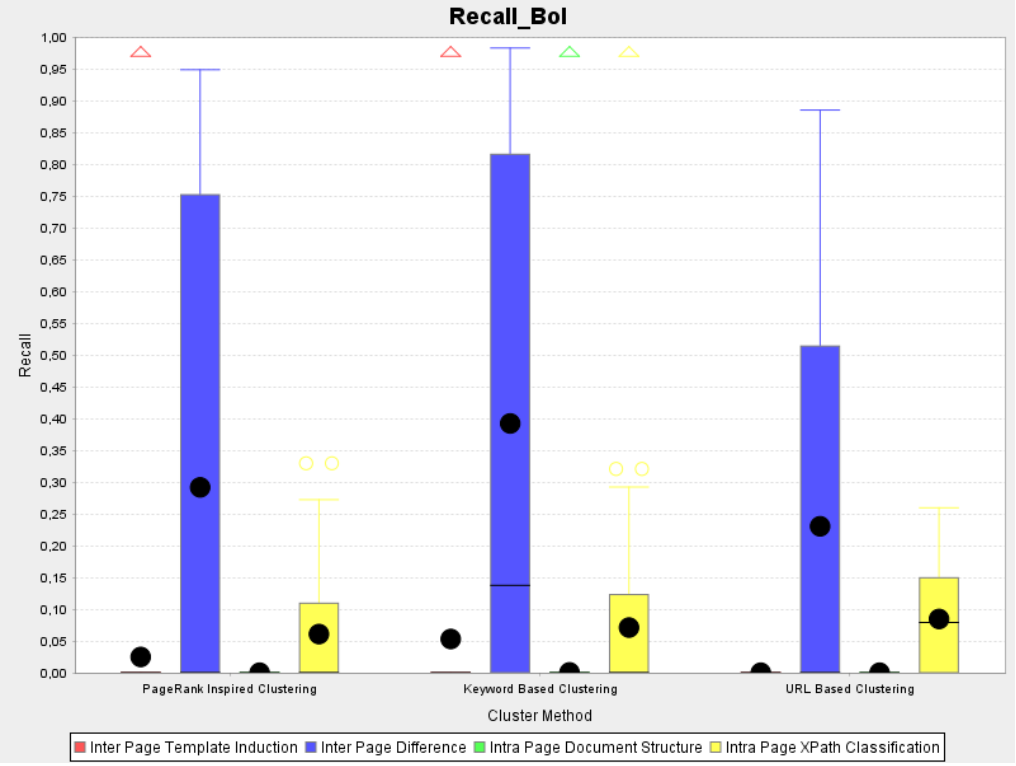

Figure 31: Bol.com recall, grouped by cluster method

Again, with the corresponding precision and recall for the clustering methods:

Method	Precision	Recall
Pagerank Inspired Clustering	0.03250	0.08500
URL Based Clustering	0.03909	0.00962
Keyword Based Clustering	0.05543	0.17963

Table 5: Bol.com precision and recall of the cluster methods

As can be seen, the intra page document structure extraction method does not deliver any (correct) results. The other intra page method however, does deliver some data: it has a relatively high precision and recall compared to other web pages that it is executed on.

Another very remarkable thing here, is that although the clusters are not of very good quality, the inter page difference method has an enormous recall. The precision is not too high, which indicates that it extracts a lot of data from the web pages.

In a few cases, URL based clustering creates clusters of significantly better quality than the rest of the clustering methods. An example is as follows:

