

Design and Implementation of a Parallel Hidden Web Crawler

THESIS

Submitted in fulfillment of the requirement of the degree of

DOCTOR OF PHILOSOPHY

to

YMCA UNIVERSITY OF SCIENCE & TECHNOLOGY

by

Sonali Gupta

Registration No: YMCAUST / Ph09 / 2010

Under the Supervision of

DR. KOMAL KUMAR BHATIA

ASSOCIATE PROFESSOR



Department of Computer Engineering

Faculty of Engineering and Technology

YMCA University of Science & Technology

Sector-6, Mathura Road, Faridabad, Haryana, INDIA

MAY, 2015

DEDICATION

I dedicate this thesis to the almighty GOD
who
gave me strength and patience to complete this research work.

DECLARATION

I hereby declare that this thesis entitled “**DESIGN & IMPLEMENTATION OF PARALLEL HIDDEN WEB CRAWLER**” by **MS. SONALI GUPTA**, being submitted in fulfillment of the requirement for the award of Degree of Doctor of Philosophy in the Department of Computer Engineering under Faculty of Engineering and Technology of YMCA University of Science and Technology, Faridabad, during the period May 2011 to January 2015, is a bonafide record of my original work carried out under the guidance and supervision of **DR. KOMAL KUMAR BHATIA, ASSOCIATE PROFESSOR, DEPARTMENT OF COMPUTER ENGINEERING** and has not been presented elsewhere.

I further declare that the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

(SONALI GUPTA)

Registration No. YMCAUST/Ph09/2010

CERTIFICATE

This is to certify that this thesis entitled “**DESIGN & IMPLEMENTATION OF PARALLEL HIDDEN WEB CRAWLER**” by **MS. SONALI GUPTA**, being submitted in fulfillment of the requirement for the award of Degree of Doctor of Philosophy in the Department of Computer Engineering under Faculty of Engineering and Technology of YMCA University of Science and Technology, Faridabad, during the period May 2011 to January 2015, is a bonafide record of my original work carried out under my guidance and supervision.

I further declare that to the best of my knowledge, the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

Dr. Komal Kumar Bhatia

ASSOCIATE PROFESSOR

Department of Computer Engineering

Faculty of Engineering & Technology

YMCA University of Science & Technology, Faridabad

Dated:

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my Supervisor **Dr. Komal Kumar Bhatia** for giving me this opportunity to work in this area. It would never be possible for me to take this thesis to this level without his innovative ideas and his relentless support and encouragement.

I am thankful to him for his aspiring guidance, invaluable constructive criticism and friendly advice during the research work. I am sincerely grateful to him for sharing his truthful and illuminating views on a number of issues related to the work. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, many thanks to you!!

Before I acknowledge the contributions made by other people, I would be failing in my duty both as a teacher as well as a student; if first and foremost, I do not perceive the opportunity to pay my homage to the seat of learning and wisdom that is **YMCA University of Science & Technology**. My head bows in deep respect and humility to this great institution which has shaped many minds and nurtured seeds of intellect in others. I have been among those fortunate ones who have received a big milestone in their developed career by not only gaining skills and knowledge in the best possible way but also the values that will inspire me for a lifetime.

Over the last six years that I have been associated with YMCAUST, I have made many great friends and found support from many of my colleagues, both within and outside of the Computer Engineering department. I would like to express my sincerest appreciation to **Dr Manjeet Singh** for his valuable technical discussions. It has been my privilege to know and work with him.

Also, I would like to express my sincerest admiration for **Dr C K Nagpal, Dr Naresh Chauhan, Dr. Atul Mishra, Dr. Neelam Duhan, Dr. Jyoti , Dr. Anuradha, Dr. Payal, Ms. Deepika, Ms. Shilpa Munjal, Mr. Sushil , Mr. Vedpal** and other faculty members for their useful and kind cooperation.

I would also like to thank my sincere students of M.Tech, who directly or indirectly helped me.

Every challenging work needs self-efforts as well as the guidance of elders especially those who are very close to our heart. Words cannot express how grateful I am to my father-in-law, **Mr. Ashok Kumar Gupta** , mother-in-law **Mrs. Premlata**, my father

Mr. K.L.Goel and my mother, **Mrs. Saroj Goel** , for all of the sacrifices that you've made on my behalf. Your prayer for me was what sustained me thus far. It is my radiant sentiment to place on record my heartily thanks for their unwavering support and encouragement over the last four years. I would like to express my gratitude to my brother **Dr. Varun Goyal**, Dental Consultant, PGIMS Rohtak, sisters **Ms. Mahima**, Lecturer DAVPS and **Dr.Monika**, Senior Consultant ESI Hospital, Gurgaon and my relatives too.

Finally, my deepest gratitude to the most important individuals in my life, my husband **Vishal ji** for his care, concern, personal support and great patience at every inch and step in pursuing this noble task and my sons **Aarush & Aaditva** for always relieving me of my tensions with their beautiful and inspiring smiles. Love you all!

I can't end without thanking and appreciating my friends for their constant encouragement and love I have relied throughout my research work.

Above all, I must thank the **Almighty**, the all-knowing and omnipresent God for giving me all these wonderful gifts of my lifetime and carrying me safely in his arms all these years with the hope that you will continue to do so forever.

(SONALI GUPTA)

Registration No. YMCAUST/Ph09/2010

Abstract

World Wide Web (WWW) is the largest repository of information that covers data from almost all the areas known to mankind. It is a source of information that is most frequently accessed publicly. This information over the WWW comprises of the hypertext markup language (HTML) documents interconnected through hyperlinks. The Surface Web or the Publically Indexable Web (PIW) includes the content that can be accessed by purely following the hyperlink structure and thus can be crawled and indexed by popular search engines. On the other hand, the Hidden Web refers to the content that is stored in Web databases and distributed through the creation of dynamic web pages. These dynamic web pages are generated based on the results retrieved in response to queries specified at the interface offered by the underlying web database.

Crawling the contents of the hidden Web is a very challenging problem especially because of the fundamental reasons of its scale and restricted search interfaces offered by the Web databases. To overcome the issue of scale, a parallel architecture of the Hidden Web crawler that seems to be an improved option in comparison to the single-process crawler architecture, has been proposed in this work. The proposed crawler is also targeted to automatically extract and integrate the search environment by modelling the search forms and filling them in to retrieve the Hidden Web contents from databases in different domains like Books, travel, Auto etc. But, when multiple instances of the crawler run in parallel, the same web document might be downloaded multiple times as one instance of the web crawler may not be aware of another having already downloaded the page. Thus, it is very important to minimize such multiple downloads to save network bandwidth and increase the crawler's effectiveness by coordinating the parallel processes must be coordinated to minimize overlap. However, the coordination between individual crawling processes needs communication which consumes network bandwidth. So, an important objective is to minimize the communication and the network bandwidth consumption while still achieving the advantage of scalability.

Thus, in this thesis, a novel framework for a parallel Hidden Web Crawler has been designed and implemented. The proposed work not only effectively but also efficiently crawls and extracts the contents in the Hidden web databases. The

proposed work also adopts a domain-specific approach to overcome the problem of heterogeneity across numerous domains and minimize the communication overhead along with reduced network bandwidth consumption. The proposed crawler offers scalability in design as new instances of the various components may be incorporated in the system as per the requirements. In addition, the proposed crawler is also extensible in the sense that third party components or modules can be added as per the requirements.

