Text

Description automatically generated

**Evaluation of the performance of a web crawler within an error message based search engine application.**

by

Owen Skilton

Supervisor: Dr Michael Dacey

Second Marker: Dr Tim Bashford

Project submitted as part of the requirements for the award of MSc Software Engineering

January 2022

**Table of Contents**

1. **Introduction 3**
2. **Literature Review4**
3. Brief History4
4. Search Engines5
5. Web Crawlers7
6. Crawl Policies7
7. Crawl Limit9
8. Web Scraper10
9. Legal Issues11
10. Performance Evaluation of web crawler11
11. **Research and Development methodologies15**
12. Experimental Design15
13. Development Methodology16
14. **Design and Implementation16**
15. **Results18**
16. Top Ranked URL based on similarity measure18
17. Crawler Evaluation19
18. **Discussion24**
19. **Conclusion27**
20. **Reflection27**
21. **References29**

*Abstract* - **Extracting information from the World Wide Web is an incredibly important process. For developers, it’s a resource they are able to turn to when solving and debugging complex software based problems. Most search engines utilise a web crawling based framework. A web crawler is an automated robot that is able to download and extract all the resources available from a web page. In this paper, an error message based specific web crawler will be designed and evaluated based on the returned results as well as the time taken to perform these actions. It will utilise a selection policy algorithm to return the most appropriate results based on a keyword search and the contents of the web pages found. It will be evaluated against different resources that contain vastly different software architectures in order to accurately measure how the crawler is able to perform in different software environments.**

1. Introduction

The World Wide Web is an ever expanding network, meaning that tools that are able to search effectively across it, such as search engines, are only becoming increasingly more important as time goes on. Along with the World Wide Web, the software development industry continues to grow year on year [1]. A report done by the university of Cambridge [2] estimate that developers spend almost half of their time debugging applications during the development lifecycle, the most popular tool utilised to find information related to these problems is the World Wide Web. Being able to search the Web effectively and efficiently, containing millions of interconnected documents through URLs and hyperlinks, will drastically decrease this lead time in development leading to lower overhead meaning a much more productive and efficient software production life cycle. This led to the idea of creating an error message based search engine that’s able to rank the pages found based on the contents in order to supply the most appropriate solution to a user’s query.

There are many different ways of ranking webpages based on a keyword or phrase contained within it contents. Some of these algorithms include the page ranking algorithm introduced by google in 1998 [3] as well as the Hyperlink-Induced Topic Search (HITS) algorithm using link analysis [4]. Web ranking is a crucial step in the search engine process as it allows for the device that will access and download the information to be as efficient as possible, the web crawler. A web crawler (also known as a spider or harvester) is an automated script that begins on any website, known as the seed URL, and extracts all the external webpages that it points to via URLs and hyperlinks as well as usually performing some function such as graphical indexing or downloading of the entire contents of the webpage [5]. These links found, assuming not been found previously, are added to the stack to be crawled in the future and the process is repeated until no new links are able to be added. This process creates a huge tree of webpages that are interconnected and a vast amount of data, usually a downloaded version of the web pages contents.

These contents can then be analysed in order to determine which web page discovered matches the user query best using any of the multiple web ranking algorithms [6][38]. The crawler developed within this report will begin on popular error based forums or documentation websites in order to find and evaluate the best resource to find software based error message problems. It will crawl the website to find any internally pointing hyperlinks and downloaded the entirety of the web pages into a repository. Any duplicate URLs found will be rejected to ensure maximum efficiency of the crawler, which will then be evaluated based on the results achieved and the performance it adhered to. An architecture of this process is able to be seen below in figure 1.

Diagram

Description automatically generated

Figure 1, architecture diagram for a simple web crawler. [7]

1. Literature review:
2. *Brief history*

The first search engine was designed as an internet searching tool within the university of Montreal

In 1991. “Archie”, short for archives, was a script based program that would download all the files located on Free Transfer Protocol (FTP) sites and then created a database that was searchable via filenames. The year after, “Gopher” was created at the university of Minnesota which was described as “a distributed document search and retrieval network protocol” [8]. Meaning it was able to index plain text documents. This was shortly followed by “Veronica” and “jughead” which were able to search the Gopher index more effectively before the launch of the world wide web in 1993. The first web search engine released was “Wandex” which possessed the first web crawler known as “World Wide Web Wander”, created by Matthew Gray [9]. “World Wide Web Wander” was primarily used to keep track of the size of the growing platform rather than a traditional web crawler that we know today. [10] The first modern Day web crawler-based search engine was created in the same year, 1993. “JumpStation”, was the first engine that relied on a fully automated bot to index links available to be displayed to the user as a list of URLs that match some form of keyword criteria. “JumpStation” is considered the first search engine that behaves similarly to modern day crawler-based search engines.

In today’s day and age, modern search engines are able index billions and billions of web pages, images and video or audio files within mere seconds thanks to the evolution of web crawlers. In the very early days, website creators would participate in submitting their own indexing for pages they wish to be listed on a search engine, while engine administrators would manually edit the links into a particular format to allow indexing. The first engine that indexed over 1million pages occurred in 1995 with the Pittsburgh's Carnegie Mellon Universities search tool “Lycos” that was able to supply page ranking for relevance as well as snippets of content from induvial sites [11]. Modern day engines such as google utilise an entire network of crawlers in order to scour the billions of indexed pages on the world wide web. Early crawlers would typically be dealing with small data sets ranging from anywhere between a few kilobytes up to a few dozen megabytes. Whereas Google estimates that their entire search index contains over a few hundred billion web pages alongside over 100 million gigabytes of content. [12]

It’s difficult to pinpoint the origin of web scraping but the first popular library that allowed developers to harness the ability of web scraping was a Python library known as BeautifulSoup created in 2004 [13]. BeautifulSoup allowed developers to parse HTML elements an extract key information from any indexed web page. This had the potential to save developers hours of time when creating applications that wished to extract data from hundreds upon thousands of sites such as price monitoring on e-commerce sites, financial data aggregation for stock based applications or for modern data crypto currency apps and finally news tracking applications that could congregate a large volume of material across the web onto one website. Soon after the release of BeautifulSoup web scraping began growing in popularity. Dedicated software began to be created such as Web Integration Platform V 6.0 by Stefan Andersen. Users were able to highlight key information within a web page and extract the data to an excel spreadsheet or supported database of their choosing for easier access and analysis. Web scraping is continuing to be utilised more and more as the World Wide Web appears to be growing endlessly and with the rapid development of artificial intelligence, the future would suggest a more predictable nature for web scraping that could allow more accessibility and freedom to data contained online.

1. *Search engines*

A search engine is defined as a software system that is able to carry out textual based search queries that scour the world wide web in a systemic way following strict protocols in order to return a list of indexed URLs that are commonly sorted by the relevance associated with the search query provided. [14] Common search engines include Google, Yahoo and Bing are all characterised as Crawler-based search engines [15], that is they share common characteristics surrounding the crawling of websites in order to extract information for a search query. The other type of search engine is known as human-powered directories [16], these can be defined as performing searches based on organized subject categories. Human-powered directories utilise human engagement in order to filter and provide a condensed list of relevant results to a specific topic or subject, examples in real life include NowNow which was created by Amazon and ApexKB [17].

The characteristics of a crawler-based search engine are able to be boiled down into a few main components:

* A crawler that is able to systematically browse through the web in order to perform some sort of indexing of potential websites to be returned
* A database that is able to store the type of indexing performed by the web crawler
* A series of programs or algorithms that is able to assess / rank the sites returned from the web crawl that provide the most relevance to the search query
* A search interface that acts as the intermediary between the user and the database

These characteristics form the building blocks to almost all commercial search engines that exists today, most search engines providers thus have their own individual applications and systems that are able to perform / meet these characteristics. For example, Google utilises their own web crawler Known as Googlebot [18], a database that is able to hold an incredible amount of large data known as Bigtable [19], a site ranking algorithm that is able to measure the importance based on the search query known as PageRank [20] and finally a search interface such as google.com.

