

SMART DISTRIBUTED FOCUSED WEB CRAWLER FOR HIDDEN WEB

A Thesis

Submitted in partial fulfillment of the requirements for the
award of the degree of

DOCTOR OF PHILOSOPHY

in

(Computer Science and Engineering)

By

Sawroop Kaur

(11412869)

Supervised By

Name of Supervisor

(Dr. Aman Singh)

Name of Co- Supervisor

(Dr G.Geetha)



L OVELY
P ROFESSIONAL
U NIVERSITY

Transforming Education Transforming India

LOVELY PROFESSIONAL UNIVERSITY

PUNJAB

2021

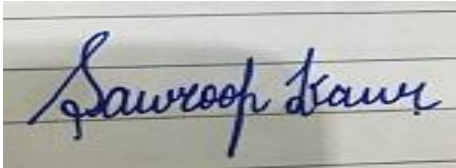
DECLARATION

I hereby declare that the thesis entitled "Smart Distributed Focused Web Crawler for Hidden Web" is submitted by me for the Degree of Doctor of Philosophy in Computer Science and Engineering is the result of my original and independent research work carried out under the guidance of Dr Aman Singh, Associate Professor, Lovely Professional University and Dr G.Geetha, CEO Advanced Computing Society. It has not been submitted for the award of any degree, diploma, associateship, fellowship of any University or Institution.

Place: Amritsar

Date: 22-06-2021

Signature of the Candidate

A photograph of a handwritten signature in blue ink on lined paper. The signature reads "Sawroop Kumar".

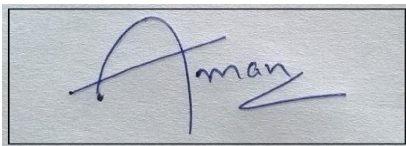
CERTIFICATE

This thesis entitled "Smart Distributed Focused Web crawler for Hidden Web" submitted by Sawroop kaur of Lovely Professional University is a record of bona fide research work done by her and it has not been submitted for the award of any degree, diploma, associateship, fellowship of any University/Institution.

Place:

Date:

Signature of the Guide

A handwritten signature in blue ink, appearing to read "Aman", enclosed within a rectangular border.

Signature of the Co-Guide

A handwritten signature in purple ink, appearing to read "Geetha", written in a cursive style.

ACKNOWLEDGEMENT

I am thankful to almighty for making things possible at the right time. I owe my success to my supervisors Dr Aman Singh , Dr. G. Geetha, and my Mother. Their continuous support and patience have brought me to this time. I am deeply influenced by my supervisors way of guidance, and sincerely thankful for standing by side in all the problems I have faced. I wish I could acknowledge your kindness and big heartedness in terms of candid interactions, guidance and inspiring advices. Reaching destination was not possible without you. This work would have been a wild goose chase without your guidance.

I would like to thank my parents, my siblings and my husband for their continuous support. My special thanks to my friends Divya Anand, Raj Kamal kaur and Rekha Chaudhary and for their encouragement.

Abstract

From its origin to the present date World Wide Web is still evolving. It is a collection of huge repositories of interlinked web pages. It can be broadly classified into surface and hidden web. The fraction of visible data or surface web is very less as compared to its hidden counterpart Hidden web. Most of the data on the web is hidden from the users, but not the machines. Machines can read the data but the web is designed for visually navigating, finding, clicking and downloading files. And machines navigate by following logical rules. Humans are interested in interactive content while machines are structure and logic dependent. So hidden web data is hidden from the human eye, not machines. This data is hidden under the forms. So, for filling a form and retrieving the information, the right tool is required. From available methods, developing a hidden web crawler is a righteous way.

The hidden web has better quality and quantity of data than the surface web. Surface web engines can crawl and index a large number of web pages daily to give a great start for information retrieval. Yet may not be adequate for complex data queries that require relevant classification of a large volume of results. Hidden web crawlers could be the result of it. Moreover, as the size of the web is increasing, distributed web crawling is the call of duty. Crawling data from hidden web consist of surfacing and virtual integration. But information from underlying databases can be curated only when the HTML form is filled with appropriate values. The general web crawler does not make difference between a webpage with a form and a web page without a form. It simply discards the web page or a part of the web page if a form is encountered. General search engines are not endowed with the ability to go beyond query interfaces. So this task is performed by hidden web crawlers. And in hidden web crawling to crawl data at large scale the crawlers are required to be efficient. It was found that in this area has no other focused crawler that can efficiently work in distributed manner.

This work comprises four objectives that are based on developing a novel architecture of distributed hidden web crawler for the focused web. The entry to the hidden web is not based on forms only but also rules are developed as rejection criteria. Crawling is based on three stages. Crawling is initiated in stage one, followed by ranking and classification. Hidden web documents have slim chances to be retrieved. The enhanced priority-based ranking algorithm has tackled the problem when the document is missed if it has a low rank. This algorithm is a triplet formula to calculate the rank of the website. By including site frequency, the documents which have a low rank earlier can have a high rank. By ranking the website, the crawler minimises the number of visits and maximize the number of websites with embedded forms. In the second stage, the links extracted from the first stage are ranked and classified. The third stage extracts the underlying content. The duplication detection technique of simhash is

implemented on the Redis server to improve the efficiency of the crawler. The resultant URLs can be for harvesting and indexing in search engines. The approach not only shows a better coverage rate but also in comparison with the existing approach harvest rate is high. The system can detect more status codes as compared to existing systems. A goal of crawler is to find maximum searchable forms in minimum visits,

TABLE OF CONTENTS

DECLARATION.....	ii
CERTIFICATE	iii
ACKNOWLEDGEMENT	iv
ABSTRACT	v
TABLE OF CONTENTS.....	vii-xiv
CHAPTER 1: INTRODUCTION	1-17
1.1 Information retrieval.....	1
1.2 Retrieval model and ranking.....	2
1.3 Models of Information retrieval.....	2
1.4 Search methods.....	3
1.5 Web Crawler.....	3
1.5.1 Working of web crawler.....	4
1.5.2 Types of crawling techniques.....	5
1.5.2.1 Generic Crawling.....	5
1.5.2.2 Focused Crawling.....	5
1.5.2.3 Intelligent Crawling.....	6
1.6 Classification of World Wide Web.....	7
1.6.1 Surface web.....	7
1.6.2 Hidden web.....	9
1.6.2.1 Steps of Hidden web crawling... ..	11
1.7 Distributed web crawling	16
1.8 Performance metrics.....	17
Motivation	19
CHAPTER 2: LITERATURE REVIEW	20-36

2.1	General or broad search crawlers.....	20
2.2	Preferential crawlers.....	20
2.2.1	Topical crawler	20
2.2.2	Focused crawlers.....	20
2.2.2.1	Semantic similarity	21
2.2.2.2	Machine learning and adaptive.....	21
2.2.2.3	Form focused	22
2.2.2.4	Context /link based	22
2.2.2.5	Application Based.....	22
2.2.2.6	Miscellaneous categories.....	22
2.3	Forum crawler	23
2.4	Mobile crawler.....	23
2.5	Continuous and Incremental crawler.....	23
2.6	Hidden web crawler.....	25
2.7	Distributed Crawlers.....	29
2.8	Focused Hidden web crawlers	36
2.9	Focused Distributed web crawlers.....	36
2.10	Distributed Hidden web crawlers.....	36
	Research Gap.....	42
	Problem formulation.....	42
CHAPTER 3 : Novel Architecture of Smart Distributed Hidden web crawler.....		43-61
3.1	Proposed Architecture.....	47
3.2	Preprocessing of URLs.....	50
3.3	Weight calculation of terms.....	51
3.4	Learning.....	54

3.5	Ranking.....	55
3.6	Domain Classification.....	55
3.7	Form Structure and extraction.....	56
3.8	Form response analysis.....	57
3.9	Query Probing.....	58
3.10	Form Submission.....	58
3.11	Stopping Criteria.....	58
3.12	Assumption and threshold.....	59
3.13	Distribution.....	59
3.14	Job Scheduling.....	60
3.15	Implementation of Scrapy , Redis and Beautiful soup.....	61

CHAPTER 4:Creating algorithms for proposed crawlers.....64-86

4.1	Dispatcher Algorithm.....	64
4.2	Parsing Algorithm.....	64
4.3	Algorithm for rejection criteria.....	65
4.4	Algorithm of Crawl Supervisor.....	66
4.5	Algorithm of Learning.....	66
4.6	Algorithm of Similarity	67
4.7	Algorithm of Ranking.....	67
4.8	Building Naïve Bayes Classifier.....	68
4.9	Building SVM classifier	69
4.10	Algorithm to find similar domains.....	71
4.11	Algorithm for pre query	72

4.12	Algorithm for post query	72
4.13	Form identification and analysis	72
4.14	Form structure extraction and form filling.....	73
4.15	Job Scheduling algorithm.....	73
4.16	Configuration.....	73
4.17	Evaluation.....	74
4.18	Experimental setup	74
4.19	Path learning	83
4.20	Application of crawler as an approach for atmospheric emission.....	84
4.21	Comparative advantages.....	86
CHAPTER 5: Comparison of results.....		87-98
5.1	Comparison in terms of performance issues.....	93
5.2	Comparison of Mercator.....	95
5.3	Constraints and barriers.....	98
CHAPTER 5: Conclusion and Future scope.....		99-102
6.1	Conclusion	100
6.2	Future Scope.....	102

LIST OF FIGURES

1. Information retrieval activities.....	2
2. Working of web crawler.....	4
3. Components of Generic crawler	5
4. Working of focused web crawler.....	6
5. Classification of World Wide Web.....	7
6. Ways to access the hidden web.....	13
7. Shows the website with search interface.....	15
8. Results after the search button is clicked.....	15
9. Result of query of titan store Amritsar.....	16
10. Steps wise difference between generic and Hidden web crawling.....	16
11. Working of focused crawler with classifier.....	24
12. Shows reported the web crawlers and its types	37
13. Link extraction from seed urls	43
14. Shows the extraction of target page.....	45
15. Architecture of proposed crawler as a single entity focused crawler	48
16. Analogy of use of single URL.....	49
17. Computation of finger print	53
18. Formation of sumtable	53
19. Diagrammatic view of rules for searchable forms.....	56
20. Distribution of proposed crawler based on Redis server.....	59
21. Job scheduling in proposed crawler.....	61
22. Show the detailed working of proposed crawler	63
23. Comparison of Precision for varied values of K in KNN.....	79
24. Comparison of Recall for varied values of K in KNN.....	80

25. Comparison of macro average and weighted average for precision, recall and F1 in KNN	80
26. Comparison of F1 using SVM for variation of 20% to 50% of testing data	81
27. Comparison of precision for 20% to 50% of testing in SVM.....	81
28. Computation of precision for 20% to 50% of testing data for K=5.....	81
29. Computation of recall for 20% to 50% of testing data for k=5.....	81
30. Comparison of F1 score for 20% to 50 % of testing data	82
31. Comparison of Accuracy for KNN and SVM.....	82
32. Comparison of accuracy for varied values of k.....	82
33. Comparison of FC, FFC, EEFC and proposed crawler in terms of harvest rate	83
34. Depth of crawl vs percentage of forms found at particular depth.....	84
35. Comparison of PM 2.5 in cities of Punjab	86
36. Comparison of PM 10 in cities of Punjab.....	86
37. Comparison of coverage for GET and POST methods.....	89
38. Comparison of domains for number of document and new document captured using GET method.....	91
39. Comparison of domains for number of document and new document captured using POST method.....	92
40. Coverage of crawler in terms of forms submitted.....	92

LIST OF TABLES

1. Comparison of techniques in terms of heuristic and machine learning, pre query and post query, and features of forms.....	25
2. Comparison of research in terms of focused crawling and classification algorithm.....	26
3. Comparison of hidden web crawlers.....	28
4. Comparison of hidden web crawlers.....	28
5. The comparison of existing distributed web crawlers based on their performance measures..	31
6. The comparison of web crawlers based on performance attributes.....	32
7. Comparison of distributed web crawlers	33
8. Open research problems associated with different types of web crawlers.....	37
9. Categories of web pages encountered by crawler.....	44
10. Contents of repository.....	57
11. Search term description as per domains.....	72
12. Status code and their description.....	75
13. Comparison of Precision, Recall and F1 score for varied values of K in KNN.....	76
14. Comparison of Precision, Recall and F1 score for variation of 20% to 50% of testing data in SVM.....	77
15. Comparison of Precision, Recall and F1 score for variation of 20% to 50% of testing data for k=2 in KNN.....	78
16. Comparison of Precision, Recall and F1 score for variation of 20% to 50% of testing data for k=5 in KNN.....	79
17. Comparison of accuracy for KNN and SVM.....	79
18. Comparison of running time and number of searchable form.....	87
19. Shows the number of forms retrieved per domain	88
20. Shows the number of forms submitted using the GET method	88
21. Shows the number of forms submitted using the POST method	89
22. Comparison of GET and POST method w.r.t number of documents per domains vs new document captured.....	90
23. Comparison of proposed crawler with hidden web crawlers.....	93

24. Comparison of Proposed crawler with other technologies based on forms features.....	96
25. Comparison of proposed crawler with distributed hidden web crawlers.....	97
26. Comparison of proposed crawler in terms of coverage.....	98

CHAPTER 1

INTRODUCTION

1.1 Information Retrieval

Information organization dates back to the third century B.C. and alphabetization was probably the oldest one devised by the Greeks. Tables of content were found in Greek Scrolls in the second century. It was first considered as the first Information retrieval (IR) system. In Ancient times papyrus scrolls were used by Greeks and Romans to store information. Some of the systems had used tags that consist of a summary. This was done to save time for searching [1].

The first computer-based searching was implemented in the 1940s. The Cornell SMART System was the first computerised repository developed in the 1960s. Before the birth of the World Wide Web, IR systems were used by expert librarians as reference retrieval systems. The birth of the World Wide Web has brought the revolution in the methods of storing, accessing and searching of collections. As an academic and research discipline, IR has been defined in various ways [2].

IR in today's context can be defined as a process of searching, exploring and discovering the information from the repository, to satisfy the information needs of the users. An information retrieval system is a combination of content, computer hardware and software. Computer hardware that stores the content is called a collection, database or in terms of modern web phrasing, it is called a site. Content is made up of information units, which may themselves be a webpage of a website or a book section [3],[4]. Computer software processes the content or retrieves the content from the collection. Items in the web-based collection are web pages. The collection of pages is called a website. Users query the system for its information needs. It is retrieved by matching metadata [5].

The most fitting analogy is that suppose if a user has an information requirement. Initially, it is vague. And it is required to be expressed into request. The request here is a search statement or query. However, the information is stored in databases. As the databases are the most valuable resource, but need to be found first to give relevant results. To be found, they must be indexed. The challenge for IR is to provide a good match between query and database to be found. If these two are correct this meant that the information is correct. Activities of information retrieval systems are shown in Figure 1. Broadly this process consists of indexing, parsing, matching, ranking, and query modification. These processes are commonly a part of search engines.

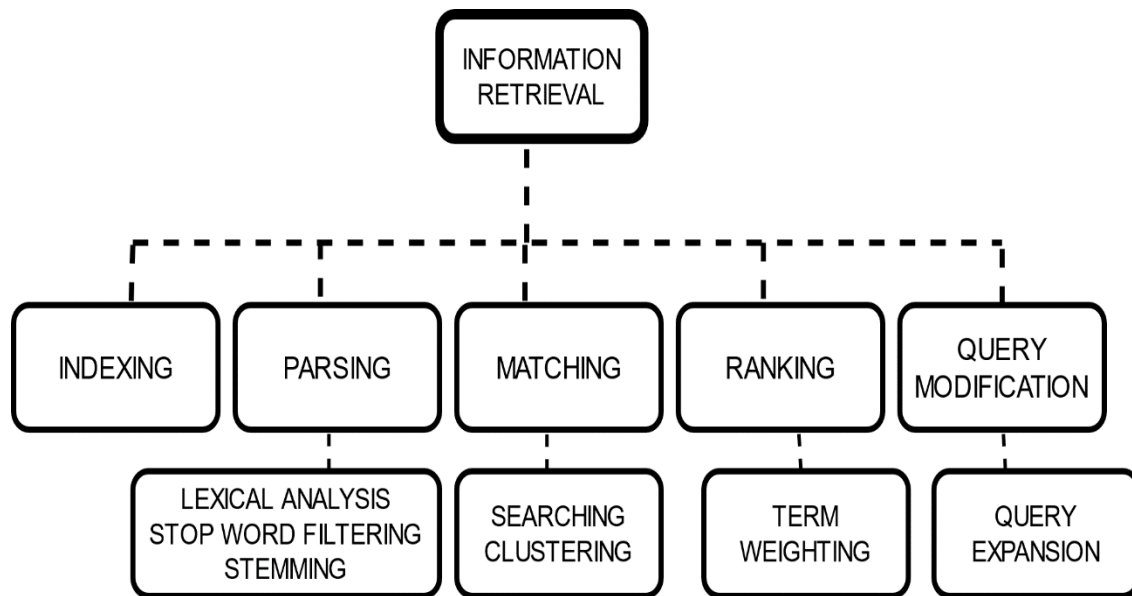


Figure 1: Information retrieval activities

1.2 Retrieval Model and Ranking

Information retrieval is the process of matching the query against the information. An index is an optimised data structure that is built on top of the information objects. It allows faster access to the search process. The indexer tokenises the text, removes words with little semantic value and indexes them for the search. The same is done for the query as well. The relevance of the document is a subjective matter. It is different to the different users. A strong retrieval strategy and ranking can bring the best results in IR. In the context of IR, the retrieval strategy model is an algorithm that takes a query and a set of documents. It assigns a similarity measure between the query and each document. This similarity represents relevance to the user query. Documents are then ranked based on their similarity to the query, and presented to the user. This process can be repeated and the query can be modified.

1.3 Models of Information retrieval

Following are the three main models of information retrieval systems.

- The Boolean model: This model uses an exact strategy to classify documents as relevant and non-relevant.
- The Vector Space Model: Queries and documents are mapped to the Vector Space model. It uses spatial distance as a similarity measure. The document and query are considered vectors. Most of the time the similarity measure is the cosine angle between them.
- The probabilistic model: It estimates document relevance as a probability using user feedback for iterative improvement.

1.4 Search methods

A broad classification of search methods includes Keyword search, Metasearch, Semantic search and Web search.

- **Keyword search:** Keyword search is another way of querying the database. It is easy as it requires only some keywords relevant to the desired data. Most internet users are acquainted with it.
- **Metasearch:** Meta search engines bring the results by accessing multiple search engines in a unified way. The underlying principle of this type of search is to access multiply data on the fly. Metasearch is very much similar to the federated search as these two words are most of the time used synonymously.
- **Semantic search:** Combining the search and semantics have given birth to semantic search. It helps in improving the accuracy of the search by understanding the semantics of the search.
- **Web search:** According to the Kobayashi and Takeda (2000) survey report, it is claimed that 85% of the users rely on search engines to find information. Two thirds to three-quarters use the web as their primary source of information while two-thirds to three-quarters were unable to get the information they want. We are living in the modern age of the web. Search engines play a prominent role in our lives. If we see the search engine statistics, 93 per cent of the web traffic is generated by search engines. And Google processes 2 trillion web searches a year. As the size of the web is increasing, information seeking is becoming complex. It is not confined to web search but search engines play an important role in this. One important tool of the search engine is a web crawler.

According to authors in [6], web search and information retrieval are different. From the spectrum of information retrieval, web search is an important part but not a whole. Queries are used to extract diverse information and web search engines are not effective for all types of queries.

1.5 Web crawler

A web crawler component of web search engines is a system that downloads web pages in bulk, figure (2) shows its working. Web crawlers are used for multiple purposes [7]. These systems first collect a corpus of web pages then index these web pages. When a user issues queries, from the index of the web pages, matched webpages are retrieved [8]. The nature of the web is dynamic. Web pages

are updated frequently and repeatedly. This changing web environment has left a major effect on the way crawlers have evolved. In recent times the scalability and dynamic nature of the web had made

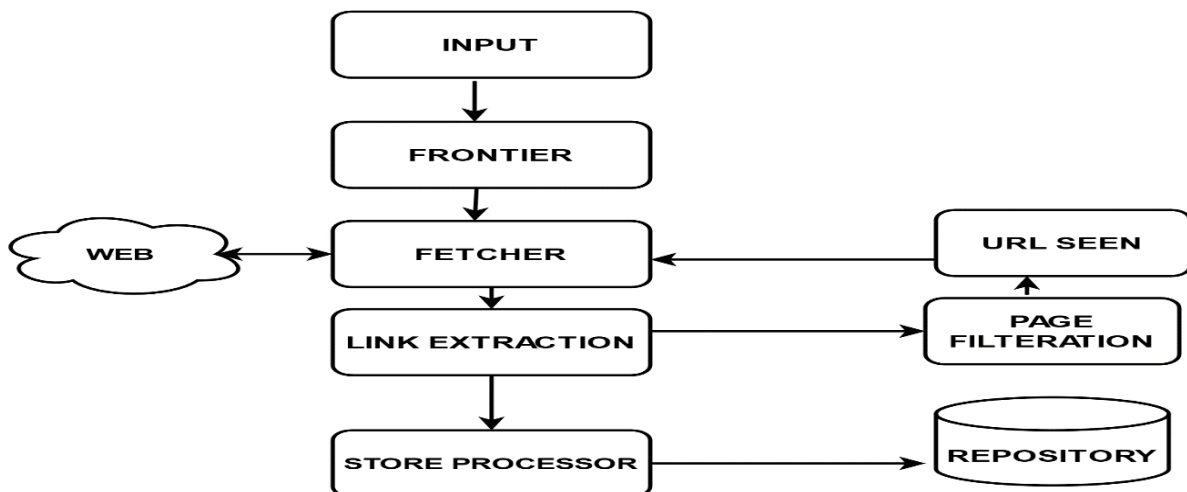


Figure 2 : Working of a web Crawler

the crawling process more difficult. The conduct of any web crawler is the result of a blend of the following strategies:

1. The selection policy states which pages are to be download [2].
2. Re-visit policy states when to revisit web pages to check changes in case if pages are refreshed [9].
3. Politeness policy states help in avoiding overloading of web sites [10].
4. Parallelization policy brings coordination in distributed web crawlers [11].

1.5.1 The working of web crawler

The web crawler selects one URL from the seed set to download the connected web pages, and extract the URL of a web page. These URLs are added to the fetcher component [8]. If not discovered earlier, it is added to the frontier. Implementation of a crawler on a big scale is complex. At large scale policies discussed earlier plus robot .txt standard is to be followed during crawling [12]. As shown in figure 2, each component plays an important part. The fetcher is used to fetch the requested web pages. Link extraction module, extract all the links present on the web pages. Page filtration module based on any criteria either keep the pages or discard them. URL seen section decide if the page has been visited earlier or not. The crawler either start crawling either randomly like generic crawling or collect as many pages as it can, otherwise follow some strict rules of crawling i.e focused or intelligent crawling. The following section discusses the three main crawling techniques.

1.5.2 Types of crawling techniques:

1.5.2.1 **Generic Crawling:** Generic crawling is the term used for crawlers using breadth-first search for covering a large number of pages, the pages collected could be irrelevant [3]. The goal of the web crawler in generic crawling is to visit as many pages as it can. And extract the maximum number of hyperlinks. The crawler keeps updating its index. The working of the generic crawler is shown in figure 3.

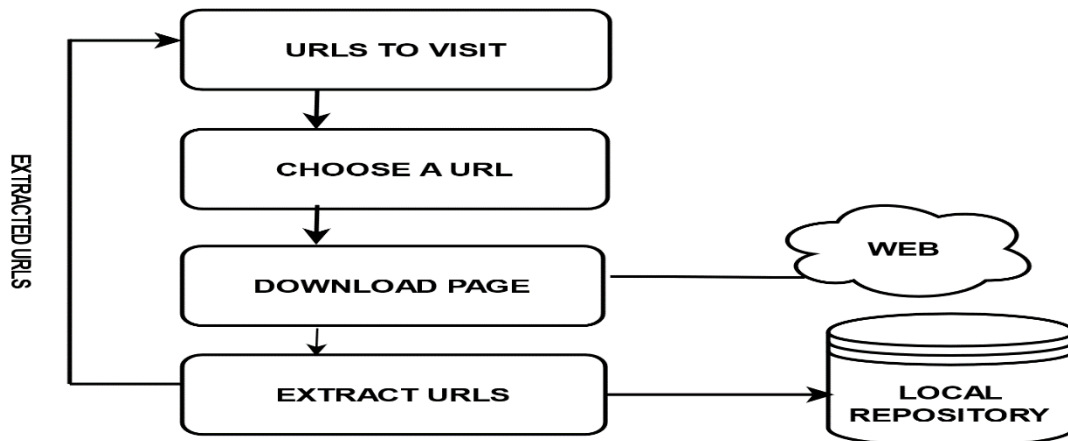


Figure 3: Components of Generic crawler

1. Choose the first URL from the list.
2. Download the corresponding web pages at a local site.
3. Extract the URLs, put the URLs in the list.
4. Follow steps 1,2 and 3, until no URL is left in the list.

In this category of crawling, the crawlers have no focus, it crawls and index everything that comes in its way, either its webpage, image or video etc [13]. When the focus of crawl or direction is expected from the crawlers, focused crawlers are used.

1.5.2.2 Focused Crawling

Focused crawling is meant to crawl relevant pages from a pre-defined set of topics. Consideration is given only to relevant links and discarding irrelevant links [14],[15],[16],[17], [18]. A focused crawler gives priority to the URLs which have a high probability of user interest. This type of crawler aims to selectively search for web pages that are similar to a set of predefined topics [17]–[22]. These crawlers unlike generic crawlers, do not search all the links. It can be said that focused crawlers are the specific tools to crawl topic-specific information. These classifier based crawlers also require negative examples. Crawling algorithms for this approach are divided into two type's

algorithms [19] with background knowledge or without background knowledge [23]. Every focused crawler has at least three components.

- Classifier: The classifier decides on the relevant pages to include potential links to the frontier.
- Distiller: The distiller identifies the home pages that point to topic relevant pages to decide the priority of URL's that are to be visited.
- Crawler: This module fetches the web pages using the list of pages provided by the distiller.

Figure 4 shows the components of a focused web crawler.

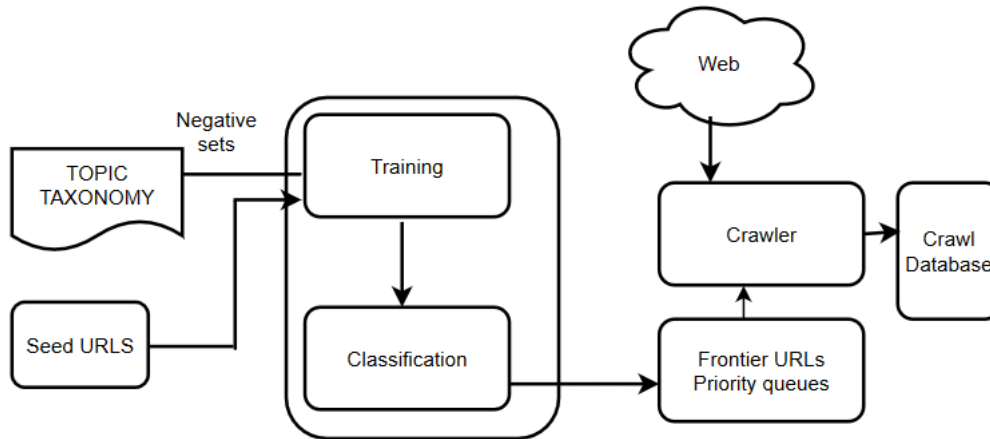


Figure 4: Components of focused web crawler

The focused crawler has the following advantages as compared to the other crawlers:

- The focused crawler steadily acquires the relevant pages while other crawlers easily lose their way, even though they start from the same seed set [24].
- It can discover valuable web pages that are many links away from the seed set, and on the other hand, prune millions of web pages that may lie within the same radius. In this way collection of web documents on specific topics is of high quality [202].
- It can also identify regions of the web that are dynamic or grow more as compared to that are relatively static [202].

1.5.2.3 Intelligent crawling

The Ultimate goal of any web crawler is to collect and process the web pages. The intelligent crawlers not only work on the similarity and priority of URLs, but they also undertake the semantics as well [25]. Semantic crawlers and ontology-based crawlers fall in this category. Intelligent crawlers attempt to give nearest to desired result [26]. Their search methodology is to retrieve information and have three main steps: identifying semantic relationships between table cells; converting tables into

data in the form of a database; retrieving objective data by query languages [21]. These crawlers also use genetic rules and priority queues.

1.6 Classification of World Wide Web

From the time of its invention in 1989 to till date, the World Wide Web is expanding its size. Indexing all the pages of the World Wide Web is difficult. But the formation of an index is vital for the quality of results. Crawler gathers the information to be indexed. Search engines build indexes either manually or automatically. The vast majority of indexing is automatic. Google holds the largest index system on World Wide Web. As shown in figure 5, WWW can be broadly classified into surface web and hidden web.

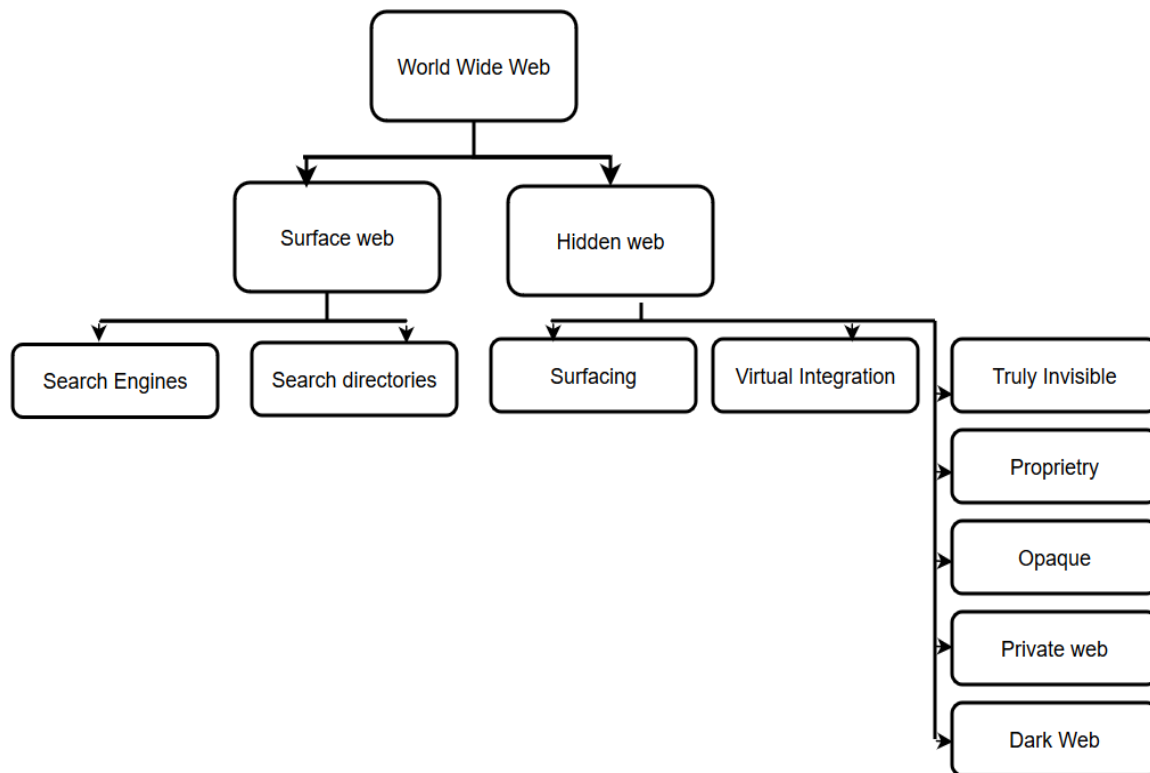


Figure 5: Classification of World Wide Web

1.6.1 Surface Web

The web that does not deal with retrieving data using FORM submission is called surface web. It is also called visible web or indexable web. It is visible to all users. Web crawler despite the use of different techniques- fetch and index the web pages. The most widely used search engine Google is surface web-based. Users can search anything by posing a query or writing any keyword. Google,

Wikipedia and Bing hold the major space in surface web crawling. Search engines are required to be powerful enough because they have to cater for the needs of every user.

But it is interesting to know that surface web index only 16% of the total online information [27]. **It is widely used even if its share is less.** The reason for this is users search patterns in not specialised. Along with search engines, the surface web consists of web directories. Search engines have been there for a long time. Their configurations depend on the application. Web search engines like google and yahoo can crawl terabytes of data. Millions of the queries are responded to in sub-seconds. There are different types of search engines

Crawler Based search engines: Search engines that have a crawler to find and download the web page are called crawler-based search engines. The loop of find, download and analyse web page keep continue for this type of S.E. When the user poses the query, S.E check its database of the webpage to retrieve the best possible match. This type of S.E always finds the new pages and changes in a web page to update its databases. Google and Yahoo are two examples of crawler-based search engines.

Web directory: It is an online catalogue of websites. Directories store the information on a hierarchal basis. A directory has a label to its theme. The label help identifying the subject. It has a root directory that orders the web pages **into a subject and sub-subject-specific hierarchal order.** A user has to search further into it to find suitable and relevant information. For example, if a directory has label culture, and the user has to find the poetry. Then the search would be ‘country’/ ‘India’/ ‘literature’/ ‘poetry’. Though useful **but users may find spending more time finding the information. The user might not have a crisp idea about his search.** Many portals are based on web directories to start a search point for browsing. The best point of directories is that they are designed by human experts. Most existing web directories were created manually by human specialists, putting lots of effort into it. Yahoo has one of the most famous directories widely used for general search as well as research purposes.

Hybrid search engines: This category is again occupied by Google and Yahoo, as they have crawlers as well as directories.

Meta search-based search engines: This type of search engine retrieve the results from more than one search engine. Results are combined to make a listing. Dogpile and **Metacrawler** are two famous meta-search engines.

Specialized search engines: Suppose if a search engine is used to search only shopping-related data, like yahoo shopping, searchnz etc are specialised search engines

1.6.2 Hidden web

World Wide Web and corporate intranets make the information access direct and easy for the organizations. But on the web, not all the information is democratized. A person usually faces difficulty in navigating the hyperlinks on the web. The response lies in hands of search engines. There is always competition between the search engines companies to create the optimized index. Information tools and techniques have reached the point where quick information is being made available. However, the deeper solutions can't be reached by developing a "one search engine serve all". Deliberately much of the information is hidden. Indeed, its value lies in the degree to which it is hidden or not easily available.

For example, if a person is interested in a trade of ABC Motor corporation. He must be interested in cars models featuring the next year. These kinds of information are available in the database but not given access to. Many reasons are behind it. Thus, the search for information comes down to the person who holds it.

For example, the personal home pages and internal directories of experts are not available through a simple search. In this case, this piece of information becomes the part of hidden web. For example, to write on a certain topic an expert is required. So typically, a small set of persons are contacted. These persons again referred to some other persons. In this way through this referral chain- a few layers deep, an expert is found. Similarly, in the hidden web, information is indeed available but due to technical or deliberate reasons, it is found under the layers of the hidden web. The layers are formed with the help of forms. So, hidden web content is that content that is hidden behind the web forms. To reach the content, a user has to fill the form. In [28] authors believed that the hidden web is a source of structured data.

The surface web is just the tip of the iceberg, huge information is hidden under the layers of the hidden web or sometimes called the deep web. The crawlers have the job of finding the web page and indexing it. Mainstream crawlers are not designed to find the data hidden in forms. Traditional crawlers rely exclusively on hyperlinks. Hidden web crawler has to find <form> tag, to find the entrance to the hidden web. The data will be retrieved only if the form is submitted with correct values.

As mentioned in [29], with the rise of server-side programming and scripting languages, such as PHP and ASP, databases became online accessible. Interaction with a web application help finding the desired data. The applications implement a common gateway interface for creation, generation and execution. Data is hosted on databases. Databases are queried using HTML forms. So, merely by following hyperlinks and downloading the web pages, the desired content can not be retrieved from

online databases. These contents are hidden from the web crawler point of view and thus are referred to as hidden web [30],[31].

Or in other words, we can say that the hidden web describes the hidden information available behind the search query interfaces that act as an entrance to backend databases. Hidden web consists of HTML pages produced as an answer to user requests submit through query forms [32]–[35]. Now the goal of hidden web crawlers is more specific than generic crawlers i.e is to search and index pages from the hidden web. And it is made possible by first finding the entrance to the hidden web i.e find form tag. Then submit the form and after submission, new URLs are generated. These URLs can as such be directed towards the surface web search engine. Under the term hidden web, there are two types of hidden web crawlers either its hidden web crawler based on surfacing and the other is vertical search engine web crawlers. But the hidden web vertical search engine crawlers are based on schema matching [33], [36].

No matter the category, finding entry is mandatory. Some online databases offer access to query interfaces that are dynamic query-based. Query interfaces act as a doorway to a hidden web. For example, if a user wants to buy an online air ticket, the search box of a search engine is to be filled by his query. A search engine will get back with result indexes that contain search forms now the user has to submit the form according to his specifications and will get the desired result. However, a traditional crawler cannot fill the form on the user's behalf and there is no mechanism for the crawler to go inside the database tables and extract the data. Hence, database content is, therefore "hidden" to the user [37].

To retrieve this hidden information, the web crawler must submit the HTML form. Most of the time a single submission is not sufficient, each time is filled with a different dataset and multiple submissions are required. Thus, the problem of crawling the hidden web got reduced to the problem of assigning proper values to the HTML form fields. Now the challenging task is to design a hidden web crawler that can to meaningfully assign values to the fields in a query form [38]–[40]. As it is explained in [14] the challenges in assigning values to fields of certain types such as radio buttons. Dealing with text box input is most difficult. It is proved in [29] that the size of the hidden web is about 440 times more than the surface web. If this huge amount of data is available, then it must be some ways using which we can find useful data. Following are few ways:

- User should increase their ability to find, evaluate and use information from all kinds of resources and collect experiences in constant practice and make full use of conventional search engines.

- A directory is a hierarchical presentation of hyperlinks of web pages and it is divided into topics and subtopics. Even some directories are a passage to HW to find relevant databases and then using a search tool needed information can be extracted.
- Use of hidden web crawler.

If it solely depends on the user to increase their ability to be found. This category is the private web. The second choice though useful require lots of human intervention. The most suitable way is the use of a hidden web crawler. The following section describes the detailed steps in hidden web crawling.

1.6.2.1 Steps of Hidden Web Crawling

1. Finding Sources of hidden web content: A human or crawler must recognize sites containing structure interfaces that prompt hidden web content. [41] discussed the configuration of the crawler for this reason.
2. Selection of similar sources: For a hidden web crawling task, one must choose a pertinent subset of the available content sources. In the unstructured case, the issue is known as database or resource selection [42] [43]. The first step of resource selection models the available content at a particular hidden website.
3. Underlying Content Extraction: A crawler must extract the content lying behind the form interfaces of the selected content sources.

Now if we know that designing a crawler is the best way to dive into hidden web, and there are some functions crawler has to performed, there must be some types of hidden web. The hidden web is also categorized into the following types as shown in figure 5.

- **Truly invisible web:** This type consists of the websites that don't have hyperlinks to follow and all the websites that are unlinked fall in this category.
- **Private web:** Private web poses restrictions on indexing, as access is limited to only a few people. Access to only specific IP addresses, Personal, internal, password-protected databases falls in this category.
- **Proprietary web:** Proprietary web demand identification and can be accessed only by providing registration.
- **Opaque web:** Traditional web search engines cannot index pages with disconnected URLs, and if the depth of crawl is high. This type of web is called opaque web [44].
- **Dark web:** This type came into existence when the owners do not wish to be indexed by the traditional search engines. The webmaster used no index policy. Usually, this type of web is not preferred due to its anonymity.