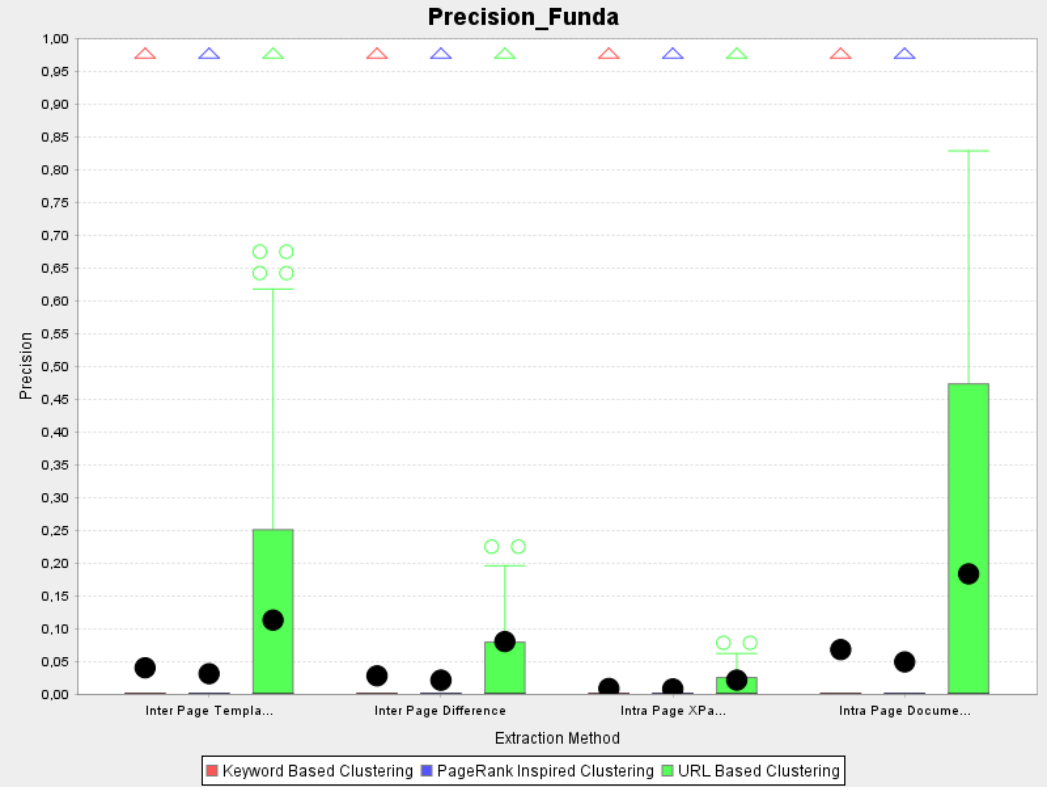

Figure 32: Funda precision, grouped by extraction method

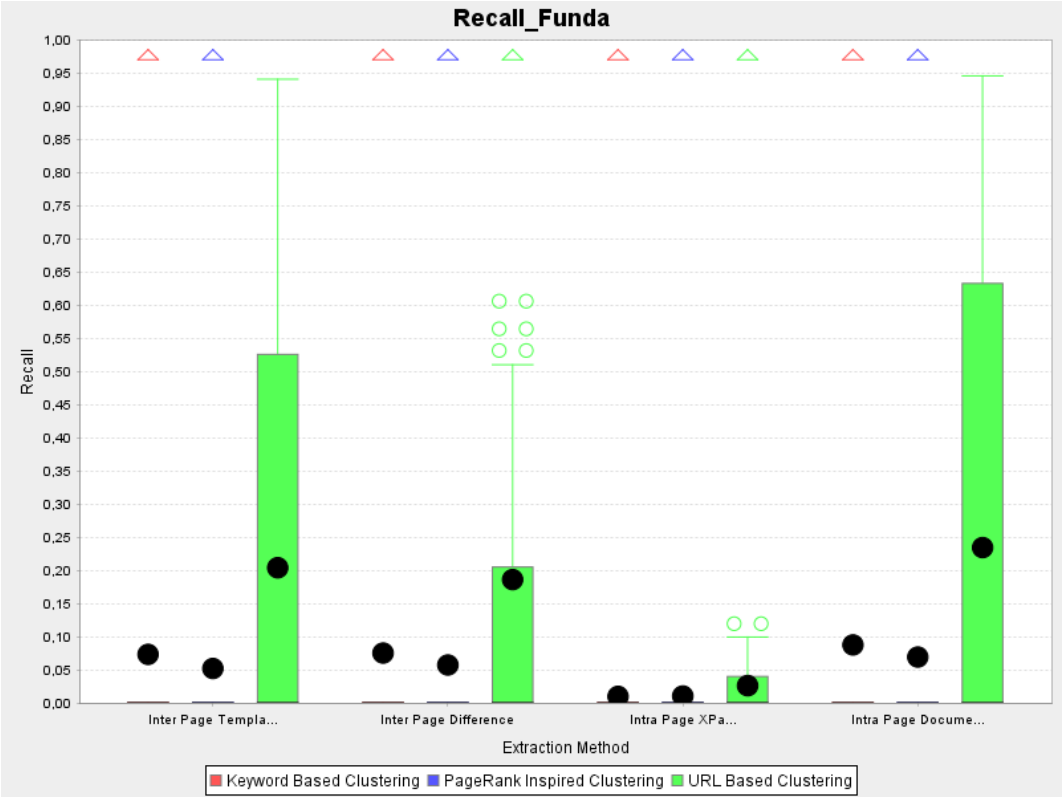

Figure 33: Funda recall, grouped by extraction method

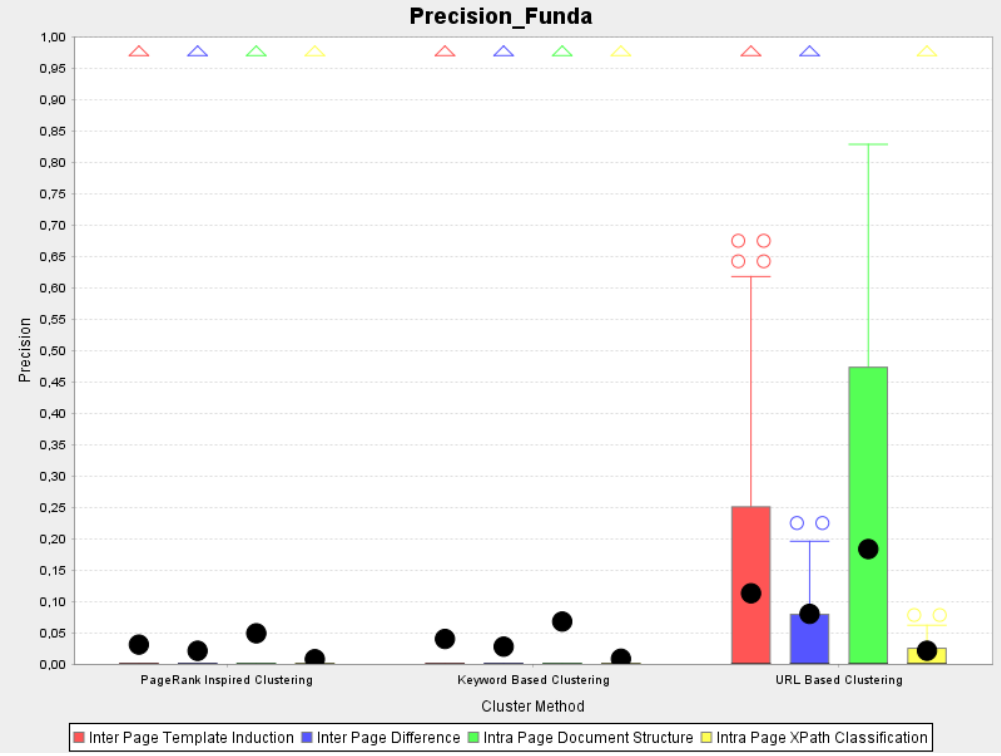

Figure 34: Funda precision, grouped by cluster method

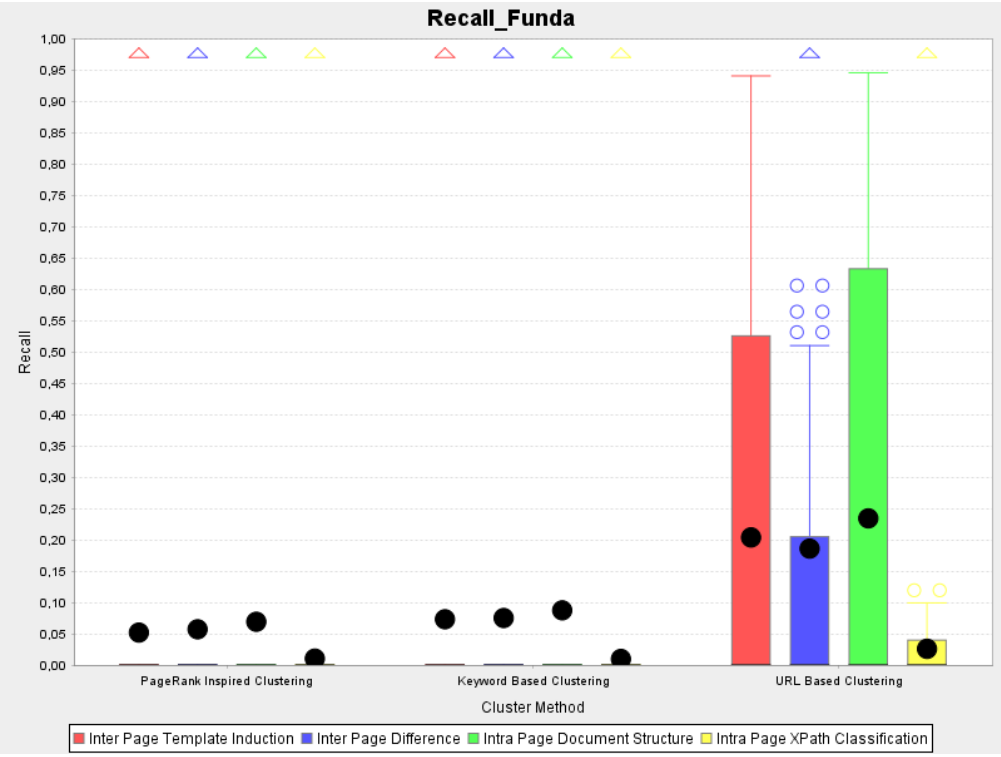

Figure 35: Funa recall, grouped by cluster method

The following precision and recall is yielded by the clustering algorithms:

Method	Precision	Recall
Pagerank Inspired Clustering	0.09459	0.01596
URL Based Clustering	0.33319	0.85975
Keyword Based Clustering	0.12735	0.12429

Table 6: Funda precision and recall of the cluster methods

As can be seen, URL based clustering performs much better. This has a direct effect on all clustering algorithms, but especially on the inter page methods. The precision of those is much higher when compared to other web sites.