The ability to accurately rank and assess the quality of a web page relative to the keyword search is want distinguishes a good search engine from a bad one fundamentally. A large reason for Googles success in the early days of their development was down to the page ranking algorithm known as PageRank [20]. PageRank is defined as a link analysis algorithm, name after co-founder Larry Page, that assigns a numerical weight to each element of a hyperlink set in order to evaluate the importance / popularity of a web page. A higher PageRank value denotes that the web page has a higher volume of traffic of users in a given period of time as well more web pages within a given set that point to the specific web page as a hyperlink, known as “link popularity”.

The algorithm possesses both a complex version involving eigenvalues of matrices and an element known as a dampening factor, a constant that’s described as if a user browsing through links will eventually stop clicking, the probability of a user stopping at any given point is known as the dampening factor (usually = 0.85). The simplified algorithm is best described using a scenario where only 4 web pages exist. Modern day versions of PageRank apply a probability distribution of between 0 to 1 between the 4 pages, therefore the initial value of PageRank for the 4 pages is 0.25.

The formula to calculate the PageRank value is denoted below in figure 2. It describes the PageRank value of a web page being equal to the sum of the PageRank of all documents that possess an outbound link that points to the web page, divided by the number of outbound links in total.

An example scenario would be if we were to calculate the PageRank for web page A and all 3 other web pages had an outbound link that pointed to it. Page B had 2 outbound links, C only 1 to page A and D a link to all 3. The formula to calculate this can be seen in figure 3 which would equate to a PageRank value of 0.458 (assuming an initial PageRank of 0.25 for each page B though D).

A picture containing diagram

Description automatically generated

Diagram

Description automatically generatedFigure 2, formula to calculate PageRank of a web page with 3 nodes that possess an outbound link pointing to it. [21]

Figure 3, formula to calculate scenario problem [21]

1. *Web Crawlers*

A web crawler, often called a spider, was created in the 90s as a tool used by search engines or other applications for the purpose of indexing or copying information in order to perform fast and efficient searches. A web crawlers process usually begins with a list of appropriate URLs to index, known as seeds [22][39]. A crawler will then crawl these seeds and depending on its function, indexing or archiving for example, will perform a number of actions such as downloading the HTML contents or constructing a graphical representation of the traversable URLS as well as extracting all the possible hyperlinks that are accessible on the individual web page with the purpose of storing these in the list of URLs for indexing in future. Most crawls will iterate through the list of URLs they intend to visit to ensure that no duplicates are re-entered. It is important that crawls act as efficient as possible so parsing through a web page that has already been crawled will not only take more time for the crawl to complete but will consume network and hardware resources as well.

1. *Crawl Policies*

The function of a web crawler is relatively simple as it repeats the same function(s) until all hyperlinks are indexed. However, the behaviour of a crawler in the way it conducts these functions are a product of 4 main crawling policies, selection policy, re-visit policy, politeness policy and parallelization policy.

*Selection policy:*

The selection policy defines a metric for the importance of a web page and creates an order of prioritization for the crawler to carry out its intended function, whether that be downloading or monitor for maintenance to ensure hyperlinks direct to the intended resource. This selection process is comprised of multiple factors but the most important and highest weighted ones for the algorithm is the quality of the content within the web page, the popularity of the page, the number of visits it has had in a set time frame and the number of pages containing a hyperlink to the same endpoint and finally how relevant the contents of the URL itself matches the search parameter supplied [21].

*Re-visit policy*:

The 2nd policy that effects the behaviour of a web crawler is the re-visit policy. This is a short policy that tells the crawler how frequently to return to the current web page being index in order to check any editions that have been made to the resource. With crawls that can take anywhere from hours to a few weeks, many pages during will have had edits. Therefore, it is important to return to these web pages within a specified time frame, especially for crawlers whose primary function is to download and maintain an accurate and update version of any page that may be indexed in the near future.

Vipul Sharma of Punjab university proposed an interesting analysis of combining the PageRank algorithm assignment on the relevance of a web page, with the Prato 80 – 20 principle [23] which states that 80% of effects comes from 20% of the causes. The idea is to better allocate a crawler’s resources during this revisitation policy as to crawl web pages more effectively and efficiently while minimizing resource consumption, such as power and bandwidth. They surmise that pages that possess a higher PageRank as well as a documents with a more recent modification date should be crawled more frequently as they tend to receive a higher volume of traffic and be updated more regularly. The assumption being that a web page crawled too frequently leads to wasted resources and one crawled too infrequently diminishes the quality of the search engine itself. By allocating web pages into collections denoted 20% and 80%, with the 20% collection containing web pages that are required to be crawled more frequently, they attempt to find a balance between the revisitation policy of popular and minor web pages to provide both a more accurate search result and a more efficient solution to the resource consumption problem. [24]

*Politeness policy:*

Due to the scale and automation of a web crawler, it is able to retrieve and index data incredible fast, a single web crawler alone can send 1000s of request per second to an individual site. Most crawlers work within a network as well, meaning multiple at any one time can be interacting with the same server, although accessing a different web address. A biproduct of this speed and automation is the vast consumption of network resources that are needed. This can have a crippling effect on a website or servers’ ability to process request and maintain visibility to other users. Due to these factors the implementation of a politeness policy is crucial. The politeness policy states that the crawler will not intentionally cause a negative impact to a network or server by restricting the number of requests made within a period of time or specified webpages that a web administrator deems as either sensitive or not useful for a crawler’s function. Matijn Koster noted that crawlers that are left to their own devices can have an incredibly negative impact on the quality of life for web users, thus the importance of the politeness policy. [25]

The solution that was devised to avoid this problem was presented by Martjin Koster in 1994 known as the robot’s exclusion protocol [26][40], Google then proposed that this be made an official standard to the Internet engineering task force (IETF) [27]. It’s a standard utilised by websites administrators to inform crawlers about which areas of a certain website should not be processed. A robots.txt file will exist utilising the root URL with a */robots.txt* (E.g. <http://site.com/robots.txt>) extension that will contain strict instructions for what URL extensions the crawler is prohibited from accessing. This policy is usually the first to be executed during the crawl process. A common process for the crawler to adhere to is first perform a get request on the root URL with the extension of robots.txt to retrieve the text file, this is performed before any crawling is applied to the root URL. The file is then read for any commands that contain the disallow keyword with extensions present. The crawler will identify itself within the HTTP request using the header ‘user-agent’ and will then read the disallowed extensions related to that user-agent. An example of this can be seen in figure 4 for youtube.com. If this file doesn’t exist the crawler will just assume that the site has no limitations on what can and cannot be accessed.

Text

Description automatically generatedFigure 4, yotube.com robots.txt file for disallow URL extensions for any user-agent.

This disallow keyword meets the criteria for restricting access to pages the site administrators do not wish to be crawled due to either privacy or the irrelevance of the contents for a public facing search engine. However, this does not solve the issue of restricting the interval of connections made to one endpoint. The solution for this is use of the crawl-delay command within the same txt file and the associated URL extension. The intention is for a crawlers to read this delay, usually represented in seconds, and restrict itself from visiting the specified URL within the delay proposed. This is not apart of the standard presented to the IETF however, meaning that crawlers may interpret this delay command differently. An example of this can be seen below in figure 5, twitters robot.txt file that implements a 1 second delay for all crawlers traversing the site.



Figure 5, twitter.com/robots.txt, containing the delay command to limit reconnections for crawlers traversing the site.