When we receive no results on Google, that doesn't mean that there is no associated webpage. It may be the page from the hidden web because it is not always sure that the search engine has indexed it. There exist multiple reasons why a page may be invisible. Some pages are only temporarily unavailable, conceivably scheduled to be indexed later.

Sometimes browser doesn't display few documents, file formats, or any non-standards file formats all consist of the truly invisible web. So, the line that differentiates the surface web and the hidden web is that the crawlers cannot put human-like knowledge otherwise technically it could be indexed in a search engine. Based on access, the hidden web has two methods to access it: virtual integration or schema matching and surfacing the hidden web. Access methods are shown in figure 6.

Virtual integration

- This approach is based on creating a mediator form for the specific domains. The mediator is the master form. Semantic mappings take place between each form and master form. Summaries are precomputed. The relevant forms are selected based on these summaries. Data is retrieved and combined from the selected form before presenting it to a user. In [45] virtual integration is compared with modern-day shopping portals.
- The data retrieved in this approach is homogenous.
- The cost of maintaining mediators is high.
- Relevant form identification is challenging.
- This approach is not suitable for general web crawling.

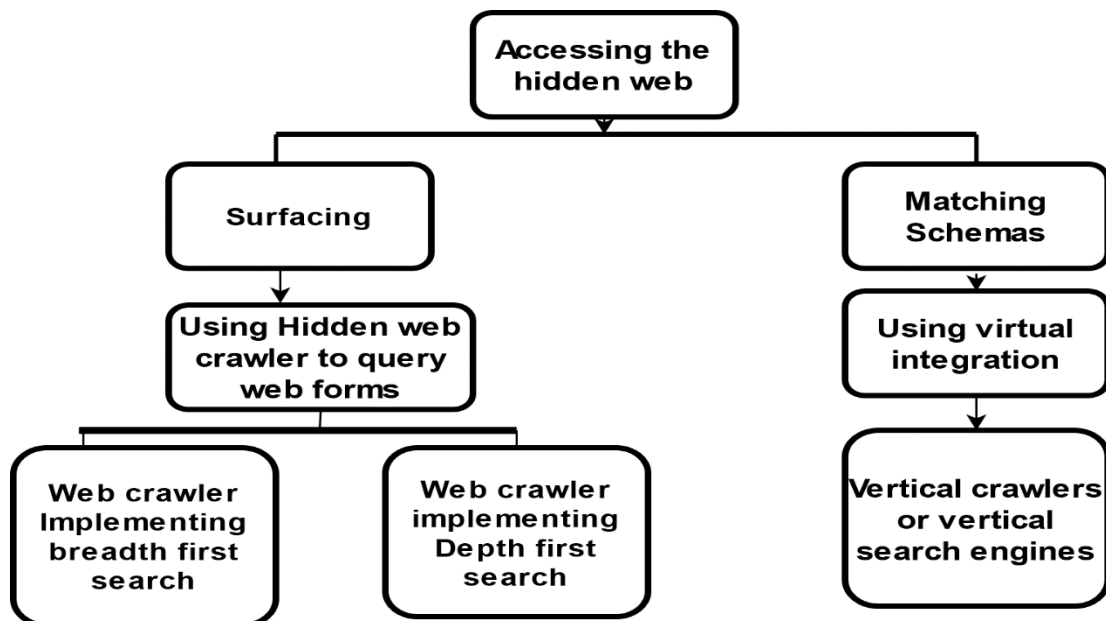


Figure 6: Ways to access the hidden web

Surfacing

This approach precomputes the submission of the most appropriate HTML form. With each submission, a new URL is generated. Generated URLs can be indexed by any search engine. Surfacing leverage the existing web search engines. According to [34], endless pages can be included using hidden web pages. Web pages can be included as direct traffic or as ranked sources. In the surfacing approach, there is no need of building query models. This is already solved by analysing the contents of the retrieved pages resulting from form submission. The real challenge is to pre-compute the queries for forms. Another challenge is to find the suitable values of the forms. If the form has the select menu, the values can be gathered by parsing the forms because the values are already known. The challenge is to minimize the number of visits to check if the value is correct or not. More the number of visits more is the unreasonable load on the system.

Both techniques have their challenges. In surfacing biggest challenge is to decide which form should be taken as the entry to the hidden web. Which values are accurate to generate positive results? Each web form has more than one type of inputs. Which inputs can be filled or not? Even the text input could be generic i.e keyword-based or one with specific values. Since the time this term being coined hidden web is considered the most valuable. According to [29], the hidden web is 550 times larger than the surface web. The key findings of this white paper are:

- The hidden web had 7500 terabytes of information in 2001, as there is no source to measure the information of surface web and hidden web, so reliability is totally on the research published.
- The hidden web consists of the most quality information.

- Half of the hidden web is based on topic-specific information.
- The quality content of the hidden web is 1000-2000 larger than the surface web.

Almost all of the studies reported on the hidden web rely on Bergman's findings for research. Since 2001, Bergman and the bright planet has not made their research toward the quality of the content of hidden web public.

Virtual integration vs Surfacing

The virtual integration method is chiefly a way to integrate the data to access hidden web content. This type of hidden web is mediator based. The mediator forms are created for each domain. The forms are analysed to identify the domain of content. Semantic mappings are created from the inputs to form the elements of the mediator form. Queries are formulated and then creates semantic mappings from the inputs of the form to the elements in the mediated schema of that domain. Queries over the mediated form can then be redeveloped as queries over each of the underlying forms. Results retrieved from each of the forms can potentially be extracted, combined, and ranked, before being presented to the user.

The hidden web is not all about indexing the web pages. Much of the retrieved data by issuing queries are used for harvesting. Google extract all the content though it stores it for temporary reference. The analogy that can be used here is- Google is like everything to everybody while the hidden web is everything to somebody. Instead of searching the entire web, the hidden web is directed towards a section to harvest data called directed harvest.

Finding accurate and relevant data is not easy. Resources are visible to machines and the web is designed for humans to view, catch and download data. Machines navigate through the set of rules, while humans look for engaging content. The rise of interactive applications has placed a new layer in the code. Following figure 7 shows if a query for store location is posed on titan's website.

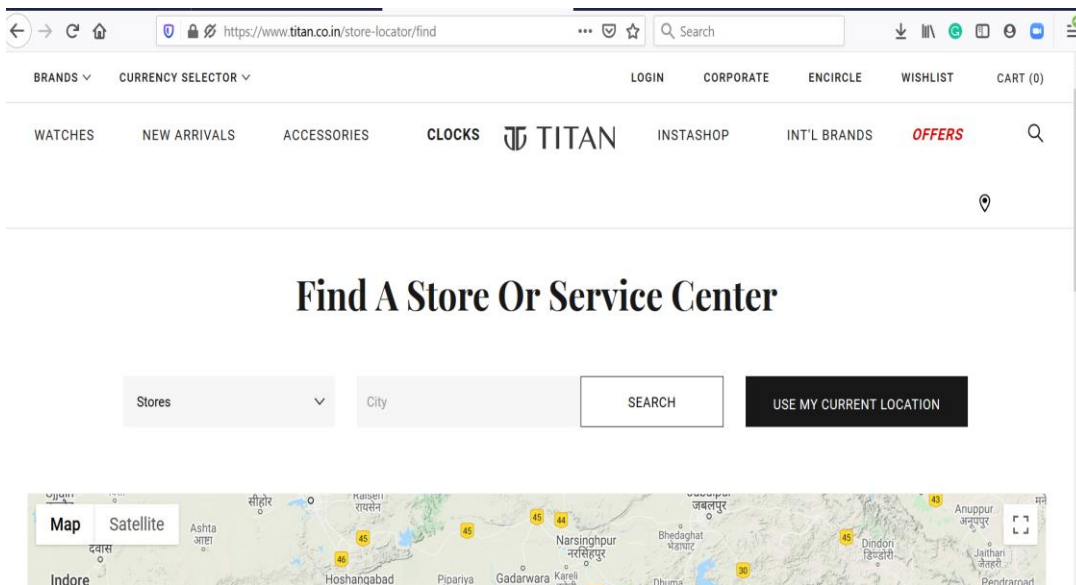


Figure 7: Shows the website with a search interface

1. The first step in hidden web crawling is to find the entrance to it by finding the search interface. Website first fulfils the criteria of being a part of the hidden web, due to the presence of a search interface. The results will be retrieved after the initial values are filled and the search is hit.
2. Suppose if the query is to find the titan store in Amritsar city. The user has to click the store's section option. Then in the city section, Amritsar is to be clicked.
3. The next step in hidden web crawling is to submit the forms. So using the same example user has is to hit the search button.
4. After the search button is hit, results are produced as shown in figure 8.

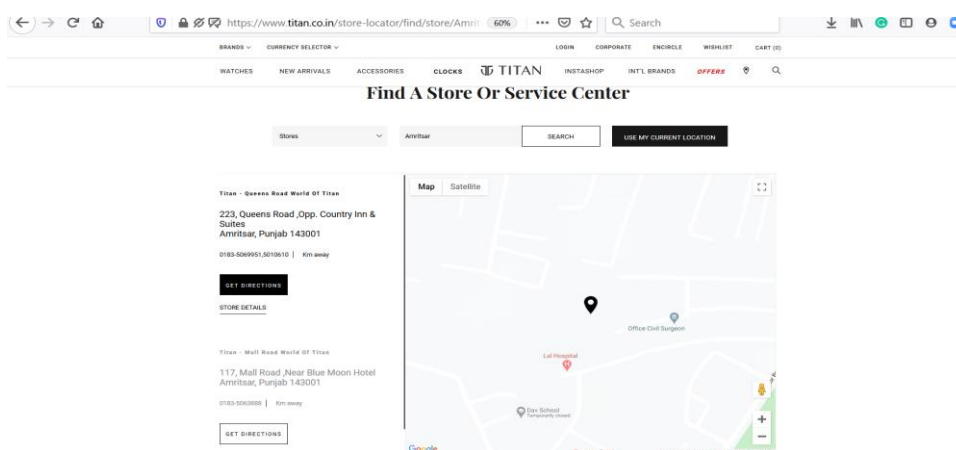


Figure 8: Results after the search button is clicked

A hidden web crawler has to automate all these steps. Suppose if the same query is posed on google as “titan store in Amritsar”. Even if it is posed as titan store Amritsar, or other keywords, a user has to start its search from the titan’s website. Now it would be easier if instead of using three steps to find relevant data it is available at step 1. This is how data is hidden. This shows the process

of engaging surface web search engines for hidden web search. The URLs generated after submission can be sent directly to a surface web search engine, otherwise used for harvesting.



Figure 9: Result of a query of titan store Amritsar

So, this example shows the difference between Google search and the steps required to perform hidden web crawling. The following diagram concludes the difference between the steps of general and hidden web crawling.

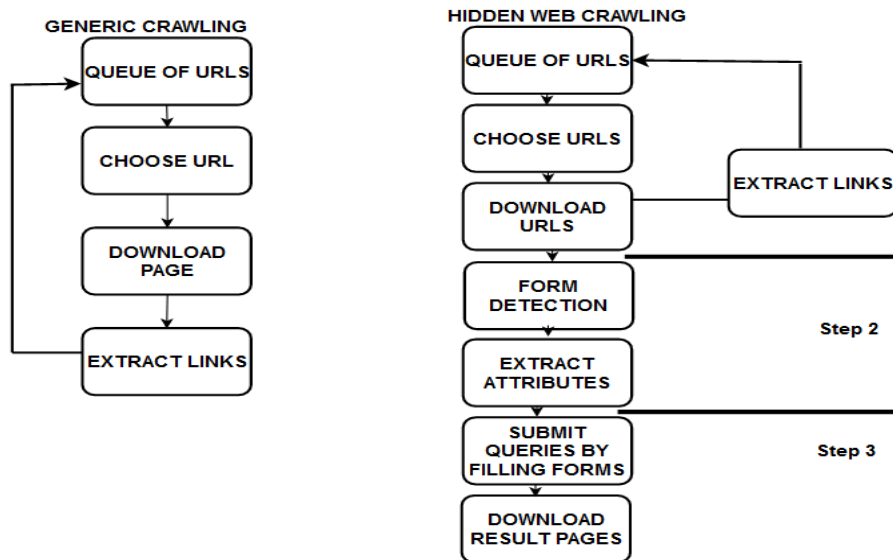


Figure 10: Steps wise difference between generic and Hidden web crawling

Web data is available in huge amounts. No matter how efficient the crawler is, a single crawler won't be able to crawl the entire web. So, distribution of crawler is required i.e more than one crawler are employed, each one doing their assigned job. The huge size of the web demands the use of distributed web crawlers. The following section describes the distributed web crawlers.

1.7 Distributed Web Crawler

A distributed web crawler (DWS) should be designed to scale to many pages per second. Crawl manager, Domain Name Server (DNS) and downloader are the main components of DWS. Crawl manager put URL in a queue, accept requests for URL's and download web pages. DNS resolver is requested for the IP addresses by the crawl manager and then it requests the robots.txt file in the web server root directory. When **robot.txt** files are parsed, excluded URL's are removed and the requested URLs are sent to a downloader. Downloader is required to have reasonable speed because it has to accept requests for a large number of pages. The responsibility of the crawling application is to check the pages that are downloaded, for hyperlinks. If pages are not visited earlier, then these are sent to the crawl manager in the form of a batch. In addition to low-cost components, the performance and network speed of distributed web crawlers can be scaled up [46], [47].

Importance of Distributed Web Crawlers

- The web environment is not static so the search engine needs to manage the web expansion, number of users and their changeable searching pattern. This is the main motivation for the system to process a growing workload as the load is shared in the system.
- Distributed web crawlers can also provide high capacity, where the capacity of the system is the maximum number of web users a system can maintain at any given time, also fulfilling both response time and throughput goals.
- Distribution helps in increasing download speed. Each task can be performed in a fully distributed fashion means no central coordinator exists.

Developing a distributed web crawler obliges major engineering challenges, all of which are eventually associated to scale. To retain the corpus of the search engine, a reasonable state of freshness the crawler must be distributed over multiple computers. The literature review gives a deep insight into web crawlers and their types.

1.8 Performance metrics

A performance metric is the degree to which a crawler holds some property. As per the best of our knowledge, precision, recall, coverage and effort were first used by [48] in their technical report. It is also inferred from the literature that precision and recall are commonly used performance measures in web crawling. Precision is defined as a portion of the information that is relevant to a search request. Effectiveness is the ability of the web crawler to satisfy the user in terms of the retrieval of relevant documents. Authors in [13] also mentioned the importance of precision and recall. It is also mentioned that maintaining high precision with the growing size of the web is difficult. The same

is with recall. For a focused crawler discarding irrelevant web pages is also important. The rate at which relevant web pages are identified from irrelevant web pages is called harvest ratio. Another important measure is robustness. It is the ability of web crawlers to stay on topic relevant web pages.

According to [49] recall and precision are useful if found from a finite set of URLs. In some studies, execution time and threshold of crawled webpages are also mentioned. It is mentioned in [50] that if no other measure is available, then a total number of crawled web pages is used as the metric measure. It is not always possible to collect all the relevant web pages. Authors in [51] proposed maximum average similarity and accumulated similarity. Authors in [52] suggested that along with scalability and freshness, coverage is another important measure for hidden web crawlers. As scalability and freshness cannot measure the effectiveness of form-based crawlers. Coverage is defined as the ratio of the total number of relevant web pages that the crawler has extracted and the total number of relevant web pages in hidden web databases. For this, a crawler is dependent on database content. Another metric suggested by authors is submission efficiency. It is defined as the ratio of response web pages with search results to the total number of forms submitted by the crawler during one crawl activity.

Suppose a hidden web crawler has crawled N_c pages, and let N_T denote the total number of domain-specific hidden web pages. N_{sf} is the total number of searchable forms that are domain-specific. Then harvest ratio is defined as the ratio of N_{sf} and N_c . Coverage is defined as N_{sf} and N_T . In [41] harvest ratio measures the ratio of relevant forms crawled from per web page. While in [53] coverage is defined as the ability to crawl as many relevant pages with a single query. Whether the crawled content is relevant to the query or not is measured by precision. Authors in [54] have introduced another measure called specificity for the hidden web. Coverage is defined as the number of web pages that can be downloaded by updating query keywords. The literature shows that different studies have defined coverage in different ways.

Motivation

1. The hidden web is inaccessible to the generic crawlers. This is the first motivation to develop a hidden web crawler. As it is mentioned in [29] users are unaware of the useful content hidden behind the forms.
2. The existing hidden web crawler considers the form tag as the entry to the hidden web. This motivated us to search for more rules of finding searchable forms as an entry to the hidden web.
3. The size of the hidden web is huge, and there are lots of free accessible databases available that can be searched and indexed by developing a hidden web crawler.
4. There exist no crawler that has implemented focused hidden web with distribution in web crawling.

CHAPTER 2

LITERATURE REVIEW

The literature review is split into studies related to general and broad search crawlers, preferential crawlers, hidden web crawlers and distributed crawlers.

2.1 General or broad search crawlers

These crawlers keep on following the links without any condition. Their main task is to fetch the page and collect all the links for further navigation [7].

2.2 Preferential crawlers

Preferential crawlers work on certain conditions submitted by a user. These crawlers do not collect all the web pages. These crawlers work by selecting the relevant pages before the actual crawling begins. Also, the topic and domains of crawling are predefined.

2.2.1 Topical crawlers

As the name suggests, these crawlers work on collecting information related only to a specific topic on World Wide Web. Not every time the labelled data is available to the crawler to remain focused. The seed URL's can consist of one or more pages as examples. Topical crawlers are expected to be smart enough due to the absence of text classifiers. In [54] authors have worked on using topical crawlers for finding domain-specific information. They have not only considered the link context but also the importance of links.

2.2.2 Focused crawlers:

The focused web crawler uses certain judicious criteria to reach relevant web pages. In this crawling technique, while collecting web pages, a crawler is forced to focus on a certain theme. It starts with seed URLs, which are already trained with a data set [55]. To keep the focus on the relevant pages, this technique is entirely dependent on hierarchical ordering. Hierarchy is made using hard and soft focusing rules to identify candidate webpages for maximum appropriate search. Very hard focus rules worsen the search process. Seed URLs play important role in relevant webpage exploration [56]. These crawlers start with some labelled relevant and non-relevant examples of web pages. Focused web crawlers have the following components called:

1. Fetcher or downloader which fetches the web page and retrieves its contents.

2. Frontier is a queue that stores the URLs to be processed. These URLs are yet to be visited, and then URLs available on a web page are extracted for further processing.
3. In addition to these three components, a focused web crawler has a topic-specific crawling model, relevance estimation and ranking module. Focused crawler first collects several URLs as seed sites. From these URLs, a crawler begins its crawling process and give results in the form of webpages crawled. Based on the literature following are the categories of focused crawlers.

2.2.2.1 *Semantic similarity based*

In this category, focused crawlers are endowed with the special ability to exploit the semantics of web pages. These crawlers are more focused on the meaning of data instead of the structure of data. The resource description framework is used to store data and ontologies for knowledge representation. Focused and semantic crawling strategies are combined in [57], crawling starts from the random webpage. Multithreaded semantic web crawler [58] is focused on learning educational content. It learns with help of ontologies. Priority is given to the semantically relevant web pages. Semantically focused web crawler in [59] adapts to the changes in the environment. Semantic relevance is computed from the statistic of downloaded web pages for source information discovery.

2.2.2.2 *Machine learning-based/ Adaptive*

These crawlers learn the linkage structure of the web while they crawl. Each web page is classified based on some features like in-links, sibling pages, and tokens in URLs. It is interesting to know that how these features contribute to collecting relevant web pages. Here crawler is not given any hard-focused rule, instead, it relies on the acceptance criteria for relevant web pages. Learning crawlers start with few general starting points to collect user-specified web pages. Initially, the crawl is general but gradually it is focused on the user-specific web pages. These types of crawlers are trained on either supervised, unsupervised, semi-supervised or any other learning criteria.

Training criteria is used to either classify relevant/non-relevant content. A focused crawler is called adaptive, when it employs learning rules to adapt its behaviour during crawling in a certain environment. The learning process of non-adaptive crawlers ends before the searching process starts. The learning capabilities for classification schemes are compared in [60] resulting in naïve Bayes as a weak classifier. While [61] being the pioneer in this work have used two classifiers for path learning and online training. For path, learning context graphs are used in [62] but adaption is limited only to adding context graphs of newly found target documents. A classifier in [63] is built on a weighting scheme. First, the term frequency and inverse document frequency is improved in terms of the expression ability of web pages. Instead of considering a web page as a whole document, it is divided

into body, anchor, headline and keywords. Different weights, based on the expression ability of each of the parts. Link priority algorithm along with joint feature evaluation strategy combine anchor text and link context to predict the relevant links.

2.2.2.3 *Form focused*

These special forms of focused web crawlers come with an additional component called form classifier that distinguishes between the form that can be searched and non-searchable forms. Form classification is mostly based on the structure of forms. Form Focused Crawler (FFC) efficiently discover forms on publicly indexable web [64]. Web forms are an entry point to hidden web crawling.

2.2.2.4 *Context/link focused crawlers*

Hyperlinks are a rich source of information for any web crawler. Link contexts act as a clue for further exploration. Focused crawlers in this category can imitate the actions of human users and exploit these important indications to conduct a further search. A technique based on a combination of context and link analysis of web pages is proposed in [65]. The application of this research is a vertical search engine. Focused crawling under supervised learning requires training for finding similar documents. The proposed algorithm assumes both terms and links important for finding similar documents. While in [66] link and context analysis are done exploiting maximum entropy Markov model, and linear-chain conditional random field.

2.2.2.5 *Application based focused crawler*

These crawlers are focused on a single area. A crawler crawls geographically aware web page. The collaborative policy considers URL and the anchor-based full content of web page, classification and IP address-based policies. Geo-coverage, geo- focused and geo centrality are the evaluation metrics proposed in [67]. E- health-related content is crawled by crawler proposed in [68]. This adaptive crawler dynamically prepares the priority list. URLs with high priority are crawled first by the focused crawler. This crawler meets the data curation needs of end-users focused on cancer-related data. While in [69] area of application is crime-related data.

2.2.2.6 *Miscellaneous categories*

Other categories of focused crawlers include Treasure crawlers [70], bootstrap crawlers[71], web crawlers that employ the structure of web page instead of the content of web page [72], sentiment focused crawlers [54]. The size of the frontier is a big issue in web crawling. A strategy named Sydney

strategy [12] helps to reduce the size of frontier and increase the coverage and quality of retrieved pages. It is also stated that if a crawler is more dependent on out-links, it worsens the coverage.

2.3. *Forum crawlers*

Web forums are repositories of information, popular for their open discussions in form of either discussion forums or community question answering forums. In forums, the user creates the content and is stored in databases on receiving a request from the user. The response page is generated dynamically based on a predefined template. The forum site is connected by a very complex graph. For interrelated discussions, forums are divided into classes. The sub-forums are at the middle level. Threads are at the lowest level. Members do their discussions under threads. Forums are ordered into a fixed set of topics with one major topic, driven and updated by members, and govern by moderators. For web forums, the task of the crawler is to download all the similar pages [73]. The application of CrimeBB crawler is to curate information being used by criminals in underground forums. Web forums have no centralized index that's why crawling web forum is difficult [69].

2.4. *Mobile crawlers*

This method helps in reducing the load. Web pages are selected and classified on the server-side. Authors in [74] have given a perspective of mobility as the skill of migrating the data source before the crawling begins. The mobile crawler access only the resource required at a time then move to the next resource. Except for mobility and autonomy, adaptive learning are the discussed features for mobile agents. While in [75] remote page filtering and compression, remote page selection, network load reduction as the benefits of mobile agents are discussed.

2.5. *Continuous /Incremental crawlers*

The web is dynamic, and web pages keep on changing. Incremental crawlers help in maintaining the fresh repository [76]. These crawlers visit pages that have changed or have a high probability of being changed. Continuous crawlers keep on revisiting every page it has visited earlier. Incremental crawler in [77] is scrapy based incremental web crawler. Incremental web crawling is made possible using bloom filters. But this crawler is not universal. Crawling rules varies with different websites. Authors in [78] have worked on the freshness and extraction of relevant content from the social web using focused crawling. To curate fresh content crawler has to incrementally visit the web page. Freshness is computed based on the web page's creation date and web page content features. Any type of focused web crawler that is working on classification have the following baseline workflow.

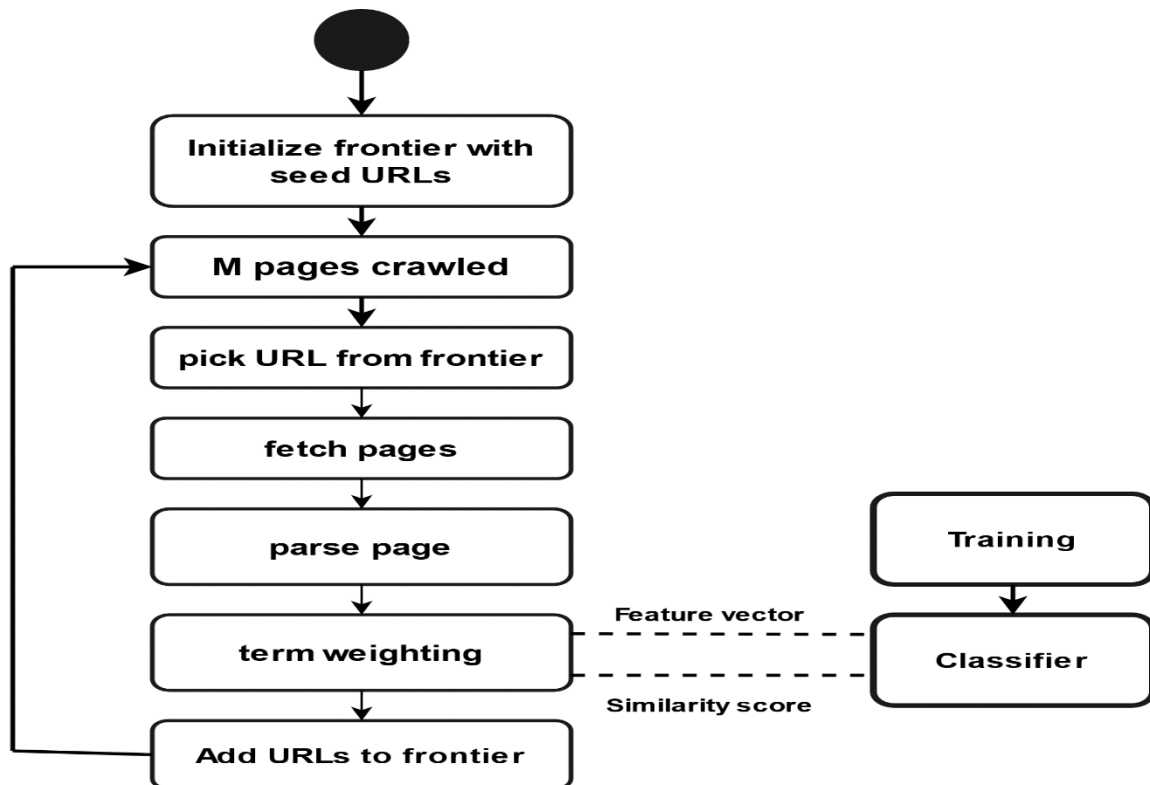


Figure 11: Stepwise working of focused crawler with classifier

Figure 11 sums the working of the focused crawler. URLs are picked from the frontier. The page is fetched, downloaded and parsed using the parser. Parser collects the terms. Term weighting is performed over the terms. Terms with weight are used to prepare the feature vector for the classifier and the score is computed. Further extracted URLs are added to the frontier.

Issues and Challenges with Focused Crawler

The following four issues were found with the focused crawlers:

- To give the immediate benefit to web pages, focused crawlers may let pass significant pages by crawling only those web pages that are expected to give immediate advantage [118].
- Many HTML pages need to be refreshed on a daily, weekly or monthly basis. For such pages, the crawler has to update the database by continuously adding fresh pages to the database to provide up-to-date information to the users. If these kinds of pages are large in number it puts stress on Internet traffic. A major issue is to develop a strategy that manages these pages [147].
- The issue is to develop some method to retrieve only highly related pages and alternate techniques to poll the Web server so that the underlying resources are not overloaded.
- The decision for the starting URL's for frontier is another challenge [133].

2.6. *Hidden web crawlers:*

To crawl the data hidden behind the web forms, the following steps are performed.

1) **Automated hidden web entry point discovery**

The hidden website can be discovered either using heuristic or machine learning. Authors in [79] have used heuristics either to discover form tag or finding other features of forms like- presence of a number of a text box and other heuristics to discard forms with short input used in [80], While in [41] and [81] machine-learning algorithms are used to classify forms to find entry to the hidden web. It is mentioned that mainly if the website has an associated form tag available it's taken into consideration. But all forms are not searchable forms. There could be login forms, registration forms and survey forms that are considered non-searchable forms. Now for the hidden web crawler question is: How to find searchable forms. To access the hidden web database finding a search interface is necessary. Mainly two techniques exist for this either use heuristic or machine learning. Following researches have used different heuristics for forms.

Table 1: Comparison of techniques in terms of heuristic and machine learning, pre query and post query, and features of forms.

Ref	H/ML	Pre/Post	Form features
3	H	Both Pre/post	1. Input text box, with less than six characters. 2. Password fields.
15	ML	Pre	Term frequency.
35	ML	Pre	1. Submission method. 2. Keywords. 3. The number of fields of each type.
65	MI/H	-	Word email, password control, radio and text control, hidden control, select control, submit control, advance search etc using DOM tree.
66	ML	Pre	Automatic

Form modelling techniques in the hidden web can be compared using form type i.e simple form, complex forms, whether the data extraction requires supervision or not, types of information a

model require such as form fields necessary to submit a form or the semantic relationship between fields.

Table 2: Comparison of research in terms of focused crawling and classification algorithm

Reference	FC	AC	Supervised/ semi-supervised
5	No	-	Semi-Supervised
15	Yes	Naïve Bayes	Supervised
18	Yes	Bootstrap algorithm	Semi-Supervised
35	Yes	Naïve Bayes	Supervised
66	Yes	EEFC classification	Semi-Supervised

Table 2 shows the comparison of research studies in terms of focused crawling (FC), Classification algorithm used, and learning techniques either supervised or semi-supervised. Classification and learning are an inseparable part of hidden web crawling. These two factors are considered in all four steps of hidden web crawling.

2) Form modelling

After entry to the hidden web, the next step is form modelling. Form modelling includes- if the classification is based on heuristic or machine learning. Another way to classify the form is - classification of forms before submission i.e pre-query, and classification of form after submission i.e post query [82]. The post query case response page is a source of classification. The feature of each form is also the source of classification. Form modelling is also based on a supervised technique or unsupervised.

3) Query selection

The concept of static hierarchy for query selection is implemented in [83]. BioNav has been used for the hierarchies. The performance measure is the overall cost of the queries. The aim is to retrieve a greater number of records than size. Authors in [84] have worked on both the content and structure of the form for queries as well as databases. The quality of the query is measured in terms of difficulty over the database. This model estimates the number of queries compulsory to retrieve the whole content of the hidden website. Performance is measured in terms of correlation of average precision. In [85] it is proved that the load on the system increases by increasing the number of submissions.

As some queries generate duplicate results. So, the query selection technique should have the goal of “minimize the number of queries and maximize the accurate response”. In [86] the proposed model for unsupervised keyword selection. This model starts with keywords extracted from the form page. First, most frequent words are calculated, and submission is repeated until maximum results are obtained. Performance is measured in terms of the effectiveness of the technique with and without using a wrapper. In [87], a technique in which for submission is done with the user-provided keyword. It also extracts keywords from the response pages. The keywords with higher informativeness are selected., which is calculated as their accumulated frequency.

Query interface analysis techniques except for detection of hidden web entry points can be summed up as the use of the following techniques

- Best effort parsing
- Search form schema matching
- Domain ontology identification

A query is searched either by domain or by URL. This research is based on a search by URL. This search URL is classified into the hidden and non-hidden web. If it falls in the category of hidden web then feature extraction is applied and domains are classified. The results are processed by the query processor.

4) Crawling path learning

Path learning plays an important role in finding web pages that lead to searchable forms. It is the order of pages that is followed to reach the relevant pages i.e which forms lead to the correct response page, which values can lead to successful submission of the form and other interactions of the webpage. The path learning techniques include filling of forms and submission, and path crawling after successful submission. Based on path learning crawlers are of the following types:

- Which can download as many pages as they can [88].
- Focused crawlers [89], [90] that are based on some intelligent rules leading the crawler to the subject relevant web pages. The proposed crawler is based on focused crawling.

Like other categories of hidden web crawlers, there are no fixed measures to compare the performance of the hidden web crawlers. So, following tables 3 and 4 show the comparison of hidden web crawlers based on common features.

Table 3: Comparison of hidden web crawlers

Crawling techniques	[52]	[91]	[92]	[93]	[94]	[95]	[80]
Parameters							
Technique	Supervised	supervised	supervised	Domain-specific	supervised	supervised	supervised
Freshness	No	No	No	yes	Yes	No	No
Query selection	Not automatic	Not mentioned	Automatic	Automatic	Automatic	Automatic	Not mentioned
Focused	Yes	Yes	Yes	No	Yes	Yes	Yes
Sampling	Yes	Not mentioned	Not mentioned	Yes	No	No	Yes
similarity	No	Yes	Yes	Not mentioned	Not mentioned	Not mentioned	Not mentioned
Classification	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 4: Comparison of hidden web crawlers

Crawling techniques	[31]	[50]	[61]	[62]	[63]	[68]	[73]
Parameters							
Technique	Supervised	Supervised	Supervised	Supervised	Not mentioned	Not mentioned	Not mentioned
Freshness	No	No	No	Yes	No	No	No
Query selection	Automatic	Not mentioned	Not mentioned	Automatic	Automatic	Automatic	Automatic
Focused	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Sampling	Yes	Not mentioned	Not mentioned	Yes	Yes	No	Yes
Similarity	Yes	No	No	Yes	Not mentioned	Not mentioned	Not mentioned
Classification	Yes	Yes	Yes	Yes	Yes	Yes	Yes

On concluding the performance measures in literature, most of the work in the hidden web is based on (1) finding entry to hidden websites, (2) measuring classification accuracy forms, (3) finding the forms and submission of forms. Hidden web crawling neither have a standard dataset nor comparison framework and testing environment to compare features of techniques.

Issues with Hidden Web Crawler

1. Efficiency, coverage, harvest rate, high quality and relevancy of hidden web sources are the main challenges for hidden web crawlers [10, 12, 13, 117, and 132].
2. Due to the unstructured nature of the surface web, finding forms is another considerable challenge [19, 20, and 48]. Pages with dynamic and decentralized nature pages are added, modified and deleted autonomously and forms are distributed on the PIW [12, 13].
3. The crawler has to process and interact with form-based search interfaces automatically. These interfaces are designed principally for the consumption of human beings [30, 55].
4. Hidden Web crawlers have to provide input in the form of “fill out” forms. This raises the question of how best to prepare the crawlers with the required input values for constructing search queries [119].
5. To run an IE system over a hidden web collection, a key challenge is to effectively and efficiently retrieve its useful documents, mainly the documents from which the IE system manages to extract tuples [127, 15].

2.7 Distributed crawlers

Authors in [96] have designed a distributed vertical crawler using a template-based periodic strategy. The domain of crawling is internet forums. Performance has been measured in terms of the number of URLs processed. Results have shown that distributed crawling has gathered more number of URLs as compared to a separated vertical crawler. A geographically distributed web crawler is presented in [97]. The approach is tested on various crawling strategies. From all, URL based and extended anchor text-based have performed best.

Jiankun Yu et al. [46] have presented a scalable cluster-based distributed crawler implemented as a data server. This crawler is shopping product-based. It performs feature extraction based on products. Web server is presented with processed data. Scalability is provided using a Hadoop platform. Huge data is stored in Hbase. The assumption for load balancing is that when all the nodes finish their crawling task at the same time. Performance of crawler is compared with Nutch crawler. With 8 crawling nodes between 3500 -4000 pages are crawled per minute.

Feng Ye et al.[98] have implemented a distributed crawler based on Apache Flink. On the cluster, Redis and other databases are deployed to store the web pages that are crawled. Scrapy is selected as an underlying crawling framework. Duplication detection is employed by combining the bloom filter with Redis. Performance is measured in terms of crawled pages and execution time. The crawler has managed to crawl 20000 pages in seven hours. CPU utilization rate even at the fourth hour is less than 35% as compared to a single crawler. Duplication detection is compared with bloom filter, link list, hashmap and treemap, from all bloom filter approaches has given promising results. The number of fetched pages increases to 7000 when the system used Mesos/Marathon platform.

A geographically distributed web crawler called UniCrawl is presented in [99]. Performance is measured in terms of crawling rounds. 50 crawling rounds have yielded 5000 new URLs and throughput between 10^6 to 10^7 for 6000 seconds. Authors in [100] have developed a dynamic web crawler as a service. Each stage of this architecture worked as a separate service and deals with its load, so scalability is also based on individual stages. The whole system does not need to be scaled. Along with being dynamic, this architecture is customizable and provide standalone service using elastic computing. The system has used Amazon RDS service. Performance is compared for fetched pages vs time graph. This crawler can fetch more than 250 pages in less than 400000 seconds. Then using 5 virtual machines 300 pages are crawled in 153.04 seconds. With the same configuration number of discovered URLs are 8452. This system has also worked on discovering new domains from newly discovered URLs. Comparison is made between response time for multithreaded crawlers and virtual machines. For 300 pages, the response time of multithreaded crawler is 142132.4 and for virtual machines on cloud computing are 512159.8.

Gunawan et al.[15] have proved that too many threads lead to a decrease in the performance of web crawlers. The system has divided crawling based on large sites first and then smaller size sites. Results are compared for CPU and memory utilisation. For 2000 threads CPU utilisation is 70% at 550 Mbps bandwidth. Choosing a suitable approach to divide the Web is the main issue in parallel crawlers.

Achsan and Wibowo [16] have worked on politeness property. Bosnjak et al. [102] proposed a continuous and fault-tolerant web crawler called Twitter Echo. This crawler continuously extracts data from Twitter-like communities. Performance is measured in terms of classification accuracy with 99.4% of the highest classification accuracy for non-Portuguese sites.

A distributed crawler in [102] is a platform-independent distributed crawler that can handle AJAX-based applications. They have also supported the breadth-first search for complete coverage. Performance is compared up to 64 active threads to crawl two-page applications and mediums sized

applications. Authors in [103] have implemented distributed crawler based on Hadoop and P2P. All the files are stored and shared in the distributed file system. Performance is measured as time to crawl vs nodes.

DG- Distributed General

DF- Distributed focused

CT- crawling time

DS- Downloading speed

MT – Maximum threads

CPU-U – CPU utilisation

T – throughput

Table 5: Comparison of existing distributed web crawlers based on their performance measures.

Ref	DG	DF	Max no of nodes	CT	Pages/url	DS	MT	Cpu-u	T
[46]	-	-	3	-	26136	361.50	-	-	-
[98]		-	-	60 secs	4000	-	-	-	-
[47]	✓	-	-	7HRS	7000	-	-	35%	-
[101]	✓	-	50 rounds	6000	-	-	-	-	10^6 -10^7
[16]	✓	-	5VM	153.0 4	8452	-	-	-	-
[104]	-	✓	-	4 hours	-	-	2000 at 550 Mbps	70	-
[103]	-	✓	-	-	-	-	-	-	-

Crawlers cannot be compared using one parameter fits all so the research reported in the literature is compiled based on attributes. The following table differentiates different types of crawler-based on their attributes.