Remarkably enough, the intra page document structure method still performs better than the inter page methods.

Lastly, there are cases in which there is a large recall, but the precision lags behind a lot. An illustrating example is nationalevacaturebank:

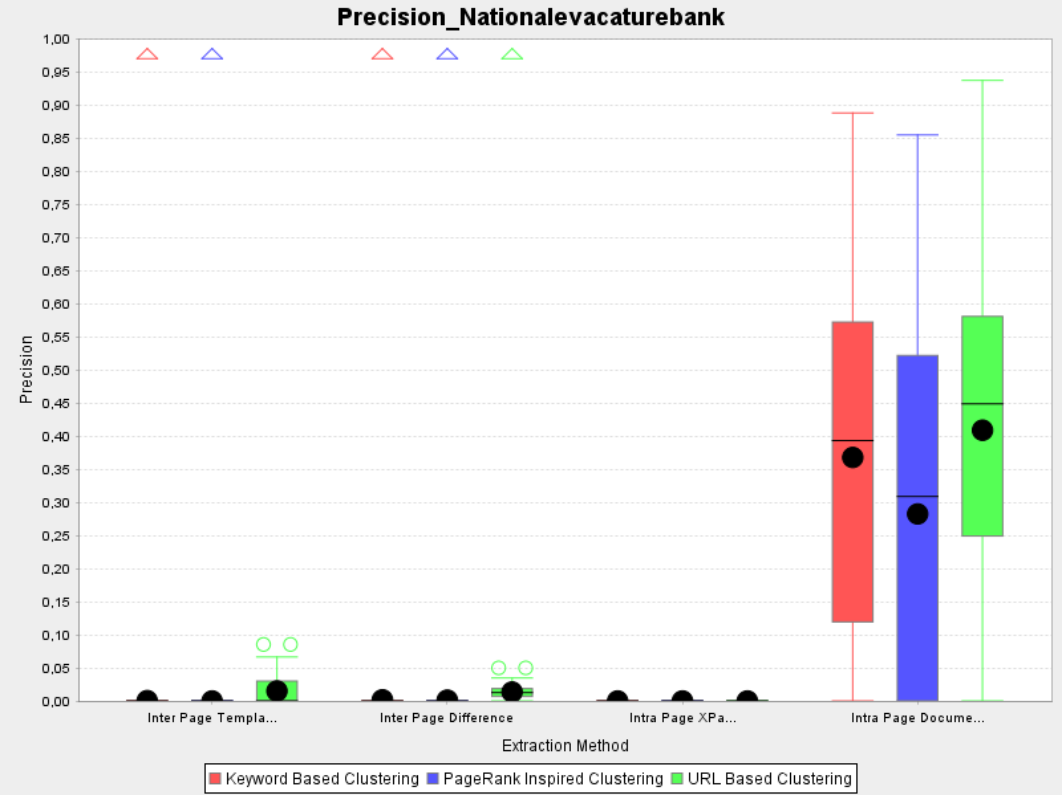

Figure 36: Nationale vacaturebank precision, grouped by extraction method

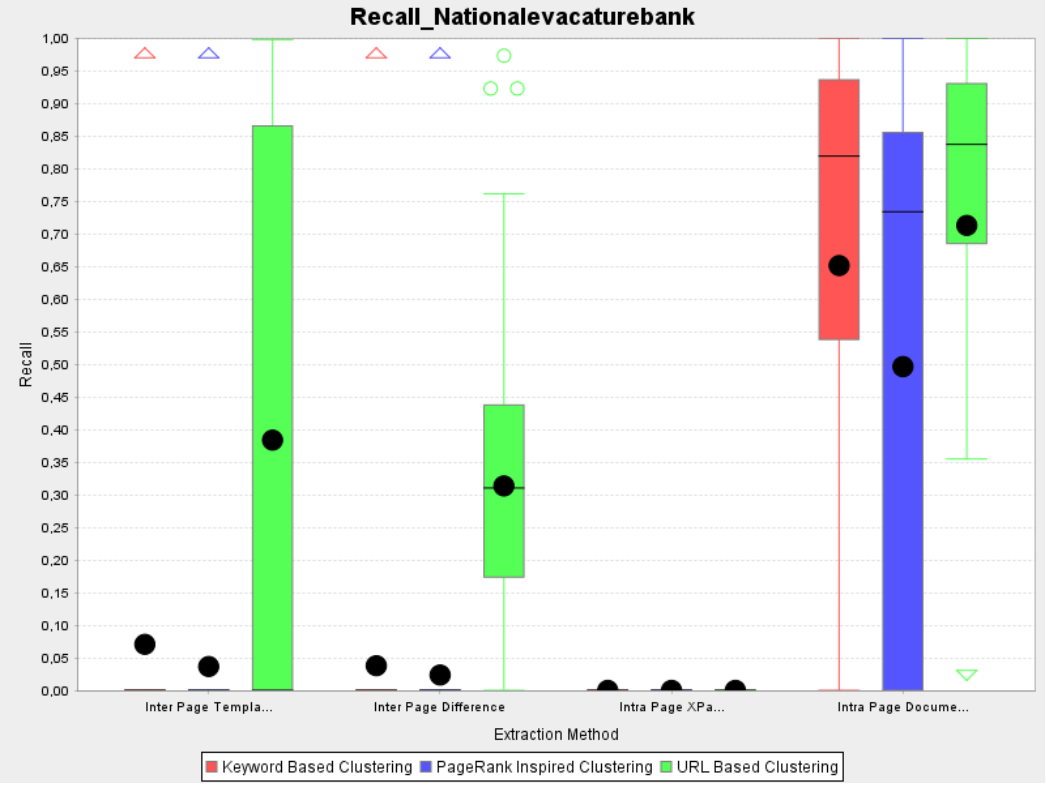

Figure 37: Nationale vacaturebank recall, grouped by extraction method

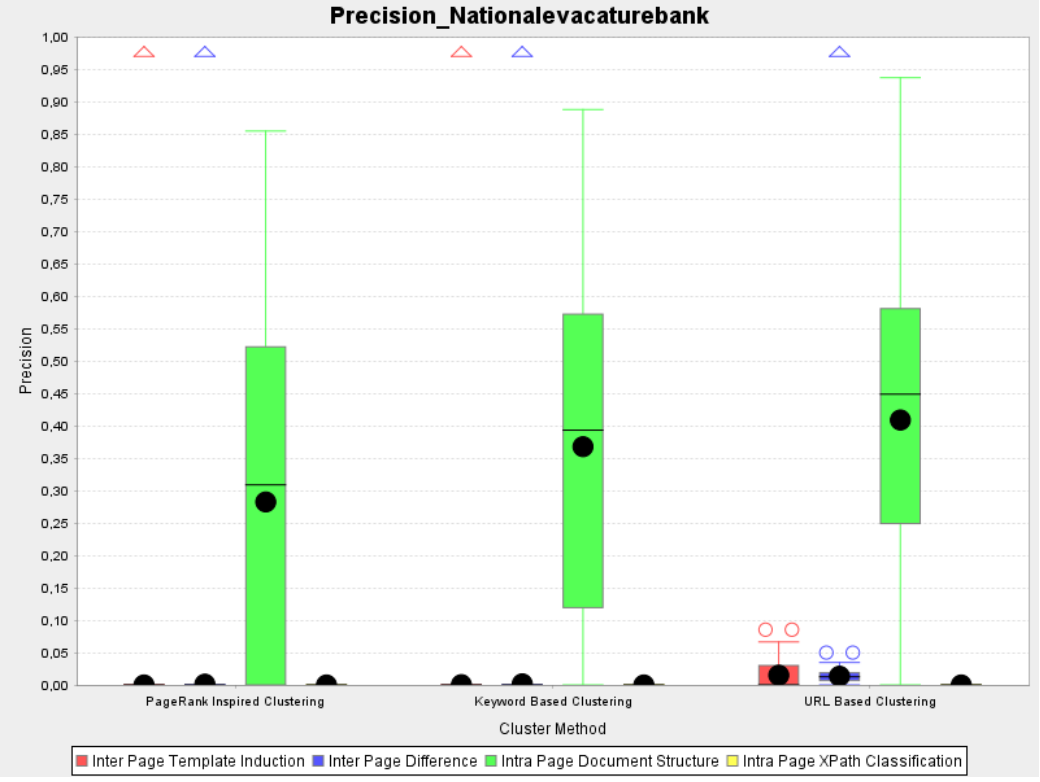

Figure 38: Nationale vacaturebank precision, grouped by cluster method

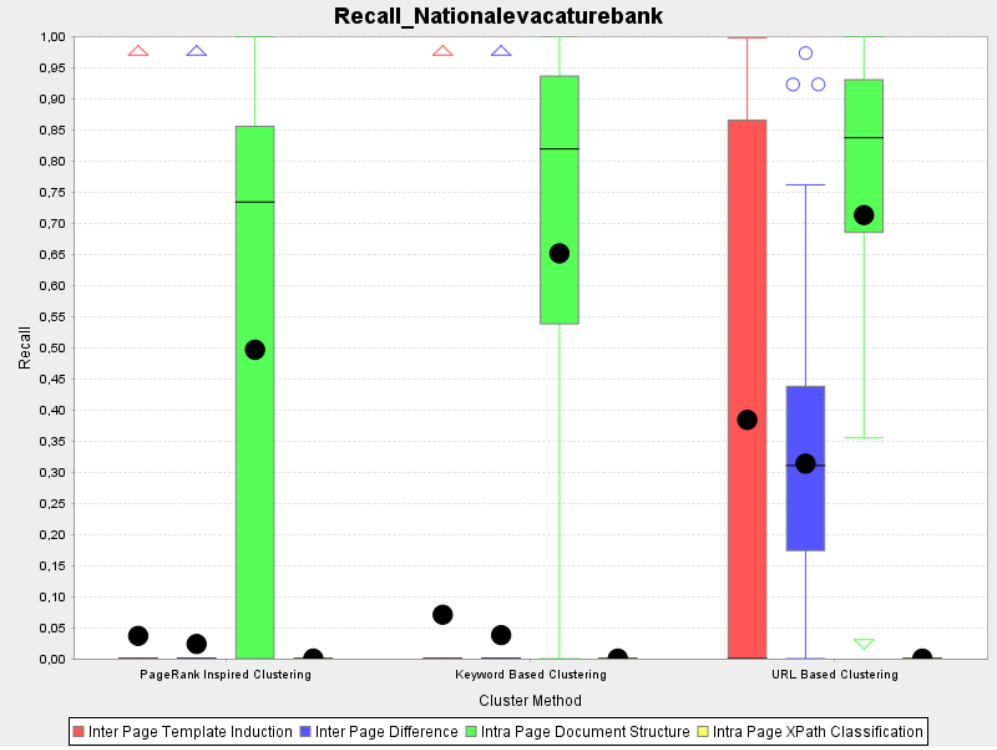

Figure 39: Nationale vacaturebank recall, grouped by clustering method

This corresponds with the following performance for the clustering algorithms:

Method	Precision	Recall
Pagerank Inspired Clustering	0.01286	0.05752
URL Based Clustering	0.44062	0.87832
Keyword Based Clustering	0.05059	0.20796

Table 7: Precision and recall of the clustering methods used on nationale vacaturebank

There are a few things that stand out on this web site:

- URL based clustering performs very good
- Intra Page Document structure extraction works very good
- Both of the inter page methods have a really high recall, but an almost nonexistent precision

5. Conclusions

