



School of Information Technology and  
Engineering at the  
ADA University



School of Engineering and Applied  
Science at the  
George Washington University

**GRAPH-BASED  
VISUALIZATION & ANALYSIS OF AZERBAIJANI WEB**

A Thesis

Presented to the Graduate Program of Computer Science and Data Analytics  
of the School of Information Technology and Engineering  
ADA University

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science in Computer Science and Data Analytics  
ADA University

By  
Aydin Bagiyev

**April 2022**

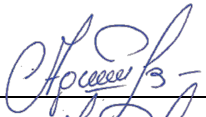
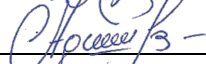
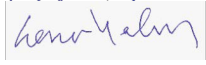
## THESIS ACCEPTANCE

This Thesis by: Aydin Bagiyev

Entitled: *Big Graph: Visualization & Analysis of Azerbaijani Web*

has been approved as meeting the requirement for the Degree of Master of Science in Computer Science and Data Analytics of the School of Information Technology and Engineering, ADA University.

Approved:

Dr. Abzatdin Adamov (Adviser)		28.04.2022 (Date)
Dr. Abzatdin Adamov (Program Director)		28.04.2022 (Date)
Dr. Sencer Yeralan (Dean)		28.04.2022 (Date)

## ABSTRACT

A full understanding of the local Azerbaijani web space is necessary to analyze information flow patterns and influences in the local network and review the dependency of Azerbaijan on external sources in case of cyber-attacks or national emergencies. To develop this knowledge and to create efficiency in local data collection processes, a web crawler with a subsequent graphical analysis is a must. The goal of this research is to create a big graph of Azerbaijani web, analyze its linkages and most influential nodes. This study aims to develop a catalog of local websites, create a web crawler to browse each web page and outgoing links, construct a graph-based visualization with valuable information and apply a ranking algorithm to measure the influence scores. A multiprocessing program in Golang is developed to crawl the database of local webpages supplied by the Ministry of Communication & Information Technologies. The program consists of a master, multiple concurrent workers, and a Postgres database. The constructed graph consists of nodes representing web pages, and edges which are connections in-between. A page ranking algorithm is implemented to measure the importance of nodes. The observations are such that the graph is not too strongly connected, and governmental web pages are the most linked ones due to redirections to various services.

**Keywords:** *web crawling, graph theory, big data, page ranking, multiprocessing*

## TABLE OF CONTENTS

<b><u>LIST OF FIGURES</u></b>	<b>6</b>
<b><u>LIST OF TABLES</u></b>	<b>7</b>
<b><u>LIST OF ABBREVIATIONS</u></b>	<b>8</b>
<b><u>1 INTRODUCTION</u></b>	<b>9</b>
1.1 PROBLEM DEFINITION	10
1.2 OBJECTIVE OF THE STUDY	11
1.3 SIGNIFICANCE OF THE PROBLEM	12
1.4 REVIEW OF SIGNIFICANT RESEARCH	13
1.4.1 WEB CRAWLING REASONS & PROPERTIES	13
1.4.2 WEB CRAWLER ARCHITECTURE & RELATED GRAPH THEORY	13
1.4.3 TYPES OF WEB CRAWLERS	14
1.4.4 DATA STORAGE	17
1.4.5 CRAWLING SEARCH	17
1.4.6 SMART QUERYING	18
1.4.7 SINGLE- OR MULTI-THREADED WEB CRAWLING	20
1.4.8 GRAPH RANKING	21
1.5 ASSUMPTIONS & LIMITATIONS	23
<b><u>2 METHODOLOGY</u></b>	<b>25</b>
2.1 DATA COLLECTION	25
2.2 PROGRAM WORKFLOW	26
2.3 DATABASE DESIGN & IMPLEMENTATION	27
2.4 WEB CRAWLER ARCHITECTURE & IMPLEMENTATION	31
2.5 GRAPH CONSTRUCTION	39
2.6 PAGE RANKING	41
2.7 METADATA COLLECTION	41
<b><u>3 RESULTS &amp; ANALYSIS</u></b>	<b>43</b>

<b>4</b>	<b><u>CONCLUSION &amp; FUTURE WORK</u></b>	<b>48</b>
<b>5</b>	<b><u>BIBLIOGRAPHY</u></b>	<b>50</b>

## LIST OF FIGURES

FIGURE 1. BASIC WEB CRAWLER ARCHITECTURE	15
FIGURE 2. PARALLEL CRAWLER ARCHITECTURE	17
FIGURE 3. GRAPH SEARCH STRATEGIES	18
FIGURE 4. THE ARCHITECTURE OF THE INFORMATION MANIFOLD SYSTEM [21]	20
FIGURE 5. GENERAL PROGRAM WORKFLOW	26
FIGURE 6. DATABASE ENTITY RELATIONSHIP DIAGRAM	30
FIGURE 7. MASTER'S WORKFLOW	33
FIGURE 8. WEB CRAWLER WORKER'S WORKFLOW	35
FIGURE 9. SAMPLE OF COLLECTED DATA	43
FIGURE 10. BIRD-EYE VIEW OF AZERBAIJANI WEB GRAPH	45
FIGURE 11. SAMPLE OF HIGHER-WEIGHTED NODES	45
FIGURE 12. SAMPLE AZERBAIJANI WEB GRAPH DEEP-DIVE VIEW	46

## LIST OF TABLES

TABLE 1. REGULAR EXPRESSION PATTERNS FOR DOMAIN PURIFICATION	25
TABLE 2. LARGEST NODES IN WEB GRAPH	44
TABLE 3. LARGEST EDGE WEIGHTS IN WEB GRAPH	44
TABLE 4. TOP-5 PAGE RANKING RESULTS	47
TABLE 5. TOP LANGUAGES	47

## LIST OF ABBREVIATIONS

Abbreviation	Explanation
API	Application Programming Interface
DoS	Denial of Service
HITS	Hyperlink-Induced Topic Search
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICT	Information Communication Technologies
IM	Information Manifold
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
PHP	Hypertext Processor
RPC	Remote Procedure Call
SCC	Strongly Connected Component
SQL	Structured Query Language
TLS	Transport Layer Security
URL	Uniform Resource Locator



## 1 INTRODUCTION

As a vast and unstructured repository, the web can deliver massive amounts of data dynamically. The web is constantly growing at an almost exponential rate with an estimated 2 billion indexed pages scattered across the globe on hundreds of servers [1]. These pages, i.e., websites, are the containers of data stored on servers and accessible through the Internet. Various online resources are being used by people daily for buying products and services, watching video content, and sharing news. Businesses utilize these platforms to market their offerings and target potential customers. Governments reap the benefits of sharing information with citizens in real-time, quickly handling any national emergencies and providing easy access to numerous services, such as administrative, healthcare and education.

Developing countries around the world are gradually catching up with this trend and nourishing modern technological cultures within their borders. Azerbaijan is keeping up with recent novelties and is becoming more dependent on technology day by day. Azerbaijan's ICT industry has been expanding at a rate of 25%-30% since 2005 [2]. Telecommunications liberalization, upgrading and extension of the national telecom infrastructure, and deployment of e-government have all contributed to this growth. Around \$2.5 billion was invested in the national IT sector with the government contributing 28% and local enterprises and foreign investors contributing 72% [3]. According to the World Bank, approximately 80% of the population has been using the internet as of 2019 statistics [4]. The most recent observed year-on-year growth of digital users constituted 2,5%, with a dramatic 16,2% increase in active social media users [5]. This is also propped by statistics showing the highest usage of YouTube, Facebook, and Instagram platforms. Most of the web traffic is coming from mobile devices (circa 60%). A progressive extension of mobile and fixed broadband networks and reduced prices helped steer the increase in internet usage by spreading across more locations at affordable costs.

However, technological development does not come without its own set of issues. Despite its listed pros, the Internet can also be employed for harmful purposes such as:

- targeting national security systems (cyber-attacks),
- promoting hate speech,
- online crimes (credit card frauds),
- hacking into confidential and potentially secret data sources.

As of 2019, according to the Geneva Center for Security Sector Governance [3], 42% of computer users in Azerbaijan were exposed to cyber dangers at the risk of malware spreading to their memory cards or hard drives. There were also several attacks targeted at the country's energy sector and governmental organizations, especially in the Second Karabakh War period. Diplomatic passports of several governmental officials were hacked by unknown agents in recent years. At some point, the government was even considering developing its own search engine to increase information security - however, no advances in this area were reported since [6].

Hence comes the necessity of controlling the local online space and fully understanding the structure of the existing web traffic along with its limitations. For this purpose, a successful local web crawler is a must. This program aims to download and index content from the local web by

fetching website URLs (Unique Resource Locators) in an organized way, crawling them for content and finding references to connected websites to add to their queue.

This project's main goal is to develop an interactive graph of Azerbaijani web and analyze the most influential local websites by applying web crawling and ranking algorithms. The purpose of this delivery is to analyze patterns in data linkages and the strength of the local online space. The paper is divided into four main modules: introduction, methodology, results and analysis, summary, and future work. The introductory chapter consists of the following subsections: problem definition, objectives of the study, significance of the problem, review of significant research, assumptions, and limitations.

### **1.1 Problem Definition**

Currently, Azerbaijan is missing a well-established local internet web crawler capable of reviewing and categorizing web pages. Its absence leads to concerns explained below.

1. National businesses and governments do not effectively utilize the huge amount of data that can be collected from local websites for such purposes as lead generation, marketing clustering, public sentiment, and policy analyses. The lack of a web crawler to gather all this information means missed opportunities to adjust prices based on competitive insights, create market-relevant products, and target the appropriate audience.

2. Academic institutions often do not pull data collectively from all connected sources for further research such as for economic studies, educational level analyses and others. Effective retrieval of information in this sector is of particular importance to analyze the existing research papers, network analyses and population clusters.

3. It is hard to analyze country-level dependency on external sources and evaluate multiple scenarios in case of national emergencies such as war or natural disasters. The lack of solutions for the listed issues can lead to such problems as increased exposure to cyber threats and an inability to circumvent them. As an instance of a very high-ranked cyber-attack, the 2012 attack on the websites of the Ministries of Internal Affairs, Communication and High Technologies serves as an example. According to studies at the time, 24 of the 25 IPs (Internet Protocols) utilized in the attack came from Iran.

4. There is no country-level understanding of the local internet structure, its most influential nodes, and linkages to external sources. This further leads to missed opportunities in applying the most recent advances in machine learning to categorize websites by ownership, topic content, sentiment classes, hosting service provider locations, amongst others.

Hence, the designed solution framework and reported results must cater to all these varying needs and be flexible enough to allow users to view the data they need. In particular, the users must be able to:

- a. interactively observe and analyze the most influential websites and their in-/out-connections;
- b. see relevant pre-calculated metrics for websites, such as closeness centrality, betweenness centrality, importance score;
- c. analyze websites by categories, language, topic and other useful meta information.

## 1.2 Objective of the Study

Therefore, the objectives of this project are the following:

- *Creating a database (catalog) of national domain names*

This is needed to ensure that the most used website names are collected and stored in one repository for maintenance and further processing. The database will store a vast number of records efficiently, make it simple to easily find sought-after information, add or edit new data. At the same time, storage of records in a database can make it easy to import them later into other applications. This needs to be done in a structured format for easy usage, necessitating a relational database.

- *Creating a web crawler to browse the local web and index pages*

This is necessary to enable collection of such important data as website linkages to in-country or external websites, metadata gathering, content parsing to identify topic categories and/or language, etc. This tool will gather information about each page, such as titles, keywords, etc., and save and index that information. The pages get indexed to sanction the process of storing and organizing linkages between web pages. Indexing can also optimize the performance of the web crawler itself. The queue of websites to crawl is created by switching to hyperlinks to which each currently crawled website is linked. Contents are parsed and entries for a search engine index are constructed.

- *Creating a graph-based interactive visualization to analyze web connectivity and strength*

Nodes of the interactive graph shall embody all the crawled online entities (i.e., websites) whereas edges are the connections in-between. Graph node representations are to be differentiated by many parameters, such as degree of centrality and/or betweenness, influence level. Edges shall vary in thickness and color according to such metrics as the number of internal or external linkages, weights, etc. The visualization is planned to be interactive with user flexibility to zoom in and out, switch modes, enabling or disabling graph features, amongst others.

- *Developing a ranking algorithm to identify the most influential websites*

This algorithm will count the number and quality of links to each website and evaluate its importance based on those characteristics. The more important the website, the more it is being referenced by others. Potentially, this computation will need to be performed on an iterative basis to approximate the true importance values. There are various ways to measure these scores, ranging from betweenness centrality to identifying community clusters in a network analysis. It is usually best to analyze how tightly grouped the resulting communities are.