TABLE OF CONTENT

CHAPTER 1. INTRODUCTION	1
1.1. GENERAL	1
1.2. MOTIVATION	4
1.3. RESEARCH OBJECTIVES OF THE PROPOSED WORK	6
1.4. CONTRIBUTION	7
1.5. ORGANIZATION	8
1.6. CONCLUSION	ERROR! BOOKMARK NOT DEFINED.
CHAPTER 2. WWW AND INFORMATION RETRIEVAL TOOLS : A REVIEW	10
2.1. INTRODUCTION	10
2.2. INFORMATION RETRIEVAL	11
2.2.1. Analysis of Web Content	13
2.2.2. Information Retrieval Tools	15
2.2.2.1. Web Directories	15
2.2.2.2. Search Engines	15
2.2.2.3. Meta-Search Engines	18
2.3. WEB CRAWLERS	19
2.3.1. Robot.txt: Standard for Robot Exclusion	20
2.3.1.1. User-Agent	21
2.3.1.2. Disallow	22
2.4. CHALLENGES FOR A WEB CRAWLER	23
2.4.1. Information Retrieval Terms And Performance Metrics	24
2.5. CLASSIFICATION OF WEB CRAWLERS	26
2.5.1. Focused Crawlers	26
2.5.2. Parallel Crawlers	29
2.5.2.1. Challenges faced with Parallel Crawling Architectures	31
2.5.2.2. Advantages of Parallel Crawler Architecture	32
2.5.3. High-performance Crawlers	33
2.5.3.1. Issues and Challenges with High-Performance Crawlers	38
2.6. THE PROBLEMS WITH THE SEARCH ENGINE LANDSCAPE	39
2.7. CONCLUSION	ERROR! BOOKMARK NOT DEFINED.
CHAPTER 3. HIDDEN WEB: A REVIEW	41
3.1. INTRODUCTION	41
3.2. CHARACTERISTICS OF THE HIDDEN WEB	42
3.3. USER-INTERACTION IN THE HIDDEN WEB	43
3.3.1. Automatic Approach to Access the Hidden Web	44
3.3.2. Difference between a Conventional crawler and Hidden Web crawler	47
3.4. CRAWLING THE HIDDEN WEB	48
3.5. RELATED WORK IN HIDDEN WEB CRAWLING	50
3.5.1. Breadth-Oriented Hidden Web crawlers for Resource Discovery	50
3.5.1.1. Automated Discovery of Search Interfaces on the Web	51
3.5.1.2. Crawling For domain Specific Hidden Web Resources	51
3.5.1.3. ACHE : An adaptive crawler for locating hidden Web Entry points	55
3.5.2. Depth-oriented Hidden Web crawlers for content Extraction	56
3.5.2.1. HiWE	57
3.5.2.2. Hidden Web crawler for the Hidden Seek	59
3.5.2.3. AKSHR: A Domain-specific Hidden web crawler.	61
3.6. COMPARISON OF THE VARIOUS HIDDEN WEB CRAWLERS	63
3.7. PROBLEM STATEMENT	65

CHAPTER 4. DESIGN OF A PARALLEL HIDDEN WEB CRAWLER	68
4.1. INTRODUCTION	68
4.1.1. Proposed Design of the Parallel Hidden Web Crawler.....	70
4.2. PHASE I: CREATING A URL FRONTIER FOR PARALLEL ACCESS.	73
4.3. PHASE II: WEB PAGE COLLECTION AND ANALYSIS	74
4.3.1. URL Allocator	75
4.3.2. A Multi-threaded document Downloader	77
4.3.3. Web page Analyzer	77
4.3.4. URL Ranker	79
4.3.4.1. Domain Score	81
4.3.4.2. Link Score	85
4.4. PHASE III: CREATION AND MAINTENANCE OF DOMAIN-SPECIFIC DATA REPOSITORIES	87
4.4.1. Page Classifier	88
4.4.2. Page Content Extractor	93
CHAPTER 5. PHASE IV: DISCOVERING THE HIDDEN WEB RESOURCES IN A DOMAIN 100	
5.1. INTRODUCTION	100
5.1.1. Form Extractor	101
5.1.2. Label Extractor	101
5.1.3. Match Value Generator	103
5.2. PHASE V: PARALLEL CONTENT EXTRACTION FROM THE HIDDEN WEB	110
5.3. PHASE VI: RESPONSE ANALYSIS	116
5.3.1. Size extractor	118
5.3.2. The Page Content extractor	120
5.4. QUERY RANKER	120
CHAPTER 6. IMPLEMENTATION & RESULT ANALYSIS OF PARALLEL HIDDEN WEB CRAWLER	131
6.1. GENERAL	131
6.2. PERFORMANCE METRICS	132
6.2.1. Data Sets.....	133
6.3. EXPERIMENTS & RESULTS	135
6.3.1. Finding the Form Pages and the PIW Pages.....	135
6.3.2. The Page Classifier and the Page Content Extractor	137
6.3.3. Form analyzer	149
6.3.4. Parallel Processing of Search Forms	158
6.4. COMPARASON OF ALL DOMAINS	165
6.5. COMPARASON OF THE PROPOSED PARALLEL HIDDEN WEB CRAWLER WITH EXISTING WEB CRAWLERS.....	167
CHAPTER 7. CONCLUSION & FUTURE SCOPE.....	170
7.1. CONCLUSION	170
7.2. FUTURE WORK	171
BIBLIOGRAPHY	173
APPENDIX A.....	183
APPENDIX B.....	185

LIST OF TABLES

Table	Page No.
Table 2.1: Percentage of web pages with words in HTML Tags	14
Table 3.1: Comparison of various Hidden Web Crawler	63
Table 4.1: Extracted Back-links for the child URL and their associated domains.	85
Table 4.2: Relevance Score Calculation for new/child URL.	86
Table 4.3: Sample Domain-Specific databases.	93
Table 4.4: A sample of the Form Element Table as generated by the Form Analyzer for Figure 4.16	95
Table 5.1: Parsed Representation for the above form as Form Element Table (FET)	102
Table 5.2: Match value computation using edit distance method for the label ‘ Name’ and other key terms included in different Domain Definitions.	104
Table 5.3: Match Value Computation based on relationship in Domain-Specific Thesaurus	105
Table 5.4: An example of a Match Value Matrix	107
Table 5.5: Labels and type of input associated with them for the Search Interface Form in Figure 5.6	109
Table 5.6: Sample Domain-Specific Database showing the 3 attributes or controls of the search form and the domain of each such attribute.	122
Table 5.7: Set of 8 possible query combination	122
Table 5.8: The query space Q and the assigned query id’s	123
Table 5.9: Computation of random rank values using equation 5.6 for the first run of the crawler	125
Table 5.10: The contents of the page statistics repository for the queries	126
Table 5.11: The static rank of the queries for the second run of the crawler	127
Table 5.12: Dynamic rank values for the second run using equation 5.7	127
Table 5.13: Computation of random_rank for the queries for the second run using equation 5.7	127
Table 5.14: Number of records retrieved by each query	128
Table 5.15: QRS and the dynamic_rank values for the next run of the crawler	129
Table 5.16: Computation of random_rank for the queries for the second run using equation 5.7	129
Table 6.1: Biases and Activation Functions Used	140
Table 6.2: Experimental Results Obtained for the Page Classifier	142
Table 6.3: A sample of candidate values with their occurrences in the web pages in the DSPR	147
Table 6.4: Number of candidate values and valid values generated for the label ‘departure city’ using the six extraction patterns.	148
Table 6.5: Results of the Form Analyzer	157
Table 6.6: Experimental Evaluation of the Query Ranker	163
Table 6.7: Number of Form submissions: Valid, Invalid and Failures	165
Table 6.8: Precision, Recall and F-measure values averaged over all the search forms processed in a domain.	166
Table 6.9: Comparison of proposed parallel hidden web crawler with the existing crawlers	168

LIST OF FIGURES/ GRAPHS

Figure	Page No.
Figure 1.1: A typical search form and a dynamic (or hidden) web page	2
Figure 1.2: Example of user interaction with a hidden web database.	4
Figure 2.1: The overall organization of the World Wide Web	11
Figure 2.2: HTML Document	13
Figure 2.3: Elements of a Basic Search Engine	16
Figure 2.4: Basic Architecture of a Meta-Search Engine.	18
Figure 2.5: Control process of a crawler for the Surface Web	19
Figure 2.6: Algorithm of a crawler.	20
Figure 2.7: a) A standard crawler following each link. b) A focused crawler trying to identify the most promising links	27
Figure 2.8: Focused Crawler Architecture	28
Figure 2.9: Generic Architecture of a Parallel Crawler.	30
Figure 2.10: The architecture of Mercator Crawler	34
Figure 2.11: The UbiCrawler.	36
Figure 3.1: An example of user-interactive search form that offers online search of books.	41
Figure 3.2: The principle of using server-side CGI Programs	43
Figure 3.3: User interaction with a search form interface	44
Figure 3.4: A crawler interacting with the search form interface	45
Figure 3.5: Keyword-based Search Interface	46
Figure 3.6: A multi-attribute search form interface for an online book store	46
Figure 3.7: Principle behind a traditional crawler and the Hidden Web Crawler.	47
Figure 3.8: Architecture of Hidden Web Crawler	53
Figure 3.9: Classes of HTML forms	54
Figure 3.10: The Form Focused crawler (FFC)	55
Figure 3.11: ACHE Architecture	56
Figure 3.12: Crawler Form Interaction	57
Figure 3.13: Architecture of HiWE.	58
Figure 3.14: Algorithm for crawling Hidden Web Site.	59
Figure 3.15: Algorithm for selecting the next query term	61
Figure 3.16: Architecture of a Domain-specific Hidden Web Crawler (AKSHR)	61
Figure 3.17: Two examples Query Interfaces from Books domain and their Hierarchical representation	62
Figure 4.1: A Search Engine with several crawling threads to achieve parallelism.	69
Figure 4.2: Architecture of the proposed Parallel crawler for the Hidden Web.	72
Figure 4.3: Structure of the Prioritized URL Queue for a domain.	74
Figure 4.4: Example of a domain-specific priority queue	76
Figure 4.5: URL Allocator Algorithm	76
Figure 4.6: Multi-threaded Document downloader Algorithm	77
Figure 4.7: Example of a web page having new or child URLs	79
Figure 4.8: The Page Type Identifier Algorithm.	79
Figure 4.9: Example web page containing hyperlinks (Source: www.placetoseeindeli.wordpress.com)	83
Figure 4.10: HTML tag structure of the web page in Figure 4.9.	83
Figure 4.11: Integration of the Page classifier and the Page content Extractor for the creation of Domain-Specific data repository.	88
Figure 4.12: The proposed system for domain identification and page classification.	89
Figure 4.13: Block Diagram of a basic Neural Network (back propagation) [79] and the abstract architecture of the same as used by the proposed system.	90