It is possible however for crawlers that have malicious intentions, such as spam or DDOS attack and phishing attacks, to not follow the convention of restricting its access to the specified domains or adhere to the request delay. To avoid this it is the job of the security administrators to implement a policy to recognize and restrict access to these malicious scripts to prevent any damage such as down time or the exposure of sensitive information. In order to prevent these malicious crawls, it is important to identify common characteristics amongst these crawlers as well as prevent access once they have been detected. Many tools exist for this detection such as the PathMarker tool proposed by Shengye Wan et al [28] to detect or simply slow down the efficiency of malicious crawlers, based on their visitation path as well as their time features, to prevent them from downloading server based content.

*Parallelization policy:*

The final policy that crawlers adhere to is the parallelization policy. Most crawlers that wish to index a large quantity of URLs and download their contents for indexing such as search engines, will execute in parallel with each other. This is known as a parallel crawler. [29] This policy relates to maximising the download speed of the web page while avoiding repeated download of content within the same web address. If 2 crawlers in parallel retrieve the same URL during the indexing of a web page, it is important for the assignment of this new URL is only added to the stack once to avoid this download duplication.

1. *Crawl limit:*

Within the politeness policy for a good Samaritan web crawler whose main intention is to crawl through a site peacefully and does not intend on throttling bandwidth to other users trying to access a website, another parameter can be assigned known as the crawl limit or crawl budget. This is defined as the number of parallel continuous connections that will be accessing the resource simultaneously as well as the time to wait between future connections to the same endpoint (assuming a robot.txt file is not present). Depending on the crawler, this value may fluctuate depending on a number of factors such as response time for loading a resource as well as direct instruction on the number of connections or request time between fetches. Google for example monitors these using a statistic they define as “crawl health”, which is used to measure the response time of the server where the resource is located and allocating an appropriate number of crawl bots to minimize the impactfulness of the crawl. [30]

1. *Web Scraper*

With the vast growth of the internet during the early 2000s, being able to accurately gather and maintain data in a timely manner was growing too cumbersome for a simple copy and paste job. Tools and libraries were being developed that allowed the extraction of key information from a vast number of websites to be harvested in a shorter period of time, much faster than a human being able to copy the information from one location to another. Thus, the web scraper was born.

A web scraper, also known as a harvester, is an automated process that usually involves the interaction of a web crawler whose intention is to extract key, unstructured data from a website usually in the form of interaction with HTML or DOM based elements and convert it into structured data. This data is usually extracted and saved to some sort of database or excel spreadsheet for use and analysis in the future. A common action for a system whose intention is to scrape information form specific web pages is to use a focused crawler whose seed URL will relate to a specific website or keyword for a topic. The crawler will begin to download the contents of these pages and extract other URLs and add them to the queue of crawl items. Commonly some form of selection policy is applied on the list of URLs. This will allow the scraper to begin acting on sites deemed most sufficient to match the topic or search keywords in the hope of supplying the most accurate and up to date information as possible. The scraper will parse through the downloaded HTML elements to extract any data deemed necessary whilst saving or presenting them in a specific format to a user or database.

Web scraping is an incredible versatile technique for extracting data from web sources and is able to be tailored and modified depending on what resource needs to be accessed [31]. Some common applications of web scraping include:

* Website editions detection
* Price comparisons
* Scrape job posting / message board
* Extracting business details from business directories such as a long chain of employee emails
* Weather data monitoring

Multiple different techniques can be utilised in order to maximise the efficiency of data extracted thus minimizing the time for the scrape. The most common technique involves HTML parsing. A technique most efficiently used when web pages encapsulate data within some form of wrapper that possess some form of common template between components. Another technique commonly used across scraping software and applications is DOM parsing. Embedded programs within web browser are able to retrieve content dynamically from the generated DOM trees formed from client side scripts.

Many commercial web scraping tools and software are available for purchase or even free use. Popular packages include Visual web reaper, web content extractor, UiPath, import IO and Web scraper. Visual web reaper and web content extractor are user friendly oriented web scraping software packages that allow for unstructured data extraction to either CSV files or a multitude of database software support such as MySQL or ODBC. UiPath is similar however attempts to replicate a user within a web browser more accurately and offer web service integration for other applications. Import IO is a free online software tool and maintains extracted data in the cloud rather than locally. Finally, web Scrapper is a browser extension based scraper and allows scheduling and parallel crawling similar to a search engine crawler in order to maintain data sets up to current format.

1. *Legal issues:*

Within the topic of web scrapers, the legality of which is a very grey area across various regions. The action of scraping in general, especially for non-commercial motivation is considered acceptable but once the action enters the area of copyrighted material uncertainty arises. A famous case involved LinkedIn and a data analytics company known as HiQ in 2019. HiQ began scraping LinkedIn using automated bots to extract information from public LinkedIn profiles. The ninth circuit court of appeals ruled in favour of HiQ creating a precedent and implying that scraping public information from social media no matter the level of personality is legal. LinkedIn have stated that they intended to escalate and take the case to the supreme court of the United States.

Both the UK and US share similarities on actions that may lead to an individual being on the receiving end of legal action. The first, as mentioned previously, is the scraping of copyrighted material that may infringed on intellectual property. A federal court ruled in 2013 that a software as a service news aggregator who offered scraped news articles as a subscription service was deemed to be in breach profiting from copyrighted material and “substantially” causing a negative effect of the potential market value of said work. This case is similar to the LinkedIn vs HiQ which leads to the grey area of what is deemed acceptable scraping practices in terms of personal information and intellectual property. Another issue with website scraping could be that it is in breach of a site’s terms of service. This is viewed as a contractual agreement meaning that a user is bound to not breach this contract or may be liable to legal action. Many sites include a non-scraping clause in their terms of service. This is especially present in social media websites term of service such as Facebook.

Other legal protections that are deemed to be breached by malicious scraping are both in the US and are known as the Computer Fraud and Abuse Act, a civil cause of action that can be taken out of it is against someone who accesses a computer system without proper authorization or intends to disrupt or prevent access to a website that may impact other users experience or the functionality of a website in general. Finally, the Data protection act, mainly recognized in the EU as the GDPR which states the use of data mining and harvesting tools in terms of gathering and collecting data from EU residents if there are no legal grounds for it is a prosecutable offence.

1. *Performance evaluation of a web crawler:*

In order to be able to evaluate a crawler effectively, it’s important to understand the limiting factors that affect crawlers in general. As stated previously, crawlers that run in parallel will almost inevitably come across a previously crawled resource. In order for the maximum efficiency to be calculated effectively it’s important that the crawler is able to distinguish between pages that have been crawled previously in order to minimize overlap. Another limiting factor relates to the quality of the web pages provided I.E., the value calculated by the page ranking algorithm is deemed sufficient enough to warrant the crawler performing its function such as downloading or indexing. The most limiting factor for most small scale analysis on the performance of a web crawler is down to the network and bandwidth restrictions on the machine running the crawler or the resource being accessed. Regardless of the efficiency of the crawler performing a particular function, if the bandwidth necessary to maintain a constant stream of web pages is insufficient and there is down time between crawls, the efficiency metric will be affected by this. [32]

Filippo Ricca and Paolo Tonella [33] performed an easy to understand evaluation on a web crawlers’ performance. They describe how the behaviour of a crawler is down to the policies mentioned previously. They state that “A crawler must carefully choose at each step which pages to visit next”, this statement is able to boil down the difference between a good crawler and a bad crawler in terms of behavior. A good crawler is able to traverse the next best web page in terms of usefulness to the topic at hand in the list of URLs found, whereas a worse crawler will simply crawl through in a more linear fashion or won’t be able to distinguish between higher priority web pages more effectively.