- *Classifying crawled websites by chosen criteria*

This includes analyzing website content to classify it by language, category it falls into by topic or association, identifying the location of its hosting service provider (in-country or outside).

Overall, this project is multiple-staged consisting of firstly, the data stage during which a database of domains is collected. The next step is data cleaning to avoid duplicate entries or errors in reporting website names by respondents. Following that is the crawling, storing, and analyzing period, after which the ranking algorithm, graph creation and potential content-based analysis are applied.

### **1.3 Significance of the Problem**

Delivering an end-to-end product for analysis of the local web has the following significance:

- Unveiling information flow patterns and influences in the local network structure.
- Servicing the gathered information in a user-friendly way with visuals and minimum manual effort for further analysis.
- Serving as a platform for further advanced analyses for the purpose of natural language processing, such as identifying entities and topics, performing sentiment analysis, etc.
- Serving as a common data source for business purposes, for example in cluster analyses, effective advertising strategies and customer lead generations.
- Serving as a common data source for academic research, for example in economic policy analyses.
- Serving as an analysis of the country's level of reliance on outside sources and simulating possible outcomes in national emergency situations.

## 1.4 Review of Significant Research

### 1.4.1 Web Crawling Reasons & Properties

The reasons why web crawling was originally invented are numerous. Search engines being in need of returning relevant answers to user queries, automated web application and security testing are amongst some of those reasons. In their original design, web crawlers were purported to have the following characteristics [7]:

- complete coverage of all web pages given unlimited time;
- refresh policy to keep up to date with the latest changes on the web pages;
- non-overloading and not attacking any website's servers;
- scalability with an increasing number of web pages to cover;
- correct reflection of crawled content;
- efficiency in execution time per page.

Some challenges of web crawlers at their current research stage include scalability problems when it comes to analyzing tons of web pages in an increasingly growing web, ability to consider context when searching for specific topical keywords, prevention of website overloads that can hinder their performance, avoidance of copyright and privacy issues which occur often since crawlers copy a web page's information without any permission from its owner [8].

### 1.4.2 Web Crawler Architecture & Related Graph Theory

The problem of web crawling can be viewed as a graph traversal algorithm. The web can be thought of as a big graph where nodes are websites and edges are the hyperlinks between them. The crawling algorithm sends requests to the web using an HTTP request, downloads web pages by tracing those hyperlinks and parses their HTML structures for content. The crawler begins with a handful of seed URLs and gradually progresses through every linkage [9]. The URLs are usually kept in a queue object in a prioritized manner. The whole process takes a comparatively short amount of time when executed right, especially if parallelized. Figure 1 illustrates the basic process of the crawler.

Algorithmically, a set of URLs, i.e. nodes, are stored in an array. The edges, i.e. links, can be saved as adjacency lists to illustrate forward and backward traversals. Some researchers [10] store in-links to the web page in an adjacency list separate from out-links from the web page which are saved in another. In general, out-links are the pages that can be accessed from the currently crawled one, while in-links are those that lead to the current web page. The graph is usually a directed one because a path can exist between web page  $a$  leading to web page  $b$ , but it does not mean there is a path from  $b$  to  $a$  for sure. Strongly connected components imply that for any web page  $a$  and  $b$  in a chosen cluster of nodes there is always a path. Out-degree of any given node is the number of web pages the node leads to, while in-degree is the number of web pages leading to the node. Full URL is never stored in the graph since it takes up too much space, especially for longer links. Hence, each URL is indexed by a unique identifier. Subsequently sorted URLs are then encoded as differences between the current and previous URLs which comes at a trade-off between space savings and

translation requirements. The authors in [10] deal with the trade-off by saving the full URL at some intervals for efficiency. These stored URLs serve as benchmarks from which the encoded URLs can then be searched. The list spaces are increased for any new additions. This paper proposes the usage of a Connectivity Server which assigns unique identifiers to each URL, explores the neighboring nodes, and delta-encodes them, then translating their encodings back into full URLs. Execution time is faster by about 0.01ms/URL.

The resulting outputs are analyzed by building a graph of connections in-between web pages. Nodes that do not have in- or out-links are removed from the graph entirely. Edges of those nodes that are hosted by the same server are erased. AltaVista link queries are used to determine in- and out-links, stored in forward and backward sets. Graph construction time is around two hours. Kleinberg's algorithm is used to find pages that act as authorities or hubs across clusters. This score is then used for ranking nodes according to their importance. To make the graph visually appealing, URL names are shortened to include host name and ID.

Other research in this area attempts to confirm the power law distribution for degrees in a connected graph. Broder et al. [11] analyzed that the core web can be treated as one strongly connected component (SCC). The Connectivity Server software is used to handle URLs and a breadth-first search strategy is employed. In- and out-links form other two components (IN and OUT). There is also a component for web pages that do not reach the SCC and cannot be reached from it, either (TENDRILS). The authors show all components to be roughly of the same size. The paper claims that the structure of the web graph cannot be judged by looking at it at microscale - a macro picture is needed to fully grasp its complexity. For instance, there is only a 24% chance that a chosen random source node is connected to the target via some route. When a path does exist, however, the average distance is 16 nodes. Power law distribution is observed in the distribution of degrees in such a resulting web graph regardless of its scale. This law is applicable to both in- and out-degree distributions separately. The web's connectivity remains relatively unaffected by the disappearance of substantial chunks.

A method to hierarchically crawl the web was suggested by David et al. [12]. The paper claims that despite the massive volume of data being crawled, there is no reason for it to be chaotic. A statistical technique is employed to analyze the user's behavioral patterns and use that as information to identify the topics of hyperlinked communities. This technology makes it easier to perform targeted web crawling based on user's actions.

### 1.4.3 Types of Web Crawlers

There are different types of crawlers available in the industry:

- *Parallel Crawlers.* Given the huge number of websites on the world wide web, a sequential crawling process can take up an unnecessarily long period of time to execute. To scale the process and decrease the load on the network, parallel crawling processes can get started.

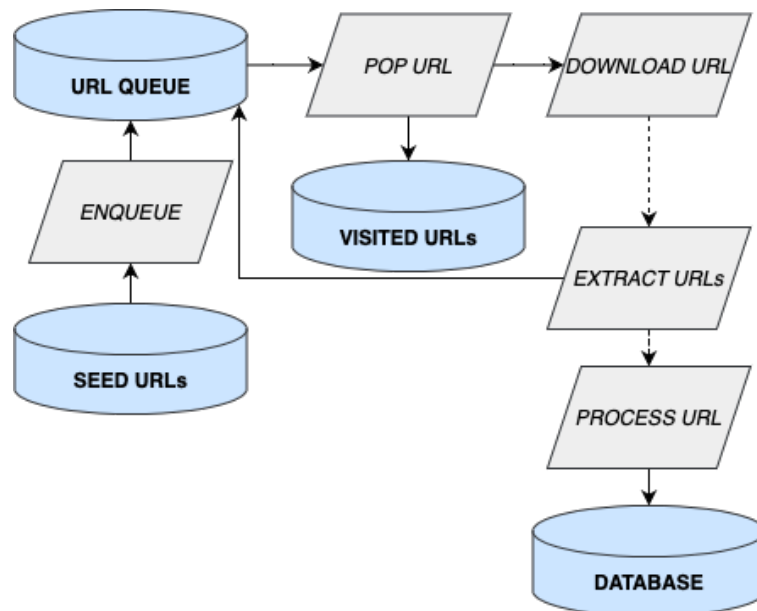


Figure 1. Basic Web Crawler Architecture

One of the not so obvious advantages of this approach is in the clustering of crawling processes by physical locations such that geographically neighboring pages get downloaded in a batch [9]. However, this does not come without its own set of issues connected with the variety of data on the web: they can come in unstructured formats or in extensions that the crawler is not able to handle. Figure 2 displays the workings of the parallel crawler.

- *Focused Crawlers.* This type of crawler retrieves only those pages that are relevant to the chosen topic, and hence is often called a topical crawler. Before crawling any page, the crawler determines the relevancy of the document to the topic to save time and resources [13]. If the relevance score does not meet a certain predefined threshold, the web page is cast aside. Crawling tasks get allocated across various workers using shared memory. However, one weak spot of this approach is the search by keywords without taking context into account.
- *Generic Crawlers.* All data is gathered from each web page regardless of its topical relevance by this crawler. This, therefore, leads to a lot of time and memory consumption [14].
- *Distributed Crawlers.* These operate on a network of computers each of which has its own dedicated crawler. Managing and coordinating the work on each node is the main issue to ensure that work does not get repeated on different nodes. Distribution can take place through such frameworks, as [14]:

- A master-slave relationship, in which a host machine leads the process of assigning and monitoring tasks on slave machines.
  - An autonomous relationship, in which a given machine communicates with every other autonomously to coordinate its work.
  - A mixed relationship, in which autonomy is maintained yet the host can intervene in case of failed tasks to re-assign them.
  - A peer-to-peer framework such that crawling is distributed across loosely cooperating nodes using a hash table for URL-to-node allocations.
- *Incremental Crawlers.* To maintain the up-to-dateness of crawled data, incremental crawlers revisit the previously crawled pages on a periodic basis. The pages that are ranked as more important than others get revisited and updated first on a priority basis [15]. One of the main issues of this crawling type is a lack of scalability and a huge load on the network. Separate modules rank web pages collected in a dataset by their importance scores, and update those based on the resultant priority.
  - *Hidden Web Crawlers.* This crawling algorithm downloads the data that is hidden from the search engine. Contents of each web page are analyzed for the presence of a search form (the “<FORM>” tag in HTML). The found form is then analyzed, candidate values are created, and then forms are filled out to get submitted to their respective URLs. Following this process, the linked hyperlinks are crawled next [16]. Despite its pros in gathering information from the sites hidden away, this crawler is not scalable and cannot operate with some file extensions.

In general, due to the massive pool of data on the web, crawling is a slow process. Yet its time execution can be improved by adding hardware resources and/or boosting network bandwidth. At the same time, focusing the crawling task only on essential web-pages can raise the bottleneck created by useless web pages [14].

The performance of the crawler can be measured by such metrics as (1) the harvest rate per some unit of time, or (2) the ratio of the number of relevant pages over total crawled pages.



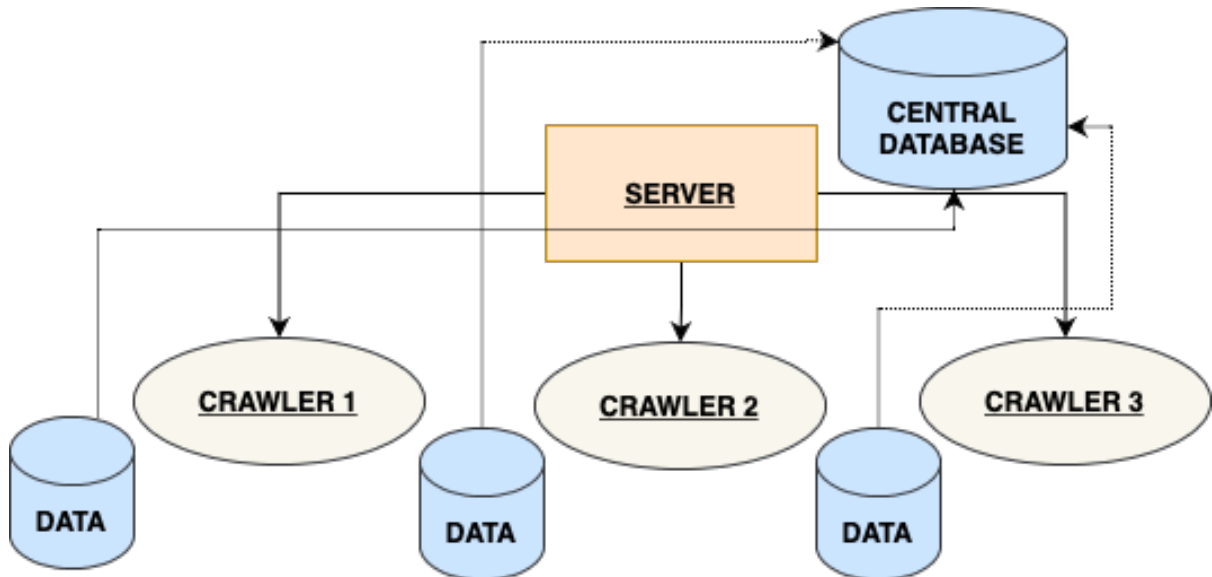


Figure 2. Parallel Crawler Architecture

#### 1.4.4 Data Storage

The crawled data can be stored either in a database or local file depending on the size. The database storage can be in a tabular or key-value format. By making each transaction atomic, possible failures and data losses are prevented. Some existing implementations have made use of a Redis key-value database which does not allow for repetition [14] and increases the efficiency of the crawler checking if a web page already exists in the database. Others made use of such NoSQL databases as MongoDB to store the crawled data. In comparison to relational databases, the latter one is schema-free and offers easy horizontal scaling at minimal cost [17].