Figure 4.14: Algorithm for selecting initial input and their weights for the neural network	92
Figure 4.15: The Page Content Extractor	94
Figure 4.16: A sample search form	95
Figure 4.17: Google web page for the query “departure city like”	97
Figure 4.18: Algorithm for Pattern Search and Validator.	99
Figure 5.1: The Form Analyzer	100
Figure 5.2: Algorithm for Form Extractor	101
Figure 5.3: Example search form	102
Figure 5.4: Match Value Generator	103
Figure 5.5: Schematic diagram of Match Value Matrix	107
Figure 5.6: A sample search form from Books domain	109
Figure 5.7: Working of Phase V of the crawler	110
Figure 5.8: Sample of the Domain-Specific search interface repository in Books domain	111
Figure 5.9: Algorithm to distribute load by the SIMs to the associated FPEs	113
Figure 5.10: Algorithm of a Form Processing Element (FPE).	113
Figure 5.11: A sequence diagram showing interaction between the SID, SIM, FPEs and the Web server.	114
Figure 5.12: The working of the Response Analyzer	116
Figure 5.13: A valid response page or a multi-match page from a hidden database in ‘travel’ domain	117
Figure 5.14: A dead Response page or a no-match page from the same hidden database of ‘travel’ domain.	118
Figure 5.15: Example of a search form to a structured database	121
Figure 6.1: Initial list of URLs for ‘Food’ domain	134
Figure 6.2: Interface of the Parallel Hidden Web Crawler	134
Figure 6.3: Interface of the Web Data Extractor employing URL Allocator, Multi-Threaded Document Loader and the Web Page Analyzer.	135
Figure 6.4: Initialization of the Web Data Extractor for URL extraction.	136
Figure 6.5: A snapshot of the list of hyperlinks extracted from downloaded web pages	136
Figure 6.6: The number of pages of each type as identified by the Page Type Identifier	137
Figure 6.7: Initialization of the key term extraction process in (a) with a sample of the input file urlforwde.txt in (b).	139
Figure 6.8: A snapshot during term extraction	139
Figure 6.9: Snapshot of extracted title and keywords from the META tag.	139
Figure 6.10: Training the neural network	140
Figure 6.11: Performance of the Neural Network (error vs iteration)	141
Figure 6.12: Performance of the Page Classifier	143
Figure 6.13: The generated Extraction Patterns in the Travel Domain	144
Figure 6.14: Result page when the extraction pattern ‘departure city like’ was raised on Google.	145
Figure 6.15: Snapshot of .mdb file when analyzing the instances of candidate values in PIW pages stored in the DSPRs.	146
Figure 6.16: Graphical representation of the number of candidate and valid instances by the PSV	148
Figure 6.17: A Sample Search Interface with Labels and Values	149
Figure 6.18: HTML Source Code of the Sample Search Form of Figure 6.17	150
Figure 6.19: The Interface of the LET.	151
Figure 6.20: Layout of the Form Analyzer during the process of label extraction.	151
Figure 6.21: Performance of Form Analyzer when extracting labels.	152
Figure 6.22: Snapshot of the temporary sheet created during label extraction	152
Figure 6.23: The number and Percentage of Labels Extracted in each domain	153
Figure 6.24: Performance of Form Analyzer when extracting values.	154
Figure 6.25: Performance of the Form Analyzer when extracting values of control elements.	155
Figure 6.26: Domain Wise performance of Form Analyzer when extracting labels and values both.	156

Figure 6.27: Overall %age Accuracy of Form Analyzer when extracting labels and values.	156
Figure 6.28: Accuracy of the Form Analyzer while discovering search forms in various domains	157
Figure 6.29: Sample search form from travel domain that is processed and the corresponding response page retrieved	159
Figure 6.30: Parallel Processing of search forms in Travel domain with the query round-trip flights from Delhi to Mumbai on specified dates.	160
Figure 6.31: Parallel downloading of response pages from the Hidden web databases behind the search forms in Figure 6.30.	160
Figure 6.32: Parallel Processing of search forms in Auto domain.	161
Figure 6.33: Parallel downloading of response pages in Auto domain	162
Figure 6.34: Comparison of different Query ranking approaches	164
Figure 6.35: Precision, Recall and F-Measure values in each domain and average over all domains.	166

LIST OF ABBREVIATION

ANN-	Artificial Neural Network
API-	Application Programming Interface
CGI-	Common Gateway Interface
C-Proc-	Crawling Process
DNS-	Domain Name System
DOM-	Document object model
DRP-	Dead Response Page
DSDR-	Domain Specific Data Repository
DSPR-	Domain Specific Page Repository
EP-	Extraction Pattern
FET-	Form Element Table
FIFO-	First In First Out
FPE-	Form Processing Element
HTML-	Hyper text Markup Language
HTTP-	Hyper text Transfer Protocol
HW-	Hidden Web
IDF-	Inverse Document Frequency
IR-	Information Retrieval
MVM-	Match Value Matrix
ODP-	Open Directory Project
PC-	Page Classifier
PCE-	Page Content Extractor
PIW-	Publicly Indexable Web
PSV-	Pattern Searcher and Validator
QRS-	Query Response Size

REP- Robots Exclusion Principle

SID- Search Interface Distributor

SIM-Search Interface Manager

TF- Term Frequency

URL-Uniform Resource Locator

VRP- Valid Response Page

WWW- World Wide Web

CHAPTER 1.

INTRODUCTION

1.1. GENERAL

World Wide Web (WWW) [1] is a system of hyperlinked documents containing useful information that can be accessed via the internet. Since its inception in 1990, WWW has become many folds in size and now it contains more than 50 billion publicly accessible web documents [1, 8, 16], distributed all over the world on thousands of web servers and is still growing at an exponential rate. Thus, WWW has a unique nature with distinctive properties like massive size, geographically distributed, much less coherent, extremely complex and rapidly changing due to which searching and retrieval of information from the Web efficiently and effectively has become challenging task. This problem resulted in the evolution of a branch of information retrieval [2, 4, 12] that is different from traditional IR in the sense that it searches the required information within the new or latest document collection.

Most of the user population takes help of search engines or other similar kind of information retrieval tools to find his or her specific information of interest from the WWW. From a user perspective, a search engine is a query interface, which allows the user to locate the documents of his interest in the WWW. The user can access the knowledge (relevant web pages) behind the query interface by using keywords as queries. Most search engines search for Hypertext Markup Language (HTML) documents from the Internet and store them into an index database. A hypertext document consists of both, the contents and the links to related documents. The content can be either in the form of text, images, videos or any other multimedia. And the links embedded within the document are commonly known as hyperlinks or Uniform Resource Locators (URLs).

But, current search engines cannot index everything the Web has to offer because of certain technical complications offered by the Web and the current search engine technology. These complications precisely divide the Web into two sections: the Surface Web and the Hidden Web.

The Surface Web is the section containing the static web pages that can be crawled and indexed by the search engines while the Hidden Web comprises of the abundant

information that is hidden behind the search forms in back-end web databases. Hidden Web comprises of the structured data often published as web pages that are dynamically generated based on the database contents. For example, if a user wants to search information about some flight, then in order to get the required information, he/she must go to airline site and fill the details in the search form acting as an interface to the web database. As a result he/she gets the details of the flights available. These types of pages are often referred to as dynamic or hidden web pages. Figure 1.1(a) shows an example of such a search form that offers a search over the flights between two cities. Figure 1.1(b) shows an example of a dynamic or hidden web page.