Within the same article they perform an evaluation on popular commercial crawlers by assessing their ability to meet certain features that would be indicative of an efficient crawler. These include completeness, the number of pages downloaded over an entire web application (stored in the web server). The latter can be a difficult value to calculate so they stipulate that this value is more of a comparison on the total web pages crawled compared to other crawlers being scrutinised. Another is the robustness, ability to function correctly when faced with both syntactical and scale based challenges. Download limiting, ability to limit the scope of the web application that is wished to be downloaded. And finally, the supported features available such as following the robot exclusion protocol, the user interface, ease of use and multithreading ability. This type of evaluation is very useful when presented with a multitude of variables that are difficult to control as well as the number of objects to be evaluated. The overall evaluation of the 7 crawlers presented is an excellent way of categorising particular crawlers based on the action required of them rather than just the hardware and network resources required to perform said action as well as the time taken to be completed. [33]

A more complex evaluation method was presented by Mohd Shoaib and Ashish K. Maurya of the Shri Ramswaroop Memorial University in India [34]. They expanded on an updated term frequency – Inverse document frequency (TF-IDF) algorithm presented by Shaojie Qiao et al of the School of Information, Science and Technology Southwest Jiaotong University [35]. Shoaib and Maurya incorporated the adapted TF-IDF algorithm into a performance analysis of a web crawlers selection policy by analysing the response time for a crawl that took place across 4 educational institute websites. They analysed the top URLs returned based on the updated similarity score. This similarity score is defined as a content score, a utilisation of the TF-IDF algorithm that is able to accurately reflect the importance of a term within a collection of documents, an adapted form of which is presented by Shaojie Qiao et al which expands on the already existing TF-IDF algorithm to incorporate a similarity measure to accurately assess the similarity of 2 documents, this will be expanded on in future. The structural score, an implementation of the SimRank algorithm that is able to effectively measure the similarity of two objects if they are related to “similar objects” based on link structure analysis [36], the formula for this sim rank is able to be seen in formula 1. And finally, the Combined similarity score that is presented below in Formula 2. K1 and K2 are constants that equal 0.7 and 0.3 respectfully.

Icon

Description automatically generatedA picture containing shape

Description automatically generatedFormula 1, the SimRank algorithm. Supplies a value between 0 and 1 that denotes the similarity between 2 objects. If 2 objects are identical, a = b, the similarity is equal to 1. [36]

A picture containing shape

Description automatically generated



Formula 2, the result of step 5 is the content similarity, and the result of step 6 is the structural score. [34]

They went on to measure the crawling time required to extract the URLs across the educational institutes and save the contents of them. Along with the crawling time they also measured the ordering time needed to convert the URLs crawled into a structured list based on their calculated similarity score. Finally, they measured the precision of the crawl which they define as the ratio of the number of irrelevant web pages that contain no content related to the keyword search, in this case “student”, to the total number of records retrieved during the crawl. The performed the experiment using 2 different crawl limits, one set to 5 and one set to 10, and used the seed URL as the 4 institutes homepage and the keyword “student”. They go on to present a comparison of the top URLs from the updated selection policy algorithm as compared to a PageRank parameter based crawler as well as the crawling and ordering time of the 4 presented websites.

The overall execution and presentation of the study is an excellent foundation to how the performance of a crawler should be presented. Not only should the crawl and ordering time be presented for the 4 different websites, but all could also have a different architecture as well as varying page sizes which will drastically impact results. The comparison of 2 different ranking algorithms supplies an excellent perspective on the actual logic behind a crawler’s behaviour. It’s all well and good for a crawler to be incredible fast in its crawl time a return in a timely manner, but if the content returned is not in the most appropriate order the overall efficiency of an application of a web scraper or the presentation of the data will be compromised in some shape or form.

The algorithm presented by Shaojie Qiao et al is an adapted page ranking algorithm based on the similarity measure which aims to bridge that gap between the link relations concept of crawlers being led along a tree within a web page to the actual contents contained within the web page and how that relates to the target information that the users is seeking. The algorithm is composed of 2 main components, a similarity metric that is able to quantify the likeness between two web pages and an implementation of an adapted PageRank algorithm that apply two distinct weightings to the title and body of a web page. This is similar to formula 2 where Shoaib and Maurya used the same weighting metric except applied it to the content similarity of a term within a collection of documents and its structural similarity principle of two object being similar if they are related to similar objects based on a link collection structure analysis where objects point to and are pointed at by many objects within the same collection.

The first component is what they define as a similarity metric and is combination of an adapted model for TF-IDF to find the likeness of the contents of a webpage. If a user wishes to find term *Ti* within a collection of documents *N*. The algorithm is as follows:

* Text

  Description automatically generated with medium confidenceTerm frequency scheme (TF-scheme). Simply put the number of times a keyword appears within a single document divided by the number of words within that document, Formula 3.

Formula 3, formula for the term frequency scheme

* TF-IDF scheme. The log of the total number of web pages searched and the number of pages term *Ti* appears in, Formula 4.

Formula 4, formula for the TF-IDF scheme.

* Text

  Description automatically generated with low confidenceAn adapted formula proposed by Salton and Buckley [37] that’s able to calculate the TF-IDF of an individual term. N+1 is applied to ensure that log value is always greater than 0 in case term *Ti* appeared in every document, Formula 5.

Formula 5, adaption of Salton Buckley formula .

* Text

  Description automatically generatedFinally for the first component, the finalised content similarity formula that is able to compute the similarity between 2 pages. If the weight of term *Ti* with a singular page X is denoted as:  then content similarity between page x and y is represented in Formula 6.
* Formula 6, content similarity of term *Ti* between 2 pages.

The second component of the Shaojie Qiao et al algorithm as mentioned previously is the adaption of the PageRank algorithm to provide a biased weight towards the title of a page rather than the contents. Identical to the Shoaib and Maurya weighting, a value of 0.7 is applied to the title and 0.3 to the content body of the web page. Shaojie Qiao et al argue that the two different contents have a different impact on the query result returned as well as the fact that title is the leading contributor to a web user interacting with a link as they haven’t had the opportunity to interact or consume the inner content as of yet.

1. Research and Development methodology

Two distinct research methodologies were utilised through this research project, the first being a simulation based method of approach. The attempt is to accurately replicate the implementation of a specific document relational algorithm and incorporate it into an error message and documentation based web crawler to return the most appropriate resource available. This approach is perceived as the most appropriate as the solution that is being explored involves the development of a complex mathematical algorithm that interacts with the relationship between similarity of documents. The other methodology involves the experimental method. This too is appropriate to incorporate into the research performed for this project as the test conditions that are intended to be evaluated involve the editing of the crawler based method in order to provide an accurate comparison of performance. Justification for the research performed is to ensure that the experimental solution designed will accurately reflect the environment and variables that a typical user will emulate when performing a web crawl.

The evaluation criteria for the research performed relied heavily on the source of the publication. The prioritisation on publications that involve the department of information of technology from research institutions or publications that primarily produce software oriented articles led to a higher credibility and validity of data provided. With a large focus of research dedicated to the evaluation criteria itself of a web crawler and distinguishing between a crawler and a scalper, utilising sources that provide a unique integration of existing algorithm to rank documents and their similarity we’re prioritised over publications that repeat the experimental process of the initial source.

Due to the performance evaluation of crawlers being heavily oriented towards the hardware that is executing the software. An ability to evaluate the efficiency of the results produced relying solely on the ranking methods of the documents provided led to a more applicable comparison of performance between crawlers rather than a standard time experiment, although this is still the fundamental way to evaluate an effective experiment but is not the ultimate factor in a definitive answer of efficiency.

1. *Experimental design*

The experimental hypothesis for this project is to design a web crawler that is able to implement the similarity measurement algorithm presented by Shaojie Qiao et al [35] into a web crawler in order to better accurately find solutions to searches within an error message based web crawler application. This experiment will be performed on a personal computer utilising a Ryzen 5 3600 6-core processor running at an estimated 4GHz clock speed combined with 24 GB of 3000MHz DDR4 RAM for memory allocation. This will be performed over a network with an estimated 70 Mbps upload speed and a 20 Mbps upload speed as well as a 6ms latency. The software language intended to be used to facilitate this web crawler will be JavaScript. In order to facilitate the extraction of URLs within webpages, the Cheerio library will enable DOM interaction via a subset of core jQuery. This will be combined with the NodeJS runtime environment and the incorporation of the natural mathematics library to facilities the use of the complex TF-IDF algorithm. The presentation layer for the application will be done using the react framework combined with ruby on rails as the application layer to facilitate navigation and the transfer of state between components. This will all interact with a Java based spring application as the data persistence layer in order to enable the persistence of user based interactions such as search history and favouriting items.

