An Introduction to Web Crawling Challenges and Solutions

Marlou Pors Dustin Lim Daniël Mast Matthijs van Otterdijk

Abstract

Web Crawling is a set of techniques used to discover and analyse information on the Internet. In this paper, we give a brief introduction to Web Crawling and we discuss four interrelated challenges in this field, namely the Hidden Web, Scheduling, Web Spam and Ethics. We will explain what these challenges mean and thereafter give a review of several solutions that are found to overcome these challenges. We will also discuss the future challenges that arise in these fields.

1 Introduction

Internet search engines have become an integral part of our digital society. Although the size of the World Wide Web, is unknown as it is a distributed entity, it is clear that it has become massive over the years. This has made it harder for the user to find information on the Web without using a search engine. Since the beginning of the Web, search engines have existed although they were far less sophisticated compared to those of today. The quality of the returned search results was low and influenced by commercial advertising. Besides the early search engines, manually managed lists of relevant web sites in different categories also existed. However these lists could never have complete coverage and were expensive to maintain by people. With the growth of the Web, the need for mature search engines grew.

1.1 High-level Overview of a Search Engine

When a person enters a search query on a search engine web site and hits the search button, the search engine will present search results, ordered by the estimated matching degree with the search query. Although this seems quite simple, the process that has been performed in order to make this possible, is very large and complex. In order to present a list of web pages that match a search query, the search engine uses an algorithm to compare this query with the content of a database, containing structured information about billions of web pages. These web pages have been indexed. By indexing the pages, it is easier to check for a page to which degree it matches with the search query. As a result, the query does not have to be matched with a complete web page, which would take much more time. For a web page, for each word the number of times it occurs is stored. Also many other important factors are measured, for example the location of the word in the document and the relative font size of the word. After indexing, a web page can be found by someone using a search engine. Before web pages can be indexed by a search engine, a list of web pages has to be available. This list is built by a web crawler.

1.2 What is Web Crawling?

Web crawling is the part of the search engine that is responsible for filling the local repository with the source of downloaded web pages. It consists of robots that travel through web pages on the Internet, downloading the source and parsing hyperlinks that point to different (potentially unknown) web pages. Modern search engines can search through more than 14 billion pages. However, this number is only a lower bound for the size of the Internet as a search engine only knows of the existence of web pages that have been crawled. Therefore the web crawler has the

important task to crawl as much of the Internet as possible. Also, a web crawler needs to re-crawl previously crawled pages periodically to keep the information up to date.

1.3 Outline of this Paper

This paper is an introduction to web crawling. In Section 2, some of the core issues of web crawling will be discussed. With each issue, developments and solutions will be explained and future challenges will be shown. In Section 3, a conclusion will be given for the discussed content.

2 Web Crawling Issues

2.1 Overview

In order to find as much (valuable) information on the Web, web crawlers have to deal with a few major issues. Of these issues, we will discuss the Hidden Web, Scheduling issues, Web Spam and ethical issues. First we will introduce these topics shortly and create a setting in which they occur:

The primary concern of web crawling is finding all pages that a user can visit. While much of the Web's content is traversed by clicking hyperlinks, much information is also hidden behind forms, javascript links and applets. This portion of the web is called the Hidden Web. A smart web crawler has to work with all these traversal methods.

The size of the Internet is enormous compared to the processing rate of a web crawler. Though the user desires up-to-date search results, no web crawler can actually crawl all those pages every day. To stay up to date, a web crawler needs to make choices about what pages to crawl at which moment. This is the scheduling problem of a web crawler.

With search engine ranking becoming more and more important to attract visitors, some website owners have opted to mislead search engines by publishing web pages whose only purpose is to mislead web crawlers and ranking algorithms. These pages are called Web Spam. Web crawlers have adapted to this new challenge, and any modern search engine now uses a variety of web spam countermeasures.

Finally, a web crawler should not blindly crawl everything. There are ethical issues that play a role in this case. Crawling certain pages can actually be harmful - [6] contains an anecdote about their early crawling experiments, where a crawler crawled a web game, resulting in garbage data being generated in this game. Technical reasons aside, some content is personal or sensitive and should not be stored.

We have choses these four issues for this introduction for two main reasons. First, these topics are very accessible for everyone who is new in the world of Web Crawling and wants to get a global view of what is going on. It gives a good overview of the different fields where web crawlers will find challenges. Second, these topics will highlight new underlying problems that web crawlers have to deal with. For example, web crawlers that want to find information in the Hidden Web will encounter many problems regarding the size of the Hidden Web, hence also the Web itself. Also, these underlying problems will connect the four discussed topics, as will be made clear in the next section.

2.1.1 The Coherence of the Discussed Issues

While the issues of the Hidden Web are about finding as much information stored at the Web as possible, the issue of Web Spam has more to do with the quality of the retrieved information. Nevertheless, the deep Web seems to guarantee new, high quality information, so the subjects are in a way related. The scheduling problem is about the question in what order a web crawler should crawl new information or update the information it has already found. Again, (the consideration of) quantity and quality play a major role in our discussed issues. Scheduling is closely related to the ethical questions that arise in web crawling, because the construction of scheduling involves

ethical considerations. Also, scheduling solutions can be very valuable regarding searching the Hidden Web, because the Hidden Web is huge.

In the remainder of this section we will give an introduction to the already shortly discussed topics. We will sketch a picture of all the developments made in these different fields and also address problems that arise here. These problems conclude the problems we spoke of above, but also new problems we have not addressed yet.

2.2 The Hidden Web

2.2.1 Overview

The Web is an immense collection of information that is displayed on different kinds of web pages. Where previously those pages were only static, now a tremendous amount of the content on the Web is dynamic. This dynamism takes a number of different forms [20]. There are pages that are generated after some request for the page (or particular information) is received from a client. The actual information is retrieved from a database that is 'hidden' behind this form. When the same form is filled in differently, this results in a different page. As a concrete example, think of a web site like 'Bol.com'. Behind its interface, 'Bol.com' contains a (hidden) database that holds all the products of the web store. You only find the information about the particular products available when searching for it directly.

The Hidden Web (also called the Deep Web) is defined as the portion of the Web that is hidden behind search forms in large searchable databases. Earlier, traditional search engines and categorization services covered only the portion of the Web that was called the publicly indexable Web, which refers to the set of web pages reachable purely by following hypertext links, while ignoring forms and pages that require authorization or prior registration [20]. However, without including the Hidden Web, these searchers only found 0.03% of the available pages [4]. This is why searching the Hidden Web is necessary, and why it is such an important challenge.