#### 1.4.5 Crawling Search

There are several algorithms that crawlers can adopt to perform the search process for hyperlinks. In essence, the issue comes down to graph traversal using either of the following strategies: depth-first search, breadth-first search, or best-first search.

Depth-first search is a recursive traversal strategy in which the search process starts from a random root node and each linked web page is then examined before retracing. The visited nodes are marked, and the algorithm continues with other unmarked nodes. The way it works algorithmically is through keeping a seed of URLs in a stack, popping the last element from it to traverse next, adding it to a set of those nodes that are already visited, adding connected hyperlinks to the stack, and continuing this process until no web page remains [18]. The pros of this approach are in less time and space complexity, with cons being the inability to find the most optimal solution

due to being stuck in some branch or not knowing the optimal depth of the search. Figure 3a illustrates the workings of this algorithm.

Breadth-first search traverses all nodes on a given depth level first before continuing to the next depth level. A random root node is added to a queue object, then a node is dequeued - the first element is popped. The visited node is marked, and all neighboring nodes are added to the queue. The process is continued until no node remains in the queue. The advantages of this approach include optimal time complexity in a sense that the search process does not go on too deep, especially if the solution is near the root of the tree. The main disadvantage, however, is in memory requirements to store all nodes and their neighbors [19]. Figure 3b displays how this approach works.

Best-first search expands a chosen node based on some priority rule. A heuristic function is used to estimate the closeness of any given path to the solution and govern the traversal. In essence, the issue comes down to finding the least-cost path. A priority queue is maintained to pop the elements with the highest priority first. This algorithm can come in informed and uninformed forms. The most popular informed version of it is A-star, in which a cost of the path is the sum of the costs of its arcs, including a heuristic cost to the target node [20]. The disadvantage of this approach is in the inability to design a proper heuristic function sometimes. Figure 3c displays the basics of this search method.

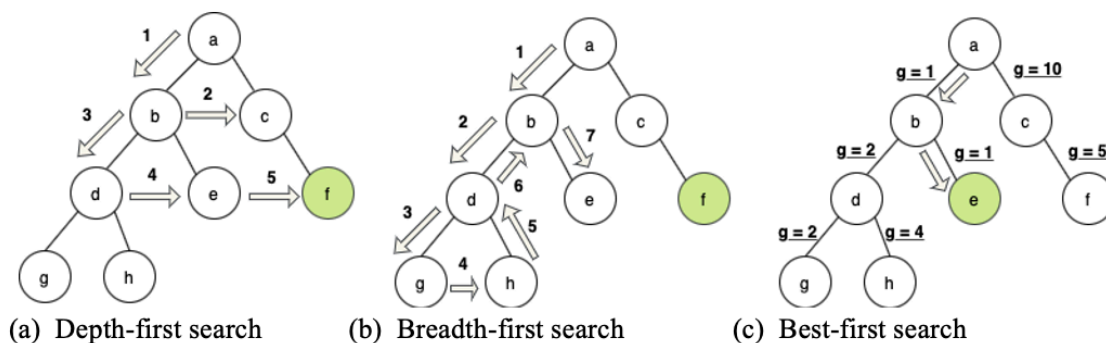


Figure 3. Graph Search Strategies

The process of identifying all neighboring nodes has long been a slow one, and several prototypes have been tested to quicken the search for predecessor and successor nodes.

### 1.4.6 Smart Querying

Although traditional keyword search serves its purpose for finding required information in a vast pool of unstructured data, the process becomes too cumbersome when dealing with structured data. Users are left with no choice but to manually examine the data they have, decide which ones they need to access, analyze source data individually and then combine the results. Hence, there is a requirement to develop smart querying algorithms to eliminate the need for such manual processing.

Some researchers have tried developing systems which can provide consistent access to a collection of documents with varying formats [21]. These systems try to address the following concerns:

- connected data stored across different sources must be recognized and exploited,
- connections must be utilized to fill in the gaps of incomplete data of any particular source.

Levy et al. [21] developed an algorithm for efficiently finding an answer to a given query from various data sources. The practicality of this mechanism is in its scalability without sacrificing performance and the ability to identify the same object referenced across different sources.

A relational data model is implemented with a class hierarchy between query subjects and objects. Each source content must be linked to these classes and their features to be able to answer a given query. The way this is done is by modeling each source content as a set of tuples (i.e., relations). For example, a source can contain a relation of *CourseList(course, teacher)* which is mapped to an overarching global relation of *Teaches(course, teacher, hour, room)* stored centrally [21]. In essence, each source is then described as a set of possible queries. Capability record for each source is maintained to showcase the kinds of queries it can answer. A series of accesses to data sources, i.e. query plans, are created to incorporate data from many sources to answer a given query comprehensively.

Traditional query plan executions using views are not scalable because they entail an exponential increase in processing with the rising number of data sources. Hence, the authors invent a new algorithm for the same problem as described below.

1. The first step of the algorithm is to create a bucket for each part of the query. For example, a tuple is searched for across every source's capability record. A query to extract this data from each source is designed.
2. The resulting sub-queries are combined and reviewed for whether they are semantically correct as a total.
3. The ordering of the combined query is considered to make sure that it is correct and will return the targeted answer.

The resulting system is called the Information Manifold (IM) which has a user-friendly interface. Users can flexibly adjust their queries using either prepared or blank templates. The system can handle data from up to 100 information sources with about 10 plans generated from those per query at an estimated 22 seconds per plan. The authors prove that time of execution increases with adding more information sources, however, the increase is not exponential. The architecture of the system is provided in Figure 4 adopted from [21].

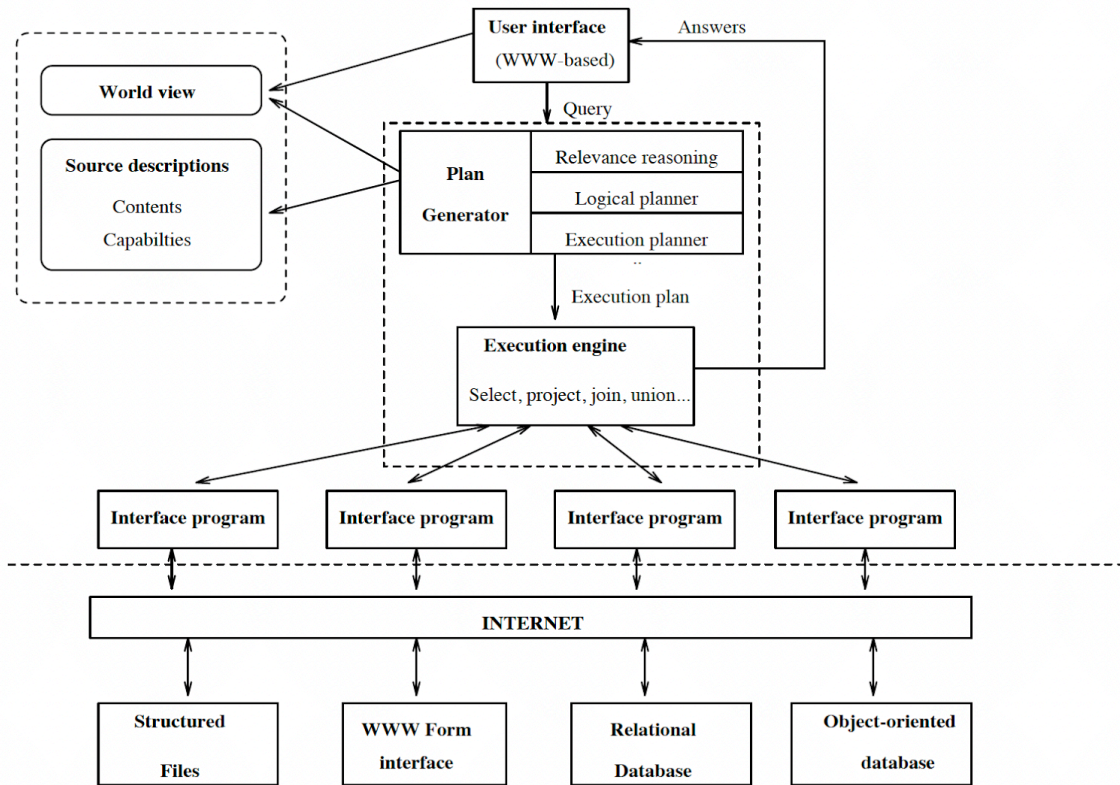


Figure 4. The Architecture of the Information Manifold System [21]

#### 1.4.7 Single- or Multi-Threaded Web Crawling

Crawling methods based on priority queues usually perform best because hyperlinks are weighted and put into a priority queue. The advantage of this approach is that undesirable hyperlinks are pruned beforehand from consideration. To speed up the process, however, multi-threading can be used.

Multi-threaded web crawler is implemented by having one main thread which controls the worker threads. Each child thread is responsible for downloading its assigned set of web pages, extracting hyperlinks from each downloaded web page, extracting domains and other relevant content, and ranking the web page based on some algorithm. The data of each crawler is then written into shared memory buffers. The structure of the working algorithm is such that [22]:

1. A root URL is chosen for further expansion.
2. Hyperlinks within the URL are extracted.
3. Text from the URL is parsed and converted into a term document frequency matrix.
4. Cosine similarity between the web page features and the original query is computed.

5. The computed score is used to rank web pages in a priority queue to achieve a best-first search traversing strategy.

Some researchers have tried carrying out the multi-threaded process using hierarchical clustering. For example, in [23] authors use an input textual file to select a root web page to start crawling. Next, hyperlinks within the crawled web page are found and indexed. The web page is then clustered according to the specified number of clusters. Java NetBeans, MySQL, and WAMP Server are used to apply and compare the performance with that of a single-threaded process. Comparison metrics used include harvest rate, execution time, recall, precision, and harmonic mean, amongst others. Harvest ratio is defined as the number of relevant documents relative to the total number of crawled documents. The researchers prove that their method works better in terms of execution time and relevancy score of the crawled web pages.

Another study [24] suggests avoiding the usage of a simple URL queue in multi-threaded executions. Whenever multiple threads try to access the same resource, a deadlock can potentially occur by one thread waiting indefinitely for another to finish. Therefore, a synchronization lock (mutual exclusion) with a binary semaphore should be present. While the mutex is locked, access is reserved only for the thread that has acquired it. After performing its operations, access is released so that another thread can pick up the resource. If at any time, any thread tries to gain access to the shared resource, such as the URL queue, and it is not available, then the operation is pushed into a stack for waiting. To avoid problems with an empty URL queue stopping the crawling process, when in fact threads are still working on adding more links to the URL queue, a novel method to put threads to sleep is used [25]. When any thread faces an empty URL queue, it is put to sleep for some time. When it is awakened, it checks for URLs in the queue once more. The number of threads that are sleeping is checked via the global monitor track. The entire crawling process stops only after all threads are in the sleeping state.

In [26] red-black tree structures are used to manage the process of fetching URLs from a database and inserting new hyperlinks. These data structures are efficient and result in low computational costs. Parsed HTML structures are stored in a HashMap format which is a key-value data store with  $O(1)$  time complexity mapping the term frequencies of words to page addresses.

#### **1.4.8 Graph Ranking**

There are several graph ranking algorithms in place, such as Kleinberg's HITS (Hyperlink Induced Topic Search) or Google's PageRank. In simple terms, a graph ranking algorithm determines the importance of any graph node by recursively computing relevant information from the whole graph.

Kleinberg's HITS mechanism computes two scores iteratively: a) degree of a node's authority, which is measured by the number of incoming links, and b) a node's hub value, which is measured by the number of outgoing links. Google's PageRank, however, takes both in- and out-links into account when computing the rank iteratively. In [27] the ranking algorithm is further extended by adding edge weights as another factor to consider. The authors develop a text ranking method - Text Rank - that considers the extra factor due to the presence of multiple links between text sentences and the need to know how strong the sentence-to-sentence connectivity is.

A pseudo-code for the calculation of Kleinberg’s HITS rank is provided in Algorithm 1. In essence, for any given node, initial hub and authority scores are assigned. Then, the scores get updated in an iterative fashion  $k$  times. In the end, normalization is performed to arrive at a comparable value scale. A node with a higher authority score is the one to which high hub scored nodes lead. A node with a higher hub score is the one to which high authority scored nodes lead. Authority scores dictate whether nodes contain valuable information about a given query. Hub scores indicate whether nodes are useful in getting to the high authority pages [28].

Positional power ranking is another method whereby a node’s importance is determined by both the number of outgoing links and the power of the neighboring nodes. The power of a node can be computed by the Copeland score or dominance functions [29].