The inputs that are intended for the experiment will be the keyword that the user wishes to find within the webpages crawled from the other input provided, the seed URL. The experiment is restricted to these 2 input variables in order to maximize the ability to accurately compare the performance of the web crawler created against already existing applications. By limiting these variables, the ability to accurately assess the impact of the adapted selection policy algorithm as well as the hardware utilised to perform the search grows with less variables being edited. The intention of this experiment is to first assess the ability of the crawler created within the Node.js environment combined with the selection policy algorithm and then compare the similarity results and time taken to perform by the crawl and ordering of URLs to the experiment performed by Shoaib and Maurya discussed in the literature review. This however differs slightly due to the fact that they utilised a crawling limit of 5 and 10 whereas this experiment will only utilise a crawl limit of 1 to minimize the network impact of the crawl and analyse the time taken more intently.

The experiment will be performed by first prompting the user into entering the keyword and seed URL that they wish to be crawled. Three seed URLs will be selected from popular software based educational and error message sites in order to provide 3 different HTML and DOM environments for the crawler to react to as well as assess the computed similarity values that the crawler calculates in order to determine if the pages retrieved contain data privy to the search term supplied.The crawler will be timed from once the seed and keyword value is entered to the point it is finished downloading the contents of all the URLs discovered. Upon completion, another timer will begin and measure how long it takes for the similarity measurement to rank the pages crawled by usefulness to the keyword supplied.

1. *Development methodology*

The development methodology selected will be Feature Driven Design (FDD). This is due to the fact that FDD is a lightweight agile method that focuses on tangible client oriented software where both functionality and testing are based around measurable features. This is applicable due to both the presentation based application and the crawler experiment being feature based applications that are able to be scaled based on the criteria that the “client” requests. Meaning the milestone based approach is an excellent method, combined with the short iteration process, for evaluating a current products development life cycle and enables effective planning for future features as well as the allocation of time and resources. A negative point to FDD one could argue is that the version rollout depends solely on the feature being developed at any one time. This can also hamper productivity as the focus is not necessarily on the “clients” needs, or in this case the experiment.

1. Design and Implementation:

During development of the web crawler and the presentation based application a couple of key design decisions were implemented during the planning stage. The original requirements were to create an efficient web crawler that is able to traverse error message based forums in order to accurately find results pertaining to a user’s input. A large time of the software development process is spent on debugging [2]; therefore, the idea was to create an application that is able to traverse a multitude of error based sites such as StackOverflow and Redhat in order to help speed up this debugging process. In order to effectively evaluate the returned results from a crawl, it was intended to utilise the same page ranking system, PageRank, that Google implemented. However, once the research portion of this experiment begin, it became clear that the assessment of the return results being based off of link analysis alone would not be sufficient. The contents of the webpages could be argued to be vastly more important rather than the popularity of a page in order to effectively find solutions to complex error messages that exist across many languages and frameworks. It was decided then to utilise both a similarity algorithm with a lower weighting than page rank as well as a content algorithm that effectively weights the interior of a document more greatly. The same as well could be argued for the design decision not to implement the identical algorithm proposed by Shaojie Qiao et al due to the weighting they supplied to the title rather than the contents. This is why an algorithm similar to Shoaib and Maurya that values similarity of contents rather than title to be more applicable as the selection policy for the crawler designed.

In order to implement the TF-IDF and content similarity algorithm, shown below in Formula 7 and 8. The decision was made to adapt the mathematical library, Natural. Many versions and implementation of the TF-IDF algorithm exist depending on the application or intent of use. By using an already existing library and editing the calculation functions to reflect the required output, time is able to be better spent developing the UI based application to enable more features for the goal of decreasing development time. The similarity algorithm was then able to be programmed by handed simply as natural is performing the heavy lifting for the TF-IDF values

 Diagram, text, letter

Description automatically generated formula 7 & 8, content frequency measurement and similarity formula used for developed crawler.

Another design decision during the development process was to no longer utilise a crawl method for StackOverflow. StackOverflow is widely considered the most popular forum for answering and asking error message based problems. Due to this popularity, there are estimated to be millions of error message based questions that exist on the site. This would provide an incredibly large problem if the user wishes to find an incredibly vague error message and supply no further detail. The crawler would begin on the search page as the seed URL and would inevitably face an error or deadlink during a crawl. It would take several minutes before the web scraper was able to extract any information from the site. The decision was made then to utilise the already existing API StackExchange when interacting with the StackOverflow site. This API is incredible dynamic in terms of searchability but was chosen to not be include in the performance evaluation as for one, it is no longer classed as a crawler and instead is directly interacting with the resource and converted into appropriate JSON content. Two, the API available is limiting on number of connections available in a 24hr period as well as the number of results returned. There would also be no sorting algorithm applied as StackExchange provides its own internal post evaluation system in terms of likes.

Finally, the search application had originally intended to include an internal voting system similar to StackExchange but would be contained within the spring layer application. This was to provide results obtained by previous searches that were deemed too closely reflect and possibly solve the issue being searched by a future user. This design decision was eventually scrapped mid-way through the design period as the implementation of an algorithm that’s also able to assess previously entered terms combined with the webpages “liked” during the search to be compared with similar terms entered in future was a very time consuming and complex task that didn’t impact the overall object of providing a service that is able to simply enable the function of speeding up the development process.

1. Results:
2. *Top ranked URL based on similarity measure*

*Case 1:*

*Text, application

Description automatically generated*

Table 1, Top ranked URL using content score from seed url: https://www.w3schools.com/java

Table 1 above shows the content score calculated along with the number of terms found against the number of words present in the document. This crawl was performed on the seed URL of <https://www.w3schools.com/java> with a crawl limit of 1 with the keyword of Classes. The most unique URL returned was: <https://www.w3schools.com/w3css/w3css_references.asp> who contained 3360 words with 30 being the keyword term “Classes”.

*Case 2:*

Graphical user interface, application

Description automatically generated

Table 2, Top ranked URL using content score from seed url: https://www.geeksforgeeks.org/java/

Table 2 above shows the content score calculated along with the number of terms found against the number of words present in the document. This crawl was performed on the seed URL of <https://www.geeksforgeeks.org/java/> with a crawl limit of 1 with the keyword of Classes. The most unique URL returned was <https://www.geeksforgeeks.org/packages-in-java/> who contained 3219 words with 47 being the keyword term “Classes”.

*Case 3:*

Graphical user interface, application, table, Excel

Description automatically generated

Table 3, Top ranked URL using content score from seed url: https://docs.oracle.com/en/java/

Table 3 above shows the content score calculated along with the number of terms found against the number of words present in the document. This crawl was performed on the seed URL of <https://docs.oracle.com/en/java/> with a crawl limit of 1 with the keyword of Classes. The most unique URL returned was <https://docs.oracle.com/en/cloud/saas/netsuite/ns-online-help/preface_1531238762.html> who contained 373 words with 2 being the keyword term “Classes”.

1. Crawler Evaluation

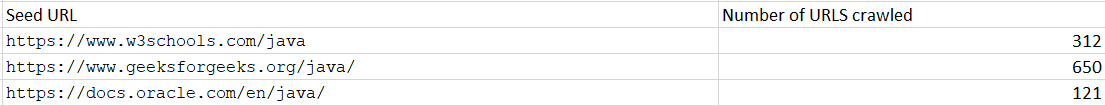


Table 4, number of URLs found from each seed URL.