First of all, methods that extract data from web pages can be categorized into two groups: methods that compare pages and methods that look for structure in single pages. Two from each group have been used in this research, one existing and one based on existing methods. In order to compare pages, clustering is needed, so three clustering algorithms are evaluated in this research. Pages need to be delivered to the clustering algorithms, therefore a crawler is needed. Crawling is not investigated in this research. This answers research question 1.1.

There are a few conclusions that can be drawn from this research, based on the finds. The most trivial conclusion is that **clustering is very important** when doing automated object extraction. This is true because of the following reasons:

- When using inter page extraction methods, such as template induction, pages of the same type need to be compared in order for the algorithms to work properly. This behavior can be seen in the results.
- If the object extraction needs to be (fully) automated, objects should only be extracted from pages that actually contain objects. If clustering is not done properly, there is too much noise in the sets of pages that the object extraction is performed on.

Even when clustering is not done properly, some inter page object extraction methods do yield relatively high recalls (with small precisions). Since these methods need to compare between pages of the same kind, and a cluster of poor quality contains pages of all different kinds, a lot of the content on these pages is considered as important (because it changes between pages). Especially the difference algorithm based method is vulnerable to this case, since it does not have strict rules as the template induction extraction method.

There is no one clustering algorithm that stands out. URL based clustering tends to create clusters of relatively good quality, if it works. If it does not work for a web site, the precision and recall drop dramatically. This is due to the URL structure of web sites. When a web site uses an hierarchical structure (for example: all product details are stored beneath /product/show?id=123), the URL based clustering tends to work very well (as can be seen for nationale vacaturebank).

PageRank inspired clustering tends to work slightly worse than keyword based clustering, from which the conclusion can be drawn that propagating terms throughout the web site is not a feasible option for clustering.

Overall, the clustering algorithms that have been used in this research, do not deliver high quality clusters for the investigated web sites. In cases where there is a very low precision and recall, the wrong cluster may be chosen. Therefore, it may be concluded that choosing the largest cluster that is created by a clustering algorithm is not a feasible way of determining the correct cluster for web harvesting.