In this section, the properties of the Hidden Web will be reviewed, as well as algorithms used to retrieve information from the Hidden Web. These algorithms have to deal with two important challenges. The first challenge is the accessibility of the information: as illustrated above, the deep Web contains much information that is not (directly) accessible. Although this information is not directly accessible, it is both important and interesting to the web user. It has been shown that there is much more quality content on the deep Web than on the surface Web [4]. This content is highly relevant to various categories of interest. For example, the category that has the most web databases is "Business & Economy", followed by "Computers & Internet" and "Education" [7].

The second problem for web crawlers that try to access the Hidden Web is that the Hidden Web is huge. A consideration web crawler designers have to make is to choose between low coverage or web crawlers that process forms they should rather leave alone.

2.2.2 Properties of the Hidden Web

From 2000, 'Crawling the Hidden Web' started to become a popular topic. Much research has focused on the properties of the Hidden Web. In [4], findings are presented about the size and relevancy of the hidden Web in 2001, based on data collected in March 2000. He did this research with the help of BrightPlanet, a well-known company that specializes in researching the deep Web [5]. Already in 2000, the deep Web was called the largest growing category of new information on the Internet. At that time, the public information on the deep Web was 400 to 550 times larger than the commonly defined World Wide Web.

Besides his findings about the size of the deep Web, [4] also stated that the quality of the information on the deep Web was better. The quality content was said to be 1000 to 2000 times greater than that of the surface Web. This is due to the structure of the deep Web. Web sites that rely on databases for content generally contain deeper and more detailed content specific to some category.

In 2004, research [7] was presented, where the researchers performed a macro experiment to study the deep Web at large, in particular its scale. The researchers concluded that at the time of their research, the size of the deep Web had an order of magnitude of 10^5 web sites. In order to get this result, [7] adopted a random IP-sampling method which estimates the Web size by testing randomly-sampled IP addresses. [4] estimated 43.000 to 96.000 deep Web sites by overlap analysis between pairs of search engines. From these results, [7] concluded that the deep Web had been expanding with an increase of 3-7 times in four years (2000-2004). But this growth was only the beginning: [15] states that the number of databases has reached 25 million in 2010.

[7] answers particular questions like "How structured is the deep Web?" and "What is the coverage of deep Web directories?". For the first question the researchers classified the web databases into two types: unstructured and structured databases. For instance, 'cnn.com' has an unstructured database of news articles, while 'Bol.com' has a structured database for books. At least three out of four deep Web data sources are structured.

Given that the deep web is so large, the question of deep Web coverage is important. [7] investigated this coverage for directory services. These services provide deep Web directoris classify Web databases in a hierarchial ordering. The best scored service in [7] was 'completeplanet.com', which covered over 70.000 databases. Although this seems like a lot, the stunning observation is that this only covers 15,6% of the estimated total web databases.

Such directory-based indexing service hardly scales for the deep Web, since it apparently relies on manual classification of web databases. For this reason, much research is being done on automating this process. This will be one of the subjects of the next section.

2.2.3 Developments

After discussing properties of the hidden Web, we now look to algorithms that try to obtain information from the hidden Web. Over the years, algorithms have been developed that attempt to collect information from the databases, like explicit data that the databases contain, but also algorithms that collect information about certain properties like the size of hidden Web databases [9].

This section discusses algorithms that fall into the first category. The discussed algorithms not only differ in the approaches they use, they also save the retrieved information in different ways.

The Hidden Web Exposer (HiWE)

One of the first Hidden Web Crawlers of its kind is the Hidden Web Exposer (HiWE), presented in [20]. The basic actions of HiWE are similar to those of other traditional crawlers, but the crawler takes additional steps for pages on which forms are detected. The goal of HiWE is to find the pages that a hidden database can generate, so that the crawler can crawl these pages as well.

The basic technique HiWE uses is filling forms it finds on the web, submitting them and using the outcome of these forms to save and search the retrieved pages. Forms are filled out by using the so-called Label Value Set (LVS) table of the crawler. This table contains different combinations of a label and a set of values this label can have, which is a fuzzy set. The algorithm can fill in forms by choosing an adequate value when it finds a label in the form. For example, if the algorithm finds a label "City", it can fill in "Delft" or another city that is in the value set.

HiWE is a task specified algorithm to counter the issue of scale of the hidden Web and uses the assistance of humans. Human assistance is needed to populate the LVS table, which is challenging for the crawler to do by itself. This happens during the initialization of the algorithm. After the initialization, the LVS table is further augmented with label value pairs learning that are encountered during crawling. When the crawler encounters form elements with a finite domain (such as a drop-down list), it can use these values to build additional pairs for the LVS table, allowing them to be used later on in other forms.

Locating Hidden Databases

In 2005, [3] proposed a new crawling strategy to automatically locate hidden Web databases. The proposed strategy is useful in the HiWE, which has two obvious shortcomings: First, the

crawler ignores small forms that have less descriptive labels, consisting merely of an unlabeled text box with an associated "Search" button, or is unable to find matches for obligatory fields (a shortcoming also mentioned in the HiWE paper [20]). Second, HiWE needs substantial human input to construct the initial LVS table. HiWE can be supplemented with the strategy of [3] which does not have these problems.

The proposed crawling strategy aims to achieve a balance between the two conflicting requirements in searching the hidden Web: the need to perform a broad search, and the need to avoid crawling a large number of irrelevant pages. The researchers also concluded that it improves the system that they developed a year earlier [2]. This illustrates the speed of the developments in the field of the Hidden Web.

The system uses a form-focused crawler to efficiently locate the searchable forms that serve as the entry point for the hidden Web. It contains a clear roadmap that uses link, page and form classifier. The link classifier is trained to identify links that are likely to lead to pages that contain searchable form interface in one or more steps. If this is the case, the classifier will give the link to the crawler. The page classifier checks if a particular page contains the form. If this is the case, the page is passed to the form classifier which checks if that form is already in the form database before it saves it.

The researchers of [3] also thought about the efficiency of their system and gave the Form Crawler two stopping criteria: The crawler leaves a site if it retrieves a predefined number of distinct forms or if it visits a predefined maximum number of pages on that site. Again, they tried to balance between low coverage versus an efficient web crawler. The stopping criteria prevent the crawler from staying too long at a particular web site without getting new forms.