Table 6: The comparison of web crawlers based on performance attributes

CRAWLERS ATTRIBUTES	Distributed web crawler	Incremental crawler	Domain-specific crawler	Mobile crawler	Breadth-first crawling
Robustness	✓	✗	✗	✓	✗
Flexibility	✓	✗	✗	✓	✗
Manageability	✓	✗	✗	✗	✗
Network Resources	✓	✗	✗	✓	✗
High Performance	✓	✗	✗	✓	✗
Incremental Crawling	✗	✓	✗	✗	✗
Cost	✓	✗	✗	✗	✗
Overlapping	✗	✗	✓	✓	✗
Communication Bandwidth	✗	✗	✓	✓	✗
Network Load Reduction	✗	✗	✓	✓	✗
Freshness	✗	✓	✗	✗	✗
Page rank	✗	✗	✗	✗	✓
Scalability	✓	✗	✓	✗	✗
Load Sharing	✓	✗	✓	✓	✗
High Quality	✗	✗	✗	✗	✗

From table 6, it is concluded that distributed, incremental, domain-specific, mobile and breadth-first crawlers have worked on above-mentioned performance measures. Here are few performance measures like freshness are the goal of incremental crawlers only. The quality web pages are not evaluated by anyone from the above-mentioned crawlers. But this feature is implemented by hidden web crawlers which shows another research gap in the studies. Similarly, there are performance measures for distributed crawlers as well. The following table shows the comparison of the distributed web crawlers based on:

- Scalability - (S),
- Load Balanced- (LB),
- Fault-Tolerant -(FT),
- Platform Independence- (PI),
- Centralized control-(CC),
- Full distributions- (FD),
- Extensible- (E).

Table 7: Comparison of distributed web crawlers

S.no	Refere nces	S	LB	FT	PI	CC	FD	E
[1]	[16]	✓	✗	✗	✗	✗	✓	✓
[2]	[79]	✓	✗	✗	✗	✗	✗	✗
[3]	[105]	✓	✗	✓	✓	✓	✗	✗
[4]	[102]	✗	✓	✗	✗	✗	✓	✗
[5]	[106]	✓	✗	✗	✗	✗	✗	✗
[6]	[107]	✓	✗	✓	✓	✗	✓	✗
[7]	[42]	✓	✗	✓	✗	✗	✓	✗

[8]	[108]	✓	✗	✗	✗	✗	✓	✗
[9]	[109]	✓	✗	✗	✗	✗	✗	✗
[10]	[110]	✓	✗	✗	✗	✗	✗	✗
[11]	[111]	✗	✓	✗	✗	✓	✗	✗
[12]	[112]	✓	✗	✗	✗	✓	✗	✗
[13]	[99]	✓	✗	✓	✗	✓	✓	✗
[14]	[10]	✓	✗	✗	✗	✓	✗	✗
[15]	[113]	✓	✗	✗	✗	✗	✗	✓
[16]	[114]	✓	✓	✗	✗	✓	✗	✗
[17]	[115]	✓	✗	✗	✗	✓	✗	✗
[18]	[116]	✓	✓	✗	✗	✗	✓	✗
[19]	[117]	✓	✓	✗	✗	✗	✓	✗
[20]	[118]	✓	✓	✗	✗	✗	✓	✗
[21]	[119]	✓	✗	✗	✗	✗	✗	✗
[22]	[120]	✗	✓	✗	✗	✗	✓	✗
[23]	[121]	✗	✓	✓	✗	✓	✗	✗

[24]	[122]	x	✓	x	x	x	x	x
[25]	[123]	x	x	✓	x	✓	x	x
[26]	[124]	x	✓	x	✓	x	x	x
[27]	[125]	✓	x	x	x	x	x	x
[28]	[126]	✓	✓	x	x	x	x	x
[29]	[127]	✓	x	✓	x	x	x	x
[30]	[128]	✓	✓	x	x	✓	x	x
[31]	[129]	x	x	x	x	x	x	x
[32]	[130]	✓	x	x	x	✓	x	x
[33]	[131]	✓	x	x	x	✓	x	x
[34]	[132]	✓	x	x	x	x	x	x
[35]	[133]	✓	x	✓	x	✓	✓	x
[36]	[134]	x	x	✓	x	x	x	x
[37]	[135]	x	x	x	x	✓	x	x
[38]	[136]	✓	x	x	x	✓	x	x
[39]	[137]	✓	✓	x	x	✓	x	x

Table 7 conclude that focus of web crawler towards distribution is mainly to satisfy the property of scalability, load balancing, and fault tolerance. Web crawler system is huge, it has to cater thousands of queries per second and even index construction takes a lot of time. On top of that, the system is required to handle faults, provide extensibility and other features at a reasonable cost. These needs point to a distributed system as the solution: a distributed system that can scale with additional components. Services are required to be distributed in such a way that these are always available. After carefully reviewing the literature of focused, hidden and distributed crawlers it is found that the following types of crawler are less reported in the literature.

2.8 Focused hidden web crawlers

Selective web crawling has been the interest of the research community for a long time. Focusing the crawl towards high-value target pages is one of the benefits of focused crawling, making focused hidden web crawling one of the eminent fields. Focused crawling on combination with the hidden web can emerge as the beneficial field for web crawling. Authors in [90] have worked on focused crawling for the hidden web to collect topic-specific web pages. This work assumes that keywords can describe the topic but not all keywords can help. [138],[139] are other studies that fall under the category of focused hidden web crawlers.

2.9 Focused distributed web crawler

A distributed focused web crawler in [16] crawls only a single web server. It collects a specific type of data from a web database. The goal of a multithreaded crawler in [101] is to improve the performance in terms of network bandwidth and storage capacity. Focused crawling is implemented using the Naïve Bayes classifier. Experiment results have proved that the performance of web crawlers will fall if too many threads are used. This will also lead to high memory utilization and low performance. Twitter Echo crawler in [104] is for crawling Twitter data. The focus of crawl and coverage are two parameters taken. The modular distributed design can easily adapt target environment to crawl high volumes of data. Modular design made it adaptable for addition of new functionalities.

2.10 Distributed hidden web crawlers

In this category, [96] have developed distributed hidden web crawler for web forums. In this system, the crawler dynamically adjusts packet size. Websites have different crawling periods and this crawler flexibly adjusts to it. Authors in [69] have discussed distribution using breadth-first strategy but it is not efficient in crawling forum sites. No other research in this area has been reported. This shows that distribution in the hidden web demands more work. The following diagram present a summarized view

of reported crawlers.

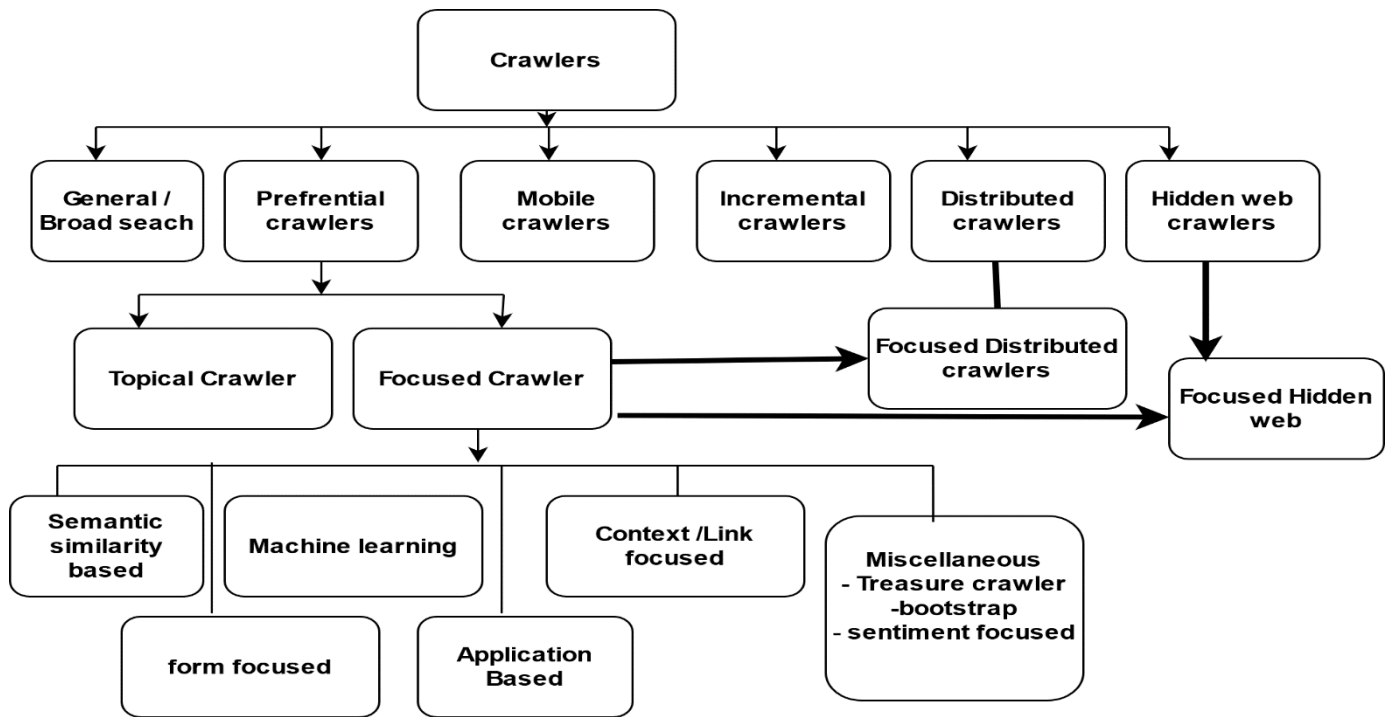


Figure 12: Shows reported the web crawlers and their types

The above diagram shows the web crawlers and their types. It also shows the combinations that are made using broad categories. Like from focused crawlers and distributed web crawlers, focused distributed web crawlers have emerged. Web crawling is still an emerging field, each type of web crawler has certain problems associated with them. The following table shows a summary of types of web crawlers and their associated research problems.

Table 8: Open research problems associated with different types of web crawlers.

Type of crawler	Method	Research problems
Classic focused	Based on guidelines retrieve the most relevant pages	The hypothesis of link and sibling locality is not accurate for always.
Semantic focused	The relevance of web pages is computed based on a property of sharing conceptual similar terms. Ontology is one of the concepts to define conceptual similarity	Semantic computation and determining the similarity of web pages to reach highly prioritized target pages only.
Learning/adaptable	<ul style="list-style-type: none"> • Learn or adapt the crawling guidelines. • Guidelines are updated dynamically. • Classification algorithms guide the crawler towards the relevant web pages and paths. 	<ul style="list-style-type: none"> • Efficient use of limited resources to perform well. • Intelligent crawling that learns the features and relations of web pages. • Web page ranking challenges.
Forum	Crawl forum data.	<ul style="list-style-type: none"> • Lack of a centralized index. • The requirement of wrappers for metadata extraction
Mobile	Compressed data is sent to the crawler after performing crawling at the server end.	How to reduce the network load.
Continuous /incremental	Crawler update only a set of topic relevant web pages	<ul style="list-style-type: none"> • Criteria to decide the probability of change, how to save bandwidth.

		<ul style="list-style-type: none"> • Frequency and type of revisit. • The decision between the frequency of change and the degree of change for a web page.
Forum	Crawl forum data.	<ul style="list-style-type: none"> • Lack of a centralized index. • A requirement of wrappers for metadata extraction
Mobile	Compressed data is sent to crawler after performing crawling at the server end.	How to reduce the network load.
Continuous /incremental	Crawler update only a set of topic relevant web pages	<ul style="list-style-type: none"> • Criteria to decide the probability of change, how to save bandwidth. • Frequency and type of revisit. • The decision between the frequency of change and the degree of change for a web page.
Hidden web	<ul style="list-style-type: none"> • Locate the hidden web source. • Understand associated forms • Select appropriate queries • Extract the relevant content 	<ul style="list-style-type: none"> • Efficiently finding the entry point to the hidden web. • Recognition of search interfaces to accept queries • Select queries that will return accurate data • How to automatically extract the content.

		<ul style="list-style-type: none"> • Increasing coverage and reducing the cost of crawling • Indexing ajax and javascript pages as much as possible. • Optimal seed URLs
Distributed	To increase coverage, crawlers are deployed in a distributed fashion	<ul style="list-style-type: none"> • Type of URL assignment Static or dynamic. • Cost per query reduction. • Web growth. • Effective partitioning of search space. • Efficient task assignment to crawling agents. • Effective cache design. • DNS, quality of service of servers. • Managing network bandwidth/ Load balancing etc.

Research Gap

The literature has reported that no web crawler yet has implemented distribution using focused crawling in the hidden web. In addition to these following points shows the research gap.

1. Most of the studies reported have used only form tag to find the entry to hidden web, using only a <form> tag is not sufficient. Because most of the websites these days come with the form tag.
2. In focused crawling ranking is based on similarity alone, a proposed crawler is based on the ranking based and site similarity, weighting and backlink count. The ranking reward function does not depend only on a similarity function.
3. Security is also a less touched field. By implementation of Redis provide security and fault tolerance.
4. The crawler work with both pre-and post-query approaches.
5. The crawlers face the issues of crawler traps and resource depletion. We have implemented stopping criteria's with which crawler resources will never deplete. As the number is fixed for forms as well as newfound URLs.

Problem formulation

The field of information retrieval has huge literature available. But when we branch it out to certain areas like distributed crawling in the hidden web, not much literature is available. As opposed to general web crawling, hidden web crawling requires a complex approach to parse, process and extract information from the hidden websites. And similarly, the process of distribution in hidden web crawling is equally challenging. The performance of the crawler is highly influenced by the architecture and techniques of crawling. From the literature review, it is found that distributed web crawler for the hidden web needs to be developed, as distributed crawlers for the hidden web are few and they face performance issues in terms of scalability, duplication, and are unable to support frequent changes in the underlying technology of web pages. Focused crawlers are software created to collect web pages that are relevant to specific topics. It was also found that there is no focused distributed web crawler for the hidden web as per the literature. Based on the above-mentioned research gaps following objectives are formulated.

1. To propose a novel architecture for smart distributed focused web crawler for the hidden web.
2. Creating algorithms for smart distributed focused web crawling that can automatically parse, process, and interact with form-based search interfaces.
3. Build web crawler based on the proposed architecture and algorithms.
4. Compare and optimize the performance of the web crawler.

CHAPTER 3

TO PROPOSE A NOVEL ARCHITECTURE FOR SMART DISTRIBUTED FOCUSED WEB CRAWLER FOR HIDDEN WEB.

Focused web crawlers play an important role in creating and maintaining subject-specific web collections. Application of focused crawlers includes search engines, digital libraries, specialized information extraction and text classification of the high-quality result page, minimizing the time, space and network bandwidth. The goal of a focused crawler is to retrieve the maximum relevant pages. Focused crawling in this architecture is based on priority computation and ranking of the sources. The ranking is again an associated term to similarity. The value of similarity is computed for the URL encountered. Seed sites play an important role in focused crawling. Crawling begins with a single seed and further links are extracted from that. The following diagram shows the working of the crawler from the seed URL.

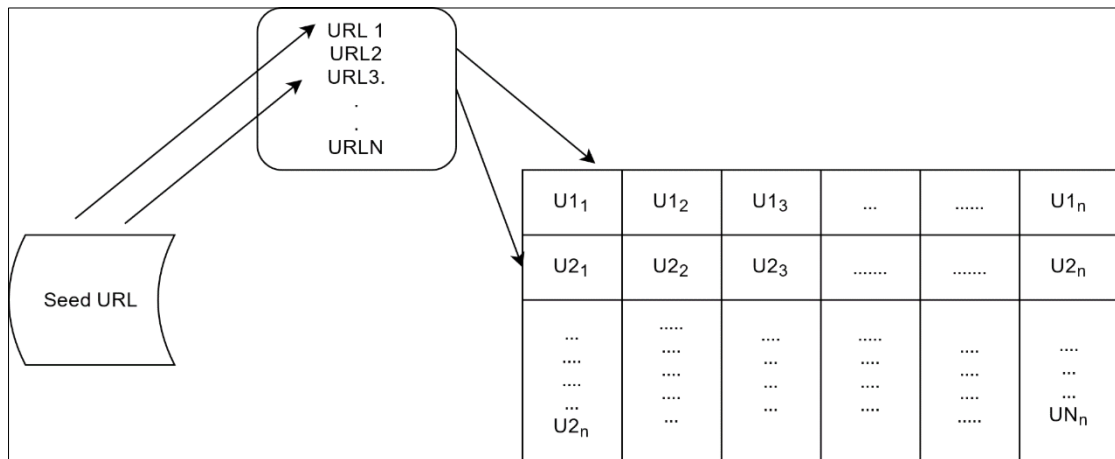


Figure 13: Link extraction from seed URLs

In the beginning, the frontier is empty. And it starts with a URL, all the methods reported in the literature falls into two categories in the context of seed sites. i.e either bootstrap based or machine learning-based. This architecture is based on bootstrapping method. The crawler is provided with the DMOZ dataset as seed URLs.

Another important aspect of focused crawling is a computation of ranking and similarity. The ranking is required in both focused web crawling and hidden web crawling. In this architecture ranking reward function is a factor of the number of backlinks, term weighting and site similarity. The value of similarity is computed between [0-1]. The similarity is measured for classifying domains, finding similar terms for form submission.

A web crawler encounters several types of web pages. Based on content and types of web pages, the following web pages fall under the category of hidden web. The following table shows the types of web pages that are encountered by a web crawler.

Table 9: Categories of web pages encountered by a crawler

Types of web pages crawler encounter	Type of content	Visible: can be indexed/ Hidden: cannot be Indexed
The web page is static HTML text.	Simple text	Visible can be indexed
HTML pages	HTML page but nothing to index	Hidden (specialized S.E. are there to search such content)
Web page made of HTML, but dynamic.	Html page with a form, consist of other controls. Sign in, user password, requiring selection. Form for user preference.	If the form itself is made up of HTML, can be indexed.
Web page with user preference form.	Other pages are straightforward HTML.	Because the form is based on user preferences so both form and its content are visible. S.E can index.
Web page with a form that has user-specified information	The form will generate dynamic information once submitted.	The form is visible, the content behind the form is not visible. The crawler will not know how to deal with form.
Dynamically generated page	Pages are displayed dynamically and a sign of "?" appears in URL	Dynamic pages are created by the script until the script is run, the crawler doesn't know what to do. So, it is part of the hidden web.
Dynamic/real-time	A web site that works with real-time data.	Hidden

	Google index 120 kb	
pdf and postscript file	All the web index 110kb. More than it is part of the invisible web	Hidden
The web page has an associated database with a web interface	User issued commands through HTML form, and the result is dynamically generation of a web page.	Hidden

Before the crawler begins crawling the hidden web sources are to be found first. i.e finding the searchable forms. A crawler is required to differentiate between searchable and non-searchable forms. Following are the basic definitions related to the crawler architecture.

- **Searchable form:** “Webform is called searchable form if it is capable of submitting a query to an online database which in turn, return the results of a query.”
- **Non-Searchable form:** “The forms, for example, login registration, mailing list subscriptions forms, and so on are called non-searchable forms. These forms do not represent database queries.
- **Pre query:** Pre-query techniques are based on identifying the entry to the hidden web based on the content and structure of the form.
- **Post query:** Post query techniques identify entry to the hidden web, based on issuing queries to the databases.
- **Hidden website:** A hidden website is associated with the searchable database. Results are retrieved upon issuing queries. A hidden website has a database associated, searchable form and result pages.
- **Depth of crawl:** It is defined as the depth at which the searchable form is located. Or the minimum number of hops required to reach the searchable form. For example, the URL abc.com/holiday/new-year/music will be similar to other Urls in its class.

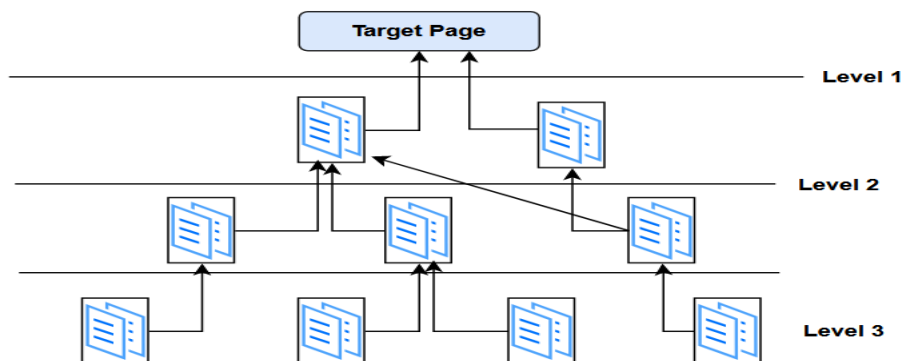


Figure 14: Shows the extraction of the target page

And the URL will be similar to abc.com/holiday in other class. Let the distance between the web page be denoted by D . S be the source document. L be another document in a class hierarchy. In our approach, we have fixed the depth of search. According to [84], most of the similar web pages are found at the depth of three. Distance '0': if document d_1 and d_2 are in the same class. Distance '1' or level 1: if document d_1 and d_2 are in sibling class. Distance '2' or level 2: if classes of d_1 and d_2 are first cousins. Distance '3' or level 3: if d_1 and d_2 are not related at all.

- **Domain and sources of search:** Domain and sources are either single or multiple. For example, the hotel domain and car rental domain. The source of retrieval could be from a single domain as well as multiple domains.
- **Type of search:** This crawler is based on structured query search and keyword search. Hidden web entry is found by filling the forms. Usually, these forms have multiple input fields. So the choice of a structured query is suitable. And the result pages are focused on the number of test domain used.
- **HTML form processing:** The job of the server is only up to when the form is submitted. Once submitted with correct values the web browser displays the results. Values of the forms are matched and once found correct the browser brings the result page.
- **Controls:** Any forms can have bounded, unbounded and calculated controls. This crawler is based on bounded controls. Bounded controls have an underlying table associated. These controls are used to enter, display and update the values. Unbounded controls display only static text. Calculated controls display data based on calculations.
- **Form elements:** A HTML form consist of the number of form elements. As the crawler work on structured data and bounded controls, button and submit are the most useful attributes.
- **Visible fields:** Each HTML has visible or hidden form fields. If the field is visible it should or could be filled to retrieve results.
- **Ranking reward:** Ranking reward is a function of the weight of terms and derived ranking. The similar the document is to the already found web data source the higher its ranking reward.
- **Use of breadth-first crawl:** For the crawling strategy, an experiment in [140] shows that for surface web crawling with the focused crawl, a breadth-first search is a better choice.
- **Stopping criteria:** There are few limits posed on the crawler. Like the depth of 3, form submission 100, form detection- 100. The reason for choosing such criteria is to stop to crawler so that it does not fall in a crawler trap.
- **Politeness policy:** The crawler should not overload the website with queries. This crawler has implemented politeness by limiting the number of submissions.
- **Selection policy:** Selection policy decide which webpage to select to crawl. This crawler has implemented selection policy by selecting only hidden web pages.

- **Fault tolerance:** Fault tolerance is implemented using the Redis server. If one server will fail others will take charge. Fault tolerance is an inbuilt feature of the Redis server.
- **Query probing:** It is a technique to classify the text database by training a rule-based classifier.
- **Response status:** When the crawler submits the page, data can either be retrieved or not based on the response status of the webform.

3.1 The proposed architecture

The proposed architecture work in three stages.

1. URL adaption and classification: Frontier is initialised in this phase, followed by parameter learning, ranking and domain classification.
 2. Relevant source selection: When frontier encountered a URL, all the links are extracted in link frontier, and fetched link frontier.
 3. Underlying content extraction: While in the third stage the form structure is extracted to fill and submit the forms.
- This system has implemented the frontier as a queue from which URLs are taken out for further processing. The frontier starts from the seed URLs. We have implemented three queues as frontiers. The frontier for seed URLs consists of URLs from the directory.
 - The frontier for links consists of URLs extracted from the seed URLs. The frontier for fetched links consists of URLs from links. The frontier depletes very easily. So as the frontier for seed URLs will have a scarcity of URLs, the frontier for links will be used.
 - A webpage can have multiple hyperlinks but not all are relevant. Aim of a web crawler is to fetch maximum hidden websites by minimizing the visited URLs. The following figure 15 shows three phases of the crawler. All the stages are interconnected with each other.

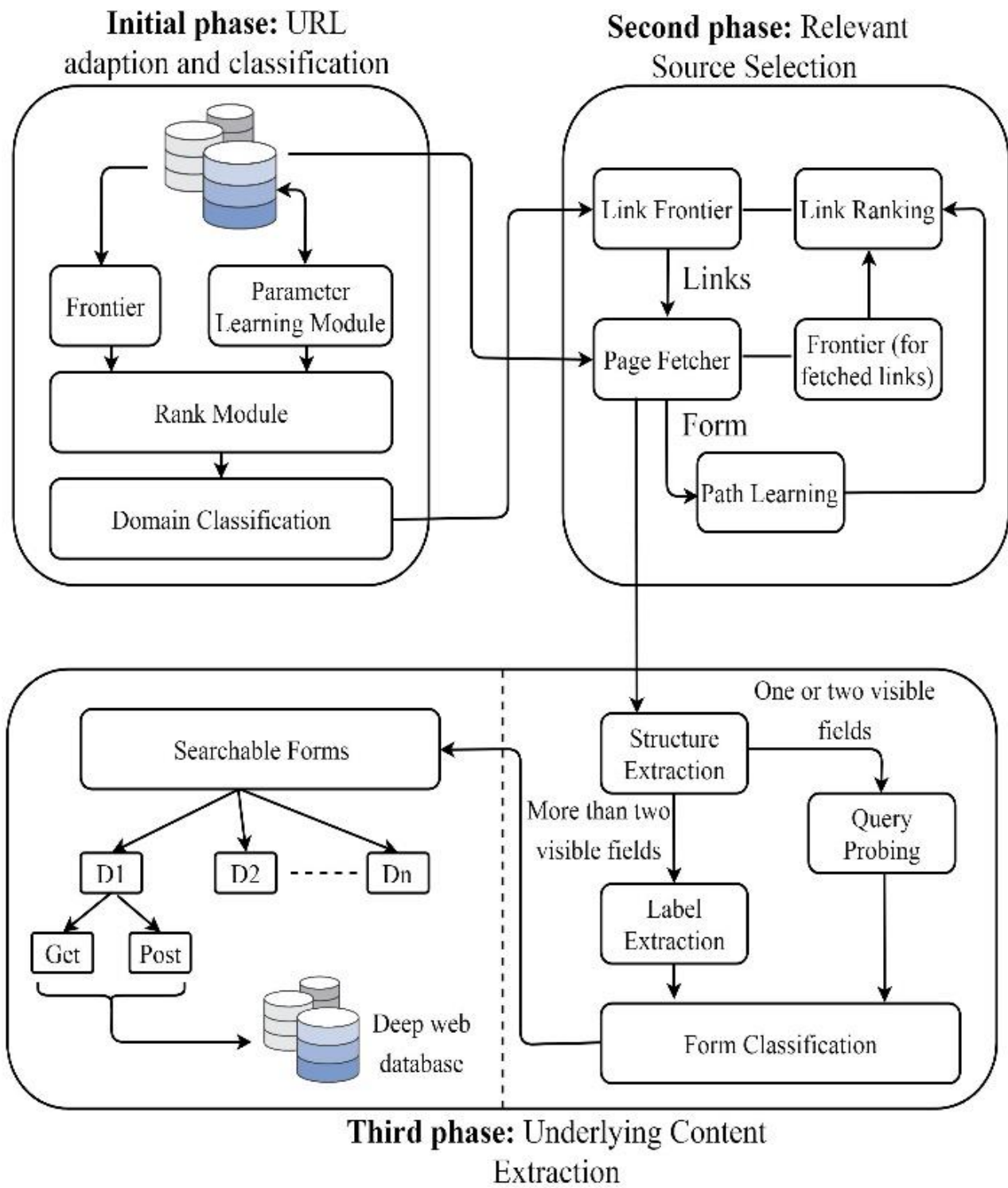


Figure 15: Architecture of proposed crawler as a single entity focused crawler

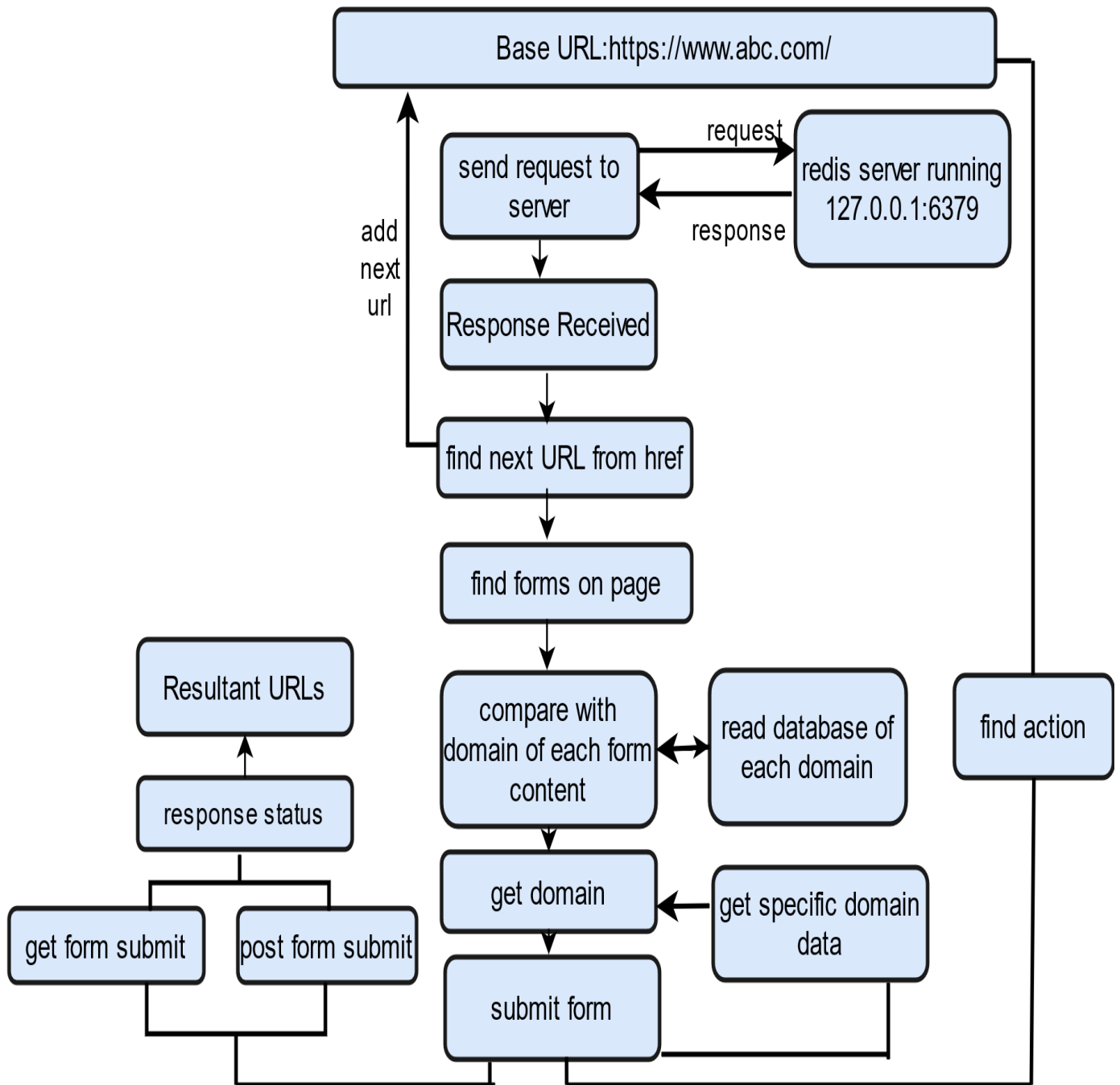


Figure 16: Analogy of the use of single URL

Figure 16 shows the working of the crawler with respect to one URL. For example, abc.in is taken. As the URL is extracted from the frontier, the next step is pre-processing of URLs.

1. A request is sent to the server, Redis server operates at port 127.0.0.1:6379
2. Find the form on the page.
3. Compare extracted content with already learned domains, find the relevant domain.
4. Get the specific domain data, for form submission and submit the form.

5. Submission could either be post query or pre query. If the number of visible form fields is more than two then use the get method, post method otherwise.
6. When the form is submitted, check the response status and add it to the result database.

3.2 Preprocessing of URLs

The baseline components of URLs are extracted. These are host, extension, documents, path etc). from all the components URL, path, anchor and text around anchor are fed to feature vector. The system has implemented python NLTK for stemming, stop word removal and tokenization. Now all the segmented words are fed to the feature vector.

Feature space for the hidden website is defined as:

$$FD = [\text{URL, anchor, text around anchor}] \quad (1)$$

Feature space for links of the hidden website is defined as:

$$FL = [\text{path, anchor, text}] \quad (2)$$

First, stop words are removed. The next step is stemming using the Porter stemming algorithm. The top m terms are selected. After pre-processing, the URL is represented as

$$U = [u, a, t, p] \quad (3)$$

Where u is URL, a= anchor, t= text around Url, p = path of URL. Now different weights are assigned to vector U

$$Tf_{ij} = u \times tf_{ij1} + a \times tf_{ij2} + t \times tf_{ij3} \quad (4)$$

$$Tf_{ij(link)} = u \times tf_{ij1} + a \times tf_{ij2} + t \times tf_{ij3} \quad (5)$$

$$\omega_{if} = \frac{t_{if_i} \times i \times df_j \times Ig}{\sum_{N=1}^N (t_{if_i} \times i \times df_j)^2} \quad (6)$$

W_{ij} = weight of term t_j in document d_i

Tf = term frequency

Idf = inverse document frequency

N = total number of documents

IG= information gain of term t_j

$$IG_j = h(d) - h(D|t_j) \quad (7)$$

$$h(d) = -\sum_{di \in D} p(di) \times \log_2 p(di) \quad (8)$$

$$H(d|t_j) = -\sum_{di \in D} p(d_i|t_j) \times \log_2 p(d_i|t_j) \quad (9)$$

The path of the URL is learned to reach the exact location of the form. A special symbol related to the path is the forward-slash (/). The path of the URL is found after the hostname. Anchors are helpful in internal navigation in URLs. And we need to find the internal links as well.

3.3 Weight calculation of terms

Based on feature vector construction, the weight of a term is computed based on occurrence in URL(U), anchor(A), text around the anchor (T) and path (P).

Term frequency of term T_i in U, A, T and P and is defined as:

$$t_{fi} = \alpha \times t_{fi} + \beta \times t_{fi} + \gamma \times t_{fi} + \delta \times t_{fi} \quad (10)$$

where α , β , γ , and δ are the weight coefficient. Ig is the information gain of terms.

$$w_{ij} = \frac{t_{fi} \times i \times df_j \times Ig}{\sqrt{\sum_{N=1}^N (t_{fi} \times i \times df_j)^2}} \quad (11)$$

It is proved in [12] that outcome of the tf-idf alone is an inappropriate distribution of the feature vector. In their approach, information is combined with the segmentation of pages in a major four sections. In our approach weights are based on URLs and associated terms. After term weighting similarity denoted by (S) is computed between the already discovered URL and newly discovered URL. The similarity is required in the ranking section is computed as follows:

$$S = sim(U, U_{new}) + sim(A, A_{new}) + sim(T, T_{new}) \quad (12)$$

Similarity has a different meaning concerning each step in web crawling. The crawler has to work on finding similar URLs so that it can prevent similar data retrieval. The two files are said to be similar if a small percentage of text is different. In the context of similarity, two related terms are resemblance and containment. Resemblance means if two files resemble each other while containment is when one

file is contained in another. The crawler has to find similarity two ways: first, it has to find the similarity to perform term weighting in pre-processing of URLs, results will be used in computing ranking. Secondly, the system has to eradicate the duplicate URLs, for this the already existing technique is combined with Redis to improve the efficiency.

The similarity is either character-based or term based. Cosine similarity is part of term-based similarity. After pre-processing, the system has a list (k) of more than 50k keywords. Now using the similarity model (V), the system read the reference file. In the next step, the elements are removed from the list (k) one by one. The similarity is computed between the two lists. For example, the flight is a word in list (k) and it has a close match in (V). If cosine similarity is 1, it means an exact match is found. The system has to extract an exact match as well as a close match. Cosine similarity is used to find terms for a query, and for finding similar URLs as well. Close match results are used for queries during repository generation in form submission.

The similarity is required for finding top k terms for ranking. And it is also required to eliminate the near duplicate URLs. To eliminate near-duplicate URLs, we have implemented the Simhash technique. It is based on counting the occurrences of binary strings. Keys are stored in a data store. And a separate database is used as the query or repository database. Simhash program will open each file in the seed set. It scans through it and looks for matches. Tags are defined as the preselected set of strings, and Sum table stores the count of matches. The hash key is computed from the sum table entries. The attention is restricted only to the term weighting scheme. The data store consists of a computed path, size, term weight filename and key. Once the database is populated with data from the test directory. SIMFIND function looks for similar keys.

The assumption of tolerance range is similar to [141]. The 16 8 bits tags are applied to the similarity measure. This measure is used to remove duplicate URLs. The websites have a complex relationship with each other. One URL can be found on more than one website. Downloading the same URLs multiple times is a waste of resources. The unique fingerprint for each request is calculated first as shown in figure 17. All the repeated requests are removed here. Simhash [141] is combined with Redis for improvement in results. The following diagram shows the working of Simhash. The files shown in figure 17 are seed URLs. Simhash is a binary similarity metric. Two files are similar if a small percentage of their raw bit pattern is different, and it operates at the word level. It does not attempt complete coverage. It is focused on the files whose degree of similarity is strong.

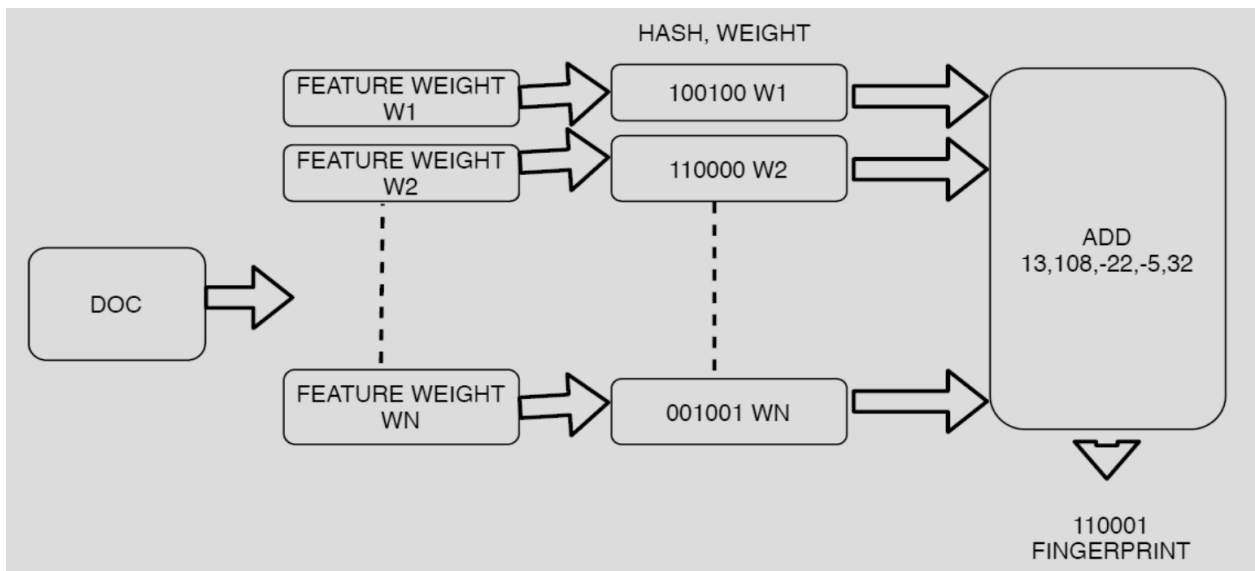


Figure 17: Computation of fingerprint

The doc file is a list of URLs from the seed database. The weight of the feature vector is computed using tf-idf. Though the actual method has implemented variation with weighting methods. But it is implemented in the proposed crawler for detecting duplicate URLs only. The fingerprint is computed by add operation. The following diagram show formation of sumtable.