Figure 1.1(a) is a flight search form with the following fields and options:

- Trip Type:** Radio buttons for "Return Trip" (selected) and "One way".
- Leaving from:** A dropdown menu.
- Going to:** A dropdown menu.
- Departure:** A date field with format "DD/MM/YYYY" and a calendar icon.
- Return:** A date field with format "DD/MM/YYYY" and a calendar icon.
- Time:** Two dropdown menus, both set to "Any time".
- Travelers (Up to 6 per booking):**
 - Adults (12+): A dropdown menu set to "1".
 - Children (2-11): A dropdown menu set to "0".
 - Infants (0-2): A dropdown menu set to "0".
- Search Button:** A green button labeled "Search for flights".

Figure 1.1(b) is a dynamic web page showing flight results for a search from New Delhi to Bhopal. The page includes a header with navigation links and a table of flight results:

Airline	Class	Duration	Actual	Price per adult	Save points
Multiple Airlines	21h, 10m	15:45	15:50 (New del)	Rs. 15,514	Save 240 points
Multiple Airlines	23h, 45m	15:05	15:50 (New del)	Rs. 16,165	Save 220 points
AI 64	21h, 10m	14:00	15:50 (New del)	Rs. 16,501	Save 230 points
AI 64	23h, 45m	15:05	15:50 (New del)	Rs. 16,501	Save 230 points

Figure 1.1: A typical search form and a dynamic (or hidden) web page

These dynamic web pages can hardly be accessed by web crawlers and are denied from inclusion in the search engine indexes and if not so, they corrupt the search result with outdated information which is not worthwhile in case of virtual marketplaces or real-time information systems [29]. The alternative names for the Surface Web include the Publicly Indexable Web (PIW) whereas those for the Hidden Web include the Deep Web or the Invisible Web.

With the help of search engines, typical users in search of the information swim at the surface of the Web leaving an enormous amount of the high quality information that is stored in web databases which can often be found in the depths of the Hidden Web. A study by Bergman [28] estimated that the content in the Hidden Web is 500 times larger than the Surface Web and approximately 7,500 terabytes of data resides inside the Hidden Web databases. Studies in [9, 10, 11] estimate that the Hidden Web consists of about 91,000 terabytes while the survey conducted and published by Madhavan [16] identified over 647,000 Hidden web resources. The survey in [16] was based on a small fraction of 25 million randomly selected web pages taken from Google's index. In contrast, the Surface Web only contains 167 terabytes of data. Hence, the typical search engines are only capable of accessing 20% of the Web [28], ignoring the structured and high quality data lying in the hidden web databases.

Below are some of the characteristics that mention and explain the importance of accessing the Hidden Web:

- 1) **Distributed and Diverse:** The hidden Web being a part of the WWW blindly follows the idea of decentralized publishing and hence does not own any single organization dedicatedly responsible for maintaining its content. As Web was developed in a decentralized democratic way, so are the Web databases drawn out of many sources, each with its own organization [32].
- 2) **Dynamic Nature:** The dynamic nature of the Hidden Web can be witnessed from the fluid nature of the WWW itself. The World Wide Web takes the form of an ever-evolving information source and, over time, webpage and even whole websites, appear and disappear, are updated or restructured etc. leading to the 'broken link' problem denoted by the HTTP Error 404 of which most of the Web users are aware.
- 3) **Size:** The Hidden Web is estimated to contain 91,000 terabytes of information compared to 167 terabytes of information in the Surface Web [16, 29, 30, 39]. And increasingly, the content rich databases from universities, libraries, associations, businesses and government agencies are being made available online, using Web interfaces as front end.
- 4) **Content Quality:** The content on this part of the Web is believed to be authentic and of very high quality as it is contained within authorized

databases. The Hidden Web now exists as an inherently reliable source for supporting research and personal communication and thus, increasing the Web usage for finding prints and papers in online repositories, for participating in various ongoing online discussions and for various other purposes like e-marketing & e-shopping (ordering product delivery online).

Also, the documents on the Hidden Web are not reachable by following the hyperlinked structure of the Web graph. This content is accessible only through the search tools purposely designed for that site allowing searching in real time and retrieving the information that is current, up-to-minute. Figure 1.2 schematically depicts a user interaction, in which a user searches for flights via one of the interfaces on makemytrip.com and gets a web page with search results.

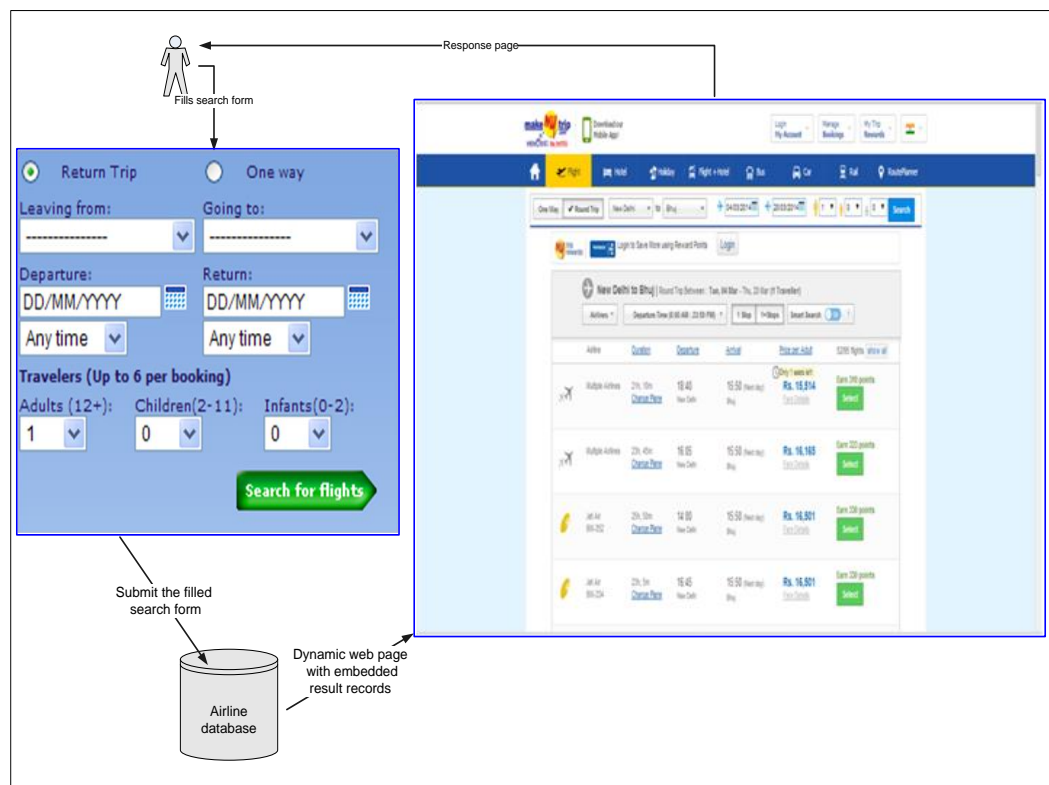


Figure 1.2: Example of user interaction with a hidden web database.

1.2. MOTIVATION

The increasing prevalence of online databases has influenced the structure of the web and the capabilities of information retrieval and search tools. As the WWW continues to grow at an exponential rate, the problem of accurately retrieving information from

this ever expanding Hidden Web also continues to exacerbate. Crawling the Hidden Web is a very challenging problem especially because of following reasons:

- Access to these databases is provided only through restricted search interfaces, intended to be filled manually. Manually filling each and every search form is not only infeasible but cumbersome task due to the huge and ever increasing nature of the Hidden Web.
- To automatically process any arbitrary search form, issuing a query is extremely complex for a crawler which neither possesses knowledge nor is intelligent enough to bypass these search forms that are primarily designed for human understanding and use. The lack of knowledge of the underlying database schema, makes the task further more complicated.
- The Hidden Web Crawler should not only be capable of automatically processing the search forms but also of making an optimal choice among the candidate queries to be raised by it.
- Moreover, the scale and the size of the Hidden Web is very large. As the volume of information in the hidden-web grows, it is expected to complete the crawl of the portion of the web for a particular domain within the expected time. This makes the current Hidden Web crawlers inefficient to crawl the Hidden Web. The Hidden Web crawler must target itself to achieve the desired download rate which can either be done by employing several processes or by making the crawler capable enough to find an optimal query among the candidate queries to be used for filling the search forms.

Thus, the parallel architecture of Hidden Web crawler seems a better option as compared to simple crawler architecture as the parallel crawlers are capable of performing the job in a much shorter span of time and cover more and more information from Hidden websites [23, 84, 85]. Therefore, in this work, a hidden-web crawler has been developed with in view to resolve the problem faced by a simple Hidden Web crawler. The proposed parallel crawler for the Hidden Web is targeted to automatically extract and integrate the search environment by retrieving the hidden web contents from different domains like Books, Travel, Automobile etc.