ViDE: Hidden Web Data Extraction

A recent developed algorithm that directly extracts detailed data from the web databases is ViDE (Vision-Based Data Extractor) [15]. This system is innovative because it is the first system that is web language independent. Furthermore, the researchers name their work as the first one that performs deep Web data extraction using primarily visual features of the web site.

ViDE represents a deep Web page as a so-called Visual Block Tree. The algorithm partitions the page into blocks, which correpsond with particular areas with texts, images or the same layout. The Visual Block tree is hierarchical; blocks are not only identified but also further divided. The system makes this tree by using the hypothesis that there are some distinct visual features for data records and data items (properties of the total data record) on the retrieved page. In other words: The algorithm is trained to recognize how a web page sorts its data in a visual manner. An example of this is a list of books in a webstore. Each entry will use a similar layout, allowing the algorithm to recognize these page segments as potentially interesting data containers.

After the algorithm has identified the specific (text or image) blocks, the actual data extraction begins. This extraction is done in three steps. First, the so-called noise blocks (blocks that do not belong to any data record, for example advertisement image) must be filtered out. In the second step all the blocks that are similar in appearance are clustered together. At this time, the blocks in the same cluster all come from different data records. For example, one cluster holds the images of all the different books that are presented on the page. The last step regroups the blocks such that the blocks belonging to the same data record form a group (i.e. the image, the title, the price of the book). This regrouping again uses visual properties from the page. The result of this last step is how the algorithms saves its information.

2.2.4 Future Challenges

During the last few years, researchers have learned more about the Hidden Web and have developed complex algorithms to retrieve more information from hidden databases. The need for these algorithms is ever increasing, as the number of databases has grown to 25 million in 2010 [15] and is still growing. The scalability of algorithms for searching the deep Web (which is tremendously bigger than the publicly indexable web) will therefore always be a major challenge for researchers.

Many algorithms are not yet ready for the size of the deep Web, but some are enhanced so that they can be used in real-time applications. For example, ViDE has generated wrappers which efficiently search through complete databases. These wrappers are generated by using a normal deep web page containing the maximum number of data records that fit on a page [15]. These wrappers can recognize the structure of a page immediately so that the algorithm can start directly with the data extraction steps.

A second issue can be reviewed also looking at ViDE [15]. ViDE is the first web-programming-language-independent algorithm for data extraction, which is revolutionary, but it has also shortcomings. ViDE can only process deep Web pages containing one data region, while there is a significant number of deep Web pages that have regions that contain more data records. The researchers are aware of a solution to this problem in prior research [25], but this solution is HTML-dependent. This is a good example that employability of algorithms is also a challenge for researchers. HTML is no longer the exclusive web language and most current works have not considered languages such as JavaScript and CSS [15].

There are many more challenges regarding the Hidden Web. For example, there are many sorts of forms that we are not yet capable of processing. And, given that all kinds of categories are present in the hidden Web [7], domain-oriented approaches might not give enough coverage. At this point, it is unclear whether or not a general solution for crawling the hidden web even exists, especially since new forms of hidden data emerge all the time. Furthermore, due to the vast size of the hidden Web, scalability is an ever-increasing issue as well.

Given these challenges, it is important that researchers continue their research on the Hidden Web. Through further research into domain-oriented solutions for Hidden Web Crawling, scaling issues might be mitigated, and by combining various domain-oriented solutions, high coverage might be achieved. In this way, researchers can keep up with the developments of the Hidden Web itself.

2.3 Scheduling problem

2.3.1 Overview

The scheduling problem is a challenge that is very significant to web crawlers. This is because the way a crawler schedules its work greatly affects the quality of the search results from the search engine. Because of this important relation, the scheduling problem is given quite some attention from the academic world. Different solutions have been proposed over the years.

A search engine contains a local repository that has all crawled web pages in it that it uses to process and answer user search queries. However, the enormous size of the Internet makes it impossible to have this repository reflect the entirety of it. There are multiple resource constrictions such as available disk space, bandwidth and processing power that are limiting factors for the search engine and therefore the web crawler. This means that the repository can never cover all of the web pages on the Internet. Also, the content on the Internet is constantly changing; web pages are added, removed and change in content. As a result, the information in the repository will slowly (or rapidly) go out of date; when this has occurred it is called **stale information**. The end result of these factors is a repository that is constantly incomplete and out of date, which in turn negatively affects the search results. The search engine may not know about the existence of a relevant web page for a user query, or worse, will return a page that is no longer relevant or broken.

To counter these issues a web crawler is given two main tasks to maintain the repository. It has to crawl unknown web pages (discovered via links to them) in order to make the repository cover as much of the Internet as possible. In addition to that it has to periodically re-crawl known web pages in order to keep the information in the repository as fresh as possible. The scheduling problem defines in what order the web crawler should crawl these web pages considering limited resources. Web pages on the Internet are not equally valuable to (re-)crawl, so at a high level a scheduling implementation will try to prioritise to crawl the most 'important' pages first.

2.3.2 Developments

Research on scheduling strategies can be found as early as at 1998 [8]. The specific scheduling strategies that mainstream search engines nowadays use is mostly kept secret. This is done in order to prevent abuse or manipulation of the crawling order by third parties. As a result, most research on this subject has compared generic scheduling strategies or newly developed strategies with each other.

Implementation
A scheduling implementation must crawl and re-crawl web pages. In addition to that there are also a few other desirable properties. Although the web crawler is a single entity on a high level, in actuality the work will be executed in a distributed network. The implementation should therefore be highly parallelizable, with many web pages being crawled at the same time. The crawler should also be polite, which means executing its work in an ethical manner. It should not cause denial-of-service (DoS) by scheduling in a way that can overload web servers. Crawling ethics in general will be discussed in Section 2.5. Combining these elements together, the implementation of a scheduling is determined by four policies:

- Selection Policy - to determine how to crawl new web pages
- Re-visit Policy - to determine how to re-crawl existing web pages
- Parallelization Policy - to enable execution in a distributed network
- Politeness Policy - to avoid overloading web servers

More concretely, a web crawler uses a list of URLs that it will crawl from top to bottom. This list of URLs is called a **crawl frontier**. The above policies require at least two frontiers; one for crawling new pages and one for re-crawling existing ones. When an unknown new URL is parsed in a crawled web page, it is added to the frontier for new pages. However, in general more frontiers are being used. One of the reasons for this will be given later.