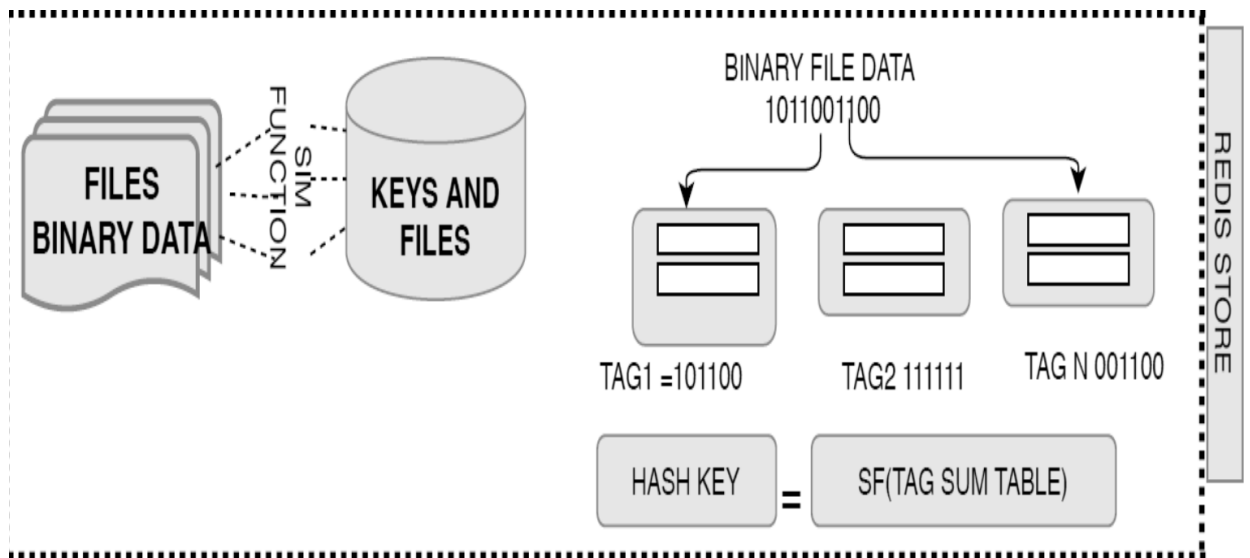


Figure 18 : Formation of sumtable

This technique is implemented over the seed URLs in the frontier. Then afterwards as the system keep on collecting the URLs, duplicate URLs are discarded.

3.4 Learning

The majority of the web crawler reported in the literature are based on learning. Each crawler has implemented it differently. The aim of a learning algorithm is to learn features for feature selection and use these features for ranking. The results from the first run will be used in successive runs. The uniqueness of our learning algorithm is learning is performed at both site level and link-level (extracted URLs from a webpage). The following steps are performed in the learning algorithm.

1. A new website (X) is encountered, extract [U, A, T].
2. For each URL the queue with sites is ordered using a similarity model w.r.t [U, A, T].
3. Extract the links from X.
4. Links are saved in the link queue. The link queue is ordered using the similarity model w.r.t to [P, A, T].
5. Check for searchable forms by following the rules of rejection criteria.
6. If the form is searchable extract path, anchor and text.
7. With this, the information in the parameter learning module in stage 1, and link ranking in stage 2, is updated. And new features are reflected in these two modules.
8. The crawler has reached the threshold of 0.8, i.e 80 new URLs and 0.01, i.e 100 new forms.

Site and link ranking are dependent on the learning process. Site learning and link learning are updated periodically. Every time a new site is entered frontier and link ranking is updated to find similarity of the links. When parsing and crawling of URL is completed, features are collected. Then FSL is updated when relevant forms are found.

3.5 Ranking

Aim of ranking in hidden web crawling is to extract top n documents for the queries. The cost is expected to be the least for this work. We have adopted the formula for ranking from [142]. But our reward function is based on a number of out-links and site similarity and term weighting. Let SF be the frequency of out-links.

$$SF = \sum I_i \quad (13)$$

$I = 0$, site has not appeared,

$I = 1$, if it has appeared

So ranking reward is a combined function of term weighting, site similarity and number of backlinks.

$$\epsilon = w_{ij} + S + SF \quad (14)$$

$$(r_j) = (1-w) \cdot \delta_j + w \cdot \text{ranking reward}(\epsilon) / c_j \quad (15)$$

w is the weight of balancing ϵ and c_j . δ_j is the number of new documents. Computation of r_j shows the similarity of ϵ and returned documents. If the value of ϵ is closer to 0, it means that returned value is more similar to the already seen document. c_j is a function of network communication and bandwidth consumption. The value of the ranking reward will be closer to 0 if the new URL is similar to the already discovered URL. Given the website with a searchable form, it is checked for similarity with known web pages cosine similarity is calculated. Site frequency is the number of times the site appears on other homepages i.e backlinks. After computing similarity, it is added to the ranking formula to compute the rank.

3.6 Domain classification

The aim of domain classification is to find topical relevance of the site and the home page. As the new URL is received. The system will extract the homepage content by parsing, removing stop words and stemming. The feature vector is constructed as explained above in the preprocessing section. The resulting vector is fed to the classifier to check if it is relevant or not. The crawler gets the base URLs or seed URLs from the frontier. The request is sent to a server. The crawler will first check for the presence of a search interface. Based on these the decision is made whether the encountered form is searchable or not. After these rules are applied, the crawler has a set of URLs that have <form> tag as well as the property of being searchable. On being provided with suitable values, these forms will retrieve the data from the associated database.

Rule1: If the crawler does not find any <form> tag, consider this a non-searchable form.

Rule2: If crawler found the <form> tag. Then extract the attribute type. If the attribute type is not in the repository call it a non-searchable page.

Rule 3: If the crawler found the <form> tag, and extracted attribute type matched in a repository. But the attributes < 3, consider this page non-searchable.

Rule 4: If the number of attributes is >3, but the submit button is not found, consider this page as non-searchable.

Rule 5: If there exists <form> tag, and attributes are similar to the repository, and submit button is also there. But button marker is not present then consider this page as non-searchable.

Rule 6: If there exists <form> tag, and attributes are similar to the repository, submit button and button marker is are present. It is a searchable form.

Rule 7: If there exists <form> tag, but crawler found login, then this is non-searchable.

Rule 8: If there exist <form> tag, but the crawler found registration, consider this page as non-searchable.

Rule 9: If there exist <form> tag, but the crawler found subscribe, then consider this page as non-searchable.

Rule 10: If there exists <form> tag, but crawler found mailing list subscription, then consider this page as non-searchable.

These days most of the web pages came with the form tag. But all forms cannot be put in the searchable categories. So above mentioned rules are implemented. Following figure 19, show the tree form of the rules.

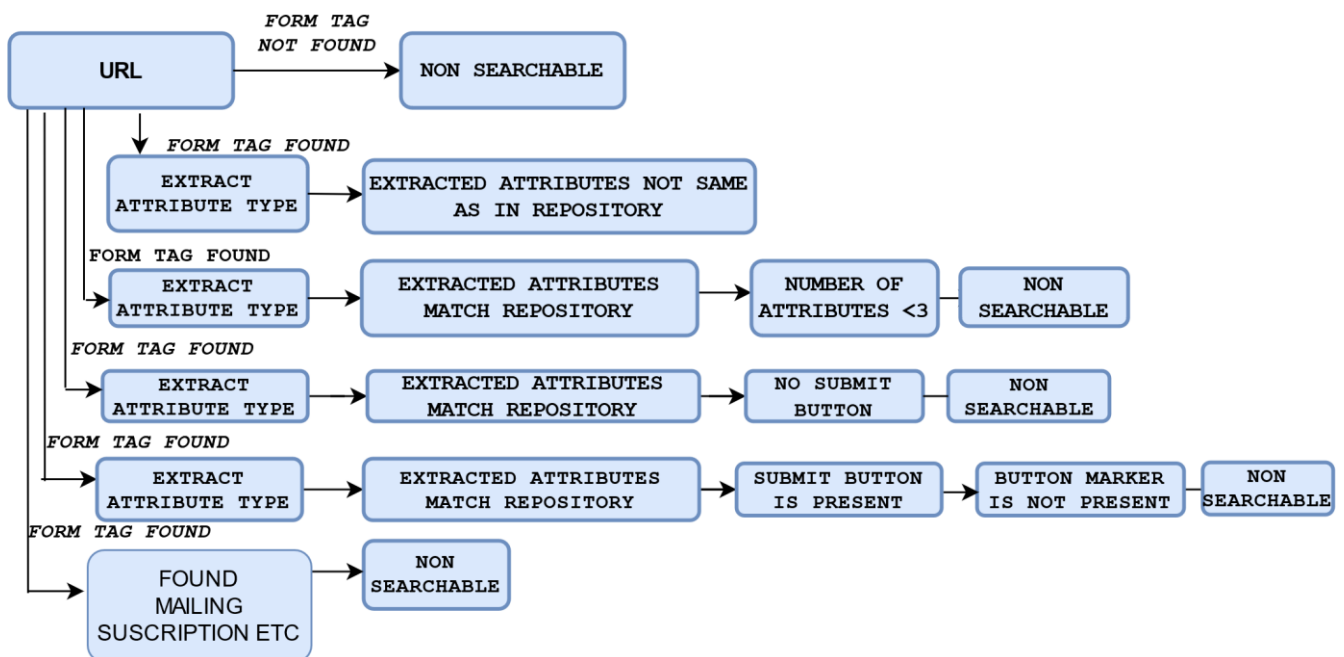


Figure 19: Diagrammatic view of rules for searchable forms

3.7 Form structure extraction

The aim of form structure extraction is to extract the content of the form. Each search form has some controls that a human can easily fill and submit. If a crawler has to fill the forms automatically it has to have a set of resources to automatically fill and submit the forms with suitable values. A task-specific database is associated with a crawler. This database contains the set of values for filling the forms, created by parsing the form. And form element table is created with a control element type, label and domain values. The crawler will adaptively learn filling values with associated forms. This database of values will be initialized with the launch of the crawler. When the first run of the crawler is completed the parsed values will be analysed to collect data. Form submission is of two types of post-form submission and Get form submission. This crawler work on both type of submissions. After

the form is submitted crawler got the response status. Response status is either a valid page or no page found code.

Steps for form parsing: Following steps are performed while forms are parsed.

- Using the Request library of python, an HTTP GET request is sent to the URL of a webpage.
- The response of HTTP request is HTML content of a webpage.
- Data is fetched and parsed using Beautiful soup.
- HTML tags and their attributes are analysed.
- Data is output in CSV file.

3.8 Form and response analysis

After the form tag is found, its elements are extracted to make a repository called form element repository as shown in table II. Forms have multiple control elements. It could be of any type:

- Text: This area of a form can be edited with multiple lines of words.
- Input: This editable area has attributes types- type as text, Submit, checkbox and radio button
- Select: Select has two options like drop-down list box and multi-choice list box.

Table 10: Contents of repository

Control Element (Visible Fields)	Label	Domain	Type Of Domain	Size	Status
Submit	Search	Submit	Infinite	More than 3 kb	VR
Radio	Flight trip	Round trip One-way trip	Bounded	More than 3 kb	VR
Select	From, to	Name of place (eg: Delhi to America)	Bounded	More than 3 kb	VR

After the repository is made as shown in table 10, as explained in [143] form submission include problems like 404 error page, duplicate information, and sometimes all information is retrieved in

single submission otherwise multiple submission are required. Two heuristics are implemented based on visible fields. If the number of visible fields is one or two -forms are classified using query probing, label extraction otherwise.

3.9 Query probing

The aim of query probing is to develop a set of queries for each class. On submitting these queries crawler will retrieve the same documents for that category. We have implemented a similar approach as [144], but we have implemented hierarchal classification, and the system can expand the number of classes as it crawls more URLs. Currently, classes are based on a seed datasets.

3.10 Form submission

Two related techniques with form submission are:

HTTP POST: In the post query technique, forms are submitted with (name, value) tuple. This pair is sent encoded in the body of the request. Query probing is implemented in the post method technique.

HTTP GET: In the get query technique, forms submission takes place by giving (name, value) pairs in URL. The pre-query technique is implemented in Get method technique. A URL has three symbols a question mark (?), equals to (=) and ampersand (&). (?) differentiates encoded (name, value) from the base URL and action path. (=) ,(&) separates the field name and field value.

3.11 Stopping criteria

Exhaustive crawling is a waste of resources. This system has implemented the following stopping criteria.

- Maximum depth of crawl: the crawler will stop following the link when the depth of three is reached. It is proved in [23] that most of the hidden web pages are found till depth 3. While at each depth maximum number of pages to be crawl is 100.
- At any depth maximum number of forms to be found is 100 or less than a hundred.
- If the crawler is at depth 1, it has crawled 50 pages, but no searchable form is found, it will directly move to the next depth. And the same rule is followed at depth 2. Suppose if at depth 2, 50 pages are crawled and no searchable form is found. The crawler will fetch a new link from the URL.

3.12 Assumptions and thresholds

- The size of the frontier should not decrease below 100 URLs at a time, as the number decreases, it will crawl URLs from the link frontier and fetched link frontier.
- The learning threshold is 80 new sites and 100 new searchable forms.
- URLs are picked out from a crawler using first in first out order.

3.13 Distribution

The above architecture (figure 16) describes the working of a single entity of a focused crawler for the hidden web. Book, hotel and flight are the domains that crawlers process. We have implemented the Redis server as shared storage for URLs. Redis stores information in cache, unlike databases, which is why information access is faster. The proposed crawler is developed in Python. Scrapy is an application framework. Scrapy helps to extract web pages and structural data. For distributed crawling, Scrapy and Redis are integrated to implement more than one server.

A crawler is implemented with a breadth-first search per host. Data can be extracted either by using the API of a website or by extracting information by accessing the webpage. From the frontier, a URL of the webpage is sent as a request to the server. The server responds by returning the HTML content of the page. Once the data is accessed next step is to parse the data. To create a tree structure of HTML data html5lib parser library is used. To navigate through parse tree beautiful soup is used. It can pull any type of data. The following figure 20, explained distribution using multiple Redis. Using multiple Redis servers, the crawler is made fault-tolerant.

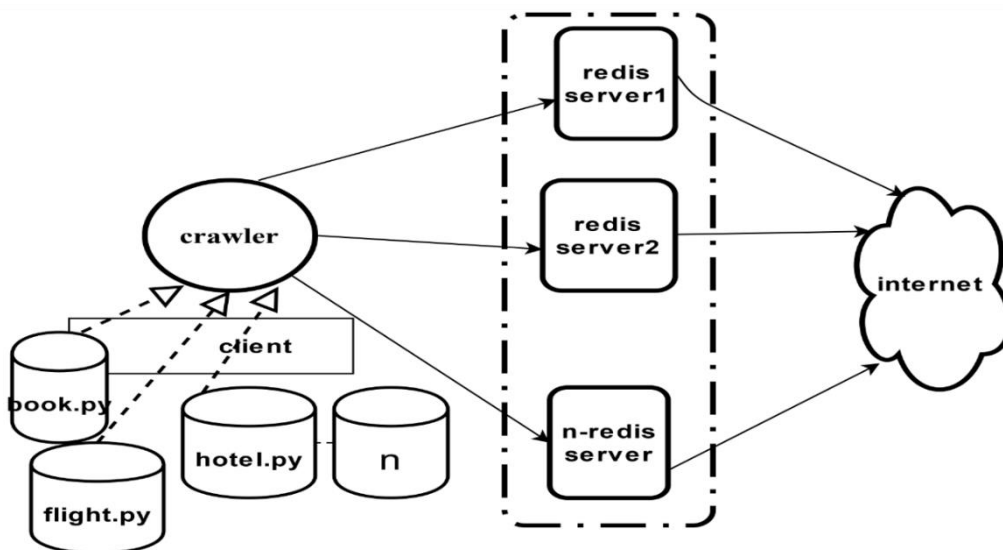


Figure 20: Distribution of proposed crawler based on Redis server

3.14 Job scheduling

In web crawling, to run multiple jobs simultaneously, the URL of a web page must be downloaded, parsed and the links captured from the web must be shared with all the crawlers. Because scrapy has no mechanism for link sharing even though it has scheduler, so the URLs are shared from the memory of crawler Figure (21) show the steps wise implementation of job scheduling. The task of job scheduler is to prevent overloading the websites. Web pages are popped out from the frontier first-in-first-out way. We have considered this as our baseline assumption. Crawling in breadth-first fashion is implemented as URL and server-based. It is proved in [12] that it yields promising results. Following steps are performed in Job Scheduling.

Step 1. To start crawling, scrapy send schedule request to message to a crawler.

Step2. As the crawler receives the request it starts crawling. From the Redis URL queue, a URL is selected and send as a request to the scheduler.

Step 3. Scheduler receives a request of URL, it sends this to Redis (request queue), and then again contact (request scheduled) is made with scrapy.

Step 4, 5. Now the associated webpage is to be downloaded, for this request is popped from the top of a request queue, and downloader on receiving the request, download the request page.

Step 6, 7. Downloader after getting the contents of a page to submit the page to the crawler.

Step 8,9. Crawler parses the webpage, collect the new URLs and send a new list of URLs to Redis pipeline. Redis pipeline sends new URLs to Redis queue. One another advantage of Redis is multiple jobs are separated using unique keys. So, jobs are not mixed.

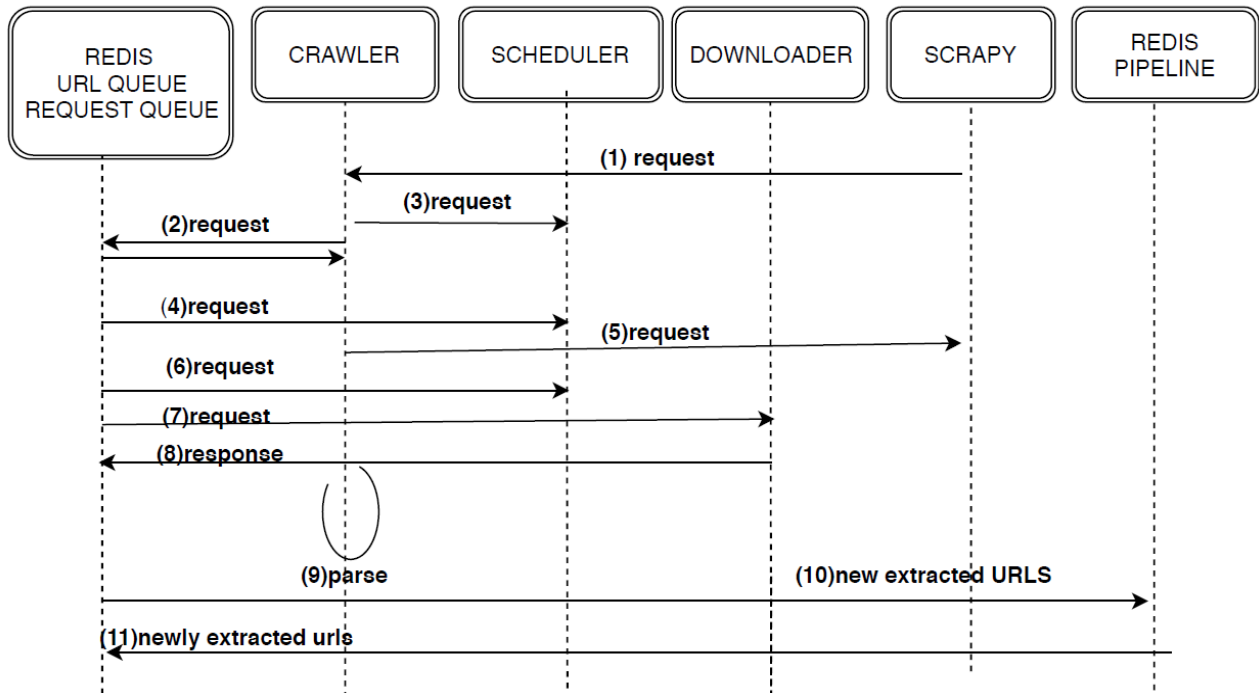


Figure 21: Job scheduling in proposed crawler

RQ scheduler is the lightweight solution used for job scheduling, it is used on top of Redis. With this scheduler only requirement is to create scheduler object with python.

3.15 Implementation of Redis and beautiful soup

Scrapy is a fast and powerful web extraction tool for structured web extraction. It is easily extensible and portable in Python. It is used as an application framework for writing web crawlers. Data can be saved into any format using scrapy. Any type of data can be scraped using scrapy selectors. At any time spidering can be closed using CloseSpider command. BeautifulSoup is implemented with Scrapy for parsing HTML responses in Scrapy callbacks. Usually, BeautifulSoup and Scrapy are used alternatively but the proposed crawler implemented both for a beautiful soup to parse and prettify data, while extraction is made possible by scrapy.

Distribution is made possible using a Redis server because Redis is exceptionally fast. It is not constrained to a single data type. Redis operations are atomic. Suppose two clients are simultaneously active with one resource, Redis always receive the updated value. Following advantages of Redis are implemented in a proposed crawler

1. Multiple Redis servers are operational, to provide fault tolerance. Redis can replicate data to any number of slave's systems.

2. Redis rarely save data on disk, so the applications like web crawling where time is the performance measure, Redis is the best choice for multiple read and write at a time.
3. It is easier to represent complex data structures in simpler forms.
4. Redis server has an efficient pipeline system. For example, suppose a client is sending multiple requests to the server, the client can have replied in a single step.
5. Redis is a secured server that demands authentication.

CHAPTER 4

CREATING ALGORITHMS FOR SMART DISTRIBUTED FOCUSED WEB CRAWLING THAT CAN AUTOMATICALLY PARSE, PROCESS, AND INTERACT WITH FORM-BASED SEARCH INTERFACES.

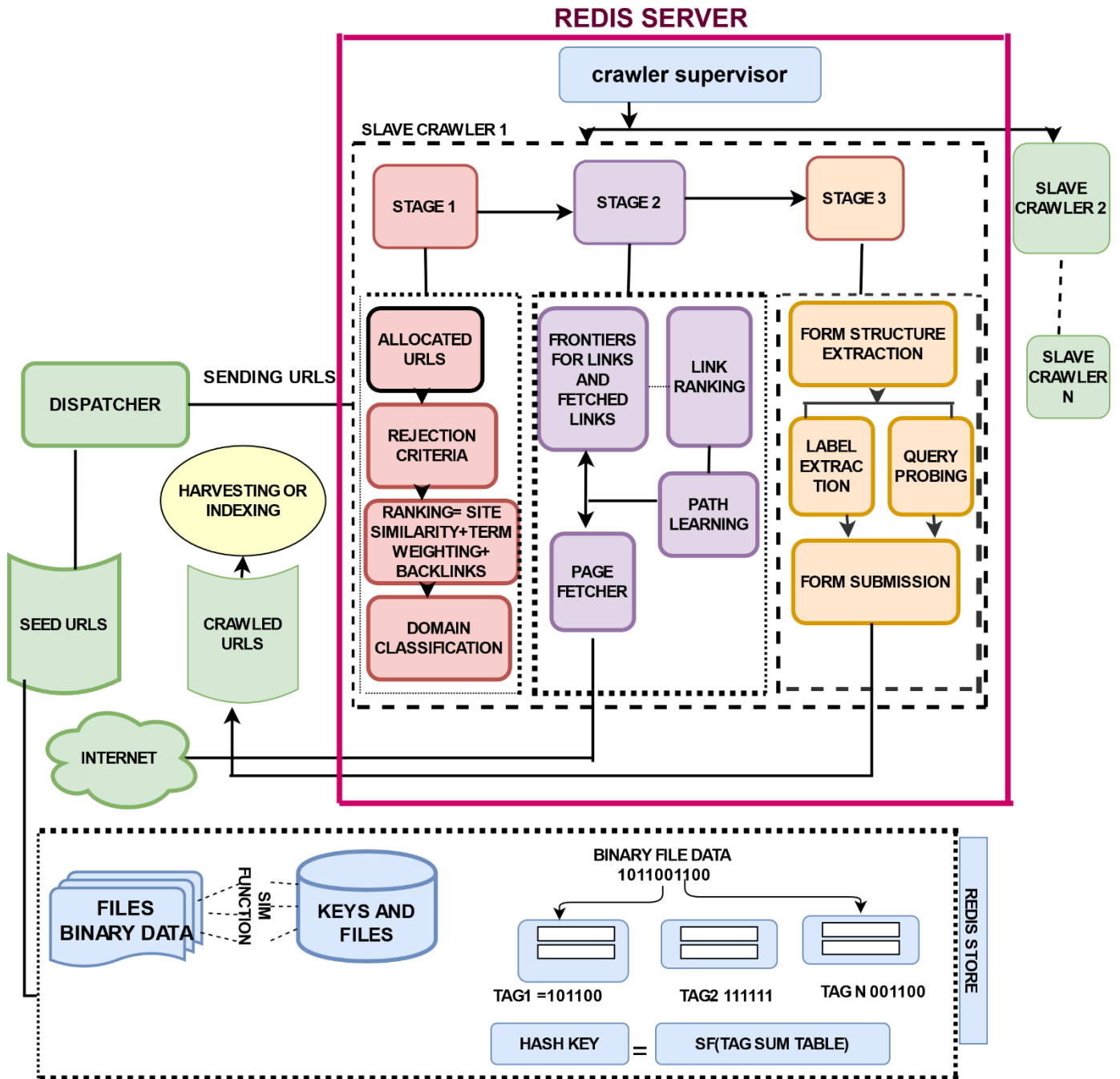


Figure 22: Show the detailed working of the proposed crawler

Figure 22 shows the working of components of the proposed crawler. The following sections explain the algorithms of each part. The first step in web crawling is to initialize the seed set. The seed set consist of a list of URLs from where the crawler starts the crawling. Frontier has this list of URLs. These URLs are dispatched to the other crawlers by the dispatcher. following steps are performed by the dispatcher.

4.1 Dispatcher algorithm

Goal: It is responsible for dispatching URLs to be crawled to the crawl supervisor.

Output: It collects results from the supervisor

1. Initialise the frontier and get URLs.
2. Dispatch URLs to crawl supervisor.
3. Collect fields. (fields are the data that is to be retrieve from the page)

4.2 Parsing algorithm

When the URL is encountered it is parsed for the desired fields. The proposed crawler is implemented with Beautiful soup. And under beautiful soup component. **Urllib.parse is used for URL parsing. It is used to parse the forms as well.** Following steps are performed by the parser.

Goal: Extract particular information

1. initialize request library
2. initialize html parser
3. `html.parser .feed (data) .` (this step is performed to feed parser with data
4. find tag that corresponds to the original html object in document.
5. Get URLs, go to baseline URL.
6. Find forms .
7. Follow rules as defined in figure (17).
8. Parse forms for fields.

This crawler is focused towards hidden web crawling, so for a URL to be a part of hidden web crawling the first condition to satisfy is the availability of a form tag. But not all URLs with the form tag can have searchable forms. The crawler has to find those forms which on being filled with suitable values to generate a valid response. The following algorithm defines the rules for the URLs that can not be included IN further search.

4.3 Algorithm for rejection criteria

1. Get the homepage of url.
2. Search for the <form> tag.
3. If crawler do not find any <form> tag, drop the URL
4. If crawler found the <form> tag. Then extract the attribute type. If the attribute type is not in repository , drop the URL
5. If crawler found the <form> tag, and extracted attribute type matched in a repository. But the attributes < 3, drop the URL
6. If the number of attributes is >3, but the submit button is not found, drop the URL
7. If there exists <form> tag, and attributes are similar to the repository, and submit button is also there. But button marker is not present drop the URL
8. If there exists <form> tag, and attributes are similar to the repository, submit button and button marker is are present. Keep this url
9. If there exists <form> tag, but crawler found login, drop the URL.
10. If there exist <form> tag, but crawler found registration, drop the URL
11. If there exist <form> tag, but crawler found subscribe, drop the URL
12. If there exist <form> tag, but crawler found mailing list subscription, then consider this page as non-searchable.

The crawl supervisor is given the list of searchable URLs. To reduce the time of crawling, the rules for rejection are followed at the frontier level. If the URL is considered searchable only that list if forwarded to crawl supervisor.

4.4 Algorithm of Crawl supervisor

The crawl supervisor assigns work to the worker crawlers. At starting the crawlers are assigned with metadata and number of URLs they are assigned to crawl.

Goal: Search URLs of seed database

1. If frontier has URLs <100.
2. Pick a hidden web site from site database or seed sites.
3. Extract links.
4. Download page.
5. Classify a page.
6. If found relevant according to rules.
7. Extract unvisited pages from the step 3.

4.5 Algorithm of Learning

1. Initialize site and link ranking
2. A new website (X) is encountered, extract [U, A, T].
3. For each URL the queue with sites is ordered using a similarity model with respect to [U, A, T].
4. Extract the links from X.
5. Links are saved in the link queue. The link queue is ordered using the similarity model with respect to [P, A, T].
6. Check for searchable forms by following rules.
7. If the form is searchable extract path, anchor and text.
8. With this, the information in parameter learning module in stage 1, and link ranking in stage 2, is updated. And new features are reflected in these two modules.
9. The crawler has reached the threshold of 0.8, i.e 80 new URLs and 0.1, i.e 100 new forms.

4.6 Algorithm of Similarity

1. Calculate term frequency i.e frequency of a word in the document.
2. Calculate inverse document frequency i.e common or rare words in document. closer to zero value indicate the more common the word is.
3. Find number of backlinks i.e SF using external _URLs = list of URLs, if the website has appeared in other links value is 1, zero otherwise.
4. Find value of ranking reward ϵ . It is sum of weighting and SF.
5. Open file in directory, scan through it at word level.
6. Look for matches of tags, if a match is found skip counter is decremented.
7. Store count of match in sum table (ST). Hash is function of ST entries (path, file size, file name, keys). Tag counts make up the sum table entries, which are then combined to form a hash key.
8. Get distance of sum tables. It is the sum of the absolute values of the differences between their entries.
9. If this distance of sum table is zero files are totally identical.

4.7 Algorithm of Ranking

A hidden web source is said to be ranked if it returns the top k sources. Keeping the assumption simple as [145], the ranking performed by this crawler is independent of queries. The sorting is done first in the first phase of crawling.

1. Extract the new coming URL for U, A and T.
2. Identify the domain of the web page under consideration.
3. Order the site frontier according to the similarity.
4. Similarity is computed as the cosine similarity of vectors.
5. Calculate the out of site links for the encountered URL.
6. Calculate the term frequency-inverse document frequency.
7. Calculate the ranking using the formula $(r_j) = (1 - w) \cdot \delta_j + w \cdot (\epsilon) / c_j$. where c_j is the factor of the network.
8. Repeat steps 1-8 for 'n' number of web pages.

4.8 Building a Naïve Bayes classifier

Term frequency is defined as how often a word occurs in a document. The more often it occurs in web page more important it is assumed. Term frequency is a ratio of one word with all the words in the document. Document frequency is how often occur in an entire set of documents. i.e. all of the web pages in DMOZ data set. In every webpage, there are few words like "a", "the", "and" etc appear very frequently. these words are discarded pre-processing. Frequencies are distributed exponentially so log values of Tf-idf are considered. It is computed for every word in the corpus.

For a given search word, documents are sorted based on their scores and the results are displayed. This system is designed to first classify URLs into the hidden website or non-hidden website by checking the <form> tag in the source code of the webpage. If the form tag is not present it will discard the URL. The DMOZ dataset has URLs in groups. From each group, URLs are parsed into tokens. First, the top-level domains such as “.com”, “. co. in”, “.gov” are excluded. Stemming is performed on terms. Then Tf/IDF is computed for the terms to construct a feature vector. We have taken eight categories from the web browsable dataset for the training of Bayes classifier. Bayes theorem is applied to compute the probability of the URL belonging to domain D_i . maximum a posteriori hypothesis is used for training.

$$H_{max} = \arg \max P(V|h) * P(h) \quad (15)$$

(where V is the vocabulary for each group in the dataset)

Each URL is split into tokens. Each token is checked with vocabularies in training data. Value of token is computed w.r.t to each vocabulary. Token will either match equally or partial.

VT = value of token

TC= token count

EQ = equal match

PM= partial match

pp = prior probability

lp = likelihood probability

$P(D_i|URL)$ = posterior probability of URL to be in D_i

$P(URL|D_i)$ = likelihood probability of URL to be in D_i

P_{D_i} = prior probability of D_i

$$VT_i = EQ + (0.5 \times partmatch) \quad (16)$$

$VSD_i =$ total count of words in vocabulary for domain D_i , where ($i= 1,2,3,4,5,6,7,8$)

$n =$ upper bound of the token count, it denotes the highest number of tokens a URL has.

$$P(URL|D_i) = \sum_{t=0}^n TD_i / VSD_i \quad (17)$$

Prior probability and likelihood probability are used to know if the URL belongs to a domain. According to the dataset, a prior probability is assumed for each domain.

pp = ratio of the total number of URLs in each domain to the total number of URLs in the dataset.

The probability for any URL to be classified under a domain (D_i) is computed as follows:

$$P(D_i|URL) = P(URL|D_i) \times PD_i \quad (18)$$

4.9 Building SVM Classifier

Almost all the real-world web data have linear inseparability. Support vector machine (SVM) is used to classify the blocks, and K- fold cross-validation is used for evaluation. Under the soft margin formulation, the linear kernel-based SVM classifier makes a certain number of mistakes and keep the class margin (CM) as wide as possible to correctly classify the points. It is expected that the system must choose a decision boundary that perfectly separates the features to avoid overfitting. Under soft marginal formulation, SVM is allowed to make mistakes to keep the margin wide. In this way, other points can be still be classified correctly.

$$L = \frac{1}{2} \|w\|^2 + v(\text{number of mistakes}) \quad (19)$$

Hyperparameter v chooses the trade-off between maximizing the margin and minimizing the mistakes.

- If v has a small value, classification mistakes are given less importance. More focus is given to maximize the margin.
- If v has a large value, the focus is more on avoiding misclassification.

More penalty is incurred by the points which are far away on the wrong side of the decision boundary.

For every data point x_i , there exists a slack variable ξ_i .

- $\xi_i =$ distance of x_i from the CM , if x_i is on the incorrect side of the margin,
- $\xi_i = 0$, if x_i is on the right side.

Each x_i has to satisfy the constraint of:

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i \quad (20)$$

The L.H.S of the equation is the confidence score denoted by CS.

- For $CS \geq 1$, the classifier has classified the point correctly.
- For $CS \leq 1$, the classifier did not classify the point correctly, and a penalty of ξ_i is incurred.

Each point P is represented by $P(x,y)$, ϕ is transformation function for point P as

$$\phi^{(P)} = (x^2, y^2, \sqrt{2xy}) \quad (21)$$

Minimization function is defined as :

$$L = \frac{1}{2} \|\bar{w}\|^2 + C \sum_i \lambda_i y_i (\bar{w} \cdot \bar{x}_i + b) \geq 1 - \xi_i \quad (22)$$

$$L = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i \cdot x_j \quad (23)$$

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \quad (24)$$

$$(P_1, P_2) = \langle \phi(x_1 y_1), \phi(x_2 y_2) \rangle \quad (25)$$

$$k(P_1, P_2) = x_1^2 x_2^2 + y_1^2 y_2^2 + 2x_1 y_1 x_2 y_2 \quad (26)$$

$$K(P_1, P_2) = (x_1 x_2 + y_1 y_2)^2 \quad (27)$$

$$k(P_1, P_2) = \langle P_1, P_2 \rangle^2 \quad (28)$$

In real-world web data, it is difficult to find exact similar data. So, we have kept the notion of similarity as to how close the points are. The main takeaway from this is we have implemented linear classification in higher-dimensional space.

Similarly, in the case of KNN, to work with maximum separability, for example, a dataset has N number of classes. μ_b is the mean vector, where $b = 1, 2, 3, \dots, N$. let x_b be the total number of samples.

$$x = \sum_{b=0}^N x_b \quad (29)$$

$$M_P = \sum_{b=1}^N \sum_{c=1}^{X_c} (y_c - \mu_b)(y_c - \mu_b)^T \quad (30)$$

$$M_Q = \sum_{b=1}^N (\mu_b - \mu)(\mu_b - \mu)^T \quad (31)$$

$$\mu = \frac{1}{A} \sum_{b=1}^N \mu_b \quad (32)$$

Distance of all instances is measured from each other using Euclidian distance metric. The instance with maximum distance is selected and is called training distance. If the boundary is 1.5 or 2 times of training

distance, it indicates classes are closer to each other. The approach has implemented non-exhaustive cross-validation. Under which k- fold cross-validation is implemented. For our approach, the value of K=5 comes out to be most suitable. With the aim of maximizing the prediction accuracy, non-perimetric neighbourhood component analysis is used for selecting features. After the domains are classified as relevant, using the varied queries forms ate submitted. If the form is correctly submitted its status code is 200. Precision, recall and F1 score are computed using SVM and KNN algorithms. Finding the ideal value of k is fixed it depends on how suitable it is to the dataset. If the value of k is decided small, it will make the crawler more blind. That is low bias and high variance. If the value of k is set high, it will make the algorithm more flexible. The output is calculated as the class with the highest frequency from the k most similar instances. It is suggested in [146] that for an odd number of classes, the value of k should be odd. If the number of classes is even the value of k should be chosen even.

4.10 Algorithm to find similar domains

Let URL be denoted by U, Domain with D, and subdomain with SD and content similarity with CS.

1. Crawl U, collect all the links present.
2. Check for search ability using rules.
3. If searchable find subdomains using beautiful soup.
4. Create dictionary for each SD.
5. Select terms above threshold.
6. Find average of top k similar terms.

$$a. CS = Average\left(\left(\max_k \left| \frac{v_i \cap S}{|S|} \right| \right)\right)$$

This approach is similar to [147]. To determine the closeness using similarity, a metric is required to determine less or high closeness. This research is based on finding the high similarity of values that reach near 1. For semantic similarity similar approach from [141] is used, but with the Redis plugin. During form submission, Pre-Query identifies web databases by analysing the wide variation in content and structure of forms. Post-Query approach identifies web databases from the retrieved results by submitting probing queries to the forms.

4.11 Algorithm for pre query

Input: relevant data to query

Output : Retrieved results from the database

1. If (query and input are same)
2. Display results (current)
3. Else
4. Search words and extract forms
5. Display results

4.12 Algorithm for post query

1. Extract the query form of the page
2. Changes in source code
3. User query processing
4. Create dynamic file

4.13 Form identification and analysis

After the crawler has identified the form, these are analysed to explore the form elements. Each form is equipped with text, HTML elements and controls. Text and HTML elements are not undertaken because these correspond to the structure of form only. Controls can either be bounded or unbounded. The proposed approach is based on bounded controls only. The following table shows the domain of experiment:

Table 10: Search term description as per domains

Domain	Description
Auto	Used car search
Book	Book search
Flight	Airfare search
Hotel	Hotel search
Music	Music CDs search
Product	Household product search

4.14 Form structure extraction and form-filling

1. Get URL from the frontier
2. Identify if it is searchable or not
3. If it is searchable, extract size of form.
4. If size is less than 3kb, discard the form.
5. Parse form
6. Extract control element, label and domain.
7. Form processor associate suitable value with each control from the repository.
8. The values of repository are initialized before the crawler bootstrap and after the completion of one cycle.
9. After the first run is over the obtained pages are collected to analyse.

4.15 Job scheduling algorithm

1. Crawler is initialised with request message from scrapy. To start crawling, scrapy send schedule request to message to a crawler.
2. Crawler upon receiving request start crawling by picking URL from the Redis URL queue
3. Send URL as a request to scheduler.
4. Scheduler receives a request of URL sends this to Redis (request queue),
5. Scheduler and then schedule request with scrapy.
6. Now the associated webpage is to be downloaded, for this request is popped from the top of a request queue, and downloader on receiving the request, download the request page.
7. Downloader after getting the contents of a page to submit the page to the crawler.
8. Crawler parses the webpage, collect the new URLs and send a new list of URLs to Redis pipeline. Redis pipeline sends new URLs to Redis queue.