1.3. RESEARCH OBJECTIVES OF THE PROPOSED WORK

In the light of the above motivation factors, the main objective of the proposed work is to develop a parallel crawler for the Hidden Web. The specific objectives of the present work are as follows:

- 1) ***Search Form representation:*** A *search form* allows the users to type a query in order to search some items on the web without changing them. These *searchable forms* serve as the entry point for the hidden-web. So, in order to efficiently process form, an important objective of the proposed work is to create a parsed internal representation of the searchable forms.
- 2) ***Scalability:*** To efficiently extract information on different topics/ domains, another important objective of the proposed work is to design a crawler that scales its performance in accordance to the increase in information, number of databases and number of domains on the WWW.
- 3) ***Domain specific Approach:*** To get the best information out of different domains and provide comprehensive coverage of the Hidden Web contents, a domain-specific approach that overcomes the problem of heterogeneity must be adopted for crawling.
- 4) ***Creating a domain specific repository:*** Since the Hidden Web content contains very large amount of information in the form of heterogeneous databases for different domains and to automate filling of the search forms from different domains, another objective of the proposed system is to create various domain specific repositories that facilitate filling and processing of search forms in a domain.
- 5) ***Synchronizing Parallel tasks/processes:*** It is very important to minimize the multiple downloads of the same page by the multiple parallel processes of the parallel crawler as this saves network bandwidth and increase the crawler's effectiveness. So, the processes must be coordinated to minimize overlapping of Hidden Web documents.

- 6) ***Politeness policy:*** A crawler should not overload web servers by issuing a large number of requests in a small interval of time. Individual requests arising from the various crawling processes should not be issued to the same server to avoid overloading or burdening it. An objective of the proposed work is to design a crawler obeying this policy of being polite.
- 7) ***Appropriate Precision, Recall and F-measure:*** Precision is defined as the ratio of relevant documents to the number of retrieve documents whereas recall is defined as the ratio of relevant documents that are retrieved to the total number of relevant documents. The metric F-measure trades off precision versus recall and is weighted harmonic mean of precision and recall. Another objective of the proposed work is to obtain better values for the stated three measures.
- 8) ***Reduced Network Bandwidth Consumption:*** To minimize the overlap and maintain the quality of downloaded collection of the web pages, the coordination between individual crawling processes needs communication that consumes network bandwidth. So, an important objective of the proposed work is to minimize communication overhead and thus the network bandwidth consumption while maintaining the quality of crawling.

1.4. CONTRIBUTION

The following contributions have been made in this work to address the above challenges.

- An effective and efficient technique to crawl and extract the content in the Hidden web databases has been proposed in this thesis. More specifically, a parallel crawler for the Hidden Web has been designed and implemented to tackle the problems as mentioned earlier.
- A match logic has been designed to identify the relevant search forms in each domain. This helps in creating the various domain-specific search interface repositories to store the relevant forms in each domain.
- An approach that automatically identifies the domain of web pages for their classification has been developed. Different domain-specific page repositories

are used to store the web pages as per their domain after classification. A framework that helps the crawler to automatically process the search forms has been designed. Domain-specific databases have been created and used for storing the labels and the values needed to fill in the search forms.

- A Query Ranker that ranks the queries in the domain-specific databases to be used by the crawler for filling the search forms has been used to suggest the most optimal query at the time of filling forms.
- Following the domain-specific approach helps the crawler in minimizing the communication overhead and save the network bandwidth consumption while maintaining the quality of crawling.
- A scalable and extensible architecture has been designed in the sense that third party components or modules can be added as per the requirements.
- The proposed parallel Hidden Web Crawler has been implemented using .NET technology and SQL Server. For the conducted experiments, high values of Precision, Recall and F-measure were obtained which indicates that the proposed work efficiently crawls the hidden web pages.

1.5. ORGANIZATION

This thesis is worked out to propose a design of a Parallel Hidden Web Crawler to access the Hidden Web contents and get the most information from it. The thesis is divided into seven chapters. The content of each chapter is summarized as under-

- **Chapter 1** introduces the theoretical aspects of the Hidden Web and the concepts that motivated this research work. The chapter also briefly presents the contributions made by the presented work and organization of thesis.
- **Chapter 2** reviews the related publications related to the WWW and tools for information retrieval from the WWW. The chapter provides an insight into the search engine architecture & behaviour along with the details on the type of crawlers that can be incorporated to provide the search functionality. It also gives a review of the problems of the current search engine landscape.
- **Chapter 3** discusses the various ways to access the contents in the Hidden Web resources. It lists the differences in the basic approach of a conventional crawler for the Surface Web with that of Hidden web crawler. It provides the

state of art in Hidden Web Crawlers through the detailed working of each with a brief comparison among them. The chapter finally presents a comparison among these existing crawlers based on the features provided by each. The Chapter also provides a description of the problem statement that is undertaken in this thesis.

- **Chapter 4** presents the proposed architecture for the parallel Hidden Web Crawler. The working of the proposed crawler has been divided into six phases. This chapter gives the details of the working of the first three phases of the proposed crawler.
- **Chapter 5** discusses the detailed working of the remaining three phases of the proposed crawler architecture. It also explains the scalability and extensibility of the proposed crawler architecture. The chapter presents how the various phases and components of the proposed crawler, collaborate together to provide the desired functionality of the proposed Parallel Hidden Web Crawler.
- **Chapter 6** presents the implementation details of the proposed design of the Parallel Hidden Web crawler. It also covers the results obtained from the proposed design and also explains the results with observations. The presented results justify the inter-relation among the various phases of the proposed crawler and the suitability of the architecture to the specified objectives.
- **Chapter 7** concludes the research work and provides directions for extending the research in this area in future.
- In Appendix A, the various domain-definitions needed for the proposed work are provided.
- In Appendix B, a list of stopwords that are used by the proposed system has been provided.
- Finally, the bibliography includes references to publications in this area.

A survey of existing search engines and crawlers is given in next chapter.

CHAPTER 2.

WWW AND INFORMATION RETRIEVAL TOOLS : A REVIEW

2.1. INTRODUCTION

World Wide Web, also known as the Web or WWW [1], is a huge repository of information resource that is served by numerous anonymous websites. The role of the WWW as the main source of information is becoming more and more significant day by day. It was developed as a collection of human generated information that allows researchers on remote sites to share their thoughts, ideas and all aspects of a common issue project. It was Tim Berners Lee who introduced the idea and created the World Wide Web at the CERN laboratories (one of Europe's largest Research laboratories) in Switzerland in December 1990 [1]. Moreover, the ability to scale its size with respect to content allowed the Web to expand rapidly, across the Internet irrespective of boundaries of nations or disciplines.

The resources on the WWW have been organized and structured in a way so as to allow the user an easy navigation from one resource to another. The navigation over the WWW is done by using an application called the WWW client often known as the *browser*. The task of the browser is to present the formatted text, images, videos, sound, or other objects like hyperlinks etc. in the form of a webpage on the user's computer screen. The user clicks on a hyperlink to navigate different webpages. But due to the hasty growth of the web data or information resources and the changing web technologies it becomes very tedious and difficult to search the information required by the users. This encouraged the researchers to develop web tools that can be used to acquire this information either by searching, querying, extraction, classification or characterization. Therefore, this chapter discusses about the research work that has been done to solve the indicated problem. The description of the related research work done in this area has been classified into following three parts:

- WWW and IR tools: The terms *World Wide Web (WWW) and the Internet* [8] are often used interchangeably without much difference. However, there is a lot of difference between these two terms. Internet is a global data communication medium or a system infrastructure that provides connectivity between computers. In contrast, the WWW is one of the services

communicated and offered via the Internet. WWW allows access to the available information over the medium of the Internet. It, thus, acts as a model that is built on the top of the Internet for sharing of information. A set of Information retrieval tools [11, 12] are now available to the users for finding the information on the Web easily in a very fast manner.

- **Surface Web and Traditional Crawlers:** Surface Web is the part of the web that consists of an immense and interlinked collection of hypertext documents which can be searched and indexed by information retrieval tools. A web crawler is an important component of such information retrieval tool that collects the documents from the web by recursively following the hyperlinks.
- **Classification of Web crawlers:** Due to enormous volume of information and the dynamic nature of World Wide Web, it is not possible to cover the entire web using a single instance of the crawler. Also, it is not possible to provide specialized type of information to the information retrieval tool for indexing. Therefore, various types of crawlers like focused crawlers, scalable crawler, parallel crawlers etc. have been designed in the research.