Importance Metrics
In order to sort a frontier it is necessary to construct a metric for measuring web page importance. Note that this is an abstract concept; there are many metrics possible and the relation between a metric and web page importance is not something that can always be assumed. Suggestions for importance metrics have been made on multiple occasions[8, 1]. A few examples are:

- **Backlink Count:** The backlink count of a web page is the number of web pages on the Internet that have a link to this page. The idea is that a page that is often linked to is likely to be more important than a page that is seldom linked to. As calculating this metric would require knowledge of the entire link graph of the Internet, a web crawler can only make an approximation using the available information from the local repository.
- **PageRank:** PageRank is used by Google to sort search results by relevance for the user. However, this metric can also be used as an indication of page importance and therefore as a way to sort a frontier. Like backlink count, this metric does not require the page source to be already known to the crawler. So it can be used for both crawling and re-crawling. Variants of this metric have turned out to be very popular and effective to use.
- **URL Structure:** The structure of a URL can also indicate page importance. As an example, the root page of a website can be considered more important than a web page that is several levels (slashes) down a site tree. Also, in the case of specialized search engines that only search in a specific domain certain keywords in a URL can affect page importance.

Metrics like the above can be combined to create new ones. The pages in the frontier can then be sorted by their metric value. It is important to emphasize that in general metrics can only be approximated using the repository. To calculate the PageRank of a web page would require knowledge of all web pages on the Internet that have a hyperlink to it. Of course, the crawler only knows about the existence of pages that are present in the repository. This gap in knowledge is relevant because there may be significant differences between metrics calculated using the repository and the 'actual' value calculated over the entire Internet.

Full- and Incremental Scheduling
Having the crawler execute on a single frontier containing all URLs from the repository for re-crawling is not acceptable. Before the entire frontier has been executed and the process can be repeated a relatively long period of time will pass. One would definitely want to re-crawl the 'most important' fraction of pages in the repository in much shorter time intervals. This is an example where the use of more frontiers is necessary. The way to handle this is to have a frontier for the entire repository and a second one for incrementally crawling the most important fraction of pages. This is called full- and incremental scheduling.

An interesting example of an incremental scheduling strategy is called Clustering-based Incremental Scheduling [22]. This strategy tracks how frequent web pages change their content and builds the scheduling around that. This aspect of the data is very relevant because web pages that change frequently (and therefore become stale faster) should in general be re-crawled more often than pages that are more static. The idea is to cluster all the web pages in the repository into a number of clusters, grouped by the average frequency that these pages change over time. This change can be measured and stored every time a page is re-crawled. The strategy is to periodically pool a number of pages from every cluster; when enough pages have changed the entire cluster is re-crawled. Pages are moved to a different cluster when needed.

MapReduce
MapReduce is a programming model that is used to do computing in a distributed network. MapReduce uses two operations, "map" and "reduce" to enable parallelization of a task. A master node receives the input data from the task that it has to process. In the map-step it divides this data into smaller subproblems that it delegates to worker nodes. These worker nodes can potentially delegate their work again to worker nodes below them. In the reduce-step solutions from worker nodes are combined in the master node again to finally return the output data.

This programming model is used in practice for the implementation of a web crawler. The task of crawling a frontier is divided into smaller tasks and executed by many worker nodes. The advantage is that more nodes can be added to the crawler while using the same programming model. This is the key to scaling a web crawler to larger sizes. The parallelization policy that is used needs to ensure that dependencies and shared resources are being handled correctly.

User-Centric Scheduling
This scheduling strategy [19] was proposed in 2005 as an improvement over earlier policies. User-centric web crawling focuses on the re-crawling task of the crawler, the re-visit policy. The main goal is to schedule the re-crawling in a way that maximizes the repository value for the user. It is important to realize that this is different from a scheduling that will try to keep the entire repository as fresh as possible. In that case, pages are implicitly given equal weight to be re-crawled, while this will not equally benefit the value for the user. Users will visit some web pages more often than others, also web pages in the repository will become stale with different frequencies.

The paper defines a metric to measure the quality of a web page result for a given query. An optimal scoring-algorithm, in this case PageRank, is assumed. The essence of user-centric web crawling is to order the pages in the frontier by the expected increase in quality that would be achieved when re-crawling it. This expectation is calculated using increases achieved in earlier re-crawls of the page. After testing this strategy with other methods the paper concludes that

an equal repository quality (for the user) can be achieved using less resources. This strategy is a great example of attempts to improve current scheduling-methods.

2.3.3 Future Challenges

As long as there are resource restrictions for the web crawler, the scheduling problem will remain an important part of the crawling domain. Even though technological progress will increase the available resources over time, it is not on the same order of magnitude as the growing complexity of the Internet. Future challenges will be to keep improving the efficiency and results of scheduling implementations to compensate for these limitations. This can be done by improving one of the four policies that define the behaviour of a search engine. An additional challenge is to prove that a certain policy is better than another one. This would need to be thoroughly tested on large datasets of web pages.

2.4 Web Spam

2.4.1 Overview

Although search engines can easily retrieve millions of web pages for any given query, the user seldomly looks past the first few pages of results. Modern search engines try to fill these first pages with those that are most relevant to the user. For web site owners, attaining a ranking on these first pages is highly desirable and a lot of work goes towards optimizing a web site that will attain a high ranking. This is called Search Engine Optimization (SEO).

While most web site administrators try to attain these rankings by providing good, relevant and easily crawlable content on their web site, others try to attain these rankings by gaming the search engine's algorithms. Respectively, these practices are called white hat SEO and black hat SEO, named after the color of the hats the good and bad guys would wear in western movies. The black hat SEO community is responsible for the phenomenon known as web spam.

Web spam pages are pages that provide little to no relevant content to users. Instead, these pages exist to generate advertisement revenue or to scam the user. Their appearance in search results is unwanted, as they are seldomly what the user was actually looking for. [21] cites that between 6% and 22% of the Web consists of spam pages and cites a 2009 estimation of $130 billion in financial losses due to web spam. Furthermore, the appearance of spam in search results is harmful for a search engine's reputation. For these reasons search engine developers make a best effort to detect such spam pages and exclude them from search results. This section will discuss web spam techniques and countermeasures relevant to web crawling.