Google’s PageRank is not as simple as counting the number of in- and/or out-links to determine how important a given vertex is. Not all links are accounted for equally, and normalization is also performed. A probability distribution of ending up on a web page is computed in an iterative fashion. In particular:

1. All web pages are initialized with the same random value for the probability of arriving at that node. For instance, the initial probability can be  $0.3$  for a set of web pages Alan, Brown, Clyde, and Dino.

2. If pages Brown, Clyde and Dino only lead to Alan, then the probability of Alan is computed as a sum of their probabilities. In essence,  $P(\text{Alan}) = 0.9$ .

3. If, however, Brown leads to Clyde and Alan, Clyde leads only to Alan, and Dino leads to all three, then  $P(\text{Alan}) = P(\text{Brown}) / 2 + P(\text{Clyde}) + P(\text{Dino}) / 3 = 0.55$ .

4. Since Brown leads to Clyde as well, and Dino leads to Brown and Clyde, the probability of Clyde must be updated. Thus, iteration starts and  $P(\text{Clyde}) = P(\text{Brown}) / 2 + P(\text{Dino}) / 3 = 0.25$ . This now implies that  $P(\text{Alan})$  must be reinstated using the updated Clyde value.

5. Iteration continues until the probability values converge.

The mathematical formula is as below, where  $A, B, C$  are pages,  $L$  is the number of outgoing links from each page,  $N$  is the total number of pages and  $d$  is a damping factor for the probability of a user leaving the page [30]:

$$PR(A) = \frac{(1 - d)}{N} + d \times \left( \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \dots \right) \quad (1)$$

The PageRank algorithm has been adopted and extended for more specific purposes across industries. For example, in [31] the authors propose using a citation count metric along with PageRank to prioritize outbound links for scientific text analytics.

In [32], the authors suggest that due to the high vulnerability of any given graph to purposeful harmful attacks executed with the purpose of increasing the ranking unjustifiably, understanding which exact graph elements contribute most to the ranking score is crucial. Algorithms to find such graph elements are proposed. Aurora-E method is presented to determine the top-K important edges in each graph with a diminishing value property. The core algorithm relies on picking an edge and then updating the gradient matrix for every  $K$  iteration. The influence of a node is determined by Aurora-N method whereby it's defined by the number of ingoing and outgoing edges [33].

---

ALGORITHM 1: Kleinberg's HITS Algorithm

---

```

N = set of nodes

for each node in N do
    node.auth = 1 // initial score for authority of the node
    node.hub = 1 // initial score for hub of the node

for iteration in range(1, k) do // perform for k iterations

    for each node in N do // update authority scores
        node.auth = 0
        for each neighbor in node.incomingNeighbors do
            node.auth += neighbor.hub

    for each node in N do // update hub scores
        node.hub = 0
        for each neighbor in node.outgoingNeighbors do
            node.hub += neighbor.auth

```

---

Other metrics can also be used to measure the importance of any given page, such as the presence of relevant keywords in a document which depends on the context the user is searching for, similarity of parsed information to a given query, cosine similarity between the seed page and its hyperlinks to determine relevance, classification score when training a machine learning model on the page contents.

### 1.5 Assumptions & Limitations

The following assumptions and limitations apply in this research:

- *Avoidance of Endless & Closed Loops.* The problem of falling into a never-ending loop whereby each page's outgoing link leads to more outgoing links in an iterative fashion must be circumvented. At the same time, it may happen that an outgoing link leads back to the already crawled page thus forming a closed loop. To prevent these issues from taking place, the implemented

web crawler will only process the depth level of 2 for outgoing links from each page in the collected dataset without continuing beyond the initial tranches. No page will be visited twice, so the crawler is not planned to be incremental.

- *API Bottleneck.* API (Application Programming Interface) requests rate-limiting is applicable such that no more than 45 concurrent requests can be sent per minute. This is due to the usage of a tool which offers this free quota. The limits are set to safeguard websites from a huge stream of same-time requests and potential denial of service (DoS) attacks. If this quota is exceeded, a “429 - Too Many Requests” or “ECONNRESET” errors get thrown.

- *DDoS Attack Bypassing Cloudflare.* Some websites make use of Cloudflare to protect from attacks by rate limiting, firewall protection, and IP access rules. Bypassing this system is a limitation.

- *Azerbaijani Content in Non-Azerbaijani Pages.* Pages containing content in Azerbaijani language on such external websites as “*facebook.com*” can be mistakenly crawled as part of outgoing links and thus, classified as local. HTML structure parsing and language identification will therefore be crucial in the correct determination of local content.

- *Azerbaijani Domains Used Externally.* Due to the relative cheapness of Azerbaijani domains as compared to others in the market, it may be the case they get purchased by external parties and do not actually associate with or contain any local content. As in the bullet point above, parsing and language determination are to be used to safeguard against such realistic instances.



## 2 METHODOLOGY

This section of the paper describes the methods and tools used for the study. The first subsection describes the data collection strategy, and the second subsection follows with a visual explanation of the implemented database. The third subsection details the workflow of the applied web crawler algorithm. The penultimate subsection focuses on graph construction and visualization techniques, with the final subsection dedicated to the implementation of a graph ranking algorithm.

### 2.1 Data Collection

Information about Azerbaijani domains is collected via two sources:

- a survey conducted within the ADA University student community, and
- data supplied by the country’s Ministry of Communications & Information Technologies.

The design of the survey is such that the respondents are asked to enter their first and last names, and the list of local domains they are aware of and often use. The survey is conducted on a sample platform created using the Django web framework. The reason for choosing this is due to the open-source, ready-to-use, and easy architectural pattern that Django provides.

There are 31 unique survey respondents. The collected data necessitates a few cleaning steps to perform for domain purification. In particular, the regular expressions defined in Table 1 are used to strip out the “*www*”, “*http*”, “*https*”, “*/*” and “*://*” characters. So, a website entry like “*http://camex.az/*” is converted into a much simpler “*camex.az*”. However, some entered domains bypass the purification stage due to having an index to individual pages like “*eyol.az/contact.html*”. These are impure domains and are subsequently neglected resulting in only 1623 unique pure domain names.

The data provided by the Ministry contains approximately 13000 unique web pages.

Table 1. Regular Expression Patterns for Domain Purification

regex pattern 1	regex pattern 2
<code>(http[s]?)://www\.(.*\.[a-zA-Z]*)/?.*</code>	<code>(http[s]?):/(.*\.[a-zA-Z]*)/?.*</code>

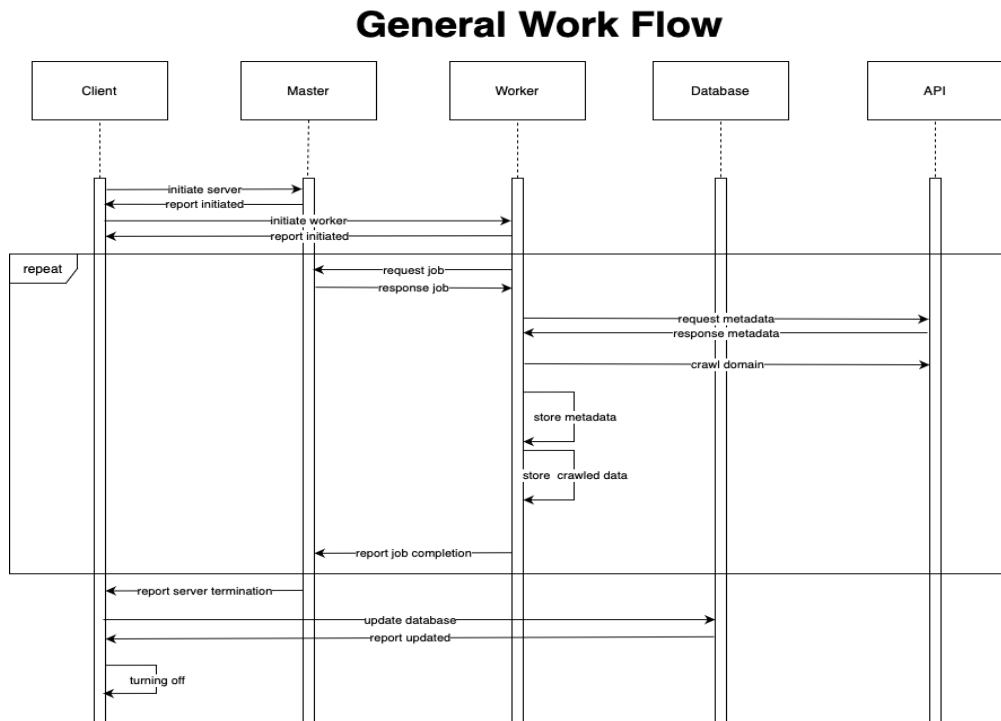


Figure 5. General Program Workflow

## 2.2 Program Workflow

There are three parts to the implemented program: master, worker, and the database. At the current research stage, the database is not fully integrated yet, however, all the necessary parts are in place.

To make the system work and have the data crawled, first the master is initialized. In the terminal, the actions that the master can do are made visible. For instance, first the message will say that configurations are set, and then it will add jobs. Each domain gathered from the Ministry is a unique input and hence, a unique job. Once all jobs are added and configurations are set, the master is ready to serve and prints the corresponding message.

Configurations entail the existence of an environment file for database connection credentials, and critical parameters for the worker and crawler – for instance, a maximum depth parameter to define how deep the crawler should dive in the job.

On a separate terminal, the worker is initialized. The reason why this takes place independently is due to multiprocessing. Asynchronous calls are created to make the program execute faster. Requests do not have to be run in a single asynchronous call because they can be made independent and have their own threads. There is a waiting period for the completion of all threads to gather results in a predefined URL structure. It is important to note that several structures are made available throughout the program – for instance, a URL structure supports easy and understandable mapping

and ingestion. Once the worker is started, it also goes through the initialization and configuration stages. Next, it starts requesting jobs from the master through remote protocol connections. A socket between the master and worker is created, and the worker calls the socket all the time. A method to request jobs is available in the master program, with an argument and a reply pointers. Once the master is ready to assign a job to the worker, the latter is added as an argument pointer and is assigned to the reply pointer, so the worker can get that reply for processing or appending to its queue.

Once the worker receives a job, it starts collecting metadata. A special API is used to return all the necessary attributes such as IP address, location, latitude, and longitude, etc. The gathered data is stored in a metadata structure and as a JSON file in a specified location. The structure of the folder for each domain is such that: 1) in the environment file, the location for the domain is specified, 2) the program creates a folder with the domain name in that location, and 3) creates a metadata JSON file in that folder once collected, and a URLs JSON file once all hyperlinks are crawled. The created URL structure is not as simple as the Metadata structure, for example. A single string is not simply stored but is in fact split into domain, subdomain names and extension before submitting to the database for easy processing. The URL structure has such attributes as protocol, subdomain, domain, extension, and path. After each URL is split, a single URL structure is added to the list. Once crawling is performed, that list is converted into JSON and written to a *urls.json* file. Even if a particular domain is not working, it should still have some metadata indicating the status of reachability and a message of why it failed. Hence, each domain has its own folder regardless of its operability.

Once crawling is over, the program moves to the phase of reporting the job status. A method to report jobs is available in the master and the status of a particular job is changed to done. In the database phase actual ingestion does not get applied yet but the structure and multi-thread synchronization are already completed. The database design is structural to ensure integrity and security of data for later analysis.

The general workflow of the program is displayed on Figure 5.

### 2.3 Database Design & Implementation

The design of the database the aim of which is to store crawled information is provided in Figure 6 and explained below:

- The main entities participating in the database are the seed domains collected via the aforescribed survey and from the Ministry, the linked domains which seed domains lead to through hyperlinks and their metadata. Metadata includes useful information about each domain, such as its IP address, geolocation, and content language amongst others.
- The *Domain* table is the table containing the core web sites in the database. Its main attributes are *name* and *extension* which are used as primary keys. In a domain like “*camex.az*” the extension corresponds to “.*az*”, and “*camex*” is the domain name.