2.2. INFORMATION RETRIEVAL

WWW is based on a client-server system in which millions of servers exist with world-wide distribution. An organization of the WWW showing the client interaction with the Web servers via a special browser application has been shown in Figure 2.1.

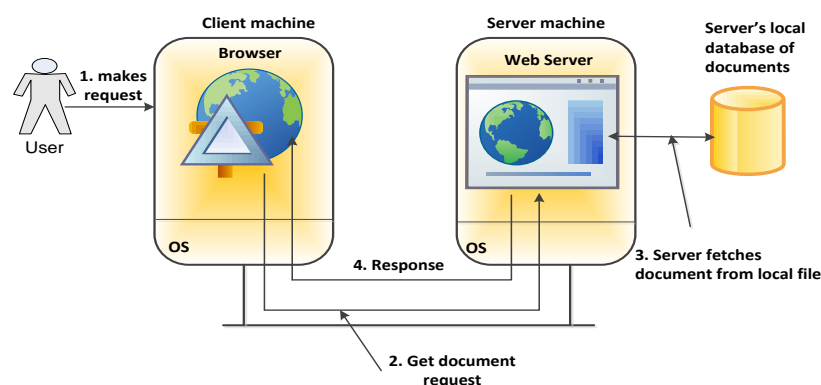


Figure 2.1: The overall organization of the World Wide Web

Each server is responsible for maintaining a collection of documents where each document is stored as a file. The browser typically accepts input from a user which is a reference to a document at the server's site. The server accepts the requests for

fetching the document and transfers it to the client where the browser holds the responsibility for displaying the document. The server is also responsible for processing the requests for storing new documents. Some of the most common terms used in context of the WWW include the following:

1. Uniform Resource Locators (URLs) are the strings used as addresses of objects like documents, images on the Web [1, 38]. It is the simplest way to refer to any document on the Web. Each URL includes the DNS of its associated server specifying the location of the document and a filename by which the server can look up the document in its local file system. For example, The URL of the main page for the WWW project happens to be

<http://info.cern.ch/hypertext/WWW/TheProject.html>

in which the string “*info.cern.ch*” refers to the web server at the CERN laboratories that hosts the requested HTML document “*TheProject*” which contains information about the origins of the WWW. The location of this html document on the web server is specified by the path mentioned in the substring “*hypertext/WWW/*”.

2. A network Protocol (HTTP) [38] used by native Web servers giving performance and features not otherwise available. The communication on the Web is based on a client-server architecture where the client makes a request to the server and waits for a response from the respective server. This communication is typically based on a network protocol typically the Hypertext Transfer Protocol (HTTP). Moreover, HTTP is based on TCP, thus need not be concerned about lost requests and responses assuming that their messages make it to the other side [8, 38].
3. A document must be properly structured to express the informational content. This is usually done with the help of a mark-up language like HTML [8, 12] that provides keywords/ tags to structure a document into different sections. It also provides features to distinguish headers, tables, forms and lists. All the HTML documents include a heading section and a main body. Other objects like images or animations can also be inserted at specific positions in a document. Even more, the HTML also provides various keywords that specify the instructions to the browser on presentation of a document.

For example, there are keywords to select a specific font or font size, to present text in italics or boldface, to align parts of text, and so on. For example, consider a simple HTML document shown in Figure. 2.2.

```
<HTML>
<BODY>
<H1> DESIGN A HTML DOCUMENT </H1>
</BODY>
</HTML>
```

Figure 2.2: HTML Document

When this Web page is requested for display in the browser, the user finds the text “DESIGN A HTML DOCUMENT” when interpreted. When any HTML document is internally parsed for a *Document Object Model* or *DOM* [9], it is represented by a rooted tree, typically called as the parse tree, where each node of tree represents an element of that document. The element is one of the types from a predefined collection of the various types of elements. Similarly, each node is required to implement a standard interface containing methods for accessing its content, returning references to parent and child nodes, and so on. Every www client is required to understand this markup language so that the transmission of information across the internet takes place.

2.2.1. ANALYSIS OF WEB CONTENT

The textual content is the most straightforward feature of a web page that is also considered for retrieval of information as it is directly available in the page. But an obvious feature that appears in HTML documents and not in plain text documents is HTML tags. It has been demonstrated that using information derived from these tags can significantly boost the search efficiency. [75] derived significance indicators for textual content in different tags. In their work, four elements from the web page are used: *title*, *headings*, *metadata*, and *main text*. A linear combination of these four elements is used to classify the web pages according to their domains. The work proposed in [105] and [106] discusses a similar approach for classifying the web pages using a modified k-Nearest Neighbour algorithm by assigning weights to terms present in different tags. But they have divided all the HTML tags into three groups and assigned each group an arbitrary weight so that the terms within different tags are given different weights. Thus, utilizing tags can

take advantage of the structural information embedded in the HTML files, which is usually ignored by plain text approaches.

The result of a survey that was conducted on a sample of 19195 web sites in [107] to estimate the number of words that are often used in the content attribute of the <TITLE> tags , the <META name=“keywords”> and the <META name=“description”> tags and in the free text found within the <BODY> tag, excluding all other HTML tags. For each tag type, the percentage of web sites with the number of words within a certain range is shown in Table 2.1.

Table 2.1: Percentage of web pages with words in HTML Tags

Tag Type	Title	Meta-Description	Meta-Keywords	Body Text
0 words	4%	68%	66%	17%
1-10 words	89%	8%	5%	5%
11-50 words	6%	21%	19%	21%
More than 51 words	1%	3%	10%	57%

The body of a web page acts as the major and most important source of text. As shown in last column of 1st row, nearly 17% of chosen web pages contain only images, frame sets or plug-ins with no usable body text. A maximum fraction of web pages contained more than 50 words with just a quarter of web pages containing 11-50 words. The amount of textual content that exist in the title tags is relatively small (only 1-10 words) for almost 89% of the web sites despite of their so common use, as can be depicted from the entry 3rd row of the second column . Moreover, the title tags typically contain the names or terms like “home page”, which are not of much help in classifying the subject of the web page.

Although only about one-third of the web sites were found to contain the Meta-tags for keywords and descriptions, still these tags play a major role in ranking and display of search results generated by the several major search engines. It was also observed that the Meta tags often include the content that is specifically intended to aid in the identification of the subject area of the web page so prove to be of great help in

classification. For most of the web sites these meta-tags contained between 11 and 50 words, with a smaller percentage containing more than 50 words (in contrast to the number of words in the body text which tended to contain more than 50 words).

2.2.2. INFORMATION RETRIEVAL TOOLS

To organize and locate the information present on the WWW, a collection of information search and retrieval tools are now available on the Web, the most popular of which are the *web directories*, *search engines* and *meta search engines*. Each of these systems is explained as follows:

2.2.2.1. Web Directories

Typically, the directories on the Web are topic-oriented catalogs that contain a hierarchical representation of Internet web sites in which the web pages have been classified by topics or subjects. This hierarchy can be broken down into topics and subtopics upto any number of levels depending on the categorization of the topic. However, classification of the web pages and the topic levels that are to be added to the web directory typically involves human intervention. Moreover, these topics vary according to the scope of each Web directory and are never standardized. The Web directories are often carefully evaluated and annotated by human experts in their respective areas or topics.

By using a Web directory, the user can select a suitable category for a topic of his interest and can move down through the hierarchy, selecting the subcategory and thus narrowing the search at each level until a list of hyperlinks relevant to his topic are offered. While traversing the directory structure downwards, the user moves towards more specific topics whereas while traversing upwards the user moves towards more general topics. But these directories being manually edited may not have been correctly classified and thus, the user can only search using the broad or general terms and on the basis of what that is visible (topics, titles, subject categories, descriptions, etc.). Some examples of Web directories are ‘Google Directory’, ‘Yahoo! Directory’ and ‘Open Directory Project (ODP)’.

2.2.2.2. Search Engines

A search engine [6, 10] is an information retrieval tool that minimizes the user’s time required to find the relevant information over the vast Web of hyperlinked documents. It is a tool that:

- 1) Enables its users to submit a query consisting of a string or phrase that describes his/her criteria about the information of interest.
- 2) Searches its locally maintained databases for matches against the query.
- 3) Returns a set of web pages that matches with the query.
- 4) Allows the user to refine and re-formulate the query as per the requirement.