2.4.2 Web Spam Techniques

Various web spam techniques are discussed in [11]. The relevant ones for web crawling are:

- Content Spam
- Link Spam
- Cloaking

Content Spam
The most important factor for a web site's ranking is its relevance to a query. This relevance is quantified by a search engine's algorithms. Content spam misleads these algorithms to achieve a higher ranking.

Although the ranking algorithms themselves are trade secrets, the SEO community has accumulated a lot of folklore about what kind of techniques work to mislead them. [21] lists five techniques used by spammers to make their page seem more relevant. All five involve 'keyword stuffing', putting as many search terms as possible in various elements of the page. The five elements listed are:

- Title
- Body
- Metadata
- Link Names
- Site URL

Link Spam
With the advent of algorithms that take the Web's link graph into account, such as Google's PageRank [18], spammers have altered their techniques as well. [21] lists some common techniques used by spammers that make use of the link graph.

- **Outgoing Link Spam:** Some ranking algorithms look at the outgoing links on a page to determine rank. When a page links to a lot of high-quality sites, this page is considered to be of high quality as well. Spammers can easily abuse this by linking to a lot of high-quality web sites. For the spammer, this is an easy, cheap way to get a high ranking. As a result, few search engines still use outgoing link information to determine rank.
- **Link Farms:** Link spam is spam that uses hyperlinks to mislead search engines. Algorithms such as PageRank [18] use incoming links to determine ranking. In the PageRank algorithm each page has a certain number associated with it, called the PageRank. This PageRank is determined by the PageRanks of all incoming links. Therefore, a page that is linked to by some high-quality web sites will have a high PageRank. However, a page that is linked to by a lot of low-quality web sites will also have a high PageRank. For this reason, spammers set up huge networks called link farms to boost the amount of incoming links for a target page; the page that needs to appear high in the rankings.
- **Open content spam:** Many web sites allow users to posts comments and other pages, such as wikis, allow the user to modify the entire page. This is an easy way for spammers to spam a link to their target page.

Cloaking
Cloaking is a technique where the page served to a web crawler differs from the page a user would see. This is very useful for spammers, because this allows them to serve a web crawler a page to optimize rank, while the user sees the page the spammer wants users to visit in the first place. For example, the spam page will contain all the right keyword combinations when a crawler visits, but when the user visits they will be served advertizements or scams.

Spam sites can easily distinguish between a crawler and non-crawler by looking at the User Agent header in the request and the IP address and domain that a request originates from. For example, a request coming from bing.com, identifying as

```
User-Agent: Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)
```

is definitely going to be a crawler for the Bing search engine. Upon seeing a request like that, a cloaked web site will send their special spam page.

2.4.3 Web Spam Countermeasures

Statistical Analysis
Since a spam page is not a legitimate page, it differs in many measurable ways from ordinary pages. [21, 11, 17] describe some of these differences. Examples include

- **URL structure:** More often than not, a long URL with many dashes points at a spam page.

- **Host names:** Some spam pages will have many domains pointing at them, in order to take advantage of search engines that use a domain name to score a page.
- **Outgoing links:** Few legitimate pages have great amount of links on the page. Rather, such pages are usually set up by spammers who wish to trick search engines in giving them a high 'hub score'.
- **Content evolution:** Some spam pages will have different content every time they are requested, while much of the Web changes slowely.
- **Word count:** Since spammers use keyword stuffing to boost ranking, a word count that far exceeds the average is evidence of spam.
- **Invisible Content:** Some spam pages will stuff a lot of keywords in a portion of the page that is invisible to common users. This is done for example by using a very small font or by using the same font color as the background. The presence of such content is evidence of a spam page.
- **Compressibility:** Some spam pages repeat keywords multiple times in an attempt to get a higher score for that keyword. This also increases their compressibility using an algorithm like gzip and therefore the compressibility can be used as statistical evidence for a spam page.
- **Word frequencies:** By analyzing the frequency of each word on a page and comparing this with the known frequencies for the language the page was written in, some automatically generated pages can easily be spotted. If their frequencies are very different from what would normally be expected of this language, it is evidence for a spam page.
- **Word cluster frequency:** As an extension of the above method, not just individual words but also the frequency of word clusters can be checked against the frequency of these clusters in the language the page was written in.

Many of these features are a result of content spam. There are many other features and much research into web spam countermeasures is about finding more features to classify a page as spam. Furthermore, the importance of each feature in the classification of spam is constantly changing, as spammers update their techniques to fool countermeasures.

None of these features alone is enough to accurately classify a page as a spam page, but together they can give enough statistical evidence to classify a page as spam and can be used to build a classifier. After calculating the above features for a large set of pages that are known to be spam or non-spam, a decision tree can be automatically generated using a decision tree building algorithm.

Link Graph Analysis

Since spammers use the link graph in special ways to boost ranking, the link graph for a cluster of spam pages will be different from the link graph of the ordinary web. This can be used to classify a cluster of spam pages (such as a link farm) as spam.

Two related approaches are discussed in [12, 13]. Both approaches use the following property: A good (non-spam) page will generally point at other good pages, but seldomly at bad (spam) pages.

In the TrustRank algorithm this propery is used to find non-spam pages. The algorithm starts with a set of known non-spam pages and assigns these pages a trust value of 1. It then normalizes this score by dividing it by the total amount of trust assigned. So if the original set contained 20 non-spam pages, each will get a trust value of $\frac{1}{20}$. The algorithm then does a predefined amount of iterations. In each iteration it will add some trust to all pages linked by pages that have some trust value, while also dampening the trust value of the original set. Specifically,

$$t^* = \alpha_B \cdot T \cdot t^* + (1 - \alpha_B) \cdot d$$

Where t^* is the trust value, α_B is a dampening factor (a value between 0 and 1, [12] uses 0.85), T is the transition matrix containing the outgoing links and d is a vector containing the starting trust values (in our example, the $\frac{1}{20}$ assigned to the initial good pages). After the predefined amount of iterations, the algorithm will have assigned a trust value to a subset of all pages. This value roughly corresponds with the chance of the page being good (and not spam) and ordering the pages by their trust value gives a good ranking from most trusted to least trusted.