- The *DomainMetadata* table stores such attributes as *ip\_address*, *status*, *continent*, *continent\_code*, *country*, *region*, *region\_name*, *city*, *zip*, *lat*, *lon*, *timezone*, *isp*, *as* and *as\_name* which represent descriptive information about each domain.
  - *IPv4* is a 32-bit web page address whose bits are separated by dots while *IPv6* can store up to 128 bits and hence can handle a much larger address space. *IPv6* separates bits by colons and contains hexadecimals. Some of its other benefits include simpler routing and easier administration. Machines connecting to a network are identified using these IP addresses.
  - The *continent* to *timezone* attributes store the geographical placement of the website's IP address. This data can be extracted using an HTTP GET request sent to an IP geolocation identification website.
  - The *isp* attribute saves the internet service provider name, while the *status* attribute returns either a success or failure of accessing the web page.
  - The *as* and *as\_name* attributes store the information about AS number and organization separated by space.
  - All the above attributes relate to each provided *name* and *extension* which act as foreign keys in this table.
- The *Url* table stores the unique outgoing links collected from seed domains. Its main attributes are *protocol*, *subdomain*, *name*, *extension*, *path* and *lang*, *context*.
  - The *protocol* attribute describes the preferred way of communicating with the server of the linked domain. This can be HTTP (HyperText Transfer Protocol) or TLS (Transport Layer Security) for instance.
  - The *path* attribute saves the full URL (Unique Resource Locator) of the outgoing link.
  - The *name* and *extension* attributes store the name and extension of the provided URL path.
  - The *subdomain* attribute stores the part additional to the main name. For example, in "*contact.camex.az*" the subdomain is "*contact*".
  - The *lang* attribute stores the language of the URL, while the *context* attribute stores the HTML document of the page.

- In terms of relationships between domains, each domain can map to many outgoing links. Hence, the relationships between *Domain* and *Url* tables are one-to-many.
- In terms of relationships between domains and their metadata, each domain can only have one entry for all metadata items. Hence, the relationships between *Domain* and *DomainMetadata* tables are one-to-one.

The implementation of the database is performed on PostgreSQL which is a relational database widely used for web applications. The reason for choosing this platform is due to the need to structurally save the gathered data for easier processing. “*CREATE DATABASE*” command is executed from the shell prompt subject to having the necessary privileges. “*CREATE TABLE*” commands are used to construct the tables described above. In particular, the following data types apply:

- *VARCHAR(n)*: this data type is for such attributes as *ip\_address*, *city*, *lang*, *domain name*, *extension*, *path*, *subdomain*, and *protocol*.
- *TEXT*: this data type is to store the value of the *html* attribute.

# DATABASE ENTITY RELATIONSHIP DIAGRAM

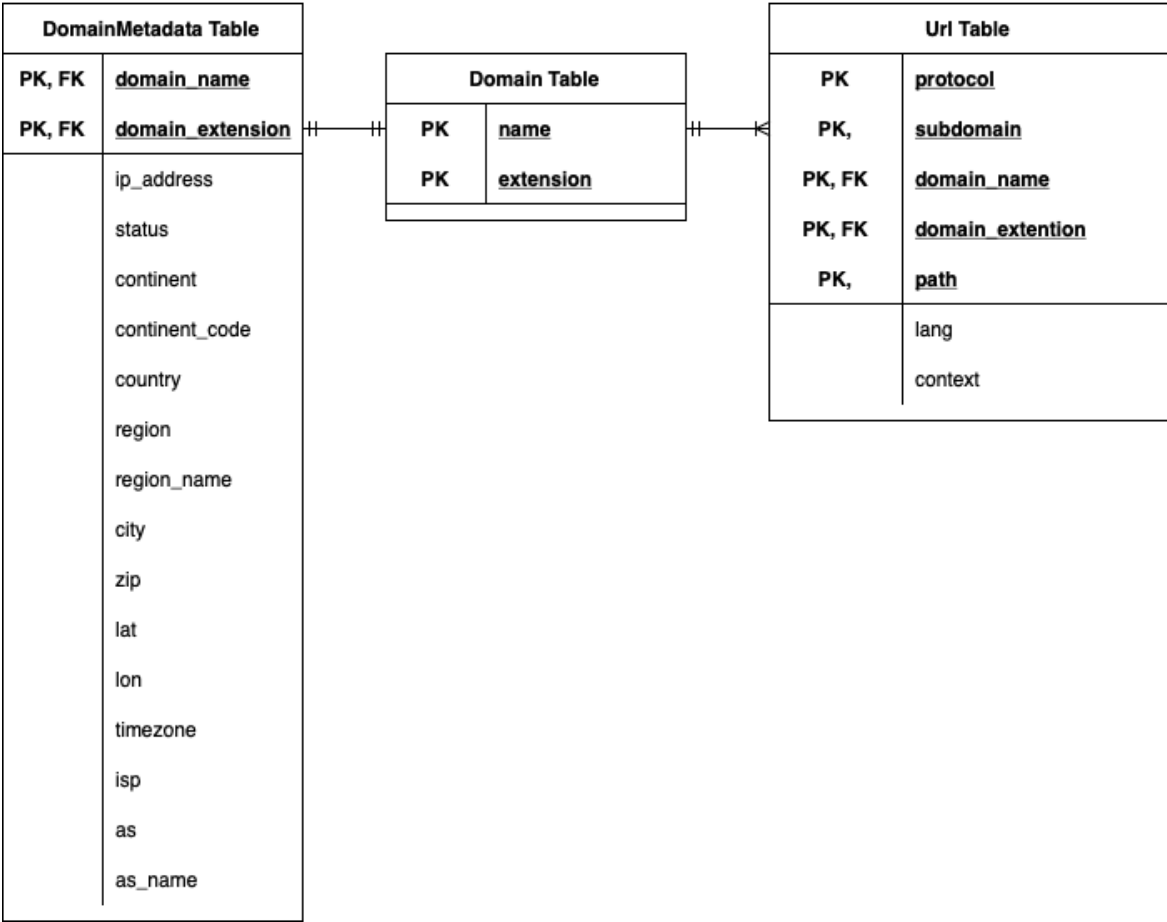


Figure 6. Database Entity Relationship Diagram

## 2.4 Web Crawler Architecture & Implementation

The web crawler consists of two main components:

- the master thread, and
- three worker threads.

The master thread is responsible for allocating tasks across workers through the thread pool and re-assigning any failed tasks back to available threads. Its workflow is provided in Figure 7. Workers perform the tasks allotted to them and are intended for re-usage after task completion. A worker is initialized and retrieves the assigned task from its job bucket. Next, a corresponding URL is extracted from the database. The worker sends an API request call to the domain. If the call is not successful, in a sense that the domain is inaccessible and throws errors, then only the metadata of the domain gets stored in the *Metadata* table of the database. The worker's task is then terminated. However, if the domain is accessible, the process continues. Two asynchronous processes are initialized:

- a process to detect the domain's IP address, find its geolocation, and
- another process to crawl the web page for content, extract hyperlinks within it and store the HTML document of the webpage.

Asynchronous processing entails the usage of lightweight threads for an almost-parallel execution. When the work by these latter threads is finished, the main thread is notified. The crawled metadata and outgoing links are then stored asynchronously as well in the *Metadata*, *Mapper* and *Linked Domains* tables. The worker then continues crawling each outgoing link from each seed domain. If the outgoing link is already in the database, such that it has already been visited, then the task is terminated. Otherwise, usual crawling techniques continue, and the data is expanded. The workflow of the worker is provided in Figure 8.

The entire implementation of this algorithm is done with Go. In particular:

- There are *client*, *database*, *master*, *worker*, *utils* and *main* folders.
- Within the *database* folder, there are such modules as *connection.go*, *server.go* and *structure.go*. Such functions as *Insert*, *Select*, *createConnection* and *MakeDatabase* are made available in *server.go* to interact with and construct the database.
- Within the *client* folder, there is a *client.go* module which contains such functions as *Client* and *call* to provide various commands for the client to use, such as “crawl”, “write”, “help” and “exit”.
- Within the *master* folder, there are such modules as *rpc.go*, *server.go* and *structure.go*. Such functions as *RequestJob*, *ReportJob*, *addJobs*, *MakeMaster*, *MonitorWorker* and *MonitorResources* are made available in the *server.go* module, while serving functions and *JobRequest/JobResponse* structs are housed in the *rpc.go* module. Each job has such attributes as *ID*, *Task* and *Status*, while the master has such attributes as *Jobs*, *Access*, *ProcessDetails* and *SocketName*.

- Within the *worker* folder, there are such modules as *server.go*, *structure.go* and *utils.go*. Such functions as *parseUrl*, *dumpUrlsToJson*, *dumpDomainMetadataToJson*, *readDomainMetadataFromJson* and *generateFinalDestinationStructures* are made available in *utils.go*. Each URL has such attributes as *protocol*, *subdomain*, *domain*, *extension*, *path*, and *metadata* pointer. The URL metadata struct has *HTML* and *language* attributes. DomainMetadata struct has *IpAddress*, *Status*, *Message*, *Continent*, *ContinentCode*, *Country*, *Region*, *RegionName*, *Timezone*, *Lat*, *Lon*, *Isp*, *Scrapable* attributes. *RequestJob*, *ReportJob*, *requestDomainMetadata*, *Work* and *crawl* functions are available in *server.go*.
- Within the *main* folder, main functions to create master and workers, initialize database connections are made available.

A more detailed description of the most significant functions and structs follows together with visual examples.

- *func crawl(task string, dirPath string, filePath string)*

This function crawls the web pages defined within an inputted task and then visits the hyperlinks of each web page. First, a default collector is instantiated which runs multiple threads and caches responses to a given directory path to prevent multiple downloads of pages even if the collector is restarted. Then a callback is called on every found *href* attribute which stands for hyperlinks. The said hyperlinks are parsed and then scraped upon gaining successful access. The function waits until all threads are done. This workflow is detailed in pseudo-code in Algorithm 2 below.

- *func parseUrl(rawUrl string)*

This function returns a custom URL struct composed of its protocol, subdomain, domain name, extension, and full path. Specifically, a raw URL string provided as an input is parsed and the protocol is then set by obtaining the scheme information of the parsed URL details. The host is then split into subdomain, domain, and extension. The full path is also accessed through the parsed URL details. The pseudo workings of this code are provided in Algorithm 3.



# Master's Work Flow

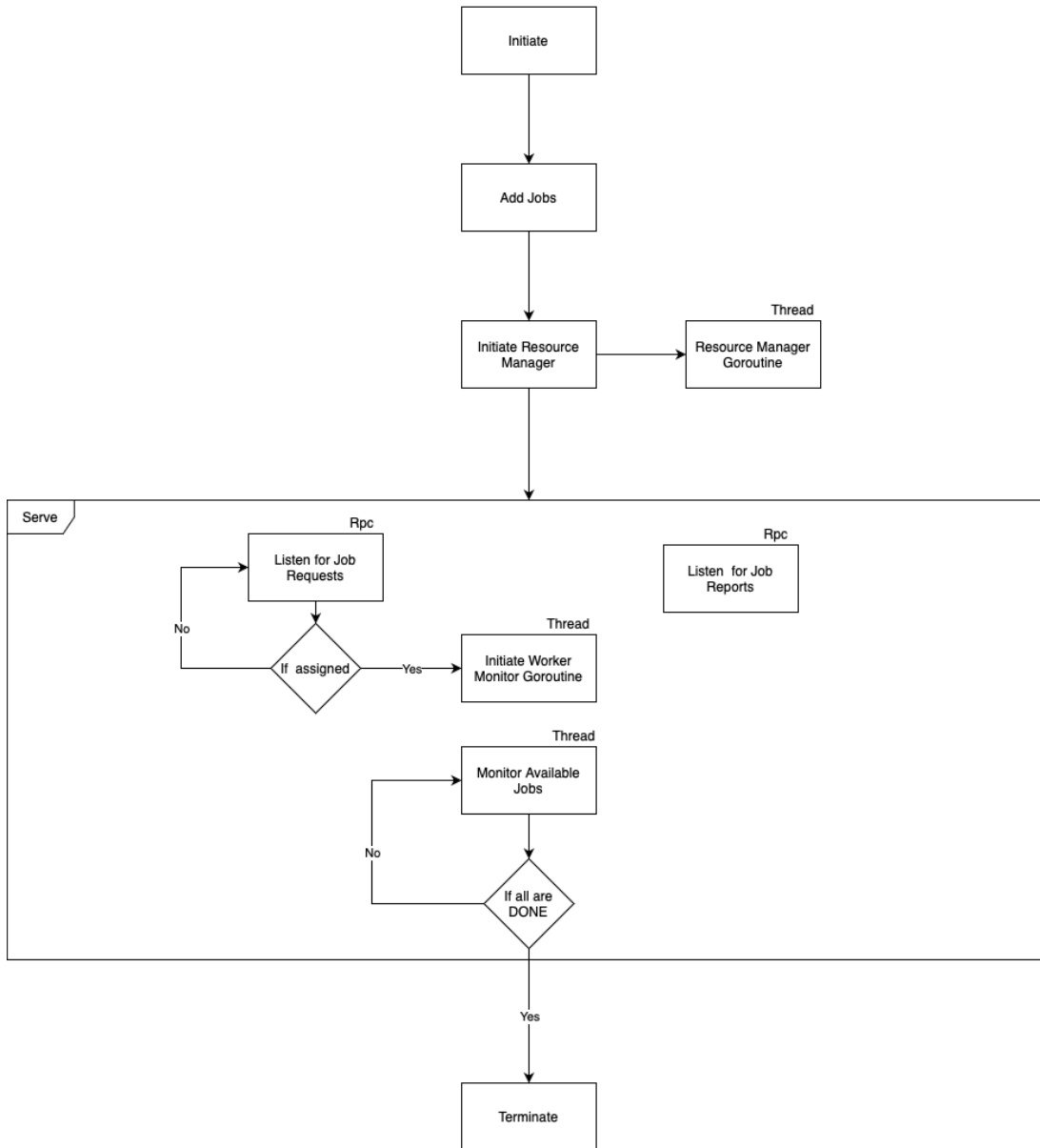


Figure 7. Master's Workflow

---

ALGORITHM 2: crawl() function

---

```
task    = task identifier
c       = standard collector from Colly Golang crawling framework
urls = [] // make an empty list to store hyperlink URLs

c.OnHTML("a[href]", func(e *colly.HTMLElement) do // for each found href

    link = e.Attr("href") // get the outgoing link
    urls = append(urls, parseUrl(link)) // append the parsed link to a common list
    e.Request.Visit(link) // scrape the found link page

c.Visit(task) // scrape the assigned task

c.Wait() // wait for all threads to finish
```