It can be concluded that the precision and recall of the combination of methods (clustering combined with an extraction method) heavily depend on the structure of the web site. This answers research question 1.4.

The best option for web harvesting, as applied in this research, is to work with intra page structure based extraction algorithms. These algorithms do not rely on clustering algorithms to do their job properly, and hence have one less variable factor. From this research, it can be seen that the performance of the algorithm that is used, performs well or does not perform at all (just like the URL based clustering). This is due to the data structures used on web sites, the algorithm that is used in

this research relies heavily on some available structure on a web page. If that structure is absent, the algorithm fails to extract the right data from a web page.

The biggest problems that remain, based on this research, are:

- Clustering needs to be improved
- There is no acceptable way of selecting the cluster that contains most data, according to this research

This answers research question 1.2.

There are a lot of different clustering methods, only two have been evaluated by this research. It would be interesting to see how other clustering methods perform on the pages. Besides the fact that the algorithms themselves can be improved, the selection of the right cluster can be improved as well. Now, the biggest cluster is selected, but as it turns out, that is most often not the cluster with the highest precision, causing some of the data extraction algorithms to fail. Another method could be selecting the cluster based on links between pages inside the cluster, and the structure of the graph resulting from those links. This answers research question 1.3.

The clustering might be improved by looking into the cluster themselves, perhaps extracting features and applying machine learning to the selection process of the clusters. In the end, all methods used for clustering in this context, come down to dividing the pages into just two clusters: one containing the pages that you want, the other containing the rest. That would basically be a binary decision on whether or not a page is important or not.

Further, research that is done on the methods that are used in this research, suggests that all extraction methods are viable for extracting information from web sites. This is in theory, but in practice and when combined with other essential modules (such as clusterers), the combination does not appear to be as viable as is implied by the results of the researches taken separately.

Overall, things need to improve in order to achieve a workable solution. For now, combining autonomous methods in order to do fully automated object extraction is too difficult. The methods need to be improved and adjusted to each other to achieve a workable solution. This answers research question 1.5.

As for the usefulness of the methods proposed and researched in this research in the scope of price data extraction, difference algorithms and template induction methods were expected to be most promising. However, this introduces the extra complexity of clustering. Therefore, intra page object extraction methods in combination with extra rules and constraints are likely to be capable of extracting specific data like prices from a web page, since these were able to extract object data (among which prices) from part of the web sites investigated in this research.

References

[1] N. Visser, “A field exploration of problems that occur when scraping web pages,” Univesity of Twente, Enschede, 2014.

[2] H.-C. Chang and C.-C. Hsu, “Using Topic Keyword Clusters for Automatic Document Clustering,” in *Third International Conference on Information Technology and Applications*, 2005.

[3] S. Poomagal and T. Hamsapriya, “K-means for Search Results clustering using URL and Tag contents,” PSG College of Technology, Coimbatore, 2011.

[4] O. Zamir and O. Etzioni, “Grouper: A Dynamic Clustering Interface to Web Search Results,” *Computer Networks,* vol. 31, no. 11, pp. 1361-1374, 1999.

[5] P. Weiner, “Linear pattern matching algorithms,” in *SWAT '08. IEEE Conference Record of 14th Annual Symposium on Switching and Automata Theory*, 1973.

[6] R. Giegerich and S. Kurtz, “From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction,” *Algorithmica,* vol. 19, no. 3, pp. 331-353, 1997.

[7] L. Page, S. Brin, R. Motwani and T. Winograd, “The PageRank Citation Ranking: Bringing Order to the Web,” Stanford InfoLab, 1998.

[8] P. Pirolli, J. Pitkow and R. Rao, “Silk from a Sow's Ear: Extracting Usable Structures from the Web,” in *SIGCHI Conference on Human Factors in Computing Systems*, 1996.

[9] M. Porter, “An algorithm for suffix stripping,” *Program,* vol. 14, no. 3, pp. 130-137, 1980.

[10] A. Hinneburg and D. Keim, “A general approach to clustering in large databases with noise,” *Knowledge and Information Systems,* vol. 5, no. 4, pp. 387-415, 2003.

[11] A. Hinneburg and H.-H. Gabriel, “DENCLUE 2.0: Fast Clustering based on Kernel Density Estimation,” *IDA,* 2007.

[12] E. Baykan, M. Henzinger, L. Marian and I. Weber, “Purely URL-based topic classification,” in *Proceedings of the 18th international conference on World wide web*, New York, 2009.

[13] I. Hernández, C. R. Rivero, D. Ruiz and R. Corchuelo, “A Statistical Approach to URL-Based Web Page Clustering,” in *Proceedings of the 21st International Conference on World Wide Web*, Lyon, France, 2012.