The Anti-TrustRank algorithm works similarly. It starts with a set of known bad (spam) pages, then propagates anti-trust to incoming links. Like the TrustRank algorithm, it continues doing this for all traversed pages for a pre-defined amount of iterations. In the end, a subset of the Web will have been traversed and appointed an anti-trust value, which roughly corresponds with the likelihood of that page being bad, or in other words, spam. The algorithm is exactly the same one as used for TrustRank, the only difference being that rather than a transition matrix for outgoing links, its transpose is used, giving a transition matrix for incoming links.

The only remaining issue is how to select the initial set of pages (the seed) to check manually for trust/anti-trust. Both papers use PageRank [18] for this purpose. Pages with a high PageRank are those pages that are well-connected, so using these pages as a seed will allow a large set of the Web to be assigned a TrustRank and an anti-TrustRank.

Both the trust and anti-trust value of a page can then be used by a search engine's ranking algorithm to weed out spam pages and prioritize trusted pages.

Cloaking Detection

Cloaking shows a different page to a known search engine crawler than it does to a random user. This property can be used to detect cloaking as well. The common method for doing so is to retrieve a page using a known web crawler and another one anonymously, from an ip address not publicly associated with a search engine and without identifying as a crawler. If the two pages differ, there's a good chance cloaking was used.

However, this is not always the case. For example, a page that contains a visitor counter will be different for every retrieval. Therefore, if cloaking is detected using a simple string comparison, the detector will inaccurately classify a lot of pages as using a cloaking technique. In order to accurately detect cloaking, pages must not be checked for equality but for similarity.

An approach to do this, described in [14], discards page text and instead focuses on the used HTML tags. By checking what tags are present or absent in one of the retrieved copies, but not in the other, one gets a measure for how similar these two pages are. [14] proposes three methods to quantify the tag difference. In the notation below, B'_i is the multiset of tags extracted from a copy retrieved while identifying as a web crawler, while C'_i is the multiset of tags extracted from a copy retrieved anonymously. These three methods are called TagDiff2, TagDiff3 and TagDiff4 because they need to retrieve respectively 2, 3 and 4 copies in order to determine a measure of difference between the crawler version of a page and the user version of a page.

- **TagDiff2:**

$$|B'_1 \setminus C'_1| + |C'_1 \setminus B'_1|$$

  In other words, TagDiff2 is the sum of the amount of tags present in B_1 but not in C_1 and the amount of tags present in C_1 but not in B_1, where B_1 is a copy retrieved as a crawler and C_1 is a copy retrieved anonymously.

- **TagDiff3:**

$$(|B'_1 \setminus C'_1| + |C'_1 \setminus B_1|) - (|C'_1 \setminus C'_2| + |C'_2 \setminus C'_1|)$$

  In other words, TagDiff3 is TagDiff2 minus the sum of the amount of tags present in C_1 but not in C_2 and the amount of tags present in C_2 but not in C_1, where C_1 and C_2 are two copies retrieved anonymously.

- **TagDiff4:**

$$|(B'_1 \cap B'_2) \setminus (C'_1 \cup C'_2)| + |(C'_1 \cap C'_2) \setminus (B'_1 \cup B'_2)|$$

In other words, TagDiff4 is the sum of the amount of tags that B_1 and B_2 have in common minus all tags in C_1 or C_2 and the amount of tags that C_1 and C_2 have in common minus all tags in B_1 or B_2, where B_1 and B_2 are copies retrieved as a crawler and C_1 and C_2 are copies retrieved anonymously.

In order to classify a page as cloaked, the result of a TagDiff calculation is to be compared with a threshold value. If it is below the threshold value, the page is classified as non-cloaking. Otherwise it is classified as cloaking. [14] calculated the precision (ratio of pages accurately classified as cloaking to the total amount of pages classified as cloaking) and recall (ratio of pages accurately classified as cloaking to the total amount of cloaking pages) of the three methods. This uses a set of known cloaking and noncloaking pages, using various threshold levels to see how the three methods performed. Their tests showed that all three methods performed well when compared with similar methods that used link difference or word difference. Furthermore, TagDiff4 had the highest precision, while TagDiff2 had the highest recall.

2.4.4 Future Challenges

Web spam, the deliberate misdirection of search engines in order to boost one's search ranking, is an ever-present problem for search engines. Web spam comes in various forms, most notably content spam and web graph spam and is further augmented by techniques such as cloaking. In order to detect web spam, search engines use statistical analysis, link graph analysis and cloaking detection.

Both web spam and web spam prevention techniques are constantly changing. Whenever a new web spam method emerges, search engine developers develop countermeasures. Whenever countermeasures are developed, spammers develop new spamming techniques. Since no countermeasure is sufficient to detect every class of spam that exists and every class that will be developed in the future, this arms race can be expected to continue indefinitely.

2.5 Crawling Ethics

2.5.1 Overview

During the years, web crawlers have evolved into very large systems, continuously downloading extremely high amounts of data. Increasing the scale is essential in order to keep a crawler serve its purpose. The information on the Internet changes every second, which requires a crawler to keep updating its collected data quickly. When a crawler crawls the pages faster, more information can be gathered in a shorter period of time, and its data will remain accurate and up to date. This entails a greater amount of server requests per time unit, which can cause various problems for the web server. Also, the fact that web crawlers are automated systems, accessing the same web pages as humans do, can cause undesirable behavior, harming the web page or its owner. There is no incentive for crawler developers to prevent these problems, because having to deal with them would result in less accurate web crawlers, reaching a smaller range of data and requiring more time to crawl. Therefore, solutions have to be invented to reduce the damaging side effect of web crawlers and ensure the healthiness of the crawled web page.

2.5.2 Ethical Issues

What and What not to Crawl

If a web crawler has not been given any restrictions of what and what not to crawl, usually the entire collection of linked web pages will be crawled. One might argue whether a robot should download every page that it can find on the Internet. There are several arguments why this should not be the case. First, although it seems that a random user (and a robot) can access the page, the page may still contain private information. Indexing this information and presenting it in a search engine's search results may harm the owner of the web site. Second, information on certain web pages may be totally irrelevant to other people and therefore should not be indexed. Third,

it may be argued whether it is even legal to download and make a permanent copy of copyrighted web pages [23]. The Internet Archive, providing great amounts of web pages and other material, does the same, but enables someone that has trouble with this, to do a request for deleting the concerning material [23]. Until now, no strict rules that prohibit search engines to do this have been established.