---

---

ALGORITHM 3: parseUrl() function

---

```
rawUrl = raw URL input string

urlDetails = url.Parse(rawUrl) // get URL details after parsing a raw string using go

protocol = urlDetails.Scheme // set protocol

hostSplitted = strings.Split(urlDetails.Host, ".") // split into domain, subdomain &
extension

path = urlDetails.Path // set full URL path
```

---

## Worker's Work Flow

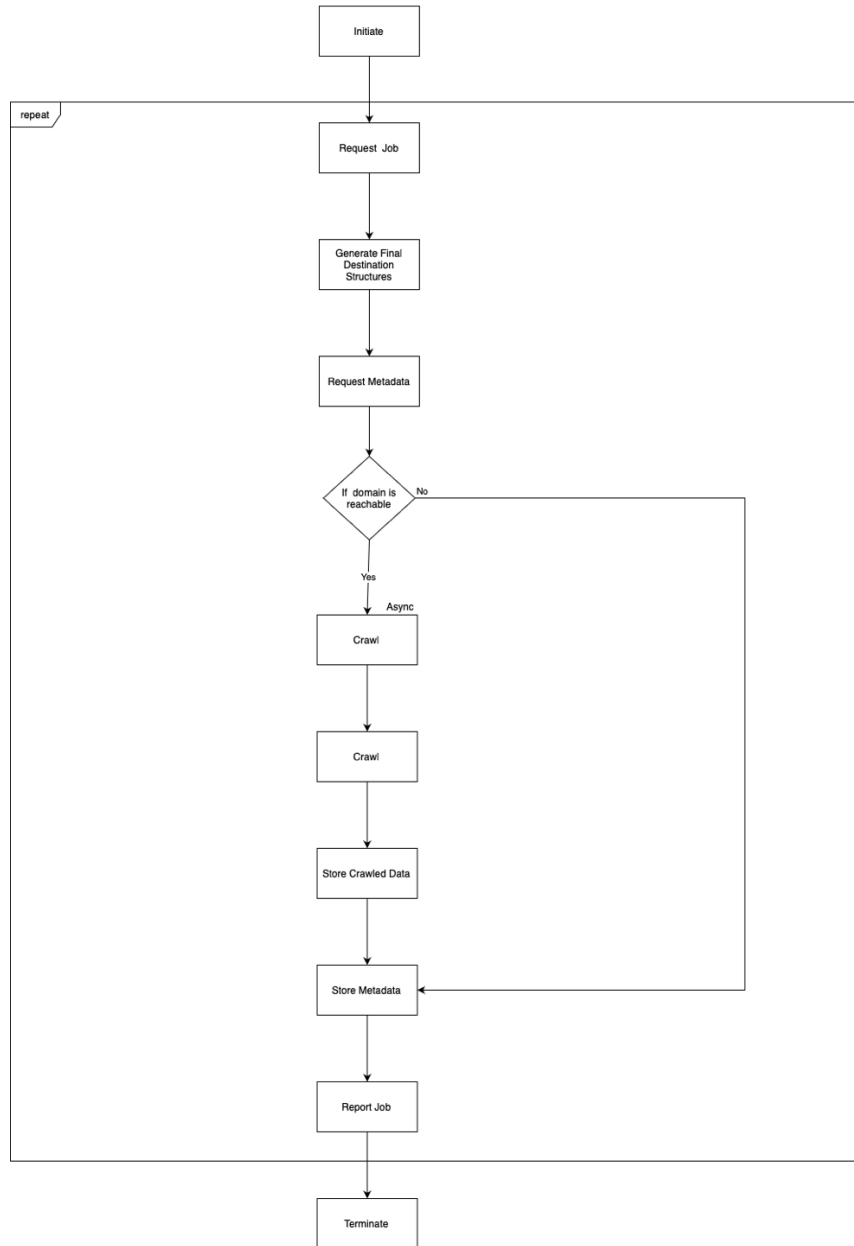


Figure 8. Web Crawler Worker's Workflow

- *func (worker \*Worker) Work()*

This function runs as an endless loop until tasks assigned to the worker are finished or the thread is terminated. First, the worker requests a job from the master thread. If no jobs are left for the worker to do, it is killed. Otherwise, domain folder path is set for the worker-assigned tasks. A check is performed to make sure the metadata of the assigned task has not been scraped yet. If the check is successful, the worker requests domain metadata and writes it to a JSON file saved to the domain folder path. Then, the file path for links is set and the worker starts crawling them, as well, storing the outputs in a *urls.json* file. Finally, the status of the job is updated through a *reportJob* method. The workflow is detailed in Algorithm 4 below.

---

ALGORITHM 4: Work() worker function

---

```

loop
    job = worker.RequestJob() // worker requests a job

    if job == nil: // if no job left, terminate the worker
        break

    outPath = os.Getenv("OUTDIR") // get the output path
    dirPath = outPath + job.Task // define directory path for the specific task

    if _, err := os.Stat("sample.txt"); err != nil: // check if the metadata was scraped

        metadataFilePath = dirPath + job.Task + "metadata.json" // path
        metadata = worker.getDomainMetadata(job.Task) // get
        dumpDomainMetadataToJson(metadata, metadataFilePath)

    outgoingLinksFilePath = dirPath + job.Task + "outgoingLinks" // the path to href
    crawler.crawl(job.Task, outgoingLinksFilePath) // crawl links & dump to JSON

```

---

- *func (worker \*Worker) requestJob()*

This function is executed by a worker thread to request a new job. *Request* and *Response* structs are initiated, and the worker calls the master asking for a task. If the status of the reply is *Assigned* then a job is returned. In case the status is *Wait*, the thread hangs on until a job is assigned. Otherwise, the returned job is *nil*.

- *func (master \*Master) monitorWorker(job \*Job)*

This function monitors the worker's job from the main thread. If no response is received from the worker thread during 300 seconds, then the worker is dubbed as idle. If the job is done, however, the monitoring ends. Algorithm 5 details the workings of this function.

- *func (master \*Master) monitorResources()*

This function monitors resource availability. The domain metadata API request count is reset to zero if no response is received within a minute.

- *func (master \*Master) addJobs(tasks [] string)*

This function adds jobs to the job pool. Looping over all tasks takes place and an incremental ID is assigned to every task, which then gets added to the total pool of master's jobs. Idle jobs count is also updated. The master thread manages idle, occupied and done jobs. This workflow is detailed in Algorithm 6.

---

ALGORITHM 5: monitorWorker() function

---

```

timer = 300 * time.Second // set timer to 300 seconds
defer time.Stop() // execute until the timer is up
loop
  if timer.C: // if timer is up
    job.Status == "idle" // set status of the job as idle
    return

  default: // otherwise
    if job.Status == "done": // check if job status is complete
      return

```

---

---

ALGORITHM 6: addJobs() function

---

```
master.Access.Lock() // reserve access by master thread

defer master.Access.Unlock() // release access

for id, task in range(tasks) do // loop each task

    newId = master.ProcessDetails.latestJobId + id + 1 // compute the next job ID
    master.Jobs[id] = &Job{Id: newId, Task: task, Status: "idle"} // add task
    master.ProcessDetails.latestJobId = newId // update latest job ID

master.ProcessDetails.idleJobsCount += len(tasks) // update idle jobs count
```

---

- *func Insert(), func Select(), func CreateConnection()*

These are made available to interact with the database. The *Insert()* function is used to insert records into the database, the *Select()* function is used to query the database, and the *CreateConnection()* function is applied to create a connection pool with the Postgres database.

- *func MakeDatabase()*

This function loads a given environment, and establishes a connection URL by getting user name, password, host, port, database name and database URL. After setting the correct parameters, the server starts working.

- *struct JobRequest, struct JobResponse, struct InsertRequest, struct InsertResponse, struct SelectRequest and struct SelectResponse*

These are various RPC calls necessary to work with the database and worker threads. Each worker needs to request for a job and record the result of the job. The crawler needs to insert requests and record the result of such insertion, as well as query and get the result of such querying.

- *other structs*
  - *Job: Id (integer), Task (String), Status (String)*

- *Domain*: Name (String), Extension (String), Metadata (\*DomainMetadata)
- *UrlMetadata*: Context (String), Language (String)
- *Url*: Protocol (String), Subdomain (String), DomainName (String), Extension (String), Path (String), Metadata (\*UrlMetadata)
- *ProcessDetails*: latestJobId (int), idleJobsCount (int), doneJobsCount (int), occupiedJobsCount (int), domainMetadataApiRequestCount (int)
- *DomainMetadata*: IpAddress (String), Status (String), Message (String), Continent (String), ContinentCode (String), Country (String), Region (String), RegionName (String), City (String), Zip (String), Lat (String), Lon (String), Timezone (String), Isp (String), As (String), AsName (String), Scrapable (Bool)

The concurrent thread execution takes place with the usage of Goroutines and channels. Goroutines are essentially lightweight threads representing workers. The main thread is called the main Goroutine. To communicate with other threads, channels are used. Data can be written to or read from Gochannels which serve as tunnels for data transfer. Goroutines are created by using the statement *go* in front of executing functions. A channel is created by using a simple *make* statement and *chan* keyword. To prevent shared resources from being mistakenly modified, a mutual exclusion (mutex) lock is implemented. This safeguards against multiple Goroutines accessing and editing data at the same time. That is why, throughout the bits of code that access a common data repository, statements like *Access.Lock()* and *Access.Unlock()* are utilized to reserve access to a single worker for the duration of the functional execution. This is applied in such functions as:

- *RequestJob()* due to access granted to a shared container of jobs,
- *Done()* to check whether all tasks are completed and increment the counter for finished jobs,
- *addJobs()* to add new tasks to the shared container and update their IDs.

Overall, the following criteria were considered during the development process:

- Ensuring the efficiency of the process by using concurrent execution whilst maintaining the integrity of shared resources.
  - Ensuring a proper handling of failed tasks by re-assigning them to available threads.
  - Ensuring system scalability due to the ability to scale up to the maximum number of threads that can be created on the user's local workstation. Hence, there is an upper limit on the concurrency that can be achieved, and limitless scalability is not guaranteed. In a distributed deployment environment, however, the system can be upscaled by increasing the amount of hardware components (i.e., increasing the number of nodes in a cluster, as an example).
- Ensuring the openness of the code base to further improvements by other developers.

## 2.5 Graph Construction

The *networkx* Python library is used to construct graphs. This library is useful because the scale of the data for this study is big but not too enormous as in that scenario massive network power will be

needed. Also, it provides flexibility in implementing various graph structures. Separate classes for directed and undirected graphs are provided for easy usage.

The directed graph is instantiated with the *DiGraph()* class whose nodes and edges are at first empty. Next, nodes are added with *add\_node()* method whose input is the domain currently collected by the web crawler. Edges are created by the *add\_edges()* method applied to the current node and each of its outgoing links. Already visited nodes are marked to disable duplicate efforts. The nodes are collected in a separate file as a list, and the same for edges which gather the “to” and “from” data to establish node-to-node connections. This information can then be directly fed into the graph instantiation using the *read\_edgelist()* method. Weight is a specific edge attribute that carries values needed to differentiate edges by some criteria. As an example, a weight of 5 between nodes 1 and 2 gets assigned by the following piece of code: *g[1][2]['weight'] = 5.0*. The graphs are drawn using the matplotlib interface. For instance, a circular graph can be displayed by calling *draw\_circular()* on the constructed graph matrix. In-degrees of a graph can be computed by using the *in\_degree()* method.