4.16 Configuration

The system hardware environment includes: CPU is Intel®, Core™ C5-7200@ 2.50 GHZ 2.70GHZ, with installed RAM-12.0 GB, and Redis 3.0.509. the crawler is implemented in python. The internet speed during the experiment was 50-100mbps.

4.17 Evaluation

This crawler has to first check if the page belongs to a hidden web or not, by following the rules mentioned in the domain classification section. After the seed database URLs are checked for <form> tag, the crawler has to pull the contents of the webpage using the URL. The request library makes use of HTTP within the python program. BeautifulSoup can extract any type of data from a webpage. After the HTML Markup's are removed page is saved for further processing. BeautifulSoup is combined with urllib3 to work with web pages. Another way is to download a copy of the webpage then use it locally. BeautifulSoup has a feature called "prettify", in which all the unnecessary tags can be dropped. We have selected 6 domains for a dataset. This dataset will be used to run machine learning algorithms.

Initially, the DMOZ dataset is used. The dataset is cleaned by excluding the non-responsive web pages. The performance of the classifier is measured using a confusion matrix. Rows of confusion matrix denote actual class, while column indicates classes predicted by SVM and KNN classifiers. We have computed accuracy for each class. The average of each class denotes the performance of the classifier. The performance metrics are precision, recall and f1. Precision is the classification of a portion of web pages that are relevant to the class. It means how correct the system is to reject the web pages that are not relevant. The recall is how correctly the classifier can find relevant documents.

4.18 Experimental setup

We have selected eight domains from the dataset. This dataset will be used to run machine learning algorithms. This dataset contains 260000 associated URLs. Initially, the DMOZ dataset is used. In our approach, the content of the web pages is not fetched. The feature vector is constructed based on URL only.

The performance of the classifier is measured using the confusion matrix. Rows of confusion matrix denote actual class, while column indicates classes predicted by SVM. We have computed accuracy for each class.

$$precision = \frac{true\ positive}{true\ positive + false\ negative}$$

$$recall = \frac{true\ positive}{true\ positive + false\ negative}$$

$$f1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Classifier prediction is either positive or negative. While true and false conclude if the prediction is correct or not. The classifier has to do two tasks. First, it will classify to which class the URL belongs. From 960000 URLs, the crawler has selected 673629 URLs which falls into the hidden web according to the rules in table [1]. For training, 80% of the total URLs are used while the rest of the 20 % is for testing. Once the URL is correctly classified. The next task is to fill in the form values. For this, a query is needed to be generated. Once the crawler submits the form. We have used k nearest neighbour and SVM classifier to check the accuracy of the form submission. When the crawler will submit the form, the pages will have the following submit status. The efficiency of form submission of hidden web crawlers depends on the number of available URLs. Cross-validation method estimates the efficiency of the learning model. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation. The macro average is the harmonic mean of the precision, recall and F1Score. It is computed to know the overall performance of the system with various sets of data. Varied values of testing and training have been used. Once the URL is correctly classified. The next task is to fill the form values with correct values. K nearest neighbour and SVM classifier is implemented to check the accuracy of the form submission. The submission status 200 shows that the system had submitted the form.

Table 12: Status code and their description

Status code	Description
200	OK
400	Bad request response status code
401	Unauthorized
403	Forbidden client error status response code
404	Page not found
405	Method Not Allowed response status code
413	Payload too large
414	URI Too Long response status code
500	Internal Server Error
503	Service Unavailable
524	A time out occurred

The analysis of the status code is required because when the crawler will submit the web page, the correct submission will yield a new URL. These URLs can be used for further analysis. Table 13 shows that for k=5, the weighted average of performance measures gives high values as compared to other values of k. On comparing the values of table 7 and 8, results are more promising for k=5. The total number of web pages with status code 200 is 13888, which makes the harvest rate of 27%. From the total harvested URLs, it is observed that only two URLs corresponds to status code 524. This is very fewer data to analyze for the machine learning algorithm. Table 17 shows that for k=5 in KNN, the value of accuracy is high as compared to the SVM algorithm. The following tables 13-17 shows the result of experiments for precision, recall, F1 score, macro and weighted average, comparison of accuracy for 20%, 30%, 40% and 50% of testing data using SVM and k=2 in KNN.

Table 13: Comparison of Precision, Recall and F1 Score for varied values of K in KNN

Status code	Precision				Recall				F1- score			
	K= 2	K=3	K=4	K=5	K=2	K=3	K=4	K=5	K=2	K=3	K=4	K=5
200	0.86	0.88	0.88	0.87	0.96	0.94	0.95	0.95	0.91	0.91	0.91	0.91
400	0.36	0.33	0.44	0.44	0.28	0.25	0.22	0.29	0.27	0.29	0.31	0.35
401	1.00	1.00	1.00	0.86	1.00	0.65	0.79	0.60	0.78	0.79	0.72	0.71
403	0.94	0.96	0.82	0.98	0.90	0.97	0.88	0.93	0.83	0.96	0.84	0.96
404	0.76	0.77	0.76	0.78	0.60	0.62	0.60	0.59	0.67	0.69	0.70	0.67
405	0.69	0.71	0.71	0.73	0.91	0.80	0.92	0.83	0.77	0.73	0.82	0.78
413	0.25	0.23	0.34	0.25	0.06	0.15	0.10	0.13	0.16	0.19	0.17	0.17
414	0.17	0.25	0.00	1.00	0.08	0.24	0.00	0.19	0.12	0.28	0.25	0.32
500	1.00	1.00	0.00	0.00	1.00	0.11	0.00	0.00	0.22	0.18	0.10	0.00
503	0.94	0.91	0.90	0.92	0.82	0.86	0.86	0.86	0.87	0.89	0.88	0.89
524	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Macro average	0.72	0.73	0.51	0.65	0.61	0.58	0.53	0.53	0.62	0.60	0.62	0.56
Weighted average	0.91	0.91	0.91	0.92	0.92	0.92	0.92	0.93	0.91	0.92	0.91	0.92

Values are optimal when k=5 and the split of training and testing data are 40/60. The table 14 shows that the weighted average of precision is more when there is a 40/60 ratio of testing and training data. But the values of the weighted F1 score is more promising in case of k=5. The ratio of testing and

training data is tested for other values of k as well, but we found our approach working well for k=5. The results for k=2 and k=5 are presented while others are skipped due to space constraints.

Table 14: Computation of precision, recall and F1 score using SVM for variation of 20% to 50% of testing data.

Status code	Precision				Recall				F1- score			
	20%	30%	40%	50%	20%	30%	40%	50%	20%	30%	40%	50%
200	0.79	0.80	0.78	0.79	0.92	0.95	0.96	0.97	0.82	0.87	0.86	0.87
400	0.00	0.75	0.00	0.00	0.00	0.06	0.00	0.14	0.00	0.12	0.00	0.23
401	1.00	0.00	0.00	0.78	0.20	0.00	0.00	0.00	0.33	0.00	0.00	0.00
403	0.83	0.71	0.00	0.77	0.35	0.68	0.00	0.78	0.49	0.70	0.00	0.78
404	0.66	0.71	0.76	0.73	0.59	0.54	0.49	0.35	0.44	0.62	0.59	0.48
405	0.71	0.71	0.68	0.65	1.00	1.00	0.99	1.00	0.62	0.00	0.81	0.84
413	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.83	0.00	0.00	0.05
414	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
500	0.00	0.00	0.00	0.92	0.00	0.00	0.69	0.68	0.00	0.00	0.00	0.78
503	0.81	0.90	0.91	0.00	0.67	0.68	0.00	0.00	0.73	0.78	0.78	0.00
524	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Macro average	0.53	0.41	0.38	0.49	0.42	0.41	0.38	0.39	0.44	0.41	0.39	0.39
Weighted average	0.89	0.90	0.99	0.90	0.90	0.90	0.99	0.80	0.87	0.88	0.88	0.88

In the case of SVM, table [15] shows the value of precision and recall when testing and training data ratio is 20/80, 30/70, 40/60 and 50/50. The weighted average of F1 is the same for 30%, 40% and 50%. Similarly, for k=2, a weighted average is the same for F1 score when the value of k is 2.

Table 15: Computation of precision, recall and F1 score using KNN for variation of 20% to 50% of testing data for K=2.

Status code	Precision				Recall				F1- score			
	20%	30%	40%	50%	20%	30%	40%	50%	20%	30%	40%	50%
200	0.79	0.80	0.78	0.79	0.92	0.95	0.96	0.97	0.82	0.87	0.86	0.87
400	0.00	0.75	0.00	0.00	0.00	0.06	0.00	0.14	0.00	0.12	0.00	0.23
401	1.00	0.00	0.00	0.78	0.20	0.00	0.00	0.00	0.33	0.00	0.00	0.00
403	0.83	0.71	0.00	0.77	0.35	0.68	0.00	0.78	0.49	0.70	0.00	0.78
404	0.66	0.71	0.76	0.73	0.59	0.54	0.49	0.35	0.44	0.62	0.59	0.48
405	0.71	0.71	0.68	0.65	1.00	1.00	0.99	1.00	0.62	0.00	0.81	0.84
413	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.83	0.00	0.00	0.05
414	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
500	0.00	0.00	0.00	0.92	0.00	0.00	0.69	0.68	0.00	0.00	0.00	0.78
503	0.81	0.90	0.91	0.00	0.67	0.68	0.00	0.00	0.73	0.78	0.78	0.00
524	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Macro average	0.53	0.41	0.38	0.49	0.42	0.41	0.38	0.43	0.44	0.41	0.39	0.39
Weighted average	0.89	0.90	0.90	0.90	0.90	0.90	0.99	0.88	0.87	0.88	0.88	0.88

Table 16: Computation of precision, recall and F1 score using KNN for variation of 20% to 50% of testing data for K =5.

Status code	Precision				Recall				F1- score			
	20%	30%	40%	50%	20%	30%	40%	50%	20%	30%	40%	50%
200	0.80	0.86	0.86	0.85	0.96	0.95	0.96	0.95	0.87	0.90	0.91	0.90
400	0.00	0.51	0.43	0.57	0.00	0.26	0.22	0.24	0.00	0.34	0.29	0.34
401	0.00	0.93	0.90	0.75	0.00	0.70	0.73	0.71	0.00	0.80	0.81	0.73
403	0.66	0.85	0.83	0.78	0.71	0.82	0.63	0.60	0.69	0.83	0.71	0.68
404	0.72	0.78	0.81	0.79	0.46	0.57	0.55	0.57	0.56	0.66	0.66	0.60
405	0.72	0.72	0.73	0.72	0.98	0.83	0.81	0.85	0.83	0.77	0.77	0.78

413	0.00	0.26	0.26	0.33	0.00	0.13	0.14	0.16	0.00	0.17	0.18	0.22
414	0.00	1.00	0.46	0.50	0.00	0.18	0.13	0.05	0.00	0.30	0.21	0.10
500	0.00	0.00	0.25	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.06	0.00
503	0.91	0.89	0.92	0.88	0.69	0.81	0.85	0.80	0.78	0.85	0.88	0.84
524	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Macro Average	0.44	0.71	0.57	0.55	0.44	0.57	0.47	0.46	0.43	0.60	0.50	0.48
Weighted Average	0.88	0.91	0.91	0.91	0.90	0.92	0.92	0.91	0.88	0.91	0.91	0.91

Table 17: Comparison of Accuracy for KNN and SVM

Accuracy					
KNN					SVM
Percentage of testing data	K=2	K=3	K=4	K=5	
20%	0.89	0.88	0.89	0.9	0.89
30%	0.88	0.89	0.88	0.92	0.90
40%	0.90	0.80	0.90	0.92	0.90
50%	0.90	0.89	0.90	0.91	0.90

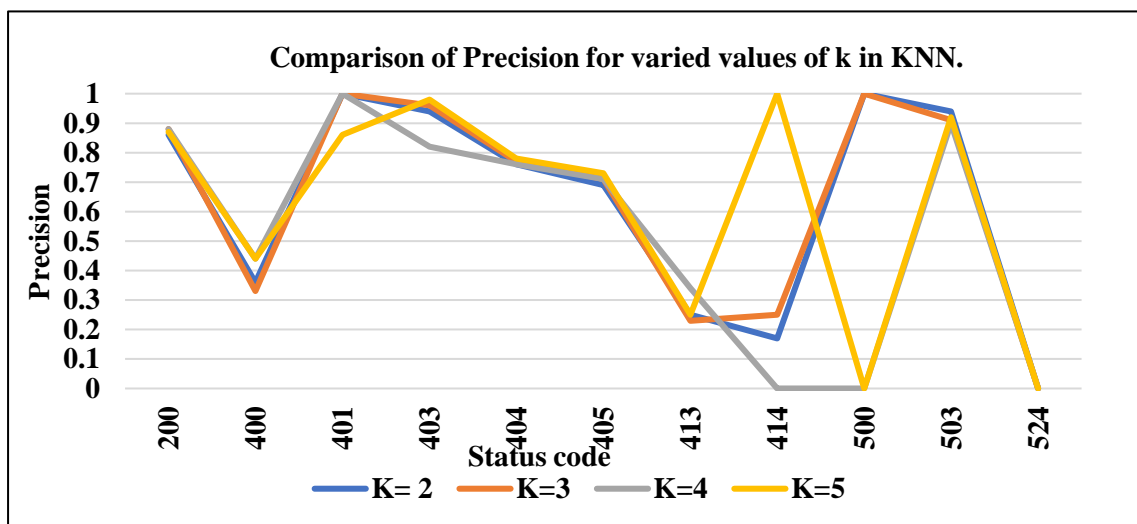


Figure 23: Comparison of Precision for varied values of K in KNN

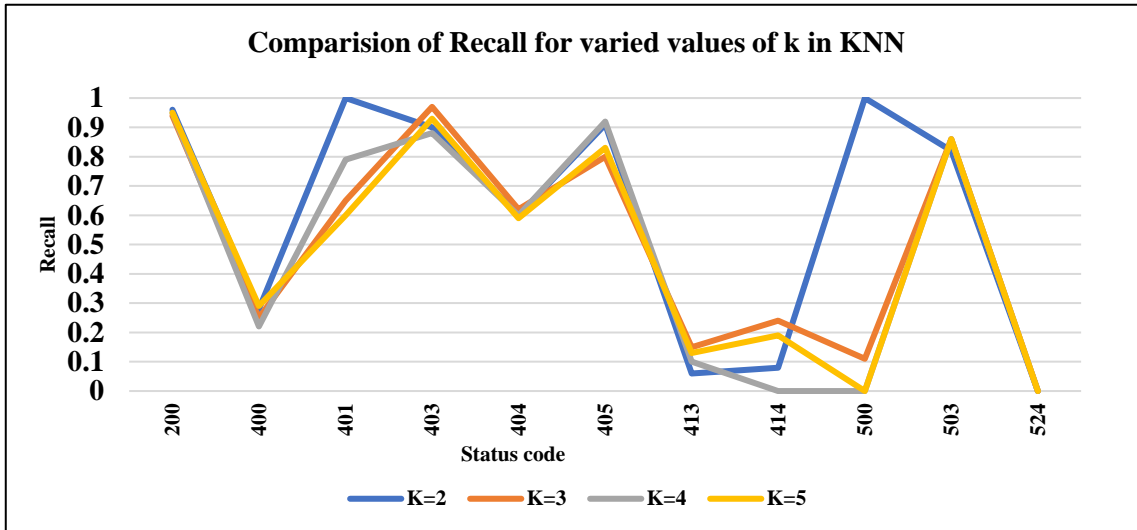


Figure 24: Comparison of Recall for varied values of K in KNN

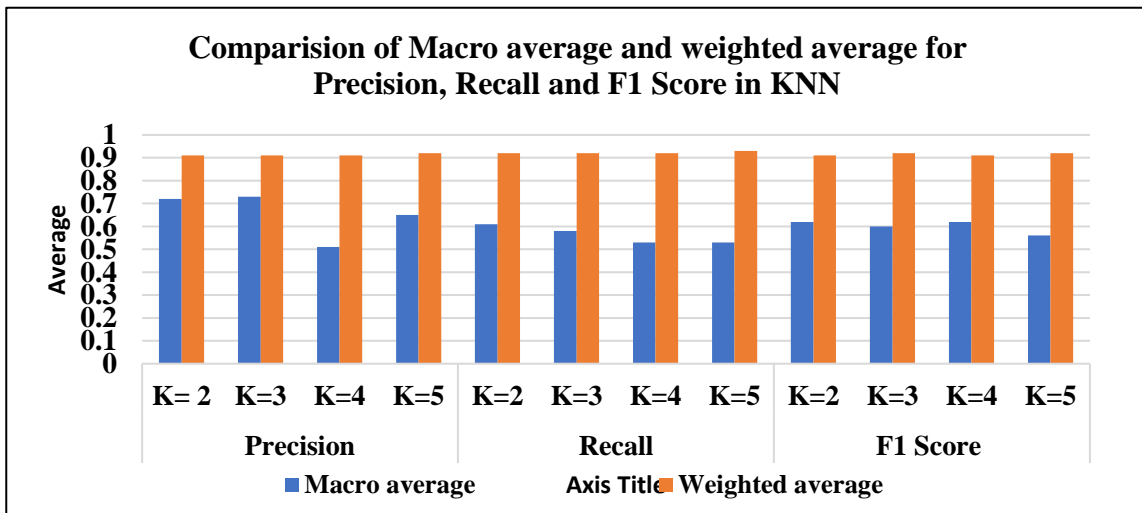


Figure 25: Comparison of macro average and weighted average for precision, recall and F1 in KNN

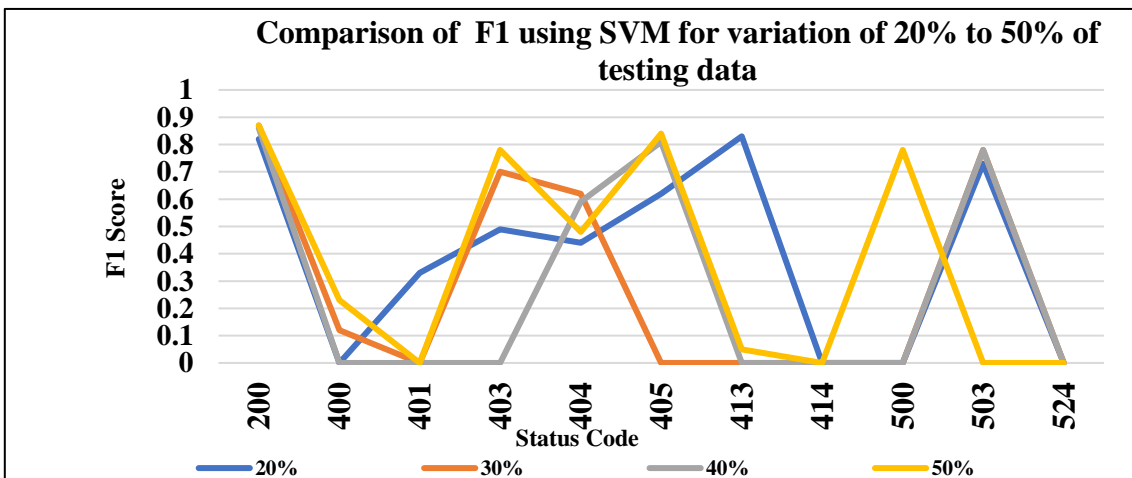


Figure 26: Comparison of F1 using SVM for variation of 20% to 50% of testing data

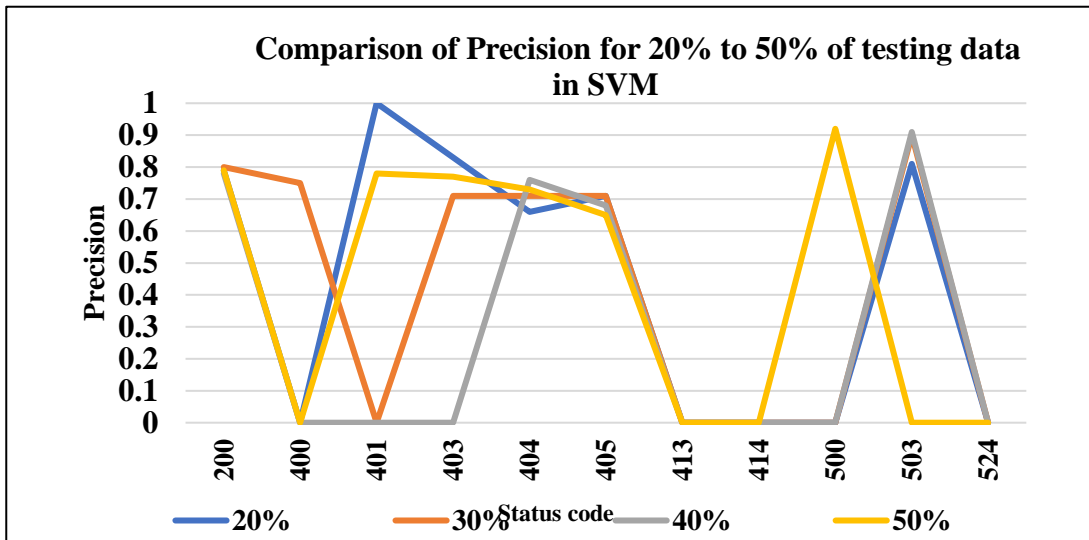


Figure 27: Comparison of precision for 20% to 50% of testing in SVM

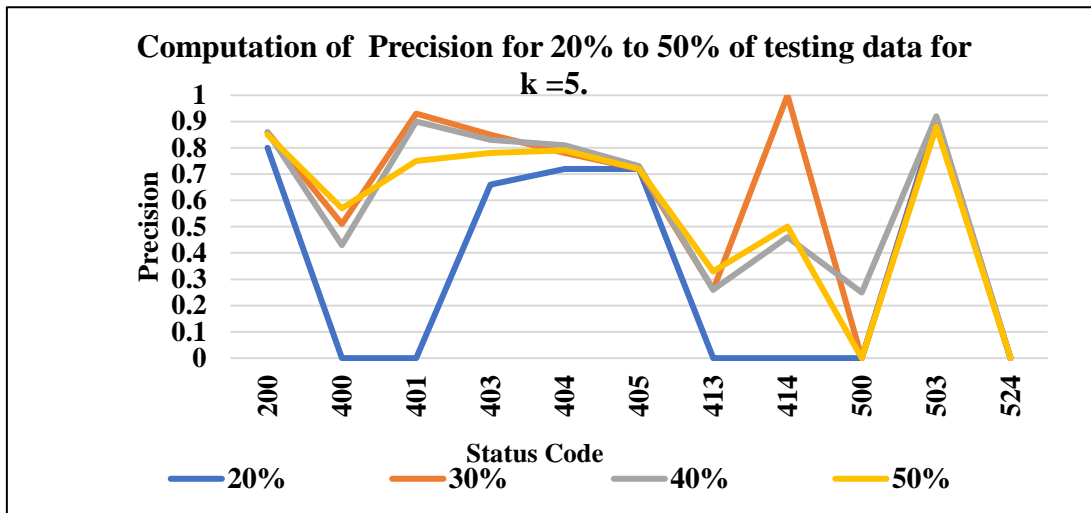


Figure 28: Computation of precision for 20% to 50% of testing data for K=5

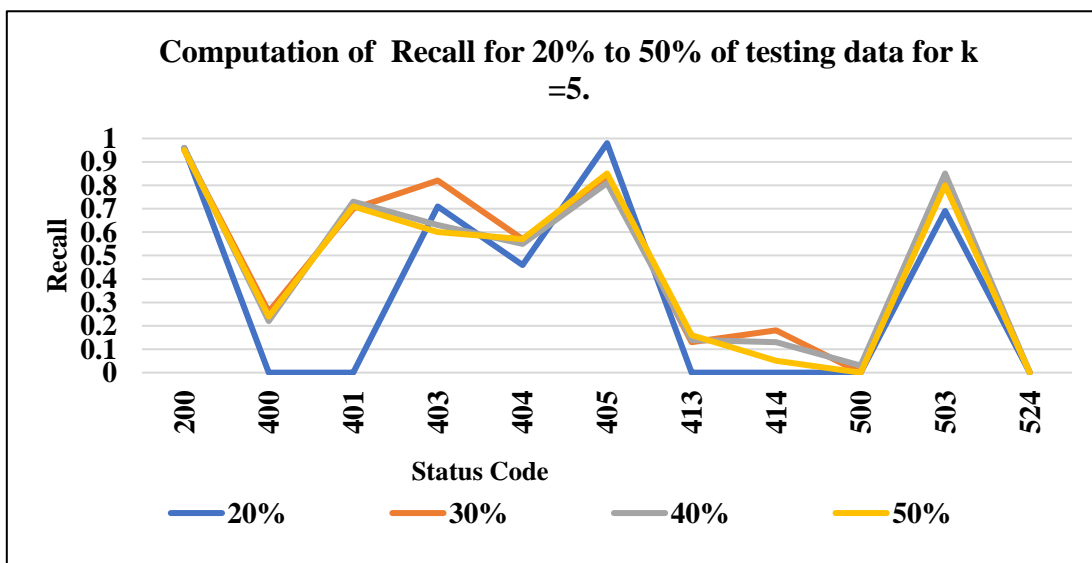


Figure 29: Computation of recall for 20% to 50% of testing data for k=5

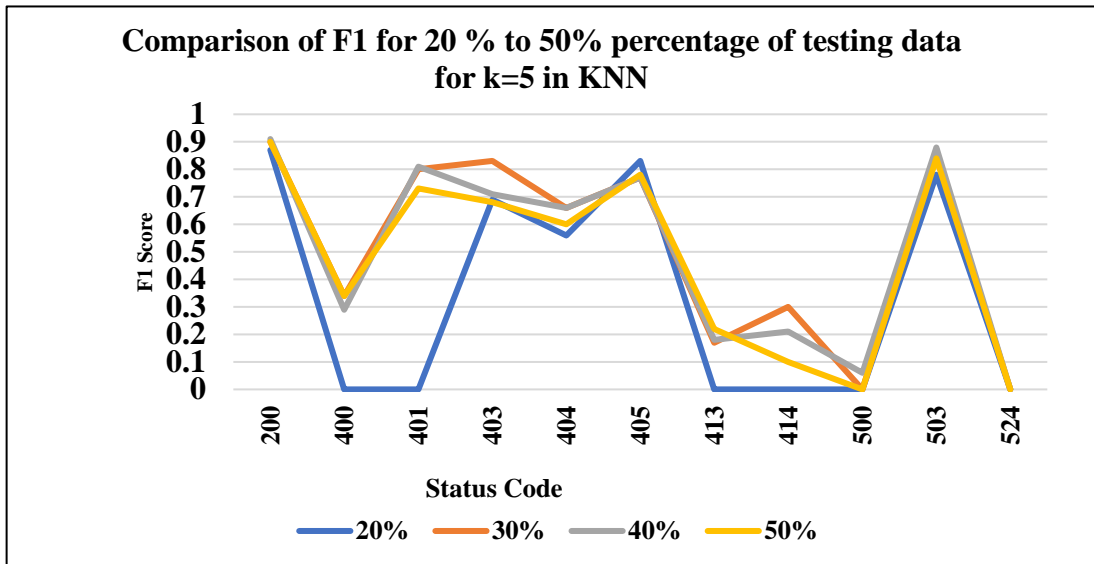


Figure 30: Comparison of F1 score for 20% to 50 % of testing data

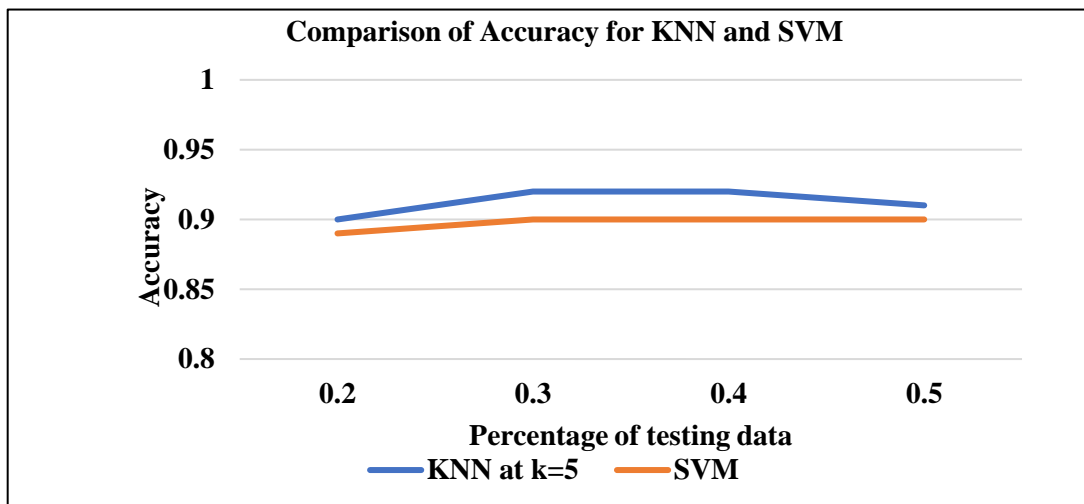


Figure 31: Comparison of Accuracy for KNN and SVM

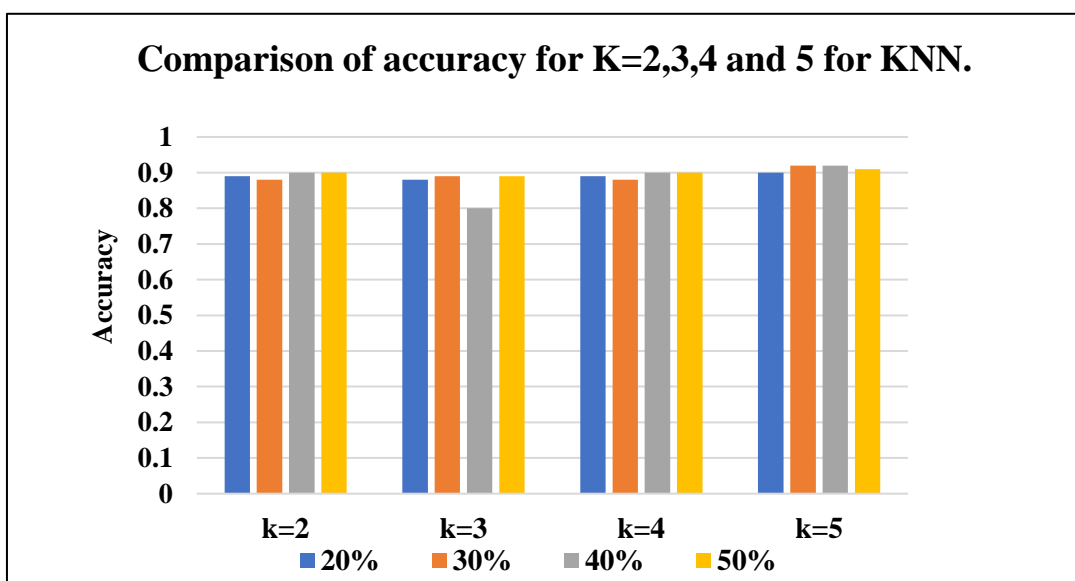


Figure 32: Comparison of accuracy for varied values of k

Figures 23-32, conclude that for the proposed approach KNN has performed better than SVM. Figure 33 shows that the proposed crawler has a high harvest rate as contrast to its pioneer contemporaries. The values of the status code as shown in tables 13-17, shows that the system has correctly classified the forms as well as submit them. Results are also shown for the ratio of testing and training data. For this approach for $k=5$ at 40% of testing, data gave promising results. On being compared with a focused crawler (FC), form-focused crawler (FFC), Enhanced form crawler (EEFC). The proposed crawler has a more than 10% high harvest rate than EEFC. There exist only a few crawlers that implement both pre query and post query approaches, ICHW also worked on both techniques. The rejection rules and stopping criteria's have impacted the harvest rate of the crawler.

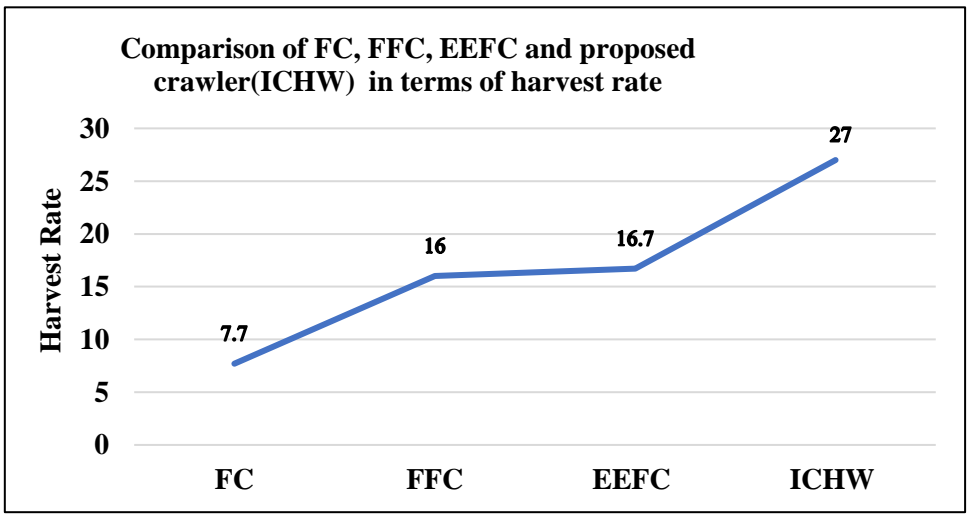


Figure 33: Comparison of FC, FFC, EEFC and proposed crawler in terms of harvest rate

4.19 Path Learning

The goal of path learning is to extract only those links which with minimum hops can lead the crawler to the hidden web databases. Some of the links are considered good, while others are discarded. Along with jasmine directory and amazon, 20 real websites from Alexa's list of top sites are exhaustively crawled to check at which depth most web pages are found. Our observation is similar to [148]. Below the depth of 6, the crawler was not able to find a considerable percentage of forms. The simplest reason for this is that form is designed for human interaction. And for this most of the times forms are put on upper levels. Due to this reason, the depth of the crawler is limited to 3. It is also observed that from the crawled URLs the number of URLs for book domain are high as compared to others. Figure 34 justify the observation. Backlinks also impact the performance of the focused web crawler. Following the connection between the web pages, crawler the good target pages. Features vector is constructed for FS and FL as explained in Equations 2 and 3. FL is calculated at each level. From a webpage, a huge number of feature vectors can be extracted. But due to length and space

constraints, the top 10 features are used and are constructed as explained in section 4.2. The good links are either immediate benefit links or delayed benefit links. Immediate benefit links are at level 1, while delayed benefit links are at levels 2 and 3. The next step is to compute the similarity between the FS and FL.

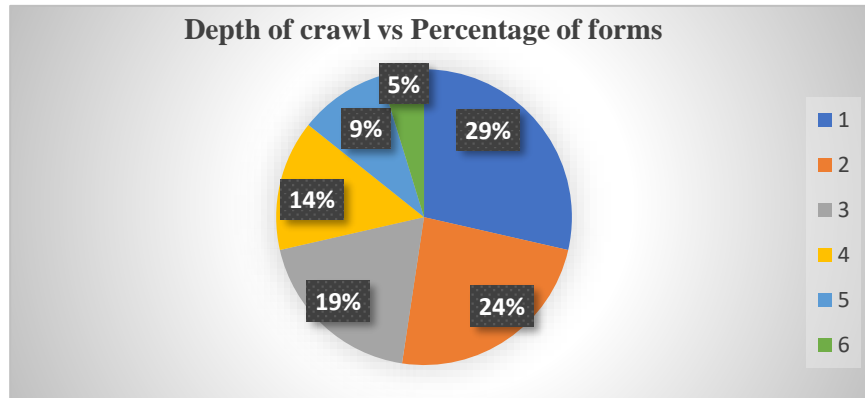


Figure 34: Depth of crawl vs percentage of forms found at particular depth.

4.20 Application of crawler as an approach for atmospheric emission.

Suppose the user has a goal to find a property with a good air quality index. Given f (Amritsar, Punjab), (Ludhiana, Punjab), (Jalandhar, Punjab) be the three cities for which search is targeted. Instead of using three different crawling nodes, the crawling is implemented as three different threads for each tuple. Let's assign $C1 =$ (Amritsar, Punjab), $C2 =$ (Ludhiana, Punjab), $C3 =$ (Jalandhar, Punjab). The location-based subdivisions of the cities are taken as the administrative divisions. Amritsar and Jalandhar have 5 administrative divisions whereas Ludhiana has 7 administrative divisions. Location-based crawling is done on these administrative divisions. Crawled data is combined for average pollution in each city. The goal is to find PM 10 and PM 2.5 values in administrative divisions. The crawler will crawl and parse the data from the real estate website and combine this with location-aware crawling. The traversing of the crawler is controlled using rejection rules.

The results will be useful for making the right investment in a property based on qualitative, relevant and empirical data. Also, suppose if a user is already living in any of the above-mentioned cities, crawling using this web crawler will help find similar properties and set a good value on their own. Users can also search for fair deals. Due to space constraints, the results regarding the submission of the form regarding each feature is not presented, moreover, most of the URLs belongs to the dynamic databases. Data are combined from both real estate and pollution URLs, by implementing expectation maximum clustering technique using a gaussian mixture model. Data normalization is

performed using MAX-MIN normalization. In this case, the expectation-maximization algorithm is implemented to find parameters. The parameters are defined as: M denote the sample of data points, μ is Gaussian distribution, Σ covariance, u is defined as input vector, 'I' denote possible curves, 'i' denote data points, C is Gaussian curve, w_{ij} is weighting factor of a feature vector, π denotes gaussian weight, θ is standard deviation and m is a number of data points in data set. Derivation of likelihood is as follows: Let θ be the random variable with binary values

$$\theta = P(I) \quad (26)$$

$$I - \theta = P(0) \quad (27)$$

$$\text{The likelihood is defined as } l(\theta) = \theta^{n_1}(I - \theta)^{n_0} \quad (28)$$

Taking derivative on both sides of equation (19)

$$\frac{\partial^2 l(\theta)}{\partial (\theta)} = n_1 \theta^{n_1-1} (I - \theta)^{n_0} - n_0 \theta^{n_1} (I - \theta)^{n_0-1} \quad (29)$$

$$= \theta^{n_1-1} (I - \theta)^{n_0-1} (n_1(I - \theta) - n_0 \theta) \quad (30)$$

$$= \theta^{n_1-1} (I - \theta)^{n_0-1} (n_1(n_1 + n_0) \theta) \quad (31)$$

If $\theta = 0$, or $\theta = 1$

$$\theta = \frac{n_1}{n_0 + n_1}$$

Let M data samples be denoted as $M_1, M_2, M_3 \dots M_n$, the maximum likelihood for the Gaussian model is derived as

$$\log l(\mu, \sigma) = \sum_{i=1}^m \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right) \quad (32)$$

$$= C + \sum_{i=1}^m -\log l - \frac{(x^{(i)} - \mu)^2}{2\sigma} \quad (33)$$

$$\frac{\partial \log l(\mu, \sigma)}{\partial \mu} = \frac{1}{\sigma^2} \sum_{i=1}^m (x^{(i)} - \mu) \quad (34)$$

$$= \sum_{i=1}^m \frac{1}{\sigma} - \frac{(x^{(i)} - \mu)^2}{\sigma^3} \quad (35)$$

$$\sigma^2 Ml = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_{Ml})^2 \quad (36)$$

Now estimation maximization for the Gaussian model is derived as follows. Suppose Y is Multinomial Distribution,

$$P(Y = k; \theta) = \mu_k \quad (37)$$

$$T \sim N(\mu_k, \Sigma k) \quad (38)$$

$$p(x = k, T; \theta, \mu, \Sigma) = \theta_k \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2} (z - \mu_k)^T \Sigma_k^{-1} (z - \mu_k)} \quad (39)$$

Expectation calculation: $p(x|z; \theta, \mu, \Sigma) = \prod_{i=1}^m p(x^{(i)}|z^i; \theta, \mu, \Sigma)$ (40)

Maximization calculation: $\max_{\theta, \mu, \Sigma} \sum_{i=1}^m \sum_{i=1}^m \sum_{k=1}^k q(x^{(i)}=k) \log(\theta_k \mathcal{N}(z^{(i)}; \mu_k))$ (41)

After applying the above-discussed technique, clusters of regions are formed according to the air quality index in Amritsar, Jalandhar and Ludhiana.