Table 4 shows the number of URLs that were found during each crawl with the associated seed URL on the left. From figure 6 below, it can be observed that the site that contained the most URLs was GeekForGeek with 650 and the least was Oracle with 121.

Chart, bar chart

Description automatically generated

Figure 6, Graphically representation of table 4, number of URLs crawled for each starting seed URL.

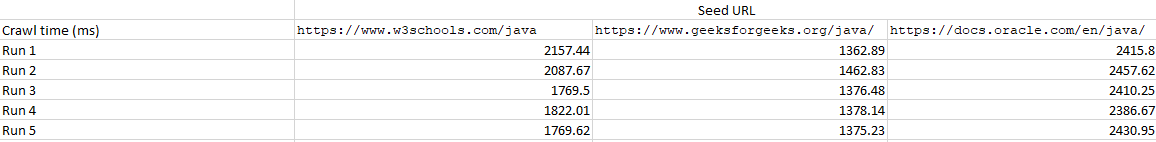


Table 5, Crawl time calculated during each run through for each of the 3 seed URLs.

Table 5 shows the time that it took for the crawler developed to complete the extraction and downloading of contents of all webpages that connect to the seed URLs in milliseconds. The shortest runtime being GeekForGeeks first run with a value of 1.362 seconds and the longest being the second run for Oracle with a value of 2.457 seconds.

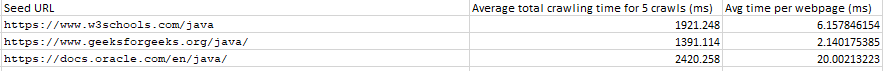


Table 6, the average crawl time taken of the 5 run throughs that were executed for the crawler developed using the 3 distinct seed URLs. Also shown is the average time taken to crawl an individual web page for the 3 seed URLs.

Table 6, shown above, represents the average time taken for the crawler to complete the extraction of the contents from all the URLs found during the crawl. Also shown is the average time taken for each web page to be crawled. This value is calculated using the number of URLs crawled from table 4, and the total time crawled during the 5 run throughs from table 5. The shortest average crawl time belongs to GeekForGeek who also has the shortest average time spent per web page with the values being 1.391 seconds and 2.14 ms respectfully for 650 webpages. The longest value for both measurements belong to Oracle who had an average crawl time of 2.42 seconds and 20.002ms per web page with a total number of pages crawled of 121.

Chart, line chart

Description automatically generated

Figure 7, the average time taken for 5 crawls for each of the Seed URLs within table 6.

Chart, line chart

Description automatically generated

Figure 8, the average time taken to crawl per web page for the 3 seed URLs, taken from table 6.

Graphical user interface, application

Description automatically generated

Table 7, sorting time calculated during the execution of the content similarity algorithm.

Table 7 shows the time taken for the downloaded contents of the web crawl for each of the 3 seed URLs to be evaluated during the 5 runs using the 2 contents measurement algorithm shown in Formula 7 and 8. The shortest sorting time being Oracle final run with a value of 67.11 ms the longest being the final run for GeekForGeeks with a value of 667.91 ms.

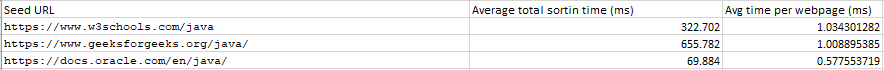
Table 8, the average sorting time taken of the 5 run throughs that were calculated utilising the content measurement algorithm. Also shown is the average time taken to sort an individual web page for the 3 seed URLs.

Table 8, shown above, represents the average time taken for the sorting algorithm to complete for all the contents of the URLs found during the crawl. Also shown is the average time taken for each web page to be sorted. This value is calculated using the number of URLs crawled from table 4, and the total time to sort during the 5 run throughs from table 7. The shortest average sort time belongs to Oracle who also has the shortest average sorting time spent per web page with the values being 69.884 ms and 0.578 ms respectfully for 121 webpages. The longest value for average sorting time belongs to GeekForGeek who had an average sort time of 655.782 with a total number of pages crawled of 650 and the longest average sort time per web page being We3Schools who took an average of 1.03 ms for 312 web pages.

Chart, line chart

Description automatically generated

Figure 9, the average time taken to sort the crawled webpages using the Seed URLs within table 8.

Chart, line chart

Description automatically generated

Figure 10, the average time taken to sort per web page for the 3 seed URLs, taken from table 8

1. Discussion:

During the completion of the experiment, a few factors were altered from the original requirements proposed during the planning period. The original plan was the run the crawler based evaluation on 3 popular error solving based web pages, We3School, GeekForGeek and Redhat.

During the early development phase of the crawler, it was discovered that many factors involving Redhat would not be compatible with performing a crawler evaluation with the site as one of its targets. One of these issues involve the sites architecture. Redhat is predominantly a subscriber based website meaning that a large amount of content is unable to be accessed and is restricted from certain users. This would mean adapting a crawler to only access material that is labelled as free to read and is not locked behind a paywall. This is a possible function to achieve but in terms of being a site to enable a crawler evaluation it is not very appropriate. A large percentage of time would be spent crawling web pages for it only to be discovered that the contents are not accessible, these webpages would need to then be rejected before performing any sort of content evaluation algorithm as the material would not be useful to the user attempting to solve their issue. Another issue is that the architecture of the web pages is split into section such as articles, solutions, documentation, errata’s etc. This would mean a further filtering method to reject web page classes that do not provide a possible solution to a user’s keyword query.

When presenting the results for the crawl performed on Redhat, a lot of context would need to be supplied to the values achieved such as the number of rejected URLs in order to offer a fair comparison on the performance of the crawl performed. Due to this and the factors mentioned previously it was decided that Redhat should be replaced by a site that offers both a compatible architecture in terms of DOM manipulation and solution based webpages to user specific queries. It was decided that a documentation specific site would be selected so that the 3 URLs are able to cover forums, documentation and code examples in the experiment. The documentation site selected was Oracle as they offer a complete index of the Java based language and would be an apt comparison within the experiment to the We3Schools and GeekForGeek respected java sections as seed URLs to offer a complete coverage in terms of finding an appropriate webpage to the keywords entered.

Its important to put into context the similarity measurement value presented in table 1-3. The higher the similarity value, the more similar 2 documents are in terms of their contents in relation to the keyword searched. It’s calculated by summing the similarity score for all webpages compared to an individual one. This means that for web page 1, it is compared to all webpages found during the crawl, 312 in the case of w3schools, and the value calculate during each comparison is added together giving a final similarity value compared to all documents. Therefore, the webpage that possess the lowest similarity score is both the most unique to the key term searched and frequently contains the most instances of this keyword. This relationship is true for all top 10 documents returned during the w3Schools crawl but not for the GeekForGeeks.

This property of the similarity metric is the ideal term to quantify if a web page is closely related to the solution needed for a user as it offers both the most unique document, usually the key factor in determining a successful solution to error message based problems, and containing the most terms of a keyword search which is usually indicative of being an appropriate resource.

As mentioned previously, the relationship of descending similarity values being identical to the descending nature of the number of terms present in the webpage would lead to the conclusion that GeekForGeeks web architecture is an ideal resource to perform this crawler performance evaluation as well as the content similarity algorithm being ideal metric in order to evaluate solutions to single worded key terms that relate to educational resources within the we3Schools hierarchy. A flaw however, is that either the architecture of W3Schools or the algorithm used returns identical similarity values for pages with an equal number of keywords found. This leads to the assumption that the contents crawled are either incredible similar to the point of identicalness or that the algorithm is not able to distinguish nuance differences. This relationship is consistent across the top 10 values returned from the We3Schools crawl.

The opposite of this fact is represented in table 2, where two documents contain 37 terms of the classes keyword yet offer different similarity values. This would lead to the conclusion that the contents of GeekForGeek webpages are appropriate resources for the content similarity metric and lead to more accurate distinctions in terms of content to We3Schools.