There are several node centralities that can be used to compute important graph metrics:

- *Betweenness centrality*

This metric can be calculated using the *betweenness centrality()* method called on the graph and is indicative of the amount of influence a node has in a network. This centrality measure computes how often a given web page lies on the path to other web pages. Despite the number of in-degree, any node can have a high betweenness centrality if it acts as a broker between its kin. The formula for its calculation includes counting the number of times a given vertex intercepts shortest paths between other nodes (2).

$$\frac{(n - 1) \times (n - 2)}{2} \quad (2)$$

- *Closeness centrality*

This metric can be calculated using the *closeness centrality()* method called on the graph and is indicative of the closeness of a given vertex to others in the network. This centrality measure computes which nodes are better at disseminating information across all vertices. The normalized formula for its calculation involves computing shortest paths between nodes (3). There are several algorithms in place for computing the shortest path between any two given nodes. This includes Dijkstra’s, A-star, Floyd-Warshall and Viterbi search algorithms.

$$\frac{(n - 1)}{\text{sum}(\text{dist}(V_i, V_j)) \text{ for all } j \text{ in nodes}} \quad (3)$$

- *Eigenvector centrality*



This metric can be calculated using the *eigenvector centrality()* method called on the graph and is indicative of the transitive influence of a node across the network. This centrality measure computes eigenvector scores, and high eigenvector scores usually indicate a node connected to other nodes which also have high scores. The formula computes this metric recursively based on the measures computed for previous nodes (4).

$$X_i(t) = \sum_1^j A_{ij} x X_j(t - 1) \quad (4)$$

## 2.6 Page Ranking

The page ranking score can be computed by adopting an already implemented method in the *networkx* library - the *pagerank()* method. The pseudocode is provided in Algorithm 7. First, degrees for each vertex get computed. Then, initial scores and probabilities are set. Afterwards, the initially computed and updated scores are compared with the tolerance level. If the difference cannot be tolerated, the probability scores are updated again until the values converge.

## 2.7 Metadata Collection

There are a few ways to extract the language of any website:

- Relevant HTML tags can be checked, such as `<meta name = "language" content = "... ">`, or the *lang* parameter within `<div>` or `<html>` tags.
- Getting a rather small text part to easily parse to determine the language using Python libraries. For instance, Python's *NLTK difflib* can be used to compare the set of parsed tokens with a particular language's vocabulary to find commonalities or differences.
- Sending a GET request with a text chunk to a free language detector API which replies with a JSON output detailing the highest scored language.

Depending on the availability of relevant tags in a document's HTML structure, either (1) or (2)-(3) will get used.

---

ALGORITHM 7: Page Rank

---

n = number of nodes

```
for edge(u,v) in G: // compute degree of each neighbor
    degree(v) += 1
```

```
for node u in G: // initialize probabilities
    const(u) = (1 -  $\alpha$ ) / degree(u)
    PR_old(u) = 1 / n
```

```
while norm(PR_old - PR_new) > tolerance: // update probabilities if tolerance exceeded
    for node u in G:
        PR_new(u) =  $\alpha$ 
        for node v in pointing to node u:
            PR_new(u) += const(v) x PR_old(v)
```

---

To find the geolocation of an IP address, a GET request can be sent to an open-source API, like *IP2Location* with the extracted IP address of a web page. The IP address can be extracted by using a *ping* function called on the web page or the *nslookup* command. The commands, however, can fail if the web page is not accessible - hence, null values are expected for such instances.

### 3 RESULTS & ANALYSIS

The internal connectivity of the resulting graph is not expected to be strong, and most likely connections over a thousand edges will not be present. The reason for this assumption is low inter-connectivity between web pages except for governmental websites which contain many redirections between each other to point to the services citizens can make use of. Many websites are expected to link to such global pages as Facebook and Twitter. These types of connections can make it easier to find security lacks by tracing which domains are affected in case of an attack on the network of any website. This analysis can also be grouped by providers, or any other metadata item through such machine learning algorithms as  $k$ -means clustering.

A total of circa 13,500 websites are attempted to be crawled, of which 9393 are reachable, and 4,106 are unable to be scraped, and 1 domain is unreachable. The reasons for the inability to crawl some web pages relate to either non-working domains or their development in PHP (Hypertext Preprocessor). The depth level for the program is limited to two, and some websites too long to crawl (e.g., news sites) due to a lot of intra-linkages. This type of pages was excluded from further execution. Overall, the program execution took place for approximately two days. The device used for execution is MacBook Pro M1 with 8-core CPU and 8-core GPU, and 1 TB added storage.

The results presented in this section are masked with random names where necessary to preserve data confidentiality and maintain the rights of each website to privacy and security. Sample results saved in a comma-delimited file are showcased in Figure 9. Each observation contains a source page, a target page, assigned node and edge weight, protocol, parsed subdomain, domain name and extension, full path and collected metadata. Metadata includes such attributes as *context* which stores the HTML document, and *language* which contains information on the natural language of the page.

The data is gathered into source and target columns to enable the creation of a directed graph, resulting in 93,545 unique observations. In terms of protocols, the majority of collected data relate to *https* protocol (circa 72,000 observations) and the rest - to *http* protocol (circa 21,000 observations). There are 1,877 unique subdomains, the majority of which are *www*, *mod* and *bilikfondu*. The number of unique extensions is 10, and the majority ones are *az*, *com*, *org* and *ru*, followed by extensions less in size, such as *net*, *me*, *tr*, *de*, *eu* and *be*.

	Source	Target	NodeWeight	EdgeWeight	Protocol	Subdomain	DomainName	Extension	Path	Metadata
0	000.az	texnar.az	1	1	https	www	texnar	az	/az/muellifer/eysar-ahmedov	{'Context': '\n\n2022 yaş\n\nIT Manage...
1	000.az	texnar.az	1	1	https	www	texnar	az	/az/teqler/IT sergi	{'Context': '\n\ninqisa məlumat\n\ninqisa ...
2	000.az	texnar.az	1	1	https	www	texnar	az	/az/teqler/TV	{'Context': '\n\nApple tvOS üçün de cüzi y...
3	000.az	texnar.az	1	1	https	www	texnar	az	/az/teqler/Apple Pencil	{'Context': '\n\nApple iPadOS 14-ü cüzi ye...
4	000.az	texnar.az	1	1	https	www	texnar	az	/az/teqler/OS	{'Context': '\n\nYeni Mac Studio haqqında ...

Figure 9. Sample of Collected Data

The largest node sizes are not described for confidentiality reasons but some of the relatively important not strategic ones include *zoo.az*, *palitraneews.az*, *extratime.az*, *interyermebel.az* and *liebherr.az*. These results are represented in Table 2, in non-normalized form.

Table 2. Largest Nodes in Web Graph

Source	Node Weight
zoo.az	670
palitraneews.az	469
extratime.az	465
interyermebel.az	424
liebherr.az	415

In terms of target edge weights, there are a lot of hyperlinks outgoing to such websites as governmental, social media websites and *alisuperdeals.com*. This is displayed in Table 3, in non-normalized form with masked names for strategically important websites.

Table 3. Largest Edge Weights in Web Graph

Source	Edge Weight
aaaa.az	15,703
bbbb.com	7,589
cccc.az	7,470
dddd.com	6,177
eeee.com	3,899
ffff.az	3,038
gggg.com	2,568
hhhh.az	1,360
jjjj.com	1,117
alisuperdeals.com	932

The size of graph nodes represents importance scores assigned to each webpage. The importance scores are measured by the PageRank algorithm. Weights are also assigned to edges based on the number of times two given nodes connect with each other. To calculate these weights, the program counts the number of times an outgoing link is referenced. Self-linkages are not accounted for since they do not entail any extra information. The graph is constructed based on those edge weights that are within a standard deviation of 20 or higher across all edges, excluding those edges that fall outside the average range. This is done to make the resultant graph comprehensible visually, as otherwise the program loads too slowly, and the output is nearly impossible to be read. The bird-eye view of the entire graph is provided in Figure 10 and weights about 81 GB, while its more condensed form with only selected edges is in Figure 11.

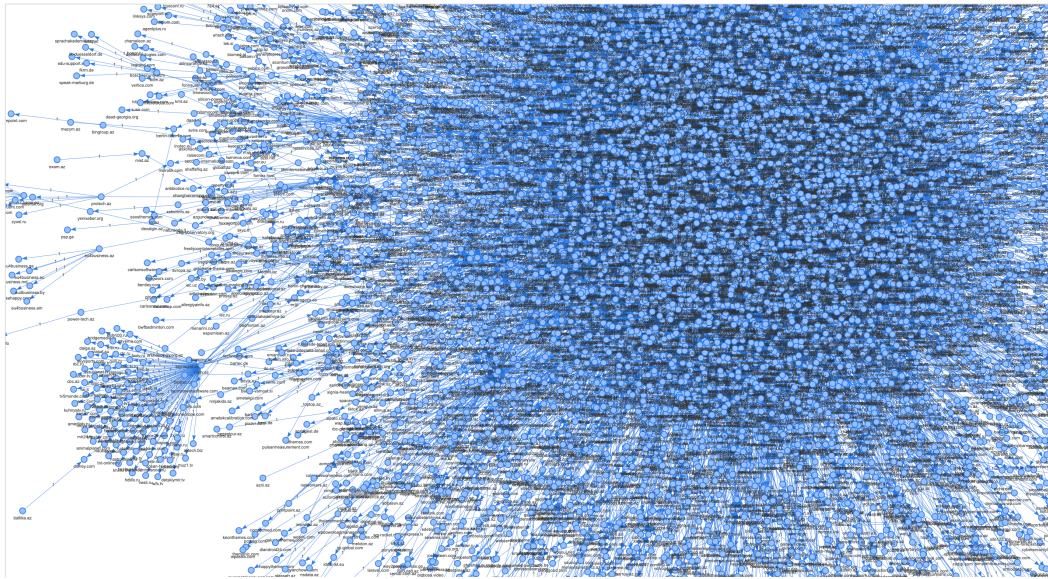


Figure 10. Bird-Eye View of Azerbaijani Web Graph

The results indicate four strongly grouped communities. The tightest groupings happen around government websites, as well as commercial and organizational pages. Also, one can observe that there are indirect linkages to other less tightly connected websites. These include news sites, such as *azertag.az*, and social media pages like *instagram.com* and *youtube.com*.

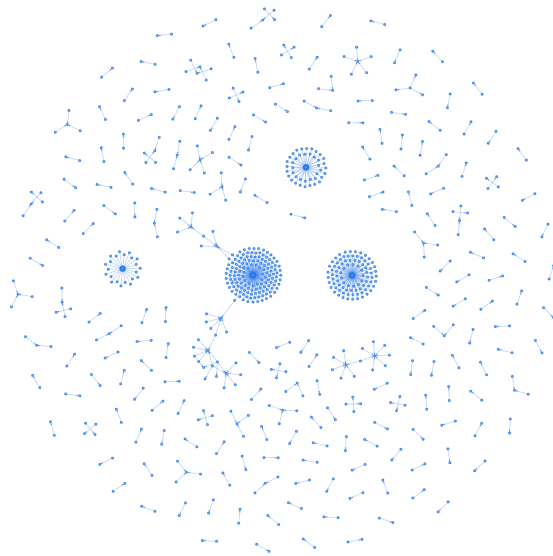


Figure 11. Sample of Higher-Weighted Nodes

Figure 12 displays a sample deep-dive view of the resultant graph. There are two tightly connected groups on the figure. Some websites from the governmental community relate to social media and news sites indirectly. For example, *azertac.az* is connected to governmental websites through a path going through *azertag.az*, which is linked to a social media site and through the latter, linked to a governmental page. There is a whole range of two-way connections, and they happen mostly between the same domains. The connections between *poly.az* and *poly.com*, *dunyaschool.edu.az* and *dunyaschool.az* serve as examples of such bilateral linkages.