Server Overload and Denial of Service
When a search robot executes its algorithm, crawling web pages without deliberate brakes, it does much more server requests than a human. Where a human will at least stay for ten seconds on a single web page, a web crawler may have visited ten pages in one second. Especially when a web site already has a large amount of visitors, the laborious web server may overload when a robot starts doing a high amount of requests [24]. This is a serious issue. Earlier research investigated some major search engines and analyzed whether they adapted their downloading speed, depending on the server workload [24]. It seemed that none of these engines changed its downloading speed during the hours that web sites had the most visitors. A situation where the overload is so severe that the server is not capable to respond to requests anymore is called a Denial of Service [23]. Depending on what type of system the server runs, the consequences can be catastrophic. A news web site that is temporarily unavailable does not directly turn into a disaster, but if the security system of a bank (accidentally) goes down, there is a bigger problem.

Traffic Costs
Besides impeding the server speed, a high amount of server requests may also incur network costs, paid by the server owner [23]. Search engines do not pay for their visits at all. There is also no suitable method to implement a small monetary compensation for downloading a web page. One way in which for example Google 'pays back' for its server usage is making the web site findable in Google's search results, resulting in a higher amount of visitors [23]. Important to note here is that not all web sites find this desirable. In contrast to Google, there are also search engines that provide no direct compensation for their server usage. Some only crawl for their own interest or for academical research. Although this may be useful research, the server owner did not choose to pay for this [23].

Responsibility
Although a search engine is a fully automated system, it is always created by human beings. When a case occurs that webmasters are harmed in any of the earlier described ways, the question that comes up is: Who is responsible? Is a creator fully responsible for the behavior of his machine or should webmasters be more careful in which information they show in public web pages? Should the search engine creator always have the chance to compensate the mistakes of the engine afterwards or are some mistakes irreversible and requiring punishment?

2.5.3 Developments

The Robots Exclusion Protocol
Soon after the first search engines came up, a need arose to limit the search range of crawlers to some extent. Webmasters wanted to be able to tell a crawler which web pages or areas must not be indexed. This can protect the server against an overflow of crawler requests. Also, it enables the webmaster to prevent test areas or other irrelevant data from being indexed [23]. Furthermore, certain pages are only meant for humans and may cause undesirable effects by being visited by a robot (e.g. an application form). In 1994, the Robots Exclusion Protocol was introduced (http://www.robotstxt.org/orig.html). A webmaster can use this protocl by placing a robots.txt file in the main directory of the web site. The content of the file could be as follows:

```
User-agent: *
Disallow: /dir1/
```

```
User-Agent: Googlebot
Disallow: /dir2/index.html
```

The first rule defines which types of user-agents the following rules apply to. In the HTTP protocol, clients always send a User-Agent HTTP Header. Most major search engines will identify themselves with this header. The star is a wildcard and states that the rules apply to all types of robots. The second rule states that pages in section *dir1* should not be indexed. It is also possible to give specific instructions to different web crawlers. In the example is shown that the Googlebot is not allowed to crawl the index.html page in section *dir2*.

An alternative to the robots.txt file is the meta robots tag [10]. This tag is placed in the head section of a web page and tells a robot whether this page should be indexed or not and whether the links on this page should be followed or not. The meta tag could look like this:

```
<META name='ROBOTS' context='noindex, nofollow'>
```

This tag tells the robot not to index this page and not to follow the links on the page.

Both methods are easy to apply for webmasters to prevent undesired behavior of web crawlers, but they do not stop all robots from visiting these pages. It is just a protocol and does not make it impossible to visit the listed pages. Therefore, spam robots looking for e-mail addresses will not adhere to this protocol. Only the web crawlers that have interest in being ethical (and having a clean public image) may obey this protocol [23].

The UPDATE File

An approach to reduce the traffic caused by web crawlers, is by informing them which files have been updated and when. This can be done by listing the URLs of the updated files in a so-called UPDATE file in the main directory of the web site [16]. When a page is updated, the URL is placed in the first line of the file. Web crawlers start by comparing the UPDATE file with their previous visit and only re-crawl what has been changed. This requires much less traffic and is therefore much more efficient compared to the method of checking each file for updates everytime. In contrast to the Robots Exclusion Protocol, this method is not used in general. A requirement is that the webmaster or an automated system, actually stores all updated file URLs into the UPDATE file. If this process is neglected, the method will not work and the details that the search engine stores about the web site will be out of date. That is a reason why this protocol has not been implemented yet. This is an important fact to notice: There are many methods that could be implemented in order to drastically improve the crawling process and make it more efficient, but it always requires flawless use of protocols. In practice, this is never the case. Therefore, search engines prefer to function autonomously, not having to deal with all kinds of protocols and limiting rules, but just crawling what they think is useful.

Workload-Aware Crawling [23]

Previous developments were examples of ways that webmasters could influence search robots. This development deals with the crawler itself. This technique shows a way in which the server workload can be estimated and the ratio of requests can be adjusted to tha. Estimating the server workload is done by measuring the difference between the HTTP Response Delay and the ICMP Response. The HTTP Response Delay already gives an estimation of the server workload, but by reducing the ICMP Response time, the delay caused by the network (which is irrelevant) is partly filtered. When this information is gathered, the search engine can lower its server requests and download speed when the Response Delay is high. This will reduce the risk of server overload.

2.5.4 Future Challenges

Most developments shown above are trying to solve these side-effects of web crawlers in a technical way, often using clever tricks to reduce these effects. One might argue if there may be something wrong about the way web crawlers work in general. Is it acceptable that without restrictions, crawlers will download everything? During the years, search engines have become indispensable

for individuals. That is a main reason why complaints have never resulted into major changes. People might still not be fully aware of the fact that the Internet (in particular search engines) never forgets and publicly released information will almost always be indexed.

3 Conclusion

Because of the massive size of the World Wide Web, it has become very hard to find everything you need without using a search engine. Over the years search engines continued improving their crawlers. This paper served as an introduction to the challenges for improving web crawlers. After a short explanation of how a search engine works, the paper focused on the web crawling itself and some important issues it has to deal with.

First, we discussed the Hidden Web, the portion of the Web that is hidden behind search forms in large searchable databases. In 2000, the hidden Web was estimated to have between 43.000 and 96.000 web sites - a number that has been expanding tremendously, reaching a number of 25 million hidden databases in 2010. There are also many algorithms developed to deal with the hidden Web. The first of this kind was the Hidden Web Exposer (HiWE), an algorithm that fills in the search forms to get to the generated pages. A disadvantage of this algorithm is that it needs human assistance and that is why other researchers tried to improve their algorithms. The newest algorithm we discussed is ViDE, a Web-programming-language-independent algorithm that uses the visual aspects of web sites for data extraction.