[14] A. Arasu and H. Garcia-Molina, “Extracting Structured Data from Web Pages,” in *ACM SIGMOD international conference on Management of data*, New York, 2003.

[15] E. W. Myers, “AnO(ND) difference algorithm and its variations,” *Algoritmica,* vol. 1, no. 1-4, pp. 251-266, 1986.

[16] D. Buttler, L. Liu and C. Pu, “A Fully Automated Object Extraction System for the World Wide Web,” in *Distributed Computing Systems*, Mesa, AZ, 2001.

[17] D. W. Embley, Y. S. Jiang and Y. K. Ng, “Record-Boundary Discovery in Web Documents,” in *Proceedings of the 1999 ACM SIGMOD*, Philadelphia, 1999.

[18] T. Anton, “XPath-Wrapper Induction by genera lizing tree traversal patterns,” GI-Fachgruppen ABIS, Darmstadt, Germany, 2005.

[19] N.-K. Tran, K.-C. Pham and Q.-T. Ha, “XPath-Wrapper Induction for Data Extraction,” in *International Conference on Asian Language Processing*, Harbin, 2010.

Appendix

Appendix A: Proof of the ranking of a pair of pages being symmetrical

The operation of calculating the score of a pair of pages is symmetrical, so that $(p_i \bowtie p_j) = (p_j \bowtie p_i)$. The proof for this claim is as follows:

The expression can be written out as:

$$\sum_{q=1}^{20}\sum_{r=1}^{20}\begin{cases}(21-q)(21-r)\ if\ l_{p_i,q}=l_{p_j,r}\\ 0\ otherwise\end{cases} = \sum_{q=1}^{20}\sum_{r=1}^{20}\begin{cases}(21-q)(21-r)\ if\ l_{p_j,q}=l_{p_i,r}\\ 0\ otherwise\end{cases}$$

For the proof of the symmetrical property of the operator ⋈, the operator '=' is used as an equality operator for strings, and is therefore symmetrical.

The aforementioned expression can be translated as the sum over all of the following elements (all elements on the right hand side of the operator are summed, and all elements on the left hand side of the operator are summed):

$$\sum \begin{matrix}\begin{cases}400\ if\ l_{p_i,1}=l_{p_j,1}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}20\ if\ l_{p_i,20}=l_{p_j,1}\\ 0\ otherwise\end{cases}\\ \vdots & \ddots & \vdots\\ \begin{cases}20\ if\ l_{p_i,1}=l_{p_j,20}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}1\ if\ l_{p_i,20}=l_{p_j,20}\\ 0\ otherwise\end{cases}\end{matrix}$$

$$= \sum \begin{matrix}\begin{cases}400\ if\ l_{p_j,1}=l_{p_i,1}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}20\ if\ l_{p_j,20}=l_{p_i,1}\\ 0\ otherwise\end{cases}\\ \vdots & \ddots & \vdots\\ \begin{cases}20\ if\ l_{p_j,1}=l_{p_i,20}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}1\ if\ l_{p_j,20}=l_{p_i,20}\\ 0\ otherwise\end{cases}\end{matrix}$$

Since the '=' operator on two elements of l is symmetrical, we can switch the operands on the right hand side of the equation:

$$\sum \begin{matrix}\begin{cases}400\ if\ l_{p_i,1}=l_{p_j,1}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}20\ if\ l_{p_i,20}=l_{p_j,1}\\ 0\ otherwise\end{cases}\\ \vdots & \ddots & \vdots\\ \begin{cases}20\ if\ l_{p_i,1}=l_{p_j,20}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}1\ if\ l_{p_i,20}=l_{p_j,20}\\ 0\ otherwise\end{cases}\end{matrix}$$

$$= \sum \begin{matrix}\begin{cases}400\ if\ l_{p_i,1}=l_{p_j,1}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}20\ if\ l_{p_i,1}=l_{p_j,20}\\ 0\ otherwise\end{cases}\\ \vdots & \ddots & \vdots\\ \begin{cases}20\ if\ l_{i,20}=l_{p_j,1}\\ 0\ otherwise\end{cases} & \cdots & \begin{cases}1\ if\ l_{p_i,20}=l_{p_j,20}\\ 0\ otherwise\end{cases}\end{matrix}$$

In order for both equations to be the same, every element that is summed on the left hand side, should occur somewhere in the sum on the right hand side of the equation. It can be seen that the diagonal of the matrix remains intact ($(1,1)$ to $(20,20)$), and each element that does not remain on the diagonal is mirrored along the diagonal (every element at (n,m) on the left hand side is positioned at (m,n) on the right hand side). Therefore, every element that exists in the sum on the left hand side of the operator, is present in the sum on the right hand side of the operator. This means that the expression on the right hand side is the same as the expression on the left hand side, thus the operation ⋈ is symmetrical.