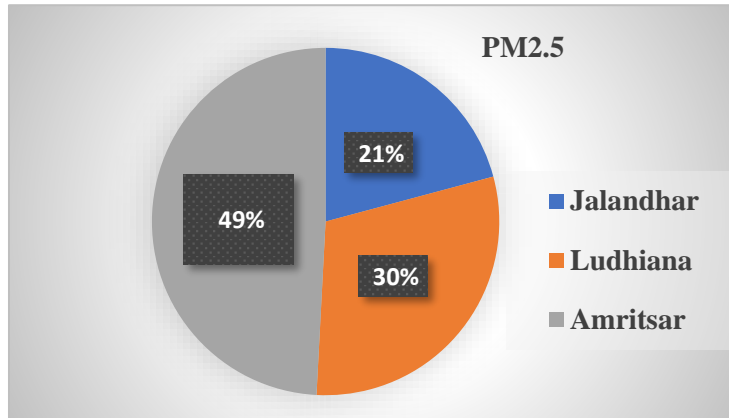


Figure 35: Comparison of PM 2.5 in cities of Punjab

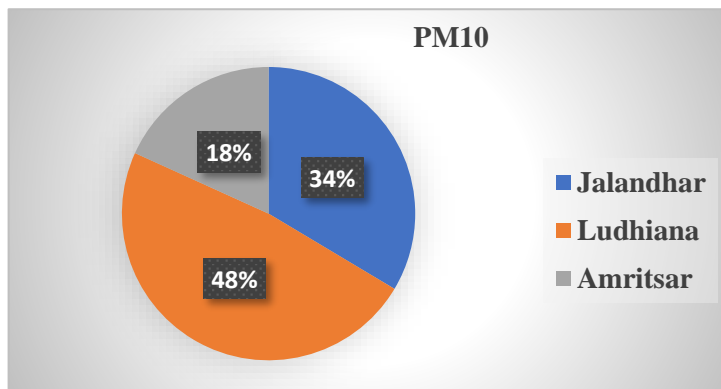


Figure 36: Comparison of PM 10 in cities of Punjab.

Further analysis could be made on the reason for the low air quality index. Due to space constraints, a tabular form of data is not presented, the above figures have shown the computed results of air quality in the three cities.

4.21 Comparative advantages

The proposed technique is one of its kind works that associate real estate data and air quality index to find a property in smart cities of Punjab. The crawler can be trained to be used for any other search terms. The results have shown that the proposed approach has a high harvest rate as compared to

existing techniques. This approach is scalable in terms of the growing size of the web, it is extensible as any third-party component for example indexer can be added. The ranking is a function of both backlinks and term weighting, due to which the chances of term bias is less. The crawler successfully saves itself from the crawler traps due to efficient stopping criteria's and accurately classify more status codes than [10]. The F1 measure of the proposed technique is higher than [149], as this technique has also implemented text clustering. One of another advantage of this technique is that it works with both GET and Post methods. In this way, the crawler can have a high number of URLs for analysis and indexing. The following table compares the running time and the number of searchable forms of adaptive crawler for hidden web entries (ACHE) and ICHW. There exists no technique as perfect that it can stop a crawler to fall in the spider traps. So, the intelligent rules of rejection are designed to prevent the crawler from falling it into infinite crawling loops.

Table 18: Comparison of running time and number of searchable forms

Domain	Running time of ACHE	Running time of proposed crawler without rejection rules	Running time of proposed crawler with rejection rules	Searchable form ACHE	Searchable form of proposed crawler
Property	Not included	7H 12 M	6H 39M	Not included	3809
Book	8H 21M	7H 21M	6H 58M	599	4589
Flight	7H 59M	6H 18M	7H 52M	1705	2843
Music	7H 59M	7H 00M	6H 58M	776	1447
Premier	Not included	6H 35M	6H 01M	Not included	668
Product	7H 50M	7H 28M	7H 48M	386	1999
Pollution	Not included	7H 26M	6H 20M	Not included	2002

The above table shows the running time of the proposed crawler is comparatively less than ACHE. Also, the number of searchable forms founds are more than ACHE. A goal of a crawler is to find maximum searchable forms in minimum visits, so the number of searchable forms without rejection rules are not included. The above results show the computation of results for the web forms that cannot be submitted due to their status code. If the status code is except the above-mentioned code, then it means that the form has been submitted correctly. In the testing phase, confusion matrix figures out the precision, recall and f1- score of the correctly submitted searchable form classification. During feature selection and top

k terms are required after performing stemming, stop words. For this cosine similarity is implemented and it is used in ranking for prioritising the URLs. But there is another issue that URLs are available on multiple websites. So, if a crawler will keep crawling the same URL again and again, it is a waste of resources. So to eliminate the duplicates, we have used simhash [141] technique. Simhash has used MYSQL as a data store. Table 19 and Table 20 show the number of forms submitted using GET method and POST method out of the total number of forms per domains.

Table 19: Shows the number of forms retrieved per domain.

Domain	Number of URLs	200	400	401	403	404	405	413	414	500	503	524
Book	13936	9969	55	71	24	463	2285	914	25	50	0	9
Product	886	496	0	0	240	115	0	0	0	0	35	0
Auto	4034	29	9	0	0	21	0	263	3	2	3707	0
Flight	6016	3075	254	0	0	338	0	0	57	12	2280	0
Hotel	911	398	0	0	17	452	0	0	15	2	27	0
Music	84	35	10	0	0	0	0	6	2	0	31	0
Premier	2	2	0	0	10	0	0	0	0	0	0	0

Table 20: Shows the number of forms submitted using the GET method.

Domain	Number of URLs	200	400	401	403	404	405	413	414	500	503	524
Book	9741	3078	0	0	0	13	2285	893	0	0	3471	1
Product	2	2	0	0	0	0	0	0	0	0	0	0
Auto	29	29	0	0	0	0	0	0	0	0	0	0
Flight	723	183	0	0	0	0	0	0	0	0	540	0
Hotel	0	0	0	0	0	0	0	0	0	0	0	0
Music	0	0	0	0	0	0	0	0	0	0	0	0
Premier	0	0	0	0	0	0	0	0	0	0	0	0

If the form is submitted with status code 200, it means that the form has been submitted with correct values. Sometimes the form is submitted but the response is not generated due to some reasons like

internal server error, service unavailable etc. For coverage, we have considered only those pages for which response is generated back.

Table 21: Shows the number of forms submitted using the POST method.

Domain	No of URLs	200	400	401	403	404	405	413	414	500	503	524
Book	7677	6891	3	24	24	450	0	21	25	2	236	1
Product	633	494	0	0	24	115	0	0	0	0	0	0
Auto	99	29	9	0	0	21	0	0	3	2	35	0
Flight	4422	2892	254	0	0	404	0	26	57	12	540	0
Hotel	959	398	0	0	17	452	0	0	15	50	27	0
Music	84	35	10	0	0	0	0	6	2	0	31	0
Premier	12	2	0	0	10	0	0	0	0	0	0	0

Figure 37 and Figure 38 shows the number of forms correctly submitted and the comparison of GET and POST method respectively. Our approach has worked better with POST methods. Which indicate the efficiency of the ranking algorithm and form submission method as explained in [148]. Table 22 shows the comparison of GET and POST method w.r.t to documents per domain and new documents.

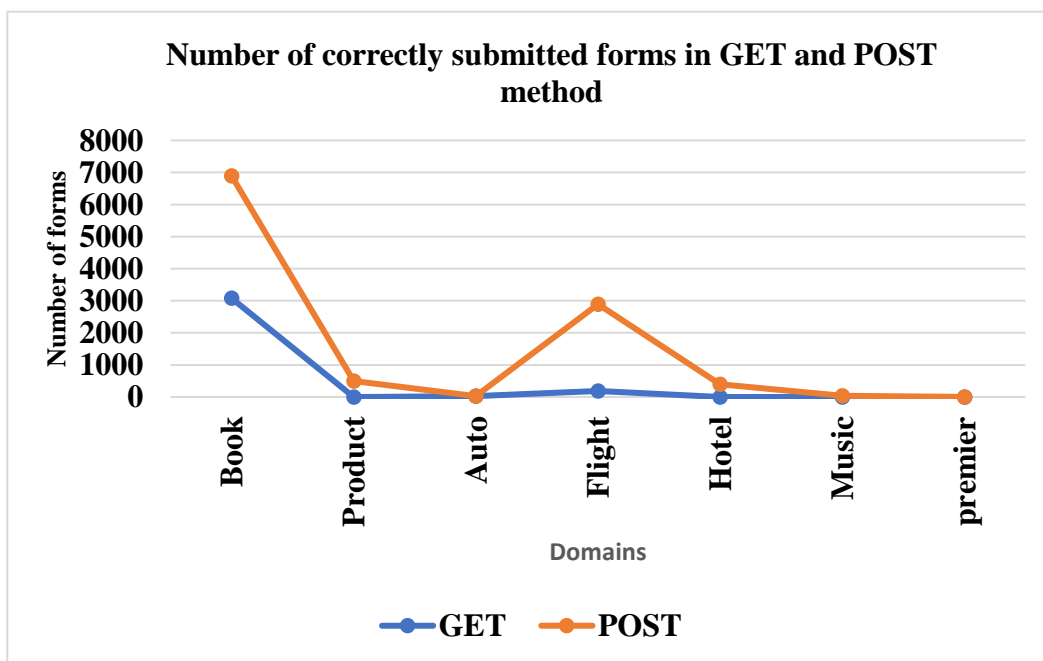


Figure 37: Comparison of coverage for GET and POST methods

Table 22: Comparison of GET and POST method w.r.t number of documents per domain vs new document captured

DOMAIN	GET				POST			
	L	Q	N	UJ	L	Q	N	UJ
BOOK	100	134	13936	9661	100	134	13936	9677
	200	146	29200	8791	200	146	29200	9781
	300	133	53200	8902	300	133	53200	9992
PRODUCT	100	91	9100	6271	100	91	9100	9281
	200	97	1900	8791	200	97	1900	8801
	300	85	3400	8902	300	85	3400	8002
AUTO	100	107	9100	5000	100	107	9100	5003
	200	90	1456	4568	200	90	1456	4788
	300	86	450	678	300	86	450	708
FLIGHT	100	128	11200	456	100	128	11200	458
	200	129	789	567	200	129	789	867
	300	105	2344	567	300	105	2344	500
HOTEL	100	54	6767	4567	100	54	6767	5038
	200	40	567	20	200	40	567	120
	300	51	450	34	300	51	450	71
MUSIC	100	39	56	35	100	39	56	30
	200	56	45	36	200	56	45	39
	300	61	32	4	300	61	32	0

In table 22, keeping the number of queries same, the methods of submission are compared. Efficiency is compared with respect to unique documents retrieved. From the total number of document new unique documents are calculated. In table22, Q (number of queries), N (Number of documents), U_j number of new documents retrieved. Since the method has not performed well in premier domain, so we have skipped its comparison in terms of number of documents. At present we have experimented with only three value of Q, i.e 100, 200 and 300. Another inference from the above table shows that our system has worked well with return limit 100. After return 100, the system retrieved lesser number of unique values.

Our method is static limit based ranking method. If we choose many high frequencies, coverage rate is decreased. This led to skipping some high-ranking documents, this is the reason our system has not worked well with premier. But in future with use of multiple query words, this problem could be overcome. Figure 39 and Figure 40 shows the comparison of submission methods in terms of new document captured.

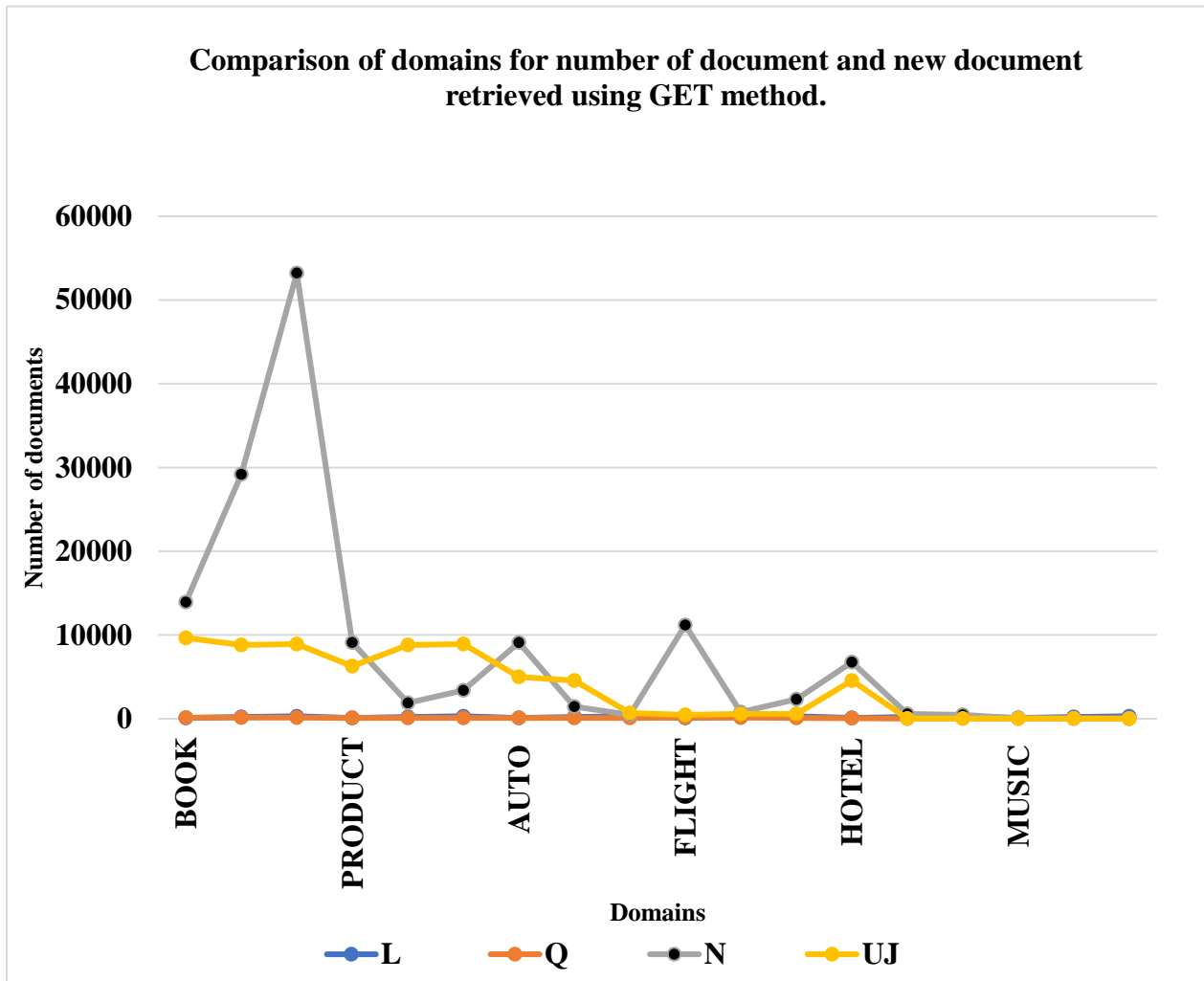


Figure 38: Comparison of domains for number of document and new document captured using GET method

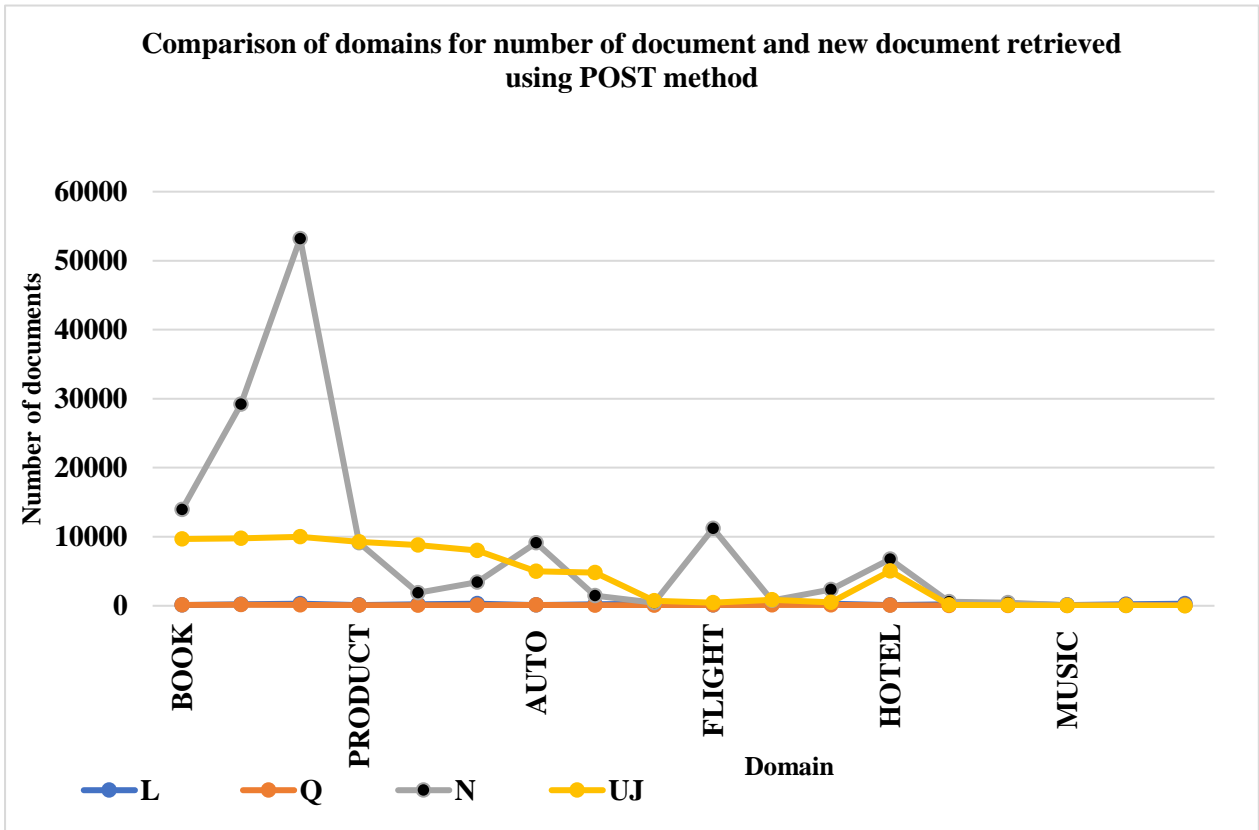


Figure 39: Comparison of domains for number of document and new document captured using POST method

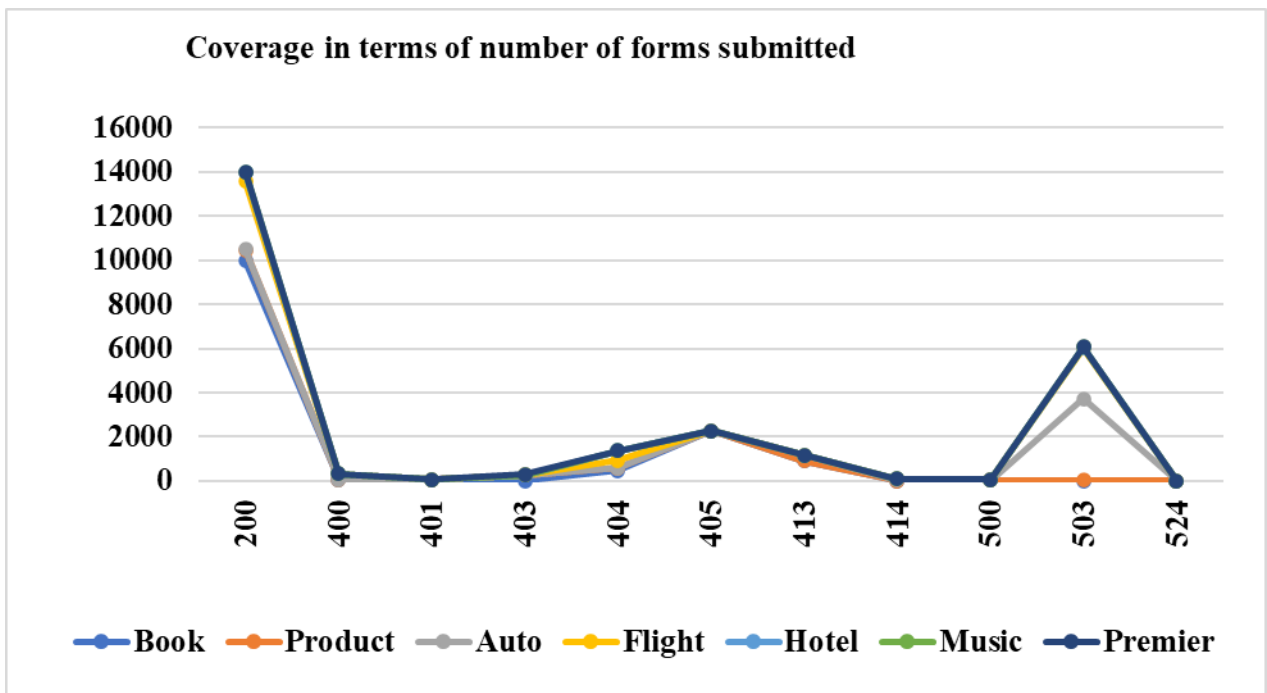


Figure 40: Coverage in terms of forms submitted

CHAPTER 5

COMPARE AND OPTIMIZE THE PERFORMANCE OF THE WEB CRAWLER

5.1 Comparison based on performance issues.

The comparison of proposed crawler is being made with all the pioneer work done in this area. But no platform exists for hidden web crawlers from which crawlers can be compared using all the performance measures. The existing technologies have worked on different performance measures.

Table 23: Comparison of proposed crawler with hidden web crawlers

Ref	Breadth search/(BFS) Depth search (DFS)	Technique	Strength	Weakness
[141]	DFS	<ul style="list-style-type: none"> • Matching of domain attributed using text similarity • Error detection using hash of important parts • Worked on multi-attributed structured data 	efficient label matching technique and incremental feedback-based crawling	Require significant human intervention and also not scalable
[141]	DFS	Stratified sampling, web pages are concatenated using navigational elements	<ul style="list-style-type: none"> • Domain independent • Hash for duplication detection 	Hash value of each sentence pose huge restrictions
[150]	DFS	Unstructured databased	<ul style="list-style-type: none"> • Query probing is adaptive • Queries are also focused 	Flat classification is not considered

[80]	BFS	<ul style="list-style-type: none"> • Crawling- domain specific • Query probing 	Can discover unstructured hidden databases as well	Deals only with full text, pre classified forms
[151]	DFS	<ul style="list-style-type: none"> • Unstructured database type • Query selection on based of frequency of occurrence 	Complete automation and high coverage	Not secure , and fixed return results
[92]	DFS	Incremental, and calculation of potential gain at each step	Hybrid policies for query selection	Huge memory requirement
[152]	-	Use of heuristics to identify forms	Extensible, can handle client as well as server-side technologies	It is assumed that one label is always associated with form elements it is not true in all cases.
[153]	DFS	Databased used is multi-attributed and set covering approach for queries.	Approach is quite effective in generating meaningful queries	Results from each round is added to the next round, it require huge resources in term of space.
[154]	BFS	Greedy algorithm and weight- based calculation for queries.	Adaptive, and retrieve homogenous data	Configuration require huge efforts
[33]	DFS	Query are selected based on the informativeness test	Navigation is easier	Cannot be scaled to

				hidden web crawling.
[155]	DFS	A unified query interface is created based on domain knowledge. Freshness of page is also undertaken using revisit policy .	The scope of specialised hidden web crawler is high for this technique	Performance measure do not include efficiency of unified query interface
PROPOSED CRAWLER	BFS	High harvest rate Priority based crawl to avoid ranking bias Works both on pre and post query	Crawler can be used as general as well as specialised crawler	Advanced form recognition will deliver more accurate results.

5.2 Comparison with Mercator [10]

Mercator is extensible and scalable web crawler that has been widely used as the base line crawler. proposed crawler on being compared with Mercator has worked on more categories of status codes. For unauthorized access and login, the forms are discarded at early stages. The status code considered are asynchronous response, bad request error, page not found, payload too large – request entity is large, payload too large – URI too long, internal server error and service unavailable error. Other performance measures cannot be compared as in Mercator the performance is measured in terms of number of URLS, while in case of proposed crawler it is classification accuracy. It is because proposed crawler submit the page as well.

Another comparison can be made with [156]. This study has presented a two-stage crawling, the forms are detected and classified according to the domains. The performance measures are in terms of site classification and form classification. This crawler is not distributed. While proposed crawler is distributed as well as it submits the forms and retrieve the results. These results can be used for further analysis. Proposed crawler is based on both pre and post query techniques .

Comparison with Hidden web crawling techniques on basis of form features, pre-query and post query, and use of machine learning and heuristics.

Table 24: Comparison of Proposed crawler with other technologies based on forms features.

Ref	Machine learning and heuristics	Pre/Post	Form features
3	Heuristics	Both Pre/post	<ul style="list-style-type: none"> • Input text box, with less than six characters. • Password fields.
15	Machine learning	Pre	<ul style="list-style-type: none"> • Term frequency.
35	Machine learning	Pre	<ul style="list-style-type: none"> • Submission method. • Keywords. • Number of fields of each type.
65	Machine learning and heuristics	-	<ul style="list-style-type: none"> • Word email, password control, radio and text control, hidden control, select control, submit control, advance search etc using DOM tree.
66	Machine learning	Pre	<ul style="list-style-type: none"> • Automatic
Proposed crawler	Both heuristic and machine learning	Both pre and post	<ul style="list-style-type: none"> • Automatic submission • Forms are dropped at early stage using rejection rules, so crawler save its time. Results from table [18] demonstrate the running time of crawler.

The following table show the comparison of distributed web crawler with other types of crawlers on the basis robustness, flexibility, manageability, network resources, high performance, incremental crawling, cost, Communication bandwidth, network load reduction and freshness. Distribution using Redis server has been proved beneficial in proposed crawler. It is fault tolerant and secured using REDIS server.

Table 25: Comparison of proposed crawler with distributed hidden web crawlers

Characteristics	DWC[30]	HWC[61]	DSHW[12]	AKSHR [62]	DGDWC	Proposed crawler
How it collect search forms	Identify but donot download	Identify but donot download	Automatic search and downloads forms	Automatic search and downloads forms	Automatic search and downloads forms	Identify but do not download until final stage
How it find entry to hidden web	Form tag	Form tag	Form tag	Form tag	Not defined	Form tag plus other rules
Do it select candidate forms	No	No	No	No	No	Yes
Is the form filling automatic	No	Not fully	-	Fully	Fully	Fully
Extensible	No	No	No	Yes	-	Yes
Scalable	No	No	No	No	-	Yes
Use of network bandwidth	High	High	-	-	-	Low , due to stopping criterias

The form crawler, form focused crawler and enhanced form crawler have coverage rate 19%, 79% and 95% respectively. But these above-mentioned crawlers have included only auto and job domains. As mentioned in literature computation of coverage is not same everywhere. We have opted total number of forms detected and submitted correctly. So total number of forms under 200 status is coverage of crawler.

Table 116: Comparison of proposed crawler in terms of coverage.

Domain	Number of forms										
	200	400	401	403	404	405	413	414	500	503	524
Book	9969	55	71	24	463	2285	914	25	50	3707	2
Product	496	0	0	240	115	0	0	0	0	0	0
Auto	29	9	0	0	21	0	0	3	2	35	0
Flight	3075	254	0	0	338	263	0	57	12	2280	0
Hotel	398	0	0	17	452	0	0	15	2	27	0
Music	35	10	0	0	0	0	6	2	0	31	0
Premier	2	0	0	6	0	0	0	0	0	0	0

The current domains under consideration are book, auto, product, flight, hotel, music, and premier and single application of pollution data . The results are manually merged for pollution and other domains. The domains depend on the seed sites. As crawling and learning progress, system can automatically add new domains. There are many restrictions on this like memory, network bandwidth and other hardware resources. In the reported literature, only one study is found in which Redis server is implemented [157]. Though distributed this crawler is only for generic crawling.

This technique outperforms the web crawler presented in [158]. On comparing accuracy and recall, in testing phase crawler has accuracy 81.06% and precision 84.62 %, whilst both performance measures have reached above 95% in our technique.

5.3 Constraints and barriers

During the initial stage of crawling, DMOZ dataset was used. After initial extraction, it was found that most of the URLs were not available upon sending request. The proposed crawler can work on both DMOZ, and jaismine directory. The URLs are collected from amazon.com as well. and ranking is function of three components.

Proposed crawler is implemented in python , it is provided with URLs obtained from above crawler, DMOZ, jaismine and amazon. Any of the mentioned can be used to start crawling.

Other barriers faced are hardware and network resource constraints. Websites are not designed same so sticking to structural features become mandatory.

1. System could work with any number of URLs, but due to restriction posed its is confined to a certain number of URLs and number of forms. If these restrictions are lifted crawler is trapped under spider traps.
2. Pages often contain invalid characters (i.e. incompatible with the encoding of the page).
3. Servers often return all kinds of HTTP errors (500, 404, 400, etc.)
4. Servers are often unreachable and cause timeouts. The domain/website might not exist anymore, or there might be DNS problems, or it might be under heavy load.
5. Some web pages are huge and cannot be downloaded in single crawl. If the crawler will do so, it will run out of memory soon.
6. Our method is static limit based ranking method. Because we have put limit on the number of web pages . If we choose many high frequencies, coverage rate is decreased. This led to skipping some high-ranking documents, this is the reason our system has not worked well with premier. But in future with use of multiple query words, this problem could be overcome.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

- Due to the large volume and dynamic nature of the hidden web, achieving wide coverage and high efficiency is a challenging issue. To overcome these issues, interlinked and interdependent three-stages of crawler efficiently harvest hidden web interfaces.
- With the adoption of focusing on relevant pages, learning link path, features of target pages, novel rejection criteria and submission system have considerably increased the throughput.
- Crawler focus on its target throughout its crawling process, and identify promising links to the target pages effectively, and can rapidly locate its target by its efficient search strategy.
- The stopping criteria and threshold rules have also been employed to avoid unproductive crawling. Experiments have proved that most of the forms are available at depth three.
- Based on effective Ranking and Learning algorithms/strategies, the crawler has ultimately achieved a good performance. An enhanced ranking algorithm for collect hidden websites based on priority by overcoming the ranking bias.
- A ranking algorithm is a triplet formula to calculate the rank of the website. By including site frequency, the documents which have low rank earlier can have a high rank. By ranking the website, the crawler minimises the number of visits and maximize the number of websites with embedded forms. Ranking algorithm has helped improving the results in terms of throughput.
- Experimental results indicate that the performance of the proposed crawler has a better harvest rate and coverage rate than of existing techniques.

The following points discuss some of the major issues faced by hidden web crawlers, and how these are resolved by proposed crawlers.

- **Identification of entry to hidden web:** Finding entry to hidden web is one of the major challenges. As it is evident from the literature that most of the studies have considered the availability of form tag as the entry to the hidden web, but this is enhanced, using some rules, which do not include every source into hidden web entry. As this crawler is designed for general search, if all the websites with form tag are included it sure will increase the number of collected URLs but then

crawler has to check each URL during form submission. If the non-searchable URLs are rejected at the early stage it will save the efforts of the crawler.

- **Social responsibility:** A crawler is required to follow social responsibility. That is, it should not overburden the website with queries. Sometimes if the crawler burdens the website it results in denial of service. To overcome this, stopping criteria's are designed.

The depth of crawl is maximum up to three, the maximum number of pages to crawl is 100, the maximum number of forms found is 100. At each depth, if crawler has crawled 50 pages but the searchable form is not found, it will jump to the next depth.

- **Interaction with search interface:** A publicly indexable web crawler cannot fill and submit the queries. To automatically parse, process and interact with query interfaces a repository is designed. That is continuously updated for the form values.
- **Selection policy:** A web crawler has to follow the selection policy. It tells the web crawler that which pages are needed to be downloaded. Duplicate URLs are removed at frontier level then first rejection criteria are followed to select which pages not to crawl. Secondly, the ranking mechanism is designed to select which URL to crawl first.
- **Implementation of distributed crawler:** The hidden web crawling lack distribution in combination with focused crawling. This crawler has implemented focused crawling in hidden web and it is distributed using Redis server. Redis is also acting as a data store.
- **Similarity and duplication detection:** Exhaustive crawling is a waste of resources. In this research, SIMHASH is implemented using the Redis plugin to detect similarities in URLs at the frontier level. Duplicate URLs will be discarded at a frontier level.
- **Form submission:** Hidden web crawlers are based on either pre or post query submission methods. The proposed crawler implemented both based on heuristics of form structure extraction. Initially, all the possibilities are tried to submit the forms before the system learn to fill and submit automatically. This also leads to limit the fields of form under consideration.
- The enhanced ranking algorithm for collecting hidden websites based on priority has tackled the problem when the document is missed if it has low rank. This algorithm is a triplet formula to calculate the rank of the website. By including site frequency, the documents which have a low rank earlier can have a high rank. By ranking the website, the crawler minimises the number of visits and maximize the number of websites with embedded forms.
- In searching for new rules to improve the efficiency, we have imposed a limit on the number of documents to be returned. This has also served as the drawback of the system as the crawler should not pose any limit on the number of documents.

- One another limitation of the system is in premier domain. The number of forms submitted is very low. For this reason, the domain could not be included in the new document retrieved factor.

6.2 Future Scope

In this research, various challenges of hidden web crawling are overcome. But still, many improvements can be made. In future applications of crawling and case studies will be taken into account. Furthermore, the followings points can be taken into account

1. We have designed the stopping rules to save the crawler from the exhaustive crawling traps. This not only saves memory and time but also help to retrieve more unique documents. On the same line, we have implemented the concept of crawling up to the depth of three and after that new URL is picked up from the frontier.
2. The efficiency of the crawler is shown by correctly submitted web pages. The inclusion of more domains and status codes remains as future work.
3. We are also going to combine the distance rank algorithm which we believe will yield better results. In future, we will also work on unbounded forms.
4. Further work can also be done in direction of indexing or resultant URLs and harvesting.
5. Crawlers can be made more efficient by making enhancements in form submission techniques and form recognition techniques.
6. Future work will also include creating a user-friendly user interface.
7. The performance of the crawler is dependent on the seed URLs, in future, the work will be done in this area. Currently, the crawler bootstrap using seed URLs, in future machine learning could be applied to carefully select the URLs.
8. More work could be done on improving the ranking and freshness of URLs.

References

- [1] S. Ceri, *Data-Centric Systems and Applications*. .
- [2] B. Arif, H. N. Qureshi, A. Un Nisa, U. E. H. Siddiqui, Q. Shafi, and T. Tariq, “Web crawlers: To detect security holes,” *ICOSST 2013 - 2013 International Conference on Open Source Systems and Technologies, Proceedings*, pp. 133–140, 2013.
- [3] P. P. Talukdar *et al.*, “Learning to create data-integrating queries,” *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 785–796, 2008.
- [4] I. Ruthven, “Advanced Topics in Information Retrieval,” *Advanced Topics in Information Retrieval*, vol. 33, pp. 187–207, 2011.
- [5] M. L. Zhang and Z. H. Zhou, “ML-KNN: A lazy learning approach to multi-label learning,” *Pattern Recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.
- [6] J. Allan *et al.*, “Challenges in Information Retrieval and Language Modeling: Report of a Workshop Held at the Center for Intelligent Information Retrieval, University of Massachusetts Amherst, September 2002,” *SIGIR Forum*, vol. 37, no. 1, pp. 31–47, 2003.
- [7] C. Olston and M. Najork, “Web Crawling,” *Foundations and Trends in Information Retrieval*, vol. 4, no. 3, pp. 175–246, 2010.
- [8] C. Castillo, M. Marin, R. Baeza-Yates, and A. Rodriguez, “Scheduling algorithms for Web crawling,” *Proceedings - WebMedia and LA-Web 2004*, pp. 10–17, 2004.
- [9] J. Jiang and N. Yu, “Schedule web forum crawling with a freshness-first strategy,” *Proceedings of 2011 International Conference on Computer Science and Network Technology, ICCSNT 2011*, vol. 3, pp. 2027–2032, 2011.
- [10] A. Heydon and M. Najork, “Mercator: A scalable, extensible Web crawler,” *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999.
- [11] J. Cho and H. Garcia-Molina, “Parallel crawlers,” *Proceedings of the 11th International Conference on World Wide Web, WWW '02*, pp. 124–135, 2002.
- [12] C. Castillo, A. Nelli, and A. Panconesi, “A memory-efficient strategy for exploring the Web,” *Proceedings - 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings), WI'06*, pp. 680–686, 2007.
- [13] S. Brin and L. Page, “Reprint of: The anatomy of a large-scale hypertextual web search engine,”

Computer Networks, vol. 56, no. 18, pp. 3825–3833, 2012.

- [14] M. Bazarganigilani, A. Syed, and S. Burki, “Focused Web Crawling Using Decay Concept and Genetic Programming,” *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, vol. 1, no. December, pp. 7–10, 2011.
- [15] A. Patel and N. Schmidt, “Application of structured document parsing to focused web crawling,” *Computer Standards and Interfaces*, vol. 33, no. 3, pp. 325–331, 2011.
- [16] H. T. Y. Achsan and W. C. Wibowo, “A fast distributed focused-web crawling,” *Procedia Engineering*, vol. 69, pp. 492–499, 2014.
- [17] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, “Focused crawling using context graphs,” *Proceedings of the 26th international conference on very large data bases*, pp. 527–534, 2000.
- [18] Y. Li, Y. Wang, and J. Du, “E-FFC: An enhanced form-focused crawler for domain-specific deep web databases,” *Journal of Intelligent Information Systems*, vol. 40, no. 1, pp. 159–184, 2013.
- [19] B. Novak, “a Survey of Focused Web Crawling Algorithms,” *Proceedings of SIKDD*, vol. 5558, pp. 55–58, 2004.
- [20] G. H. Agre and N. V. Mahajan, “Keyword focused web crawler,” *2nd International Conference on Electronics and Communication Systems, ICECS 2015*, pp. 1089–1092, 2015.
- [21] G. Gossen, E. Demidova, and T. Risse, “iCrawl: Improving the Freshness of Web Collections by Integrating Social Web and Focused Web Crawling,” 2016.
- [22] M. M. G. Farag, S. Lee, and E. A. Fox, “Focused crawler for events,” *International Journal on Digital Libraries*, vol. 19, no. 1, pp. 3–19, 2018.
- [23] M. Jacovi, “The shark-search algorithm. An application: Tailored Web site mapping,” *Computer Networks*, vol. 30, no. 1–7, pp. 317–326, 1998.
- [24] M. Li, C. Li, C. Wu, and Y. Luo, “A focused crawler URL analysis algorithm based on semantic content and link clustering in cloud environment,” *International Journal of Grid and Distributed Computing*, vol. 8, no. 2, pp. 49–60, 2015.
- [25] P. Dahiwale, A. Mokhade, and M. M. Raghuwanshi, “Intelligent web crawler,” *ICWET 2010 - International Conference and Workshop on Emerging Trends in Technology 2010, Conference Proceedings*, no. Icwet, pp. 613–617, 2010.
- [26] G. Madhu, A. Govardhan, and T. K. V. Rajinikanth, “Intelligent Semantic Web Search Engines: A Brief Survey,” *International journal of Web & Semantic Technology*, vol. 2, no. 1, pp. 34–42, 2011.