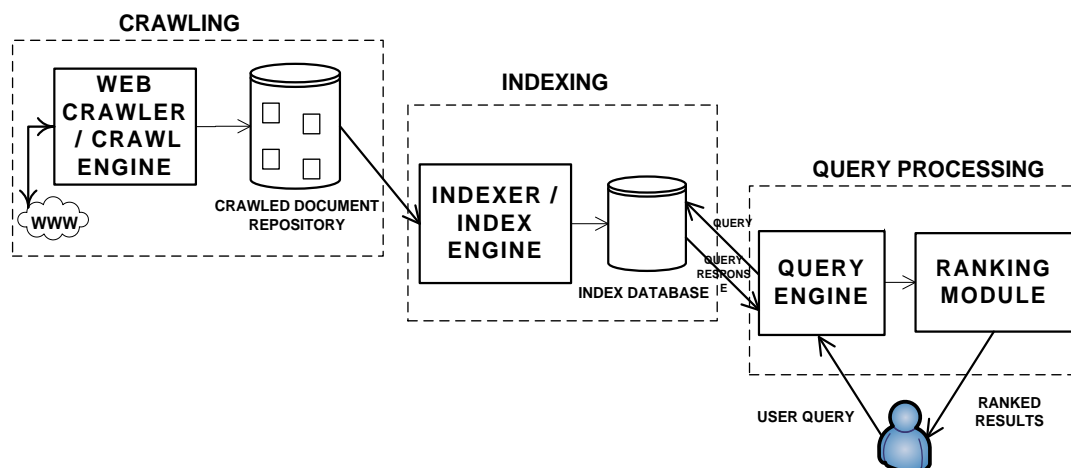


Figure 2.3: Elements of a Basic Search Engine

All the search engines share a common set of activities and components [11, 12]. The various components of a basic search engine are illustrated in Figure 2.3. Based on these components, the various functionalities performed by a search engine can be divided in the following manner:

- 1. Crawling:** This is the process by which a search engine gathers pages from the WWW, in order to index them and support the search engine. The component responsible for the process is known as a Crawl Engine or more typically as a Web crawler [11, 12]. The crawler when given a set of seed Uniform Resource Locators (URLs) as input downloads all the web pages addressed by the URLs, extracts the hyperlinks contained in the just downloaded pages and recursively downloads the web pages corresponding to the extracted hyperlinks. The objective of crawling is to quickly and efficiently gather as many useful web pages as possible together with the link structure that interconnects them.
- 2. Indexing:** Indexing refers to the task of creating a data structure that allows easy and quick searching of content [12]; or the act of assigning index terms to documents for their efficient retrieval where an index term

is a word which can semantically represent the main subject or theme of the document [12]. The storage space is commonly termed as the index or the database and the element responsible for building the search engine's index is termed as the Index Engine or typically as an Indexer.

3. **Query Processing:** This is another important task to be carried out by a search engine. It includes receiving a query from the user, describing the web documents that suit the interest to the user, searching the index data structure for relevant document entries, and presenting the results to the user. These results are thereafter ranked in the order of their relevance to the query. The component responsible for processing the user queries is often termed as a Query Engine.

A search engine can either be a general- purpose search engine or a specialized search engine. The general- purpose search engine are basically used for general queries and index all kinds of web pages. These do not focus on any specific kind of web content thus not limiting itself to particular domain. For example: Google, Yahoo!, AltaVista etc are some of the popular search engines for generic search. These kinds of the search engines are the most popular among the users that answer millions of queries per day.

The specialized search engines focus on a specific topic or domain, like the Search Engines attributed to medical research databases (MedHunt.Com, Pubmed.inc); search engines offering travel catalogues to help users plan their travel via the web (TripAdvisor.com, Expedia.com) etc. The search technology used by these IR tools target to deliver only the results that seem most relevant and popular by filtering out any irrelevant search results. This helps to obtain a fewer number of hits as the content that does not belong to the topic has been eliminated from the search results. This increases the chances of fetching more precise and relevant results with a big savings in user's time needed for searching.

Moreover, due to the limited domain or topic for access, the crawling process can be more frequent facilitating more current and updated index.

2.2.2.3. Meta-Search Engines

A meta-search engine is generally used to send the user query to multiple data sources or search engines in order to present the combined results in a formatted manner. The data sources are generally the indexes of web search engines. More clearly, a meta-search engine forwards the keywords of the user query to several individual search engines simultaneously for obtaining the results from all the search engines. The results obtained thereof are combined together in a common representation by the meta-search engine for presenting them to the user. Here, it may be observed that the meta-search engines do not possess their own index of Web pages. They completely rely on the indexes of other search engines. The examples of various meta-search engines are: Profusion [10], MetaCrawler [15] etc.

A good meta-search engine is not only capable of accepting complex queries, integrating the results in a well planned way but is also capable to eliminate any duplicate result record hits, and rank the results as per their relevance in an intelligent manner. The working of a typical meta-search engine is shown in the Figure 2.4.

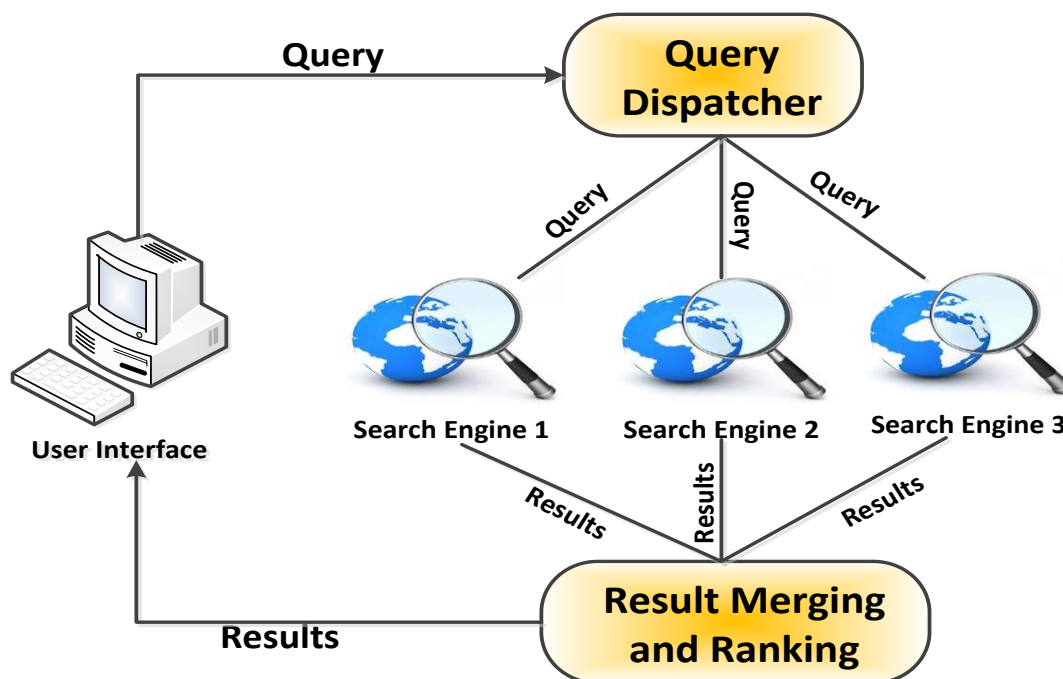


Figure 2.4: Basic Architecture of a Meta-Search Engine.

The main disadvantage of these meta-search engines is that, they are mainly dependent on the general purpose search engines and the accuracy of retrieval depends on the efficiency of the general purpose engines. Also, these search systems must keep up with the changes like search algorithms, indexing techniques etc. in the dependent search engines.

2.3. WEB CRAWLERS

The crawler downloads the web pages to be used by a search engine. These web pages are later processed by the search engine in order to create and maintain an index. The crawler traverses the web by following the hyperlinks embedded in the web page and recursively downloading the documents [7]. Technically, crawler is defined as “Software programs that traverse the World Wide Web information space by following the hypertext links extracted from hypertext documents” [32]. Web crawlers are also known as *spiders*, or *wanderers*, or *robots*, or *worms* etc. These names may be misleading as the terms “Spider” and “Wanderer” gives false impression that the crawler itself moves, and the term “Worm” implies that the crawler multiplies itself, like the infamous Internet worm [5, 6, 22]. Figure 2.5 shows the process of a conventional crawler to download web pages.