Table 3 would represent the outlier within this experiment. Across the 121 webpages accessed, only one contained the keyword “classes”. This is a difficult to comprehend statement as the assumption would be made that the documentation for an object-oriented programming language such as Java would have more than one mention of a class based structure. With that in mind, the conclusion of the experiment involving the seed URL: <https://docs.oracle.com/en/java/> is that the crawler’s mechanism is either not capable of extracting the correct links to access the required documentation or that the architecture of the seed URL itself is unable to access a majority of the resources available within the Oracle domain. It is also not possible to be able to effectively draw a conclusion on the appropriateness of the content measurement as there are not enough webpage to provide an acceptable comparison.

Table 5 and 6 offer an insight into evaluating the other performance metric, crawl time. The initial expectation was that the crawl time would represent a somewhat linear relationship to the number of pages crawled to the time taken for completion, with some deviations due to network and hardware issues. However, this was not the case as the site with the most webpages, shown in table 4 as GeekForGeek, compared to the one with the least, oracle, led to GeekForGeek vastly outperforming Oracle in terms of time taken to crawl. This relationship is best represented in the average time taken per web page in table 6, showing the time take for a GeekForGeek crawl to be faster by a factor of 10 than that of an Oracle web crawl for individual web pages. This could be down to multiple factors but the most likely of which is the network response time of the resource being accessed by the crawler. It is likely that the server that the crawler requests for the GeekForGeek crawl is able to respond much faster than that of the Oracle one. Another factor could be down to the network traffic of the personal PC used at the time of the crawl. However, with the results being repeated 5 times over the course of several minutes it is unlikely that any significant network throttling would occur to cause that much of a discrepancy. The same conclusion can be drawn for the comparison between GeekForGeek and We3Schools, the response time of the resource is much faster, especially since the average document size shown in table 1 and 2 is much greater on average for the case of GeekForGeek in terms of the top 10 responses.

Other factors that may impact the crawl time performed for each individual seed URL is the response time of the resource being allocated as mentioned previously. Another is the current hardware usage of the computer executing the get query. Its possible that other process may cause small deviations in results. The size of the web page being accessed will have a direct impact on crawl time as more data will need to be downloaded onto the local machine. However, since the average crawl time presented in table 6 for GeekForGeek was significantly lower than We3Schools, it would lead to the conclusion that network response time led to the most impacting factor.

Tables 7 and 8 offer a perspective on the sorting times performed on the final downloaded resources from the web crawl. The expectation was similar to the expectation for the crawl time except possess slightly less deviation as there is no competition for network resource rather just hardware. Therefore, a closer result was expected for the average time to sort individual web pages. This linear relationship appears to be true for both the sorting of GeekForGeek and We3School resources as the crawl time achieved led to a very low deviation. The same however can not be said for the sorting time of the Oracle resources. This makes sense however due to the fact that only one webpage possessed any sort of similarity value that offered substance. This would mean that much less processing time is being spent on comparing and moving values within an array as 99% are all equal. An assumption can be made then due to the fact that the average sorting time of the Oracle data values was 43.5% faster than that of the combined average of GeekForGeek and We3Schools, almost 50% of the time allocated to perform a calculation of the similarity metric is spent sorting the completed array and the other 50% being spent on the actual computational aspect.

During the experiment there was a key issue that arose while crawling the GeekForGeek seed URL. For both the timing aspect of the other 2 sources, they were able to be performed back to back with no delay in between experiments. However, for GeekForGeek it was required to leave a 1-2 minute gap in-between execution of the crawler. This didn’t cause any issues for the results gathering phase other than some slight inconvenience but raise a large point about a fundamental issue with the crawler being utilised in the application. If a user wised to access multiple Geek resources within a short time period, they would be met with several hundred 403 forbidden messages. A likely assumption for this is that the server is not allowing multiple requests from the same source to be executed with a short time frame, solving this problem however is a much more difficult question. Without proper diagnostic measure on what aspect of the source accessing the resource is being rejected such as number of requests per minute, the IP or the headers contained within the requests. It would be difficult to find a complete solution to navigate around this issue.

When comparing the results achieved in this experiment to the ones achieved in the study performed by Shoaib and Maurya, its important to convert the results in order to provide more context. For content similarity score, it’s possible to make a direct comparison. As mentioned previously, Shoaib and Maurya utilised both a content similarity metric as well as a SimRank based link analysis. Due to the content similarity algorithm being implemented identically, we’re able to observe that the average similarity score for the top 5 URLs returned within their figure 3 is 0.75 is vastly different to our average of top 5 URLs in figure 2 which ranges from 0.13 – 0.22. This relationship is consistent however for web pages that only contain one or two instances of the key word. GeekForGeek pages that contained only 1 instance of the keyword “classes” had an average value of 0.7, almost identical to the average value for the ones calculated within Shoaib and Maurya’s figure 4.

Due to the fact that the study performed by Shoaib and Maurya was done using a crawl limit higher than 1, some assumptions will need to be made about our values. If a loose assumption is made that if our crawler repeated the experiment with a crawl limit of 5, the values can be linearly converted by dividing by the difference. For example, Shoaib and Maurya achieved an average ordering time of the 4 seed URLs with a crawl limit of 5 of 50ms, so if we divide our value achieved for the average sorting time for We3Schools by 5, we get 64.4ms. This value is approximately 29% higher but the sort is being applied to nearly 50 times more web pages. This difference is able to be explained by either the fact the hardware Shoaib and Maurya performed their experiment on would be more outdated as it was performed on hardware at best up to date with 2014 specifications. The other issue could be that the ordering was performed during the crawl whereas for the crawler developed in this report the ordering was performed after the calculation of similarity score. This may have led to some issues with asynchronous sorting thus limiting the performance possible to perform the function leading to higher ordering times.

If we apply the same logic as ordering times to crawl times we are able to see that the average crawl time for the seed URL: <https://www.srmu.ac.in/> within Shoaib and Maurya’s table 2, the time achieved for a single crawler to access and download the contents would be an estimated 28.35 seconds. This is a drastically different value even to the highest crawl time calculated being Oracle that indexed over 100 pages whereas Shoaib and Maurya’s crawler only indexed 5. A reason for this gulf could be down to the fact that the resource they are accessing is an educational institutes website. The response time of the resource being accessed may be drastically reduced as the architecture and scalability of a web domain such as GeekForGeek will be able to handle multiple requests to the same end point much more competently than an educational institute that may not be equipped to handle numerous requests in a short period of time and thus may be throttling the network capability.

1. Conclusion:

Within this research topic a web crawler was developed to specifically download and crawl the contents of error message forum, documentation and educational resource websites. A performance evaluation was then performed to analyse the content similarity of all the web pages in order to find the most unique solution which in theory, would provide the most appropriate solution for single keyword term queries. The metrics measured include the crawl time for each seed URL as well as the sorting time for the content measurement algorithm plus the most unique URLs found during each of the unique crawls. 3 similar web sites in terms of contents were tested, a predominantly documentation based resource, combination of documentation and learning based resource and a solely learning based resource but offer vastly different DOM architectures to test the crawler’s information extracting ability. A finalised version of the web crawler was presented along with a potential application that offers a user interface in order to best interact and consume the information extracted and scraped during the crawl.

Future work could include the development of specific crawlers for individual resources in order to both maximum crawl time efficiency and extract or scrape key information based on the structure of the resource being accessed such as, documentation sites, error forums and educational webpages. Other future works could include a more in-depth comparison of selection policy based algorithms that effectively measure both the interior contents, title and URL structure with a combination of a link structure analysis to find the most popular and content specific web pages for specific error based solutions.