- [27] S. Lawrence and C. L. Giles, "Searching the World Wide Web Author (s): Steve Lawrence and C . Lee Giles Published by : American Association for the Advancement of Science Stable URL : <http://www.jstor.org/stable/2895232> Searching the World Wide Web," vol. 280, no. 5360, pp. 98–100, 2017.
- [28] J. Madhavan, L. Afanasiev, L. Antova, and A. Halevy, "Harnessing the Deep Web: Present and Future," 2009.
- [29] M. K. Bergman, "The Deep Web : Surfacing Hidden Value " Whole new classes of Internet-based companies choose the Web as their preferred medium," *World Wide Web Internet And Web Information Systems*, pp. 1–17, 2001.
- [30] B. He, M. Patel, Z. Zhang, and K. C. Chang, "Accessing the Deep Web : A Survey," *Communications of the ACM - ACM at sixty: a look back in time*, vol. 50, pp. 94–101, 2007.
- [31] R. Baeza-Yates and C. Castillo, "Crawling the infinite web," *Journal of Web Engineering*, vol. 6, no. 1, pp. 49–72, 2007.
- [32] D. Web, N. York, A. Rajaraman, and J. P. Bezos, "Exploring a ' Deep Web ' That Google Can ' t Grasp," pp. 23–25, 2009.
- [33] J. Madhavan, L. Afanasiev, L. Antova, and A. Halevy, "Harnessing the Deep Web: Present and Future," *Systems Research*, vol. 2, no. 2, pp. 50–54, 2009.
- [34] J. Liu, Z. Wu, L. Jiang, Q. Zheng, and X. Liu, "Crawling deep web content through query forms," *Proceedings of WEBIST2009, Lisbon Portugal*, pp. 634–642, 2009.
- [35] Y. Wang, H. Li, W. Zuo, F. He, X. Wang, and K. Chen, "Research on discovering deep web entries," *Computer Science and Information Systems*, vol. 8, no. 3, pp. 779–799, 2011.
- [36] J. Madhavan, D. Ko, \Lucja Kot, V. Ganapathy, A. Rasmussen, and A. Halevy, "Google's Deep Web crawl," *Proceedings of the VLDB Endowment archive*, vol. 1, no. 2, pp. 1241–1252, 2008.
- [37] S. M. Mirtaheri, M. E. Dinçtürk, S. Hooshmand, G. V. Bochmann, G.-V. Jourdan, and I. V. Onut, "A Brief History of Web Crawlers," 2014.
- [38] A. O. Mendelzon and T. Milo, "Formal models of Web queries," *Information Systems*, vol. 23, no. 8, pp. 615–637, 1998.
- [39] M. C. Moraes, C. A. Heuser, V. P. Moreira, and D. Barbosa, "Prequery discovery of domain-specific query forms: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1830–1848, 2013.
- [40] N. Gupta and S. Kapoor, "Extraction of Query Interfaces for Domain- Specific Hidden Web

Crawler,” vol. 5, no. 1, pp. 679–681, 2014.

- [41] L. Barbosa and J. Freire, “An adaptive crawler for locating hiddenwebentry points,” *Proceedings of the 16th international conference on World Wide Web - WWW '07*, p. 441, 2007.
- [42] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, “Trovatore: Towards a Highly Scalable Distributed Web Crawler.,” *WWW Posters*, pp. 7–8, 2001.
- [43] D. Rocco, J. Caverlee, L. Liu, and T. Critchlow, “Exploiting the deep web with DynaBot: Matching, probing, and ranking,” *14th International World Wide Web Conference, WWW2005*, pp. 1174–1175, 2005.
- [44] G. Pant and P. Srinivasan, “Learning to crawl: Comparing classification schemes,” *ACM Transactions on Information Systems*, vol. 23, no. 4, pp. 430–462, 2005.
- [45] B. Com, *No Title*. .
- [46] J. Yu, M. Li, and D. Zhang, “A Distributed Web Crawler Model based on Cloud Computing,” no. 66, pp. 276–279, 2016.
- [47] H. M. Moftah and S. M. Abuelenin, “Elastic Web Crawler Service-Oriented Architecture Over Cloud Computing,” 2018.
- [48] C. W. Cleverdon, J. Mills, and M. Keen, “Factors determining the performance of indexing systems,” *Aslib Cranfield Research Project Cranfield England*, vol. Vol 2. pp. 37–59, 1966.
- [49] C. Olston and M. Najork, “Web Crawling,” *Web Crawling*, vol. 4, no. May, pp. 228–235, 2010.
- [50] A. Chandramouli and S. Gauch, “A Co-operative Web Services Paradigm for Supporting Crawlers,” *Riao2007*, pp. 475–489, 2007.
- [51] M. Al Saadany, “The Reality of the Use of Learning Resource Centers Specialist for Libraries and Digital Resources as a Tool for Continuing Professional Development: A Comparative Study between Egypt and Saudi Arabia,” pp. 245–245, 2014.
- [52] S. Raghavan and H. Garcia-molina, “Crawling the Hidden Web,” *27th VLDB Conference - Roma, Italy*, pp. 1–10, 2001.
- [53] P. G. Ipeirotis, L. Gravano, and M. Sahami, “Probe, count, and classify,” *ACM SIGMOD Record*, vol. 30, no. 2, pp. 67–78, 2001.
- [54] F. Menczer, G. Pant, and P. Srinivasan, “Topical Web Crawlers : Evaluating Adaptive Algorithms,” vol. V, no. February, pp. 1–38, 2003.
- [55] S. Chakrabarti, M. Van Den Berg, and B. Dom, “Focused crawling: A new approach to topic-specific Web resource discovery,” *Computer Networks*, vol. 31, no. 11, pp. 1623–1640, 1999.

- [56] Y. Li, Y. Wang, and J. Du, "E-FFC: An enhanced form-focused crawler for domain-specific deep web databases," *Journal of Intelligent Information Systems*, vol. 40, no. 1, pp. 159–184, 2013.
- [57] A. Batzios, C. Dimou, A. L. Symeonidis, and P. A. Mitkas, "BioCrawler: An intelligent crawler for the semantic web," *Expert Systems with Applications*, vol. 35, no. 1–2, pp. 524–530, 2008.
- [58] P. Bedi, A. Thukral, H. Banati, A. Behl, and V. Mendiratta, "A Multi-Threaded Semantic Focused Crawler," *Journal of Computer Science and Technology*, vol. 27, no. 6, pp. 1233–1242, 2012.
- [59] H. Dong and F. K. Hussain, "Self-adaptive semantic focused crawler for mining services information discovery," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1616–1626, 2014.
- [60] G. Pant and P. Srinivasan, "Learning to crawl: Comparing classification schemes," *ACM Transactions on Information Systems (TOIS)*, vol. 23, no. 4, pp. 430–462, 2005.
- [61] S. Chakrabarti, K. Punera, and M. Subramanyam, "Accelerated focused crawling through online relevance feedback," *Proceedings of the eleventh international conference on World Wide Web - WWW '02*, p. 148, 2002.
- [62] M. Diligentit, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, "Focused crawling using context graphs," *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB'00*, pp. 527–534, 2000.
- [63] H. Lu, D. Zhan, L. Zhou, and D. He, "An Improved Focused Crawler: Using Web Page Classification and Link Priority Evaluation," *Mathematical Problems in Engineering*, vol. 2016, 2016.
- [64] G. Z. Kantorski, V. P. Moreira, and C. A. Heuser, "Automatic Filling of Hidden Web Forms," *ACM SIGMOD Record*, vol. 44, no. 1, pp. 24–35, 2015.
- [65] G. Almpanidis, C. Kotropoulos, and I. Pitas, "Combining text and link analysis for focused crawling-An application for vertical search engines," *Information Systems*, vol. 32, no. 6, pp. 886–908, 2007.
- [66] H. Liu, E. Milios, and J. Janssen, "ABSTRACT Probabilistic Models for Focused Web Crawling," vol. 28, no. 3, 2008.
- [67] W. Gao, H. C. Lee, and Y. Miao, "Geographically focused collaborative crawling," *Proceedings of the 15th international conference on World Wide Web - WWW '06*, p. 287, 2006.
- [68] S. Xu, H. J. Yoon, and G. Tourassi, "A user-oriented web crawler for selectively acquiring online content in e-health research," *Bioinformatics*, vol. 30, no. 1, pp. 104–114, 2014.

- [69] S. Pastrana, D. R. Thomas, A. Hutchings, and R. Clayton, "CrimeBB: Enabling Cybercrime Research on Underground Forums at Scale," *The Web Conference (WWW)*, pp. 1845–1854, 2018.
- [70] A. Seyfi, "Analysis and Evaluation of the Link and Content Based Focused Treasure-Crawler," *arXiv preprint arXiv:1306.0054*, pp. 1–13, 2013.
- [71] G. Pavai and T. V Geetha, "A Bootstrapping Approach to classification of Deep web Query Interfaces," vol. 11, no. 1, pp. 1–9, 2014.
- [72] M. Vidal and E. S. De Moura, "Structure-Based Crawling in the Hidden Web," *Computer*, vol. 14, no. 11, pp. 1857–1876, 2008.
- [73] N. H.S.Bamrah, B. S Satpute, and P. Patil, "Web Forum Crawling Techniques," *International Journal of Computer Applications*, vol. 85, no. 17, pp. 36–41, 2014.
- [74] M. A. Kausar, V. S. Dhaka, and S. K. Singh, "An Effective Parallel Web Crawler based on Mobile Agent and Incremental Crawling," *Journal of Industrial and Intelligent Information*, vol. 1, no. 1, pp. 86–90, 2013.
- [75] M. A. Kausar, V. S. Dhaka, and S. K. Singh, "Web Crawler Based on Mobile Agent and Java Aglets," *International Journal of Information Technology and Computer Science*, vol. 5, no. 10, pp. 85–91, 2013.
- [76] Q. Huang, Q. Li, H. Li, and Z. Yan, "An approach to incremental deep web crawling based on incremental harvest model," *Procedia Engineering*, vol. 29, pp. 1081–1087, 2012.
- [77] Z. Shi, M. Shi, and W. Lin, "The Implementation of Crawling News Page Based on Incremental Web Crawler," *Proceedings - 4th International Conference on Applied Computing and Information Technology, 3rd International Conference on Computational Science/Intelligence and Applied Informatics, 1st International Conference on Big Data, Cloud Computing, Data Science*, pp. 348–351, 2017.
- [78] G. Gossen, E. Demidova, and T. Risse, "ICrawl: Improving the Freshness of Web Collections by Integrating Social Web and Focused Web Crawling," *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*, vol. 2015-June, pp. 75–84, 2015.
- [79] J. Madhavan, D. Ko, and A. Rasmussen, "Google 's Deep-Web Crawl," pp. 1241–1252.
- [80] A. Bergholz and B. Chidlovskii, "Crawling for domain-specific hidden web resources," *Proceedings - 4th International Conference on Web Information Systems Engineering, WISE 2003*, pp. 125–133, 2003.
- [81] J. Cope, N. Craswell, and D. Hawking, "Automated Discovery of Search Interfaces on the Web BT - Fourteenth Australasian Database Conference (ADC2003)," vol. 17, pp. 181–189, 2003.

- [82] M. C. Moraes, C. A. Heuser, V. P. Moreira, and D. Barbosa, "Pre-Query Discovery of Domain-specific Query Forms : A Survey," pp. 1–19.
- [83] A. Kashyap, V. Hristidis, M. Petropoulos, and S. Tavoulari, "Effective navigation of query results based on concept hierarchies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 540–553, 2011.
- [84] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang, "Structured databases on the web," *ACM SIGMOD Record*, vol. 33, no. 3, p. 61, 2004.
- [85] J. Madhavan *et al.*, "Structured data meets the web: A few observations," *IEEE Data Eng. Bull.*, vol. 29, no. 4, pp. 19–26, 2006.
- [86] L. Barbosa and J. Freire, "Siphoning Hidden-Web Data through Keyword-Based Interfaces," vol. 1, no. 1, pp. 133–144, 2010.
- [87] A. Ntoulas, "Keyword Queries," *Framework*, pp. 100–109, 2005.
- [88] J. Caverlee, L. Liu, D. B. Probe, and Cluster, "and discover: Focused extraction of qa-pagelets from the deep web," *in: ICDE*, no. 1, pp. 103–115, 2004.
- [89] T. Furche *et al.*, "DIADEM: Thousands of Websites to a Single Database," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, 2014.
- [90] P. Liakos, A. Ntoulas, A. Labrinidis, and A. Delis, "Focused crawling for the hidden web," *World Wide Web*, vol. 19, no. 4, pp. 605–631, 2016.
- [91] L. Barbosa and J. Freire, "Searching for Hidden-Web Databases."
- [92] A. Ntoulas, P. Zerfos, and J. Cho, "Downloading textual hidden Web content through keyword queries," *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*, pp. 100–109, 2005.
- [93] Y. Li, T. Nie, D. Shen, and G. Yu, "Domain-oriented deep web data sources' discovery and identification," *Advances in Web Technologies and Applications - Proceedings of the 12th Asia-Pacific Web Conference, APWeb 2010*, pp. 464–467, 2010.
- [94] F. Shi, Y. Lu, G. Yang, and J. Huang, "A Deep Web Query Generation Method Using Semantic Annotation," *2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control*, no. part 1, pp. 322–325, 2011.
- [95] M. Khelghati, M. Van Keulen, and D. Hiemstra, "Designing a general deep web harvester by harvestability factor," *CEUR Workshop Proceedings*, vol. 1310, pp. 1–16, 2014.
- [96] B. Zhou, B. Xiao, Z. Lin, and C. Zhang, "A distributed vertical crawler using crawling-period based strategy," *Proceedings of the 2010 2nd International Conference on Future Computer and*

Communication, ICFCC 2010, vol. 1, pp. 306–311, 2010.

- [97] M. A. Kausar, V. S. Dhaka, and S. K. Singh, “An Effective Parallel Web Crawler based on Mobile Agent and Incremental Crawling,” *Journal of Industrial and Intelligent Information*, vol. 1, no. 2, pp. 86–90, 2013.
- [98] F. Ye, Z. Jing, Q. Huang, and C. Hu, “The Research and Implementation of a Distributed Crawler System Based on Apache Flink,” vol. 3, pp. 90–98.
- [99] D. Le Quoc, C. Fetzer, P. Sutra, V. Schiavoni, and P. Felber, “UniCrawl : A Practical Geographically Distributed Web Crawler.”
- [100] M. E. ElAraby, “Crawler Architecture Using Grid Computing,” *International Journal of Computer Science and Information Technology*, vol. 4, no. 3, pp. 113–127, 2012.
- [101] D. Gunawan, Amalia, and A. Najwan, “Improving Data Collection on Article Clustering by Using Distributed Focused Crawler,” *Journal of Computing and Applied Informatics*, vol. 1, no. 1, pp. 39–50, 2017.
- [102] M. Bošnjak, E. Oliveira, J. Martins, E. M. Rodrigues, and L. Sarmiento, “TwitterEcho - A Distributed Focused Crawler to Support Open Research with Twitter Data,” *WWW - MSND Workshop*, pp. 1233–1239, 2012.
- [103] H. Xu, K. Li, and G. Fan, “An Improved Strategy of Distributed Network Crawler Based on Hadoop and P2P,” vol. 2, pp. 849–855, 2019.
- [104] M. Bošnjak and E. Oliveira, “TwitterEcho - A Distributed Focused Crawler to Support Open Research with Twitter Data,” 2012.
- [105] A. Alkalbani, A. Shenoy, F. K. Hussain, O. K. Hussain, and Y. Xiang, “Design and implementation of the hadoop-based crawler for SaaS service discovery,” *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, vol. 2015-April, pp. 785–790, 2015.
- [106] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, “UbiCrawler: A scalable fully distributed Web crawler,” *Software - Practice and Experience*, vol. 34, no. 8, pp. 711–726, 2004.
- [107] P. Boldi, A. Marino, M. Santini, and S. Vigna, “BUbiNG: massive crawling for the masses,” *International Conference on World Wide Web - WWW '14 Companion*, no. Ga 288956, pp. 227–228, 2014.
- [108] R. Campos, O. Rojas, M. Marín, and M. Mendoza, “Distributed ontology-driven focused crawling,” *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013*, no. FEBRUARY, pp. 108–115, 2013.

- [109] J. Exposto, J. Macedo, A. Pina, A. Alves, and J. Rufino, “Geographical partition for distributed web crawling,” *International Conference on Information and Knowledge Management, Proceedings*, pp. 55–60, 2005.
- [110] D. Le Quoc, C. Fetzer, P. Felber, E. Riviere, V. Schiavoni, and P. Sutra, “UniCrawl: A Practical Geographically Distributed Web Crawler,” *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, pp. 389–396, 2015.
- [111] G. Gouriten, S. Maniu, and P. Senellart, “Scalable, generic, and adaptive systems for focused crawling,” *HT 2014 - Proceedings of the 25th ACM Conference on Hypertext and Social Media*, pp. 35–45, 2014.
- [112] R. Campos, O. Rojas, M. Mar, and M. Mendoza, “Distributed Ontology-Driven Focused Crawling.”
- [113] P. G. Ipeirotis and L. Gravano, “Distributed Search over the Hidden Web,” *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pp. 394–405, 2002.
- [114] F. Liu, F. Y. Ma, Y. M. Ye, M. L. Li, and J. Di Yu, “IglooG: A distributed web crawler based on grid service,” *Lecture Notes in Computer Science*, vol. 3399, pp. 207–216, 2005.
- [115] B. Loo, S. Krishnamurthy, O. Cooper, and 2004, “Distributed web crawling over DHTs,” *Citeseer*, no. March, 2007.
- [116] S. M. Mirtaheri, D. Zou, G. V. Bochmann, G. V. Jourdan, and I. V. Onut, “Dist-RIA Crawler: A distributed crawler for Rich Internet Applications,” *Proceedings - 2013 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013*, pp. 105–112, 2013.
- [117] D. Mukhopadhyay, S. Mukherjee, S. Ghosh, S. Kar, and Y.-C. Kim, “Architecture of A Scalable Dynamic Parallel WebCrawler with High Speed Downloadable Capability for a Web Search Engine,” *Distributed Computing*, p. 6, 2011.
- [118] A. Alexandrescu, “A distributed framework for information retrieval, processing and presentation of data,” *2018 22nd International Conference on System Theory, Control and Computing, ICSTCC 2018 - Proceedings*, pp. 267–272, 2018.
- [119] M. D. Dikaiakos and D. Zeinalipour-Yazti, “A distributed middleware infrastructure for personalized services,” *Computer Communications*, vol. 27, no. 15, pp. 1464–1480, 2004.
- [120] V. Shkapenyuk and T. Suel, “Design and implementation of a high-performance distributed web crawler,” *Proceedings - International Conference on Data Engineering*, pp. 357–368, 2002.
- [121] B. V. Mahavidyalaya, V. Vidyanagar, B. V. Mahavidyalaya, V. Vidyanagar, B. V. Mahavidyalaya, and V. Vidyanagar, “Distributed High Performance WEB,” no. 1, pp. 236–239, 2013.

- [122] S. M. Mirtaheri, G. V. Bochmann, G. V. Jourdan, and I. V. Onut, "PDist-RIA Crawler: A peer-to-peer distributed crawler for Rich Internet Applications," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8787, pp. 365–380, 2014.
- [123] M. Kc, M. Hagenbuchner, and A. C. Tsoi, "A scalable lightweight distributed crawler for crawling with limited resources," *Proceedings - 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Workshops, WI-IAT Workshops 2008*, pp. 663–666, 2008.
- [124] P. Nesi, G. Pantaleo, and G. Sanesi, "A Distributed Framework for NLP-Based Keyword and Keyphrase Extraction From Web Pages and Documents," pp. 155–161, 2015.
- [125] B. D. Processing, E. Commerce, and C. A. Keywords, "Analysis and Research of Distributed Network Crawler based on Cloud Computing Hadoop Platform Hongsheng Xu 1,2," vol. 83, no. Snce, pp. 1045–1049, 2018.
- [126] V. Dhingra and K. K. Bhatia, "Intelligent Distributed Computing - Proceedings of the Third International Symposium on Intelligent Informatics, ISI 2014, September 24-27, 2014, Greater Noida, Delhi, India," pp. 213–223, 2015.
- [127] K. T. T. E. Tjin-Kam-Jet, "Research proposal for distributed deep web search," *International Conference on Information and Knowledge Management, Proceedings*, pp. 33–37, 2010.
- [128] B. B. Cambazoglu, V. Plachouras, F. Junqueira, and L. Telloli, "On the feasibility of geographically distributed web crawling," *InfoScale '08: Proceedings of the 3rd international conference on Scalable information systems*, pp. 1–10, 2008.
- [129] H. G. Kim, J. W. Lee, T. H. Ban, and H. K. Jung, "A study on distributed crawling-based overhead optimization," *International Journal of Software Engineering and its Applications*, vol. 9, no. 3, pp. 175–182, 2015.
- [130] A. Juffinger *et al.*, "Distributed Web 2.0 crawling for ontology evolution," *Journal of Digital Information Management*, vol. 7, no. 2, pp. 114–119, 2009.
- [131] A. Kritikopoulos, M. Sideri, and K. Strogilos, "CrawlWave : A Distributed Crawler," *Proceedings of the 3rd Hellenic Conference on Artificial Intelligence*, 2004.
- [132] K. Zhu, Z. Xu, X. Wang, and Y. Zhao, "A full distributed Web crawler based on structured network," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4993 LNCS, pp. 478–483, 2008.
- [133] J. Urbani, S. Kotoulas, E. Oren, and F. Van Harmelen, "Scalable Distributed Reasoning using

- MapReduce,” *Proceedings of the International Semantic Web Conference*, vol. 48, no. 6, pp. 634–649, 2009.
- [134] N. Pappas and E. Stamatatos, “An Intelligent Distributed System for Automatic Sentiment Analysis from Topic-Specific Web Sources: Discovery and Extraction of Relevant Documents,” 2011.
- [135] Y. Q. Gao and C. L. Peng, “Design and implementation of distributed crawler system for opinion mining,” *Applied Mechanics and Materials*, vol. 347–350, no. Iccsee, pp. 2506–2510, 2013.
- [136] M. Road, “Distributed Web Crawlers using Hadoop,” vol. 12, no. 24, pp. 15187–15195, 2017.
- [137] S. Zhong and Z. Deng, “A web crawler system design based on distributed technology,” *Journal of Networks*, vol. 6, no. 12, pp. 1682–1689, 2011.
- [138] G. Almpantidis, C. Kotropoulos, and I. Pitas, “Combining text and link analysis for focused crawling-An application for vertical search engines,” *Information Systems*, vol. 32, no. 6, pp. 886–908, 2007.
- [139] G. Yan, L. Kui, Z. Kai, and Z. Gang, “Board Forum Crawling: A web crawling method for Web forum,” *Proceedings - 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)*, *WI'06*, pp. 745–748, 2007.
- [140] P. Barrio and L. Gravano, “Sampling strategies for information extraction over the deep web,” *Information Processing and Management*, vol. 53, no. 2, pp. 309–331, 2017.
- [141] C. Sadowski and G. Levin, “SimHash : Hash-based Similarity Detection,” *Techreport*, pp. 1–10, 2007.
- [142] G. Valkanas and A. Ntoulas, “Rank-Aware Crawling of Hidden Web sites,” *WebDB*, pp. 1–6, 2011.
- [143] S. Liddle, D. Embley, D. Scott, and S. H. Yau, “Extracting Data Behind Web Forms,” *Lecture Notes in Computer Science*, no. 2784, pp. 402–413, 2003.
- [144] P. G. Ipeirotis, L. Gravano, and M. Sahami, “Automatic classification of text databases through query probing,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1997, pp. 245–255, 2001.
- [145] Y. Wang, J. Lu, J. Chen, and Y. Li, “Crawling ranked deep Web data sources,” *World Wide Web*, vol. 20, no. 1, pp. 89–110, 2017.
- [146] C. C. Aggarwal, *Machine learning for text*. 2018.
- [147] E. Baykan, M. Henzinger, L. Marian, and I. Weber, “Purely URL-based Topic Classification Categories and Subject Descriptors,” *In Proceedings of the 18th International World Wide Web Conference (WWW 2009)*, pp. 1109–1110, 2009.

- [148] S. Kaur and G. Geetha, "SIMHAR - Smart Distributed Web Crawler for the Hidden Web Using SIM+Hash and Redis Server," *IEEE Access*, vol. 8, pp. 117582–117592, 2020.
- [149] A. K. Sangaiah, A. E. Fakhry, M. Abdel-Basset, and I. El-henawy, "Arabic text clustering using improved clustering algorithms with dimensionality reduction," *Cluster Computing*, vol. 22, pp. 4535–4549, 2019.
- [150] P. Clough, "Extracting metadata for spatially-aware information retrieval on the internet," *Proceedings of the 2005 workshop on Geographic information retrieval - GIR '05*, p. 25, 2005.
- [151] L. Barbosa and J. Freire, "An adaptive crawler for locating hiddenwebentry points," *16th International World Wide Web Conference, WWW2007*, pp. 441–450, 2007.
- [152] M. Álvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro, "Crawling the content hidden behind web forms," *Lecture Notes In Computer Science*, pp. 322–333, 2007.
- [153] P. Wu, J. R. Wen, H. Liu, and M. Wei-Ying, "Query selection techniques for efficient crawling of structured Web sources," *Proceedings - International Conference on Data Engineering*, vol. 2006, p. 47, 2006.
- [154] L. Barbosa and J. Freire, "Searching for Hidden-Web Databases," *Proceedings of WebDB*, vol. 5, pp. 1–6, 2005.
- [155] K. K. Bhatia, A. K. Sharma, and R. Madaan, "AKSHR: A novel framework for a domain-specific hidden web Crawler," *2010 1st International Conference on Parallel, Distributed and Grid Computing, PDGC - 2010*, pp. 307–312, 2010.
- [156] F. Zhao, J. Zhou, C. Nie, H. Huang, and H. Jin, "SmartCrawler: A two-stage crawler for efficiently harvesting deep-web interfaces," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 608–620, 2016.
- [157] L. Zhang, Z. Bu, Z. Wu, and J. Cao, "DGWC: Distributed and generic web crawler for online information extraction," *IEEE/ACM BESC 2016 - Proceedings of 2016 International Conference on Behavioral, Economic, Socio - Cultural Computing*, 2017.
- [158] N. L. H. Hien, T. Q. Tien, and N. Van Hieu, "Web crawler: Design and implementation for extracting article-like contents," *Cybernetics and Physics*, vol. 9, no. 3, pp. 144–151, 2020.

Annexure 1

Source code

Accuracy

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas.compat import StringIO
from sklearn import svm
from sklearn.metrics import classification_report, confusion_matrix

from similarityforduplication import *

class Prop2NumTranser(object):
    def __init__(self, properties=[]):
        self._properties = properties

    def fit(self, data):
        self._properties = list(set(data))

    def transform(self, l):
        for i in range(len(l)):
            l[i] = self._properties.index(l[i]) + 1

url = "F:\\\\crw\\dataset\\Link-names-final.csv"
#url = "F:\\\\crawler code extract\\data\\Link-names-final.csv"

names = ['URL', 'METHOD', 'ACTION', 'BASEURL', 'DEPTH', 'PATHVALUES', 'DOMAIN', 'STATUSCODE']

dataset = pd.read_csv(url, names=names, keep_default_na=False, encoding='latin1')

transer = Prop2NumTranser()
transer.fit(np.hstack((dataset.iloc[:, 0])))
transer.transform(dataset.iloc[:, 0])

transer.fit(np.hstack((dataset.iloc[:, 1])))
transer.transform(dataset.iloc[:, 1])

transer.fit(np.hstack((dataset.iloc[:, 2])))
transer.transform(dataset.iloc[:, 2])

transer.fit(np.hstack((dataset.iloc[:, 3])))
transer.transform(dataset.iloc[:, 3])

transer.fit(np.hstack((dataset.iloc[:, 4])))
transer.transform(dataset.iloc[:, 4])

transer.fit(np.hstack((dataset.iloc[:, 5])))
transer.transform(dataset.iloc[:, 5])

transer.fit(np.hstack((dataset.iloc[:, 6])))
transer.transform(dataset.iloc[:, 6])
```

```

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 7].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

##### KNeighborsClassifier #####

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

##### SVM Classifier#####
'''clf_ae = svm.SVC(probability=True)
clf_ae.fit(X_train, y_train)
y_pred = clf_ae.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))'''

class Prop2NumTranser(object):
    def __init__(self, properties=[]):
        self._properties = properties

    def fit(self, data):
        self._properties = list(set(data))

    def transform(self, l):
        for i in range(len(l)):
            l[i] = self._properties.index(l[i]) + 1

p=graphs()

```

FET TEST

```

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
import numpy as np
import nltk
import pandas as pd
from nltk.corpus import stopwords

```

```

from nltk.tokenize import word_tokenize
nltk.download('stopwords')
#data = pd.read_csv("E:\CRW_data\crawlset-final.csv", encoding="ISO-8859-1")
#data = pd.read_csv("E:\CRW_data\crawlset-final.csv", error_bad_lines=False)
data = pd.read_csv("F:\\crw\\dataset\\Link-names-final.csv", encoding='latin1')
print(type(data))

messages = data.iloc[:, -2].dropna()
print('-----')
print(str(messages))
#vect = CountVectorizer()
#vect.fit(messages.dropna())
#print(vect.get_feature_names())
#dtm = vect.transform(messages.dropna())
#repr(dtm)
#print(dtm)
#print(pd.DataFrame(dtm.toarray(), columns=vect.get_feature_names()))

example_sent = messages

stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 'blog', 'the', 'which'])
word_tokens = word_tokenize(str(example_sent))

filtered_sentence = [w for w in word_tokens if not w in stop_words]

filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(word_tokens)
print(filtered_sentence)

def createDTM(messages):
    vect = TfidfVectorizer()
    dtm = vect.fit_transform(messages) # create DTM
    print(dtm)
    # create pandas dataframe of DTM
    dataf = pd.DataFrame(dtm.toarray(), columns=vect.get_feature_names())
    dataf.to_csv('vector.csv', sep=',', encoding='latin1')
    return pd.DataFrame(dtm.toarray(), columns=vect.get_feature_names())

print(createDTM(messages))
df = pd.read_csv('vector.csv')
headerList = []
valueList = []

for eachHeader in list(df.columns.values):
    if eachHeader == "Unnamed: 0":
        continue
    headerList.append(eachHeader)
    valueList.append(df[eachHeader].value_counts()[1])

print(headerList)
print(valueList)
y_pos = np.arange(len(headerList))
plt.bar(y_pos, valueList, align='center', alpha=0.5)

```

```

plt.xticks(y_pos, headerList)
plt.ylabel('Number of Form submitted')
plt.title('Domain vs No of Form submission')
plt.show()

```

FINAL CRW

```

from difflib import SequenceMatcher
import requests
import csv
import operator
from bs4 import BeautifulSoup
from urllib.parse import urlparse
from redis import Redis
from rq import Queue
from page_text import page_text_at_url

from Book import bookHeaders, getBookValue
from Hotel import hotelHeaders, getHotelValue
from Flight import flightHeaders, getFlightValue
from Apartment import apartmentHeaders, getApartmentValue
from Auto import autoHeaders, getAutoValue
from Music import musicHeaders, getMusicValue
from Premierleaguereults import premierLeagueHeaders, getPremierLeagueValue
from Product import productHeaders, getProductValue

writingFile = csv.writer(open('Link-names-final.csv', 'w'))
writingFile.writerow(['URL', 'METHOD', 'ACTION', 'BASEURL', 'DEPTH', 'PATHVALUES', 'DOMAIN',
'STATUSCODE'])

textFromHotel = list(hotelHeaders())
textFromBook = list(bookHeaders())
textFromFlight = list(flightHeaders())
textFromApartment = list(apartmentHeaders())
textFromAuto = list(autoHeaders())
textFromMusic = list(musicHeaders())
textFromPremier = list(premierLeagueHeaders())
textFromProduct = list(productHeaders())

threshold = 0.1
thresholdForIDCom = 0.8
def similar(a, b):
    return SequenceMatcher(None, a, b).ratio()

q = Queue(connection=Redis('127.0.0.1',6379))

#url = 'https://www.amazon.in/'
#url = 'https://www.jasminedirectory.com/'
#url = 'http://www.dmoz.org.in/'
#url = 'https://www.bbc.com/'
#url = 'https://www.Tmall.com/'
#url = 'https://www.Baidu.com/'
#url = 'https://Qq.com/'
#url = 'https://Ebay.com/'
#url = 'https://Bestbuy.com/'
#url = 'https://Ikea.com/'

```

```

#url = 'https://www.indiamart.com/'
url = 'https://www.whatsinproducts.com/'
#url = 'https://www.flipkart.com/'
#url = 'https://www.google.com/'
#url = 'https://www.ldblog.jp/'
#url = 'https://www.cc.com/'
#url = 'https://www.rumorfix.com'
#url = 'https://www.citysquares.com/'
#url = 'https://www.cocoacontrols.com/'
#url = 'https://www.themestotal.com/'
#url = 'https://www.wtmLondon.com/'
#url = 'https://www.yatra.com/'
#url = 'https://tomandlorenzo.com/'
#url = 'https://www.paintnite.com/'
#url = 'https://facebookpostmarketing.blogspot.com/'
#url = 'https://mypapershop.com/'
#url = 'https://www.yellowpages.co.th/'
#url = 'https://www.lasvegas.com/'
#url = 'https://www.newqc.cn/'
#url = 'https://www.xiu8.com/'
#url = 'https://www.noobmeter.com/'
#url = 'https://www.themestotal.com/'
#url = 'https://www.cocoacontrols.com/'
#url = 'https://www.debijenkorf.nl/'
#url = '-https://www.home-designing.com/'
#url = 'https://www.imdb.com/'
#url = 'https://www.youtube.com/'
#url = 'https://www.bestbuy.com/'
#url = 'https://www.musicgenreslist.com/'
#url = 'https://www.melodyful.com/'
#url = 'https://www.gaana.com/'
#url = 'https://www.cinespot.net/'
#url = 'https://www.deautos.com/'
#url = 'https://www.paintnite.com/'
#url = 'https://www.bigbasket.com/'
#url = 'https://www.wclC.com/'
#url = 'https://www.xneolinks.com/'
#url = 'https://www.zhuoaiwang.com/'
#url = 'https://www.css3gen.com/'
#url = 'https://www.golestanema.com-/'
#url = 'https://www.vocuspr.com-/'
#url = 'https://www.playrust.com/'
#url = 'https://www.proxy4free.com/'
#url = "https://www.lc115.com/"
#url = 'https://www.yahoo.com/'
#url = 'https://www.newqc.cn/'
#url = 'https://www.xiu8.com/'
#url = 'https://allinchrome.com/'
#url = 'https://www.engineersedge.com/'
#url = 'https://www.erpnext.com/'
#url = 'https://www.snapdeal.com/'

```

```

ListOfURLs = []
ListOfURLs.append(url)

```

```

for item in ListOfURLs:
    try:

```

```

    job = q.enqueue(page_text_at_url, item)
    page = job.perform()
    soup: BeautifulSoup = BeautifulSoup(page.text, 'html.parser')
except Exception as e:
    print(e)

aTags = soup.find_all('a', href=True)
for eachTag in aTags:
    link = eachTag.get('href')
    goturl = ''
    if link.find('https://') != -1 and link.find('https://') == 0:
        goturl = eachTag.get('href')
    else:
        goturl = url+link
    ListOfURLs.append(goturl)

forms = soup.find_all('form')
for eachform in forms:
    methodType = eachform.get('method')
    action = eachform.get('action')
    inputName = ''
    o = urlparse(item)
    depth = o.path.count('/')
    values = o.path.replace('/', ' ').replace('.html', '').replace('-', ' ')
    baseURL = o.scheme+'://'+o.hostname
    payload = ''
    if methodType != None:
        #Find Label
        allspans = eachform.find_all('span')
        labelArray = []
        for eachSpan in allspans:
            #print(eachSpan.text)
            labelArray.append(eachSpan.text.lower())
        print('Size',labelArray)
        nonSearchableText = ['username', 'password']
        s1 = set(labelArray)
        s2 = set(nonSearchableText)
        countNonSearchable = s1.intersection(s2)
        if countNonSearchable !=None and len(countNonSearchable) == 0:
            resultDist = {}
            resultHotel = {x for x in labelArray for y in textFromHotel if similar(x, y) >
threshold}
            resultBook = {x for x in labelArray for y in textFromBook if similar(x, y) >
threshold}
            resultFlight = {x for x in labelArray for y in textFromFlight if similar(x, y)
> threshold}
            resultAuto = {x for x in labelArray for y in textFromAuto if similar(x, y) >
threshold}
            resultApartment = {x for x in labelArray for y in textFromApartment if
similar(x, y) > threshold}
            resultMusic = {x for x in labelArray for y in textFromMusic if similar(x, y) >
threshold}
            resultPremier = {x for x in labelArray for y in textFromPremier if similar(x,
y) > threshold}
            resultProduct = {x for x in labelArray for y in textFromProduct if similar(x,
y) > threshold}

            if len(resultBook) != 0 and len(resultHotel) != 0 and len(resultFlight) != 0
and len(resultAuto) != 0 and len(resultApartment) != 0 and len(resultMusic) != 0 and
len(resultPremier) != 0 and len(resultProduct):
                resultDist = {'book': len(resultBook), 'hotel': len(resultHotel),

```

```

'flight': len(resultFlight), 'auto': len(resultAuto), 'apartment': len(resultApartment),
'music': len(resultMusic), 'premier': len(resultPremier), 'product': len(resultProduct)}
    foundDomain = max(resultDist.items(), key=operator.itemgetter(1))[0]
else:
    foundDomain = ''
print('=====', foundDomain)
print('Domain Data', resultDist)
if foundDomain != '':
    #find hidden inputs
    allInputField = eachform.find_all('input')
    hiddenText = ''
    id = []
    for eachInput in allInputField:
        print(baseURL)
        print(eachInput)
        try:
            if eachInput.get('type').strip() == 'hidden':
                hiddenText += eachInput.get('name') + '=' +
eachInput.get('value') + "&"
            elif eachInput.get('type').strip() == 'text':
                id.append(eachInput.get('name'))

        except Exception as e:
            print(e)
        foundIdsAndText = {}
        if foundDomain == 'book':
            foundIdsAndText = {(x, y) for x in id for y in textFromBook if
similar(x, y) > thresholdForIDCom}
        elif foundDomain == 'flight':
            foundIdsAndText = {(x, y) for x in id for y in textFromFlight if
similar(x, y) > thresholdForIDCom}
        elif foundDomain == 'hotel' :
            foundIdsAndText = {(x, y) for x in id for y in textFromHotel if
similar(x, y) > thresholdForIDCom}
        elif foundDomain == 'apartment':
            foundIdsAndText = {(x, y) for x in id for y in
textFromApartment if similar(x, y) > thresholdForIDCom}
        elif foundDomain == 'auto':
            foundIdsAndText = {(x, y) for x in id for y in textFromAuto if
similar(x, y) > thresholdForIDCom}
        elif foundDomain == 'music':
            foundIdsAndText = {(x, y) for x in id for y in textFromMusic if
similar(x, y) > thresholdForIDCom}
        elif foundDomain == 'premier' :
            foundIdsAndText = {(x, y) for x in id for y in textFromPremier if
similar(x, y) > thresholdForIDCom}
        elif foundDomain == 'product':
            foundIdsAndText = {(x, y) for x in id for y in textFromProduct
if similar(x, y) > thresholdForIDCom}
    else:
        foundIdsAndText = {}

    payload = {}
    for eachIdAndText in foundIdsAndText:
        if foundDomain == 'book':
            payload = {eachIdAndText[0]: getBookValue(eachIdAndText[1])}
        elif foundDomain == 'hotel':
            payload = {eachIdAndText[0]: getHotelValue(eachIdAndText[1])}
        elif foundDomain == 'flight':
            payload = {eachIdAndText[0]:
getFlightValue(eachIdAndText[1])}

```