A second challenge for web crawling is scheduling. A web crawler has two main tasks to maintain the local repository. It has to crawl unknown web pages, but also has to periodically re-crawl existing web pages in order to keep the information in the repository as fresh as possible. A scheduling should also be polite and be parallelizable in its execution. These four aspects form the four policies that determine the behaviour of the scheduling. To determine in which order the crawler visits URLs in a frontier, the crawler uses metrics to measure page importance. The repository is not only fully re-crawled but also incrementally in order to reduce staleness. Finally, MapReduce is a programming model that can be used to parallelize the scheduling of a crawler and enable up-scaling the crawler.

With web crawlers working with rankings and metrics, most website administrators try to attain high rankings by providing good, relevant content on their website. Others try to get high rankings by gaming the search engine's algorithms and are responsible for the phenomenon Web Spam. We discussed three web spam techniques that are relevant for web crawling. First there is Content Spam, where spammers use 'keyword stuffing' to get as many keywords as possible in their website's content. With Link Spam, spammers try to get high rankings by getting their page linked with other pages or link their page to many other pages. A last technique is Cloaking, where the page served to a web crawler differs from the page the user eventually would see. For all these spamming techniques, we also discussed countermeasures. These are Statistical Analysis, Link Graph Analysis and Cloaking Detection.

Last, we discussed ethical issues web crawlers have to deal with, like the question: "What and what not to crawl?". For this question there has not been any strict rules that prohibit search engines to search particular web sites. Other important issues we discussed are server overload, traffic costs and responsibility problems. As an example of the traffic costs issue, Google compensates for its server usage by making websites findable in its results, but there are enough search engines that provide no direct compensation for their server usage. As a development in the ethical area, some exclusion protocols have been developed, but a disadvantage of this protocol is that only web crawlers that have interest in being ethical may obey this protocol.

Besides discussing the developments of the issues described above, we shortly stated the future challenges web crawlers have to deal with. The biggest challenge for web crawlers will always be how to handle the immense size of the Web. The World Wide Web is an growing entity, so the numbers of hidden databases and the number of spam sites will also increase. And with a bigger area of pages to crawl, what will be new scheduling schemes? Besides the size of the Web, people might still not be fully aware of the fact that the Internet never forgets. This will lead to

bigger ethical questions about web crawlers and form new challenges for the crawlers to keep their information fresh.

When we look into the near future, we say that the greatest challenges researchers have to deal with will be the combination of how web crawlers can overcome the size of the Web and how they can be automated to get the highest coverage of relevant information (and not spam, for instance) at a high speed. Now, web crawlers that get much relevant information still need specifications from humans, are domain specified or get automated results that do not meet the ethical requirements. To overcome these problems will be enough challenge for researchers for the upcoming five years. And when they do, this will definitely benefit the user.

4 Acknowledgements

We wrote this paper for a research course of the Bachelor Computer Science at the TU Delft. We want to thank Carsten Eickhoff, who supervised and supported us during our research. He was also an important reviewer who provided many helpful suggestions.

References

[1] Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. Crawling a country: Better strategies than breath-first for web page ordering, 2005.

[2] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. *Proc. of SBBD*, pages 309 – 321, 2004.

[3] Luciano Barbosa and Juliana Freire. Searching for hidden-web databases. *Eighth International Workshop on the Web and Databases*, 2005.

[4] Michael K. Bergman. The deep web: Surfacing hidden value (white paper). *Journal of Electronic Publishing*, 7(1), 2001.

[5] BrightPlanet. www.brightplanet.com.

[6] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine, 1998.

[7] Kevin Chen-Chua Chang, Bin He, Chengkai Li, Mitesh Patel, and Zhen Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3), 2004.

[8] Junghoo Cho, Hector Garcia-Molina, and Lawrance Page. Efficient crawling through url ordering, 1998.

[9] Arjun Dasgupta, Xin Jin, Bradley Jewell, Nan Zhang Zhang, and Gautam Das. Unbiased estimation of size and other aggregates over hidden web databases. *SIGMOD '10*, 2010.

[10] M. Carl Drott. Indexing aids at corporate websites: The use of robots.txt and meta tags, 2002.

[11] Dennis Fetterly, Mark Manasse, and Marc Najork. Spam, damn spam, and statistics. In *Proceedings of the Seventh International Workshop on the Web and Databases*, pages 1–6, 2004.

[12] Zoltán Győngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *Proceedings of the 30th VLDB Conference*, 2004.

[13] Vijay Krishnan and Rashmi Raj. Web spam detection with anti-trust rank. In *SIGIR 2006 Workshop on Adversarial Information Retrieval on the Web at AIRWEB 2006*, pages 37–40, 2006.

[14] Jun-Lin Lin. Detection of cloaked web spam by using tag-based methods. *Expert Systems with Applications*, 36:7493–7499, 2008.

[15] Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vide: A vision-based approach for deep web data extraction, 2010.

[16] Shekhar Mishra, Anurag Jain, and Dr. A.K. Sachan. Smart approach to reduce the web crawling traffic of existing system using html based update file at web server. *International Journal of Computer Application (0975 - 8887)*, 11(7), 2010.

[17] Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the 15th International Conference on World Wide Web*, 2006.

[18] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[19] Sandeep Pandey and Christopher Olson. User-centric web crawling, 2005.

[20] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web, 2001.

[21] Nikita Spirin and Jiawei Han. Survey on web spam detection: Principles and algorithms. *SIGKDD Explorations*, 13(2), 2012.

[22] Qingzhao Tan and Prasenjit Mitra. Clustering-based incremental web crawling, 2010.

[23] Mike Thelwall and David Stuart. Web crawling ethics revisited: Cost, privacy, and denial of service. *Wiley InterScience*, 2006.

[24] Shaozhi Ye, Guohan Lu, and Xing Li. Workload-aware web crawling and server workload detection, 2004.

[25] H. Zhao, W. Meng, and C.T. Yu. Automatic extraction of dynamic record sections from search engine result pages. *Proc. Int'l conf. Very Large Data Bases (VLDB)*, pages 989–1000, 2006.