1. Reflection:

Upon reflection, if the process of evaluating a web crawler were to be repeated I would make a few alterations. I would want to replicate the study but evaluate the performance for different crawl limits. This would allow an apt comparison of more commercial crawlers but would require some form of looping implementation to ensure that any endpoints that return a 403 forbidden would be repeated until an appropriate content is responded. Another edition would be directly comparing different content algorithms to find the most suitable for each resources accessed. Another variable that I would like to be measured if the experiment was repeated is to apply a weighting value of both the URLs content itself and the title of the resources being accessed. This would help supply more context to the content measurement allowing a user wishing to find solution or learning resources to a problem better accurately assess the appropriateness of a webpage. Finally, creating an evaluation experiment by applying any combination of the changes listed above with the robot protocols selection policy would provide a more accurate comparison to modern day crawlers being utilised across the web.

The biggest aspect that I’ve learned during this research project is that there is no one solution for a page ranking algorithm. Many factors can lead to an algorithm being the most appropriate chose for specific scenarios from the scale of architecture for a web domain to the specific functionality required for the crawler. For example, a crawler whose function is to find single word sources would be able to accurately rely on title and URL contents to find an appropriate solution but more complex terms such as long error messages or incredible specific learning resources would require algorithms that more accurately analyse the contents of a web page and show little regard for the title.

Another important learning aspect I learned is that certain resources have incredibly strict preventative measures to restrict that use of web crawlers. I found this out during the testing of the crawler through the GeekForGeek domain that accessing the same end point multiple times will be restricted usually after the second time accessing. This situation is common across many popular end points and would lead to a difficult scenario for the situation to be solved.

1. References
2. The Business Research Company. Information Technology Global Market Report 2021: COVID 19 Impact and Recovery to 2030. 2021.
3. T. Britton, L. Jeng, G. Carver, P. Cheak, T. Katzenellenbogen. Reversible debugging software. Judge Bus. School, Univ. Cambridge, Cambridge, UK, Tech. Rep. 2013.
4. M. Bianchini, M. Gori, F. Scarselli. Inside pagerank. ACM Transactions on Internet Technology. 2005, 5(1), p. 92-128.
5. A. K. Chandranna. An Online Version Of Hyperlink-Induced Topic Search (HITS) Algorithm.  San José State University. 2010.
6. M. Thelwal. A web crawler design for data mining. Journal of Information Science. 2001, 27(5), p.319-325.
7. D. K. Sharma, A. K. Sharma. A comparative analysis of web page ranking algorithms. International Journal on Computer Science and Engineering. 2010, 2(08), p.2670-2676.
8. V. Rajapriya. A Literature Survey on Web Crawlers. 2014, vol 2, p. 36-44.
9. K. R. Fall, W. R. Stevens. TCP/IP illustrated, volume 1: The protocols. Addison-Wesley. 2011
10. D. R. Nath, K. Chopra. Web Crawlers: Taxonomy, Issues & Challenges. International Journal of Advanced Research in Computer Science and Software Engineering. 2013, 3(4), p. 944-948.
11. U. Gasser. Regulating search engines: Taking stock and looking ahead. Yale JL & Tech. 2005, vol 8, p.203-205
12. B. Gerson. How Google and its rivals rewrote the rules of business and transformed our culture. 2005
13. Google. How Google Search organises information. [Online] Available from: <https://www.google.com/search/howsearchworks/how-search-works/organizing-information/> [Accessed 29 Nov 2021]
14. Google. Beautiful Soup Documentation. [Online] Available from: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> [Accessed 01 Dec 2021]
15. E. Van Couvering. The history of the Internet search engine: Navigational media and the traffic commodity. Springer, Berlin, Heidelberg. 2008, p.177-206
16. D. Sullivan. How search engines work. [Online] Available from: <https://didattica-2000.archived.uniroma2.it//prog_web/deposito/search_engine.pdf> [Accessed 02 Dec 2021]
17. S. Rana, N. Singhal. Design of Human Powered Directories using Mobile Agents. International Journal of Scientific Research Engineering & Technology. 2013, 1(11), p. 16-21.
18. DBpedia. About ApexKB. [Online] Available from: <https://dbpedia.org/page/ApexKB> [Accessed 02 Dec 2021]
19. Google. Googlebot. [Online] Available from: <https://developers.google.com/search/docs/advanced/crawling/googlebot> [Accessed 03 Dec 2021]
20. Google. Cloud Bigtable. [Online] Available from: <https://cloud.google.com/bigtable> [Accessed 03 Dec 2021]
21. L. Page, S. Brin, R. Motwani, T. Winograd. The PageRank citation ranking: Bringing order to the web. Stanford InfoLab. 1999
22. GeekForGeek. Page Rank Algorithm and Implementation. [Online] Available from: <https://www.geeksforgeeks.org/page-rank-algorithm-implementation/> [Accessed 03 Dec 2021]
23. S. S. Dhenakaran, K. T. Sambanthan. Web crawler-an overview. International Journal of Computer Science and Communication 2011, 2(1), 265-267.
24. H. Bullot, S. K. Gupta, M. K. Mohania. A data-mining approach for optimizing performance of an incremental crawler. IEEE/WIC International Conference on Web Intelligence. 2003, p. 610-615
25. V. Sharma, M. Kumar, V. Renu. A Hybrid Revisit Policy For Web Search. Journal of advances in information technology. 2012, vol 3, p. 36-47
26. M. Koster. Robots in the Web: threat or treat?. OII Spectrum. 1995, 2(9), p.8-18.
27. Y. Sun, Z. Zhuang, C. L. Giles. A large-scale study of robots. txt. 16th international conference on World Wide Web. 2007, p. 1123-1124.
28. IETF. International Engineering Task Force. [Online] Available from: <https://www.ietf.org/> [Accessed 15 Dec 2021]
29. S. Wan, Y. Li, K. Sun. PathMarker: protecting web contents against inside crawlers. Cybersecurity. 2019, 2(1), p.1-17.
30. P. Boldi, B. Codenotti, M. Santini, S. Vigna. Ubicrawler: A scalable fully distributed web crawler. Software: Practice and Experience. 2004, 34(8), p.711-726.
31. Google. What Crawl Budget Means for Googlebot. [Online] Available from: <https://developers.google.com/search/blog/2017/01/what-crawl-budget-means-for-googlebot> [Accessed 19 Dec 2021]
32. D. S. Sirisuriya. A comparative study on web scraping. Department of Computer Science, Faculty of Computing, General Sir John Kotelawala Defence University, Ratmalana, Sri Lanka. 2015
33. M. Q. Rafiq. Performance evaluation of web crawler. International conference on Emerging trends in technology. 2011, p. 25-26
34. C. Girardi, F. Ricca, P. Tonella. Web crawlers compared. International Journal of Web Information Systems. 2006
35. M. Shoaib, A. K. Maurya. URL ordering based performance evaluation of Web crawler. International Conference on Advances in Engineering & Technology Research. 2014, p. 1-7
36. S. Qiao, T. Li, H. Li, Y. Zhu, J. Peng, J. Qiu. SimRank: A Page Rank approach based on similarity measure. IEEE International Conference on Intelligent Systems and Knowledge Engineering. 2010, pp. 390-395
37. G. Jeh, J. Widom. Simrank: a measure of structural-context similarity. 8th ACM SIGKDD international conference on Knowledge discovery and data mining. 2002, p. 538-543
38. G. SaJton, C. Buckley. Term-weighting Approaches in Automatic Retrieval. Information Processing & Management. 1988 24(5), p. 513-523.
39. R. Jain, D. G. Purohit . Page ranking algorithms for web mining. International journal of computer applications. 2011, 13(5), p. 22-25.
40. F. McCown, M. L. Nelson. Evaluation of crawling policies for a web-repository crawler. 17th conference on Hypertext and hypermedia. 2006, p. 157-168
41. P. Jha, S. Goyal, T. Kumari, N. Gupta. Robots Exclusion Protocol. Internation journal of emerging science and engineering. 2014 2(5).