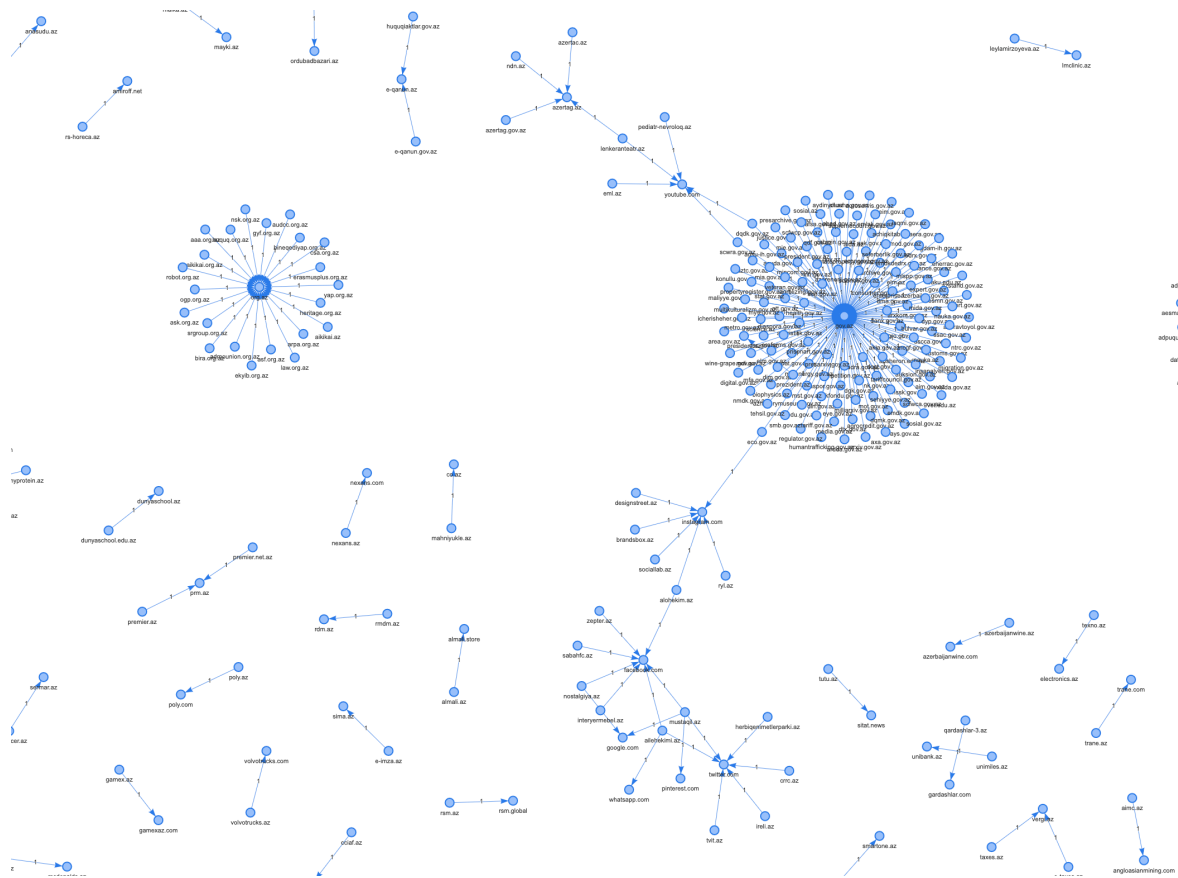


Figure 12. Sample Azerbaijani Web Graph Deep-Dive View

Google’s page ranking algorithm is applied to nodes, and their importance scores are computed. Table 4 displays top five most highly ranked websites, which highly relate to the clusters displayed on previous figures.

Table 4. Top-5 Page Ranking Results

<b>Node</b>	<b>PageRank</b>
aaaa.az	0.11
bbbb.az	0.06
cccc.az	0.02
dddd.az	0.01
eeee.az	0.01

In terms of languages, the most prevalent ones are Azerbaijani, English, Russian and Turkish. The exact counts of websites per top languages are provided in Table 5.

Table 5. Top Languages

<b>Language</b>	<b>Number of Web Pages</b>
az	46539
en	36282
ru	6729
tr	2294
de	251

#### 4 CONCLUSION & FUTURE WORK

To conclude, this study implements a big graph of Azerbaijani web to form a full understanding of the local web space, its most influential players and information flow patterns. This is especially important in the age of technology as Azerbaijan progresses its innovative development and is hence highly susceptible to factors coming outside of local sources. At the same time a by-product of this research is a local web crawler which can be used to collect data from Azerbaijani websites for business or academic purposes.

The implemented web crawling program is written in Golang and executed on a MacBook Pro M1 device for the duration of two days on approximately 13,500 Azerbaijani domains. The program consists of a master thread and several concurrent workers executing their assigned jobs. Workers crawl the provided seed domains and outgoing hyperlinks up until the second tranche. In the process, metadata, such as IP address, geolocation, hosting provider name, language and context are collected. The gathered data is stored in JSON format, in a separate folder for each unique domain. Caching is enabled to prevent multiple downloads of the same page if the program crashes or is stopped for any reason.

The processed observations are gathered into a structured data format to create a directed graph. Node sizes convey the importance scores of websites, while edge sizes represent the strength of linkages in-between. Overall, the resultant big graph is visually tightly connected although there are still some sparse parts remaining. As expected, governmental websites form the most strongly connected component, followed by websites grouped around educational, commercial, and organizational sites. The constructed graph is narrowed down into those communities where edge weights are within the standard deviation range of 20 or higher for the purpose of comprehensibility. Indirect linkages with news and social media sites are present in some tightly connected groups, such as governmental websites. Bilateral connections mostly take place between the same domains, or domains centered around the same context. The most highly ranking nodes are also sorted by their importance scores and provided in a data table. It is important to note that the program implements a depth level of two only for easier and faster processing.

Future work in this space includes the following:

- making the visualization more interactive and less space-consuming such that users can freely filter on the attributes they are interested in;
- clustering graph nodes by the industry to which they belong (e-commerce, government organizations, education, etc.);
- clustering graph nodes by natural language prevalent on their websites;
- clustering graph nodes by hosting service provider locations;
- applying machine learning techniques, and natural language processing, to collected samples from each page to find clusters based on recognized entities or topic names;
- stress testing how the connectivity of local web space breaks in case of severe impact coming from outer sources, and propagating through the graph;
- removing a limit of two-tranche depth level to enable a full computation of the Azerbaijani graph and consider all the available hyperlinks;



- running the crawling algorithm on a more powerful machine for much faster concurrent processing and without depth limitations.

## 5 BIBLIOGRAPHY

- [1] Maurice de Kunder. 2022. The Size of the World Wide Web (The Internet). Retrieved from <https://www.worldwidewebsize.com>.
- [2] The World Bank. 2019. Accelerating the Growth of High-Speed Internet Services in Azerbaijan. Retrieved from <https://www.worldbank.org/en/country/azerbaijan/publication/broadband-in-azerbaijan>.
- [3] Natalia Spinu. 2020. Azerbaijan Cybersecurity Governance Assessment. Geneva Center for Security Sector Governance, DCAF.
- [4] The World Bank. 2019. Individuals Using the Internet (% of Population) – Azerbaijan. Retrieved from <https://data.worldbank.org/indicator/IT.NET.USER.ZS?locations=AZ>.
- [5] Datareportal. 2021. Digital 2021: Azerbaijan. Retrieved from <https://datareportal.com/reports/digital-2021-azerbaijan>.
- [6] Giorgi Lomsadze. 2015. Azerbaijan Plans National Internet-Search Engine. Eurasianet. Retrieved from <https://eurasianet.org/azerbaijan-plans-national-internet-search-engine>.
- [7] Seyed M. Mirtaheri, Mustafa Emre Dinçtürk, Salman Hooshmand, Gregor V. Bochmann, Guy-Vincent Jourdan, Iosif Viorel Onut. 2014. A Brief History of Web Crawlers. Proc. of CASCON 2013, Toronto, Nov. 2013.
- [8] Mini Singh Ahuja, Jatinder Singh, Var nica. 2014. Web Crawler: Extracting the Web Data. International Journal of Computer Trends & Technology (IJCTT), vol 13, no 3.
- [9] S. Sharma and P. Gupta. 2015. The anatomy of web crawlers. International Conference on Computing, Communication & Automation, 2015, pp. 849-853, doi: 10.1109/CCA.2015.7148493.
- [10] Krishna Bharat, Andrei Broder, Monika Henzinger, Puneet Kumar, Suresh Venkatasubramanian. 1998. The Connectivity Server: fast access to linkage information on the Web. Computer Networks and ISDN Systems, Volume 30, Issues 1–7, 1998, Pages 469-477, ISSN 0169-7552, [https://doi.org/10.1016/S0169-7552\(98\)80047-0](https://doi.org/10.1016/S0169-7552(98)80047-0).
- [11] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, Janet Wiener. 2003. Graph Structure in the Web. Retrieved from <https://www.cis.upenn.edu/~mkearns/teaching/NetworkedLife/broder.pdf>.
- [12] David Gibson, Jon Kleinberg, Prabhakar Raghavan. 1998. Inferring Web Communities from Link Topology. Retrieved from <https://www.cs.cornell.edu/home/kleinber/ht98-comm.pdf>.
- [13] Bhatt, Dvijesh & Vyas, Daiwat & Pandya, Sharnil. 2020. Focused Web Crawler. Advances in Computer Science and Information Technology (ACSIT) Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 11.
- [14] Linxuan Yu, Yeli Li, Qingtao Zeng, Yanxiong Sun, Yuning Bian and Wei He. 2020. Summary of Web Crawler Technology Research. Journal of Physics: Conference Series, Volume 1449.
- [15] Kevin S. McCurley. 2009. Incremental Crawling. Google Research.
- [16] L. Saoudi, A.Boukerram, S.Mhamedi. 2015. A New Hidden Web Crawling Approach. International Journal of Advanced Computer Science and Applications, Vol. 6, No. 10, 2015.
- [17] Gu Yunhua, Shen Shu, Zheng Guansheng. 2011. Application of NoSQL Database in Web Crawling. International Journal of Digital Content Technology and its Applications 5(6):261-266.
- [18] Sheila Eka Putri, Tulus, Normalina Napitupulu. 2011. Implementation and Analysis of Depth-First Search (DFS) Algorithm for Finding The Longest Path. International Seminar on Operational Research (InteriOR).
- [19] Jason Holdsworth. 1999. The Nature of Breadth-First Search. School of Computer Science, Mathematics, and Physics, James Cook University, Australia.
- [20] Rina Dechter, Judea Pearl. 1985. Generalized best-first search strategies and the optimality of A\*. Journal of the ACM (JACM) 32, 505-536.

- [21] Alon Y. Levy, Anand Rajaraman, Joann J. Ordille. 1996. Querying Heterogeneous Information Sources Using Source Descriptions. Proceedings of the 22<sup>nd</sup> VLDB Conference, India.
- [22] Ippili Akarsh, Ravi Maithrey Regulagedda. 2021. Parallelization of Web Crawler with Multithreading and Natural Language Processing. International Research Journal of Engineering and Technology (IRJET), Vol 8, Issue 10.
- [23] Arvind K Sharma, Vandana Shrivastava, Harvir Singh. 2020. Experimental performance analysis of web crawlers using single and Multi-Threaded web crawling and indexing algorithm for the application of smart web contents. Materials Today: Proceedings 37.
- [24] Kartik Kumar Perisetla. 2012. Mutual Exclusion Principle for Multithreaded Web Crawlers. International Journal of Advanced Computer Science and Applications, Vol 3, No 9.
- [25] Gautam Pant, Padmini Srinivasan, and Filippo Menczer. 2004. Crawling the Web. Web Dynamics.
- [26] Yasin Kansu, Begum Mutlu, Anil Utku, and M. Ali Akcayol. 2017. An Efficient Multi-Threaded Web Crawler Using HashMaps. Journal of Advances in Computer Networks, Vol. 5, No. 2.
- [27] Rada Mihalcea. 2004. Graph-based Ranking Algorithms for Sentence Extraction, Applied to Text Summarization. DOI:10.3115/1219044.1219064.
- [28] Raluca Remus. 2009. Lecture 4 - HITS Algorithm - Hubs and Authorities on the Internet. Cornell University.
- [29] P. Jean-Jacques Herings, Gerard van der Laan, Dolf Talman. 2005. The positional power of nodes in digraphs. Soc Choice Welfare (2005) 24: 439–454.
- [30] Zlata Verzhbitskaia. 2021. The Past, Present & Future of Google PageRank. Retrieved from <https://www.link-assistant.com/news/google-pagerank-algorithm.html>.
- [31] Mikalai Krapivin and Maurizio Marchese. 2008. Focused Page Rank in Scientific Papers Ranking. Digital Libraries: Universal and Ubiquitous Access to Information, 11th International Conference on Asian Digital Libraries, ICADL 2008, Bali, Indonesia, December 2-5, 2008.
- [32] M. Wang, J. Kang, N. Cao, Y. Xia, W. Fan and H. Tong. 2021. Graph Ranking Auditing: Problem Definition and Fast Solutions. In IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 10, pp. 3366-3380, 1 Oct. 2021, doi: 10.1109/TKDE.2020.2969415.
- [33] Jian Kang, Scott Freitas, Haichao Yu, Yinglong Xia, Nan Cao, and Hanghang Tong. 2018. X-Rank: Explainable Ranking in Complex Multi-Layered Networks. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18). Association for Computing Machinery, New York, NY, USA, 1959–1962. DOI:<https://doi.org/10.1145/3269206.3269224>.