```

        elif foundDomain == 'apartment':
            payload = {eachIdAndText[0]:
getApartmentValue(eachIdAndText[1])}
        elif foundDomain == 'auto':
            payload = {eachIdAndText[0]: getAutoValue(eachIdAndText[1])}
        elif foundDomain == 'music':
            payload = {eachIdAndText[0]: getMusicValue(eachIdAndText[1])}
        elif foundDomain == 'premier':
            payload = {eachIdAndText[0]:
getPremierLeagueValue(eachIdAndText[1])}
        elif foundDomain == 'product':
            payload = {eachIdAndText[0]:
getProductValue(eachIdAndText[1])}
        else:
            payload = {}
    try:
        if action != None and action.find(o.hostname) == -1:
            action = baseURL + action + '?' + hiddenText
        else:
            action = action + '?' + hiddenText
        if methodType == 'get':
            resp = requests.get(action, params=payload)
            print('submit form get:', action, payload)
        else:
            resp = requests.post(action, data=payload)
            print('submit form post:', action, payload)
        writingFile.writerow([item, methodType, action, o.path, depth,
values, foundDomain, resp.status_code])
        print('writing Data--->', item, methodType, action, o.path,
depth, values, foundDomain, resp.status_code)
    except Exception as e:
        print(e)

```

SEARCH

```

import csv
import random

```

```

class Apartment:

```

```

    def __init__(self, names, price, area, unittype):
        self.names = names
        self.price = price
        self.area = area
        self.unittype = unittype

```

```

Apartment_list = []

```

```

with open('apartments.csv', newline='') as csv_file:
    reader = csv.reader(csv_file)
    next(reader, None)
    for names, price, area, unittype in reader:
        Apartment_list.append(Apartment(names, price, area, unittype))

```

```

def apartmentDetail(index):
    return Apartment_list[index]

```

```

def apartmentHeaders():

```



```

    return ['names', 'price', 'area', 'unittype']

def getApartmentValue(prop):
    apartment: Apartment = apartmentDetail(random.randrange(0, len(Apartment_list), 3))

    if prop == 'names':
        value = apartment.names
    elif prop == 'price':
        value = apartment.price
    elif prop == 'area':
        value = apartment.area

    else:
        value = apartment.unittype
    return value
#print(getApartmentValue('unittype'))

import csv
import random

#dateCrawled,name,seller,offerType,price,abtest,vehicleType,yearOfRegistration,gearbox,powerPS,model,kilometer,monthOfRegistration,fuelType,brand,notRepairedDamage,dateCreated,nrOfPictures,postalCode,lastSeen

class Auto:

    def __init__(self, dateCrawled, name,seller, offerType,
price,abtest,vehicleType,yearOfRegistration,gearbox,powerPS,
model,kilometer,monthOfRegistration,fuelType,brand,notRepairedDamage,dateCreated,nrOfPictures
,postalCode,lastSeen):
        self.dateCrawled = dateCrawled
        self.name = name
        self.seller = seller
        self.price = price
        self.abtest = abtest
        self.vehicleType = vehicleType
        self.yearOfRegistration = yearOfRegistration
        self.gearbox = gearbox
        self.powerPS = powerPS
        self.model = model
        self.kilometer = kilometer
        self.monthOfRegistration = monthOfRegistration
        self.fuelType = fuelType
        self.brand = brand
        self.notRepairedDamage = notRepairedDamage
        self.dateCrawled= dateCreated
        self.nrOfPictures= nrOfPictures
        self.postalCode= postalCode
        self.lastSeen = lastSeen

Auto_list = []

with open('autos.csv', newline='') as csv_file:
    reader = csv.reader(csv_file)
    next(reader, None)

    for
dateCrawled,name,seller,offerType,price,abtest,vehicleType,yearOfRegistration,gearbox,powerPS

```

```
,model,kilometer,monthOfRegistration,fuelType,brand,notRepairedDamage,dateCreated,nrOfPictures,postalCode,lastSeen in reader:
```

```
Auto_list.append(Auto(dateCrawled,name,seller,offerType,price,abtest,vehicleType,yearOfRegistration,gearbox,powerPS,model,kilometer,monthOfRegistration,fuelType,brand,notRepairedDamage,dateCreated,nrOfPictures,postalCode,lastSeen))
```

```
def autoDetail(index):  
    return Auto_list[index]
```

```
def autoHeaders():  
    return  
    ['dateCrawled','name','seller','offerType','price','abtest','vehicleType','yearOfRegistration',  
    'gearbox','powerPS','model','kilometer','monthOfRegistration','fuelType','brand','notRepairedDamage',  
    'dateCreated','nrOfPictures','postalCode','lastSeen']
```

```
def getAutoValue(prop):  
    auto: Auto = autoDetail(random.randrange(0, len(Auto_list), 3))
```

```
    if prop == 'dateCrawled':  
        value = auto.dateCrawled  
    elif prop == 'name':  
        value = auto.name  
    elif prop == 'seller':  
        value = auto.seller  
    elif prop == 'offerType':  
        value = auto.offerType  
    elif prop == 'price':  
        value = auto.price  
    elif prop == 'abtest':  
        value = auto.abtest  
    elif prop == 'vehicleType':  
        value = auto.vehicleType  
    elif prop == 'yearOfRegistration':  
        value = auto.yearOfRegistration  
    #  
    'powerPS','model','kilometer','monthOfRegistration','fuelType','brand','notRepairedDamage','dateCreated',  
    'nrOfPictures','postalCode','lastSeen'  
    elif prop == 'powerPS':  
        value = auto.powerPS  
    elif prop == 'model':  
        value = auto.model  
    elif prop == 'kilometer':  
        value = auto.kilometer  
    elif prop == 'monthOfRegistration':  
        value = auto.monthOfRegistration  
    #####  
    elif prop == 'fuelType':  
        value = auto.fuelType  
    elif prop == 'brand':  
        value = auto.brand  
    elif prop == 'notRepairedDamage':  
        value = auto.notRepairedDamage  
    elif prop == 'dateCreated':  
        value = auto.dateCreated  
    elif prop == 'nrOfPictures':  
        value = auto.nrOfPictures  
    elif prop == 'postalCode':  
        value = auto.postalCode  
    else:
```

```

        value = auto.lastSeen
    return value

#print(getAutoValue('LastSeen'))

import csv
import random

class Book:

    def __init__(self, title, author, genre, height, publisher):
        self.keyword = title
        self.author = author
        self.genre = genre
        self.height = height
        self.publisher = publisher

book_list = []

with open('books.csv', newline='') as csv_file:
    reader = csv.reader(csv_file)
    next(reader, None)
    for title, author, genre, height, publisher in reader:
        book_list.append(Book(title, author, genre, height, publisher))

def bookDetail(index):
    return book_list[index]

def bookHeaders():
    return ['keyword', 'author', 'genre', 'height', 'publisher']

def getBookValue(prop):
    book: Book = bookDetail(random.randrange(0, len(book_list), 3))
    if prop == 'keyword':
        value = book.keyword
    elif prop == 'author':
        value = book.author
    elif prop == 'genre':
        value = book.genre
    elif prop == 'height':
        value = book.height
    else:
        value = book.publisher
    return value
import csv
import random

class Flight:

    def __init__(self,
year,month,day,day_of_week,airline,flight_number,tail_number,origin_airport,destination_airpo
rt,scheduled_departure,departure_time,departure_delay,taxi_out,wheels_off,scheduled_time,elap
sed_time,air_time,distance,wheels_on,taxi_in,scheduled_arrival,arrival_time,arrival_delay,div
erted,cancelled,cancellation_reason,air_system_delay,security_delay,airline_delay,late_aircra
ft_delay,weather_delay):
        self.year = year
        self.month = month
        self.day = day

```

```

self.day_of_week = day_of_week
self.airline = airline
self.flight_number=flight_number
self.tail_number=tail_number
self.origin_airport = origin_airport
self.destination_airport=destination_airport
self.scheduled_departure=scheduled_departure
self.departure_time=departure_time
self.departure_delay = departure_delay
self.taxi_out = taxi_out
self.wheels_off = wheels_off
self.scheduled_time = scheduled_time
self.elapsed_time = elapsed_time
self.air_time = air_time
self.distance = distance
self.wheels_on = wheels_on
self.taxi_in = taxi_in
self.scheduled_arrival = scheduled_arrival
self.arrival_time = arrival_time
self.arrival_delay = arrival_delay
self.diverted = diverted
self.cancelled = cancelled
self.cancellation_reason = cancellation_reason
self.air_system_delay = air_system_delay
self.security_delay = security_delay
self.airline_delay = airline_delay
self.late_aircraft_delay = late_aircraft_delay
self.weather_delay = weather_delay

```

```
flight_list = []
```

```

with open('flights.csv', newline='') as csv_file:
    reader = csv.reader(csv_file)
    next(reader, None)
    for

```

```

year,month,day,day_of_week,airline,flight_number,tail_number,origin_airport,destination_airpo
rt,scheduled_departure,departure_time,departure_delay,taxi_out,wheels_off,scheduled_time,elap
sed_time,air_time,distance,wheels_on,taxi_in,scheduled_arrival,arrival_time,arrival_delay,div
erted,cancelled,cancellation_reason,air_system_delay,security_delay,airline_delay,late_aircra
ft_delay,weather_delay in reader:

```

```

flight_list.append(Flight(year,month,day,day_of_week,airline,flight_number,tail_number,origin
_airport,destination_airport,scheduled_departure,departure_time,departure_delay,taxi_out,whee
ls_off,scheduled_time,elapsed_time,air_time,distance,wheels_on,taxi_in,scheduled_arrival,arri
val_time,arrival_delay,diverted,cancelled,cancellation_reason,air_system_delay,security_delay
,airline_delay,late_aircraft_delay,weather_delay))

```

```

def flightDetail(index):
    return flight_list[index]

```

```

def flightHeaders():
    return
['year', 'month', 'day', 'day_of_week', 'airline', 'flight_number', 'tail_number', 'origin_airport',
'destination_airport', 'scheduled_departure', 'departure_time', 'departure_delay', 'taxi_out', 'wh
eels_off', 'scheduled_time', 'elapsed_time', 'air_time', 'distance', 'wheels_on', 'taxi_in', 'schedu
led_arrival', 'arrival_time', 'arrival_delay', 'diverted', 'cancelled', 'cancellation_reason', 'air
_system_delay', 'security_delay', 'airline_delay', 'late_aircraft_delay', 'weather_delay']

```

```

def getFlightValue(prop):
    flight: Flight = flightDetail(random.randrange(0, len(flight_list), 3))

```

```

if prop == 'year':
    value = flight.year
elif prop == 'month':
    value = flight.month
elif prop == 'day':
    value = flight.day
elif prop == 'day_of_week':
    value = flight.day_of_week
elif prop == 'airline':
    value = flight.airline
elif prop == 'flight_number':
    value = flight.flight_number
elif prop == 'tail_number':
    value = flight.tail_number
elif prop == 'origin_airport':
    value = flight.origin_airport
elif prop == 'destination_airport':
    value = flight.destination_airport
elif prop == 'scheduled_departure':
    value = flight.scheduled_departure
elif prop == 'departure_time':
    value = flight.departure_time
elif prop == 'departure_delay':
    value = flight.departure_delay
elif prop == 'taxi_out':
    value = flight.taxi_out
elif prop == 'state':
    value = flight.wheels_off
elif prop == 'wheels_off':
    value = flight.scheduled_time
elif prop == 'elapsed_time':
    value = flight.elapsed_time
elif prop == 'air_time':
    value = flight.air_time
elif prop == 'distance':
    value = flight.distance
elif prop == 'wheels_on':
    value = flight.wheels_on
elif prop == 'taxi_in':
    value = flight.taxi_in
elif prop == 'scheduled_arrival':
    value = flight.scheduled_arrival
elif prop == 'arrival_time':
    value = flight.arrival_time
elif prop == 'arrival_delay':
    value = flight.arrival_delay
elif prop == 'diverted':
    value = flight.diverted
elif prop == 'cancelled':
    value = flight.cancelled
elif prop == 'cancellation_reason':
    value = flight.cancellation_reason
elif prop == 'air_system_delay':
    value = flight.air_system_delay
elif prop == 'security_delay':
    value = flight.security_delay
elif prop == 'airline_delay':
    value = flight.airline_delay
elif prop == 'late_aircraft_delay':
    value = flight.late_aircraft_delay
else:

```

```

        value = flight.weather_delay
    return value

#print(getFlightValue('flight_number'))

import csv
import random

class Flight:

    def __init__(self,
year,month,day,day_of_week,airline,flight_number,tail_number,origin_airport,destination_airpo
rt,scheduled_departure,departure_time,departure_delay,taxi_out,wheels_off,scheduled_time,elap
sed_time,air_time,distance,wheels_on,taxi_in,scheduled_arrival,arrival_time,arrival_delay,div
erted,cancelled,cancellation_reason,air_system_delay,security_delay,airline_delay,late_aircra
ft_delay,weather_delay):
        self.year = year
        self.month = month
        self.day = day
        self.day_of_week = day_of_week
        self.airline = airline
        self.flight_number=flight_number
        self.tail_number=tail_number
        self.origin_airport = origin_airport
        self.destination_airport=destination_airport
        self.scheduled_departure=scheduled_departure
        self.departure_time=departure_time
        self.departure_delay = departure_delay
        self.taxi_out = taxi_out
        self.wheels_off = wheels_off
        self.scheduled_time = scheduled_time
        self.elapsed_time = elapsed_time
        self.air_time = air_time
        self.distance = distance
        self.wheels_on = wheels_on
        self.taxi_in = taxi_in
        self.scheduled_arrival = scheduled_arrival
        self.arrival_time = arrival_time
        self.arrival_delay = arrival_delay
        self.diverted = diverted
        self.cancelled = cancelled
        self.cancellation_reason = cancellation_reason
        self.air_system_delay = air_system_delay
        self.security_delay = security_delay
        self.airline_delay = airline_delay
        self.late_aircraft_delay = late_aircraft_delay
        self.weather_delay = weather_delay

flight_list = []

with open('flights.csv', newline='') as csv_file:
    reader = csv.reader(csv_file)
    next(reader, None)
    for
year,month,day,day_of_week,airline,flight_number,tail_number,origin_airport,destination_airpo
rt,scheduled_departure,departure_time,departure_delay,taxi_out,wheels_off,scheduled_time,elap

```

```
sed_time,air_time,distance,wheels_on,taxi_in,scheduled_arrival,arrival_time,arrival_delay,div
erted,cancelled,cancellation_reason,air_system_delay,security_delay,airline_delay,late_aircra
ft_delay,weather_delay in reader:
```

```
flight_list.append(Flight(year,month,day,day_of_week,airline,flight_number,tail_number,origin
_airport,destination_airport,scheduled_departure,departure_time,departure_delay,taxi_out,whee
ls_off,scheduled_time,elapsed_time,air_time,distance,wheels_on,taxi_in,scheduled_arrival,arri
val_time,arrival_delay,diverted,cancelled,cancellation_reason,air_system_delay,security_delay
,airline_delay,late_aircraft_delay,weather_delay))
```

```
def flightDetail(index):
    return flight_list[index]
```

```
def flightHeaders():
    return
['year','month','day','day_of_week','airline','flight_number','tail_number','origin_airport',
'destination_airport','scheduled_departure','departure_time','departure_delay','taxi_out','wh
eels_off','scheduled_time','elapsed_time','air_time','distance','wheels_on','taxi_in','schedu
led_arrival','arrival_time','arrival_delay','diverted','cancelled','cancellation_reason','air
_system_delay','security_delay','airline_delay','late_aircraft_delay','weather_delay']
```

```
def getFlightValue(prop):
    flight: Flight = flightDetail(random.randrange(0, len(flight_list), 3))
    if prop == 'year':
        value = flight.year
    elif prop == 'month':
        value = flight.month
    elif prop == 'day':
        value = flight.day
    elif prop == 'day_of_week':
        value = flight.day_of_week
    elif prop == 'airline':
        value = flight.airline
    elif prop == 'flight_number':
        value = flight.flight_number
    elif prop == 'tail_number':
        value = flight.tail_number
    elif prop == 'origin_airport':
        value = flight.origin_airport
    elif prop == 'destination_airport':
        value = flight.destination_airport
    elif prop == 'scheduled_departure':
        value = flight.scheduled_departure
    elif prop == 'departure_time':
        value = flight.departure_time
    elif prop == 'departure_delay':
        value = flight.departure_delay
    elif prop == 'taxi_out':
        value = flight.taxi_out
    elif prop == 'state':
        value = flight.wheels_off
    elif prop == 'wheels_off':
        value = flight.scheduled_time
    elif prop == 'elapsed_time':
        value = flight.elapsed_time
    elif prop == 'air_time':
        value = flight.air_time
    elif prop == 'distance':
        value = flight.distance
    elif prop == 'wheels_on':
        value = flight.wheels_on
```

```

elif prop == 'taxi_in':
    value = flight.taxi_in
elif prop == 'scheduled_arrival':
    value = flight.scheduled_arrival
elif prop == 'arrival_time':
    value = flight.arrival_time
elif prop == 'arrival_delay':
    value = flight.arrival_delay
elif prop == 'diverted':
    value = flight.diverted
elif prop == 'cancelled':
    value = flight.cancelled
elif prop == 'cancellation_reason':
    value = flight.cancellation_reason
elif prop == 'air_system_delay':
    value = flight.air_system_delay
elif prop == 'security_delay':
    value = flight.security_delay
elif prop == 'airline_delay':
    value = flight.airline_delay
elif prop == 'late_aircraft_delay':
    value = flight.late_aircraft_delay
else:
    value = flight.weather_delay
return value

```

```
#print(getFlightValue('flight_number'))
```

```

import csv
import random
#
uniq_id,crawl_timestamp,product_url,product_name,product_category_tree,pid,retail_price,discounted_price,
#
image,is_FK_Advantage_product,description,product_rating,overall_rating,brand,product_specifications
class Product:
    def __init__(self,
Id,Name,StockQuantity,Price,Description,Category,ProductType,PaymentMethod):
        self.Id = Id
        self.Name = Name
        self.StockQuantity = StockQuantity
        self.Price = Price
        self.Description = Description
        self.Category = Category
        self.ProductType = ProductType
        self.PaymentMethod = PaymentMethod

product_list = []

with open('products.csv', newline='') as csv_file:
    reader = csv.reader(csv_file)
    next(reader, None)
    for Id,Name,StockQuantity,Price,Description,Category,ProductType,PaymentMethod in reader:

product_list.append(Product(Id,Name,StockQuantity,Price,Description,Category,ProductType,PaymentMethod))

```



```

def productDetail(index):
    return product_list[index]

def productHeaders():
    return ['uniq_id', 'crawl_timestamp',
'product_url', 'product_name', 'product_category_tree', 'pid,retail_price', 'discounted_price', 'i
mage', 'is_FK_Advantage_product', 'description', 'product_rating', 'overall_rating', 'brand,produc
t_specifications']

def getProductValue(prop):
    product: Product = productDetail(random.randrange(0, len(product_list), 3))

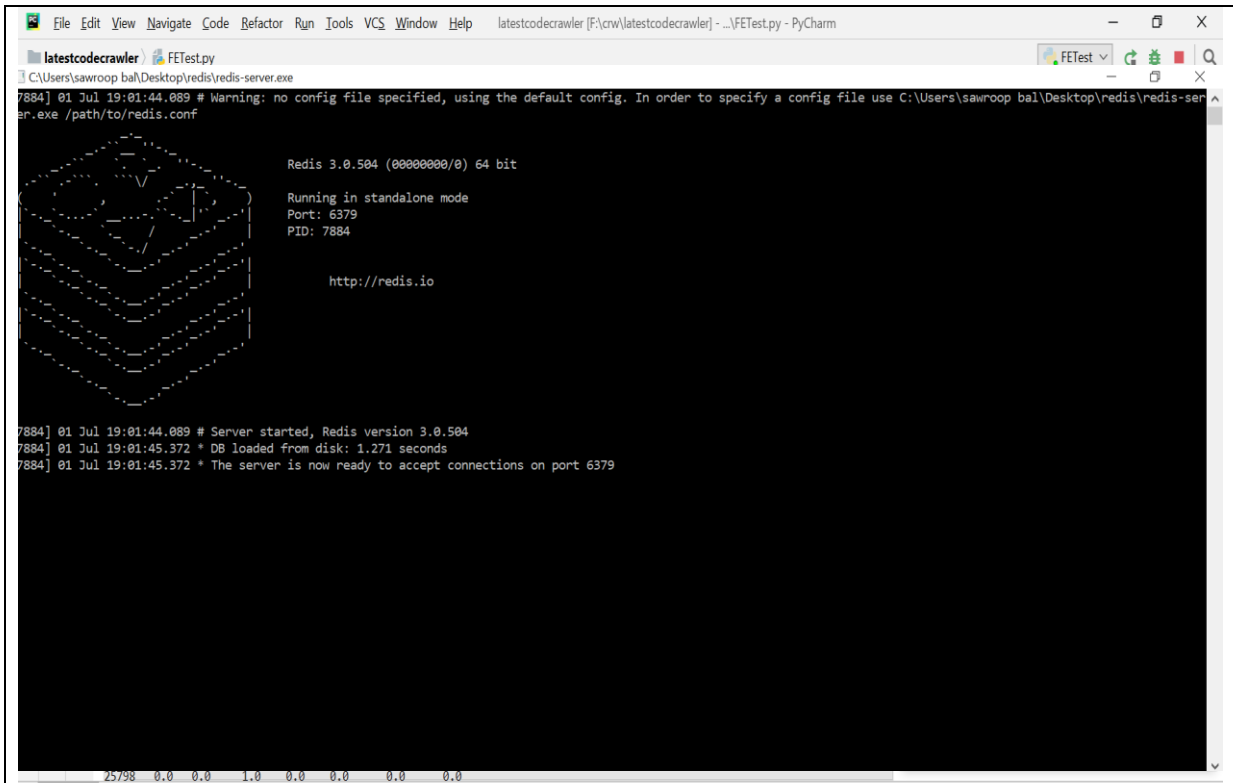
    if prop == 'Id':
        value = product.Id
    elif prop == 'Name':
        value = product.Name
    elif prop == 'StockQuantity':
        value = product.StockQuantity
    elif prop == 'Price':
        value = product.Price
    elif prop == 'Description':
        value = product.Description
    elif prop == 'Category':
        value = product.Category
    elif prop == 'ProductType':
        value = product.ProductType
    else:
        value = product.PaymentMethod
    return value

#print(getProductValue(PaymentMethod))

```

Annexure 2

EXECUTION STEPS

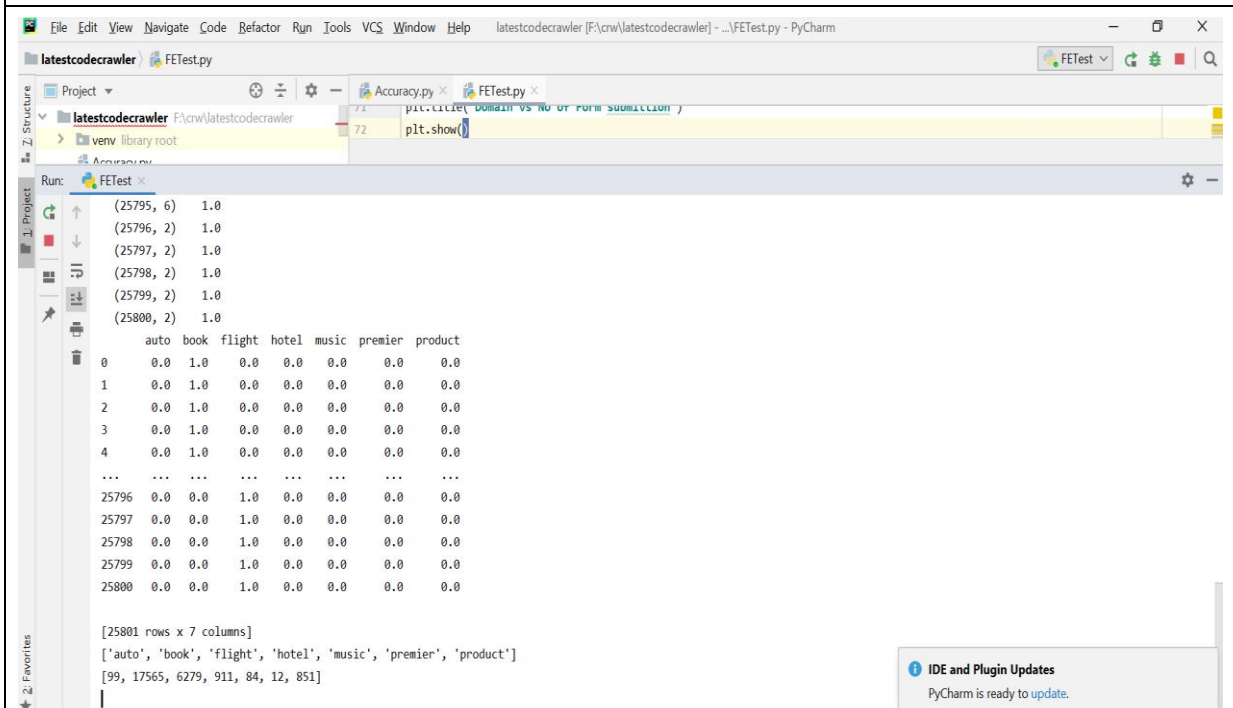


```
latestcodecrawler [F:\crw\latestcodecrawler] - ...FETest.py - PyCharm
latestcodecrawler FETest.py
C:\Users\sawroop ba\Desktop\redis\redis-server.exe
7884] 01 Jul 19:01:44.089 # Warning: no config file specified, using the default config. In order to specify a config file use C:\Users\sawroop ba\Desktop\redis\redis-server.exe /path/to/redis.conf

Redis 3.0.504 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 7884

http://redis.io

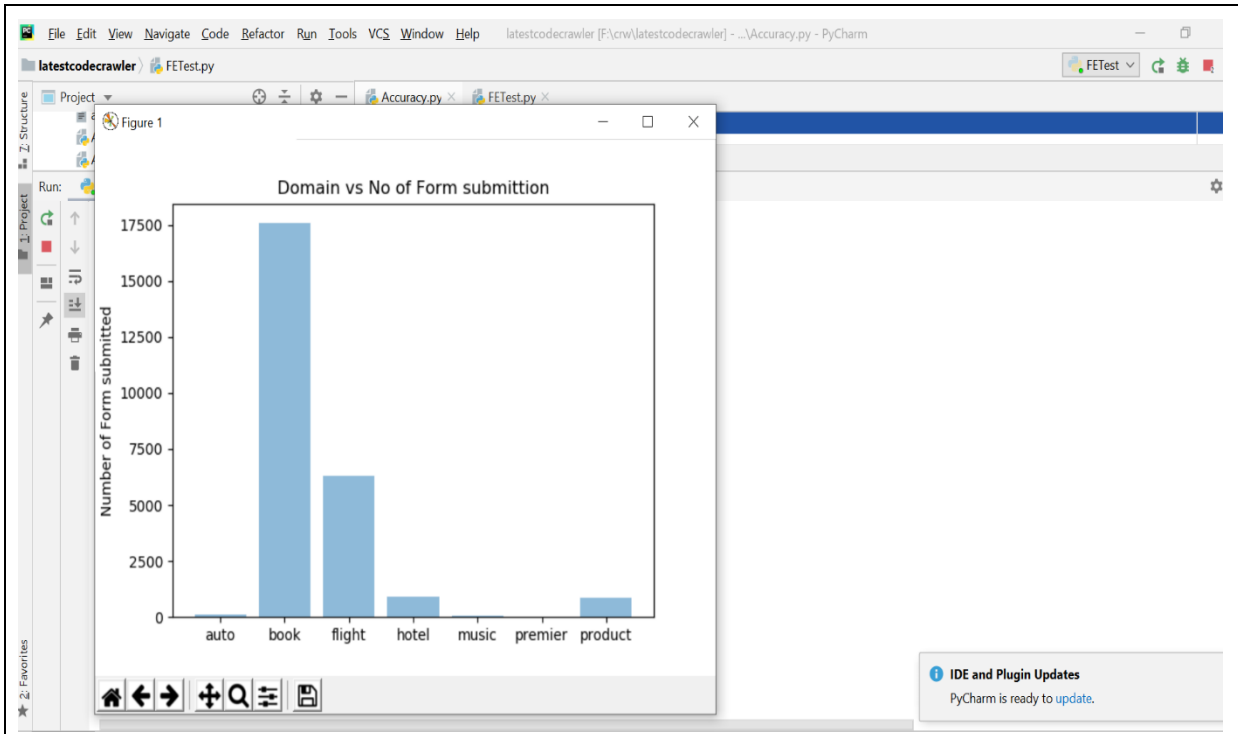
7884] 01 Jul 19:01:44.089 # Server started, Redis version 3.0.504
7884] 01 Jul 19:01:45.372 * DB loaded from disk: 1.271 seconds
7884] 01 Jul 19:01:45.372 * The server is now ready to accept connections on port 6379
```



```
latestcodecrawler [F:\crw\latestcodecrawler] - ...FETest.py - PyCharm
latestcodecrawler FETest.py
Accuracy.py x FETest.py x
latestcodecrawler F:\crw\latestcodecrawler
venv library root
Accuracy.py
Run: FETest.py x
(25795, 6) 1.0
(25796, 2) 1.0
(25797, 2) 1.0
(25798, 2) 1.0
(25799, 2) 1.0
(25800, 2) 1.0
auto book flight hotel music premier product
0 0.0 1.0 0.0 0.0 0.0 0.0
1 0.0 1.0 0.0 0.0 0.0 0.0
2 0.0 1.0 0.0 0.0 0.0 0.0
3 0.0 1.0 0.0 0.0 0.0 0.0
4 0.0 1.0 0.0 0.0 0.0 0.0
... ..
25796 0.0 0.0 1.0 0.0 0.0 0.0
25797 0.0 0.0 1.0 0.0 0.0 0.0
25798 0.0 0.0 1.0 0.0 0.0 0.0
25799 0.0 0.0 1.0 0.0 0.0 0.0
25800 0.0 0.0 1.0 0.0 0.0 0.0

[25801 rows x 7 columns]
['auto', 'book', 'flight', 'hotel', 'music', 'premier', 'product']
[99, 17565, 6279, 911, 84, 12, 851]

IDE and Plugin Updates
PyCharm is ready to update.
```



Link-names-final(4).csv - Excel

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	URL	METHOD	ACTION	BASEURL	DEPTH	PATHVAL	DOMAIN	STATUSCODE										
2																		
3	https://w	get	https://w//b		2	b	book	503										
4																		
5	https://w	get	https://w//b		2	b	book	503										
6																		
7	https://w	get	https://w//b		2	b	book	503										
8																		
9	https://w	get	https://w//b		2	b	book	503										
10																		
11	https://w	get	https://w//b		2	b	book	503										
12																		
13	https://w	get	https://w//b		2	b	book	503										
14																		
15	https://w	get	https://w//b		2	b	book	503										
16																		
17	https://w	get	https://w//b		2	b	book	503										
18																		
19	https://w	get	https://w//b		2	b	book	503										
20																		

sorted data.xlsx - Excel SAWROOP kaur

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Clipboard Font Alignment Number Styles Cells Editing

G15

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	URL	METHOD	ACTION	BASEURL	DEPTH	PATHVALL	DOMAIN	STATUSCODE											
2	https://w/post	https://w//gp/prod	5	gp produ	flight	500													
3	https://w/post	https://w//gp/prod	5	gp produ	flight	500													
4	https://w/post	https://w//Optimurr	3	Optimum	flight	500													
5	https://w/post	https://w//Optimurr	3	Optimum	flight	500													
6	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500													
7	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500													
8	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500													
9	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500													
10	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500													
11	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500													
12	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500													
13	https://co post	https://co /t5/Annou	4	t5 Annou	flight	500	flight domain with status 500 = 12												
14																			
15																			
16																			
17																			
18																			

500 flight 500 hotel 503 auto 503 book 503 flight 503 hotel 503 music 524 book

sorted data.xlsx - Excel SAWROOP kaur

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Clipboard Font Alignment Number Styles Cells Editing

I3

hotel domain with status 500 = 2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	URL	METHOD	ACTION	BASEURL	DEPTH	PATHVALL	DOMAIN	STATUSCODE											
2	https://w/post	https://w//AXE-Dark	3	AXE Dark	hotel	500													
3	https://w/post	https://w//dp/B07N	3	dp B07N	hotel	500	hotel domain with status 500 = 2												
4																			
5																			
6																			
7																			
8																			
9																			
10																			
11																			
12																			
13																			
14																			
15																			
16																			
17																			
18																			

500 flight 500 hotel 503 auto 503 book 503 flight 503 hotel 503 music 524 book

sorted data.xlsx - Excel SAWROOP kaur

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Clipboard Font Alignment Number Styles Cells Editing

136 auto domain with status 503= 35

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	URL	METHOD	ACTION	BASEURL	DEPTH	PATHVALL	DOMAIN	STATUSCODE											
2	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
3	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
4	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
5	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
6	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
7	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
8	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
9	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
10	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
11	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
12	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
13	https://w/post		https://w/Mi-Band-		3	Mi Band f	auto	503											
14	https://w/post		https://w/dp/B07H		2	dp B07HC	auto	503											
15	https://w/post		https://w/dp/B07H		2	dp B07HC	auto	503											
16	https://w/post		https://w/dp/B07H		2	dp B07HC	auto	503											
17	https://w/post		https://w/dp/B07H		2	dp B07HC	auto	503											
18	https://w/post		https://w/dp/B07H		2	dp B07HC	auto	503											
19	https://w/post		https://w/dp/B07H		2	dp B07HC	auto	503											

sorted data.xlsx - Excel SAWROOP kaur

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Clipboard Font Alignment Number Styles Cells Editing

19970 Coverage of book domain= 9969

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	URL	METHOD	ACTION	BASEURL	DEPTH	PATHVALL	DOMAIN	STATUSCODE										
2	https://w/get		https://w/blog/		2	blog	book	200										
3	https://w/get		https://w/blog/		2	blog	book	200										
4	https://w/get		https://w/blog/was		3	blog wass	book	200										
5	https://w/get		https://w/blog/was		3	blog wass	book	200										
6	https://w/post		https://w/blog/was		3	blog wass	book	200										
7	https://w/post		https://w/blog/was		3	blog wass	book	200										
8	https://w/post		https://w/blog/was		3	blog wass	book	200										
9	https://w/post		https://w/blog/was		3	blog wass	book	200										
10	https://w/post		https://w/blog/was		3	blog wass	book	200										
11	https://w/post		https://w/blog/was		3	blog wass	book	200										
12	https://w/post		https://w/blog/was		3	blog wass	book	200										
13	https://w/post		https://w/blog/was		3	blog wass	book	200										
14	https://w/get		https://w/blog/and		3	blog andy	book	200										
15	https://w/get		https://w/blog/and		3	blog andy	book	200										
16	https://w/post		https://w/blog/and		3	blog andy	book	200										
17	https://w/post		https://w/blog/and		3	blog andy	book	200										
18	https://w/post		https://w/blog/and		3	blog andy	book	200										
19	https://w/post		https://w/blog/and		3	blog andy	book	200										
20	https://w/post		https://w/blog/and		3	blog andy	book	200										

200 book 200 product 200 auto 200 flight 200 hotel 200 music 200 premier 400 autc ...

sorted data.xlsx - Excel SAWROOP kaur

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Clipboard Font Alignment Number Styles Cells Editing

I10 Coverage auto domain = 9

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	URL	METHOD	ACTION	BASEURL	DEPTH	PATHVALL	DOMAIN	STATUSCODE											
2	https://w1	post	https://w1/Mi-Band-		3	Mi Band f	auto	400											
3	https://w1	post	https://w1/Mi-Band-		3	Mi Band f	auto	400											
4	https://w1	post	https://w1/Mi-Band-		3	Mi Band f	auto	400											
5	https://w1	post	https://w1/Mi-Band-		3	Mi Band f	auto	400											
6	https://w1	post	https://w1/dp/B07H		2	dp B07HC	auto	400											
7	https://w1	post	https://w1/dp/B07H		2	dp B07HC	auto	400											
8	https://w1	post	https://w1/dp/B07H		2	dp B07HC	auto	400											
9	https://w1	post	https://w1/dp/B07H		2	dp B07HC	auto	400											
10	https://w1	post	https://w1/dp/B07H		2	dp B07HC	auto	400	Coverage auto domain = 9										
11																			
12																			
13																			
14																			
15																			
16																			
17																			
18																			

sorted data.xlsx - Excel SAWROOP kaur

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Clipboard Font Alignment Number Styles Cells Editing

I16

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	URL	METHOD	ACTION	BASEURL	DEPTH	PATHVALL	DOMAIN	STATUSCODE										
2	https://w1	post	https://w1//gp/prod		5	gp produ	music	413										
3	https://w1	post	https://w1//gp/prod		5	gp produ	music	413										
4	https://w1	post	https://w1//gp/prod		5	gp produ	music	413										
5	https://w1	post	https://w1//gp/prod		5	gp produ	music	413										
6	https://w1	post	https://w1//gp/prod		5	gp produ	music	413										
7	https://w1	post	https://w1//gp/prod		5	gp produ	music	413	music domain with status code 413= 6									
8																		
9																		
10																		
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		

Annexure 3

List of publications

1. Sawroop Kaur, Aman Singh, G. Geetha and Xiaochun Cheng, “IHWC: Intelligent Hidden Web Crawler for Harvesting Data in Urban Domains”, *Complex & Intelligent Systems*, Springer, **SCI Indexed, IF 3.791** (Accepted with revisions).
2. Sawroop Kaur, Aman Singh, G. Geetha, Mehedi Masud and Mohammed A. ALZain, “SmartCrawler: A Three-Stage Ranking Based Web Crawler for Harvesting Hidden Web Sources”, *Computers, Materials & Continua*, Scopus Journal, **SCI Indexed, IF 4.89**. (Accepted).
3. Sawroop Kaur and G. Geetha, “Simhar- smart distributed web crawler for the hidden web using simhash and redis server”, *IEEE ACCESS*, **SCI Indexed, IF 3.745**.
<https://ieeexplore.ieee.org/abstract/document/9123854>
4. Sawroop Kaur and G. Geetha, “Smart focused web crawler for hidden web”, *Third International Conference on ICT for Competitive Strategies*. (LNNS, volume 40), **Springer Indexed**.
https://link.springer.com/chapter/10.1007/978-981-13-0586-3_42
5. Sawroop Kaur and G. Geetha, “Advances in web crawler” in *Journal of Control Theory and applications*, **Scopus Indexed**.
6. Bal, S. K., & Geetha, G. (2016, February). Smart distributed web crawler. In *2016 International Conference on Information Communication and Embedded Systems (ICICES)* (pp. 1-5). IEEE.
<https://ieeexplore.ieee.org/abstract/document/7518893>
7. Kaur, S., & Geetha, G. (2019). Smart Focused Web Crawler for Hidden Web. In *Information and Communication Technology for Competitive Strategies* (pp. 419-427). Springer, Singapore.
https://link.springer.com/chapter/10.1007/978-981-13-0586-3_42
8. Sawroop Kaur¹, Aman Singh^{1*}, G. Geetha² “Bibliometric study of web crawlers”. “Bibliographic Study of Web Crawlers” *IEEE 9th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO'2021)* (Under review).