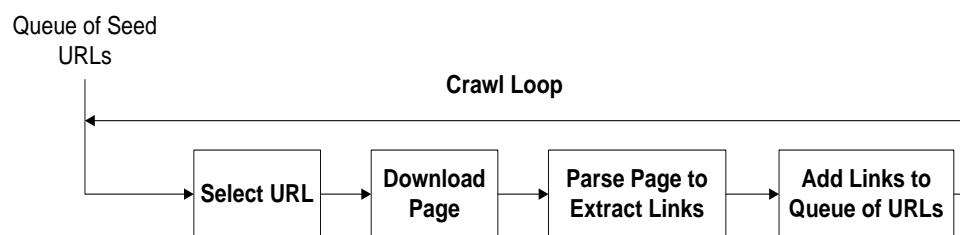


Figure 2.5: Control process of a crawler for the Surface Web

The following are some of the basic terms used in the context of web crawlers:

1. **Seed URLs:** A crawler traverses the Web by following hyperlinks recursively within its downloaded web pages. It starts by choosing a URL from a given set of initial URLs. This starting URL set is the entry point through which any crawler starts searching procedure. These initial URLs in the set are known as the “Seed URLs” for crawling. The selection of a good seed set is the most important factor in any crawling process.
2. **Frontier:** The crawling method starts with a given URL (from the seed URLs), downloading the associated web page, extracting links from it and adding them to an un-visited list of URLs. This list of un-visited links or URLs is known as, *frontier*. This frontier is implemented by using Queue, Priority Queue Data structures. The maintenance of the Frontier is also a major task of any Crawler.

3. **Parser:** Once a page has been fetched, its content is then parsed to extract information that is used to guide the future path of the crawler. The job of the parser is to parse the fetched web page to extract list of new URLs to be added to the Frontier.

Thus, crawlers are mainly used by web search engines to gather the data for indexing. Since, a crawler identifies a document from its URL, it picks up a seed URL and downloads corresponding Robot.txt file, which contains downloading permissions and the information about the files that should be excluded by the crawler. On the basis of the host protocol, it downloads the document and stores the related pages in its database. It then repeats the whole process as per the algorithm in Figure 2.6 :

Step1: Read a URL from the seed set of URLs
Step2: determine the IP address of the host name.
Step3: download the Robot.txt file that carries the downloading permissions for different files.
Step4: determine the protocol of underlying host like http, ftp etc. for downloading the file.
Step5: identify the document format like doc, html, or pdf etc.
Step6: check whether the document has already been downloaded or not
Step7: if the document is fresh one
 read it and extract the links to the other cites from that document;
 else
 continue;
Step8: convert the URL links into their absolute URL equivalents.
Step9: add the URLs to set of seed URLs

Figure 2.6: Algorithm of a crawler.

The Robot.txt is an important file housed on every server. A detailed discussion on this file is given in the next section.

2.3.1. ROBOT.TXT: STANDARD FOR ROBOT EXCLUSION

The crawlers or robots traverse many pages in the WWW by recursively retrieving linked pages [25]. In 1993 and 1994, there were occasions when crawlers visited WWW servers where they were not welcomed for various reasons such as given below:

- Certain robots (crawler) swamped servers with rapid fire requests

- Some robots retrieved the same file repeatedly
- Robots traversed parts of WWW servers, which were not suitable such as very deep virtual trees, duplicate information, temporary information, access to CGI scripts etc.

The above mentioned points necessitated the need for established mechanisms for WWW servers to indicate the crawlers as to which parts of their servers should not be accessed. Therefore, a concept to have a file named as `"/robots.txt"` came into existence that will specify the access policy for these robots.

The format and semantics of the `"/robots.txt"` file is as follows:

- The file consists of one or more records separated by one or more blank lines terminated by CR, CR/NL, or NL
- Each record contains lines of the form:
 - `"<field>: <optional space> <value> <optional space>"`
 - The field name is case sensitive.
- Comments can be included in file using UNIX borne shell conventions. The `"#"` character is used to indicate that preceding space (if any) and the remaining part of the line up to the line termination is discarded.
- The record starts with one or more *user-agent* lines, followed by one or more *Disallow* lines as detailed below.

2.3.1.1. User-Agent

- The name of the field is the name of the robot for which the record is describing access policy.
- If more than one user agent field is present, the record describes an identical access policy for more than one robot. At least one field needs to be present per record.
- The robot should be liberal in interpreting that field. A case sensitive substring match of the name without version information is recommended.
- If the value is `"*"` the record describes the default access policy for any robot that has not matched any of the records. It is not allowed to have multiple such records in the `"/robots.txt"` file.

2.3.1.2. Disallow

- The value of this field specifies a partial URL that is not to be visited. This can be a full path, or a partial path; Any URL that starts with this value will not be retrieved.

For Example: Disallow: */help* disallows both */help.html* and */help/index.html*

- Whereas Disallow */help/* would disallow */help/index.html* but allow *help.html*.
- Any empty value indicates that all URLs can be retrieved.
- At least one disallow field needs to be present.

Example: The following server:

```
# Go away
User-agent: *
Disallow :/
```

Indicates that no crawler should visit this site further.

Example: The following server:

```
# robots.txt for http://www.exampleworld.com
User-agent: *
Disallow: /internetcity/map/
Disallow: /temp/
Disallow: /cyber.html
```

indicates that no crawler or robot should visit any URL starting with *"/internetcity/map"* or *"/temp/"* or *"/cyber.html"*.

Thus, "/robots.txt" specifies which parts of the server URL space should be avoided by the robots. This facility can be used to warn crawlers or robots for black holes. This standard is voluntary but is very simple to implement.

2.4. CHALLENGES FOR A WEB CRAWLER

Though the Web crawler algorithm seems to be a simple recursive traversal of the hyperlinks and the associated web pages, but is complicated by the demand of high quality of the retrieved document collection. Following are some of the issues and challenges generic to web crawlers:

1. **Robustness:** A web server might create malicious *spider traps* (during website development) by hosting and linking to web pages that mislead the crawlers to get stuck by fetching an infinite number of pages repeatedly. The crawlers designed must be flexible enough to handle such traps.
2. **Politeness:** Web servers have both implicit and explicit policies regulating the rate at which a crawler can visit them. A crawler should not overload web servers by issuing a large number of requests in a small interval of time. This can be done by avoiding issuing the requests to the same server. A crawler must be designed with the view to respect this policy of being polite.
3. **Scalable:** The crawl of a portion of the web must be completed within a limited time. A search engine that requires daily updates to its index cannot use a crawler that takes weeks or months to harvest the data from the web. The download rate of the crawler must suit the requirements of the application that will process the downloaded collection. The crawler architecture should provide support to scale up the crawl rate either by adding extra machines or bandwidth in the existing one.
4. **Distributed:** The crawler should have the ability to execute in a distributed fashion across multiple machines.
5. **Extensible:** Crawlers should be designed in such a way that it may cope with new data formats, new fetch protocols, and so on. The information downloaded by a crawler would have little use if it could not be processed by other applications. Thus, a crawler should be designed to operate as a component of broader systems. This demands that the crawler architecture be modular so that third party modules can be added as per the requirements.

- 6. Performance and efficiency:** The crawler should utilize the available resources such as processor, storage and network bandwidth in the best manner possible.

2.4.1. INFORMATION RETRIEVAL TERMS AND PERFORMANCE METRICS

As the WWW contains huge amount of data distributed over the different geographical locations and still growing at an exponential rate, information retrieval tools are needed that allows the users to find the information on the web quickly and in a rapid manner. Some basic definitions that introduce the area of information retrieval and thus help in evaluating the performance of any information retrieval tool are given below:

- 1) The most basic unit of Information Retrieval is the *term* [4] which can be defined as a word in any known language like English, Japanese, French etc.
- 2) A *query* [12] is a phrase consisting of one or more than one term that specify the ad hoc information need of the user.
- 3) A *document* [32, 38] is generally defined as a large set of terms that are arranged usually in a meaningful manner, to form sentences. A set of documents is often termed as the *document collection or corpus*.
- 4) The terms are often weighed to measure their expressive or descriptive power. The measure *term frequency (TF)* [2, 4, 12] specifies the number of times a particular term appears in a document.
- 5) The metric *Inverse Document Frequency (IDF)*, is used to find how common or how frequently a word exists in a collection [4, 12]. The words that have a high frequency of occurrence are called Common words and thus have a low value of IDF whereas the terms that occur rarely have a high value of IDF. IDF is calculated by using the formula:

$$IDF = \log \frac{\text{collection size}}{\text{number of documents containing the term}} \quad 2.1$$

If a document has been represented as a one-dimensional array or vector that contains the term frequency of every word in the document then for each term

in the document vector, its TF is multiplied by *IDF* to get the score *TF-IDF* [2, 4, 12] which is the most popular method of weighing terms *i.e.*

$$TF - IDF = TF \times IDF \quad 2.2$$

- 6) *Relevance* [6, 17] is a measure of how well a document satisfies the need of its user. The IR tool or system takes a query from the user and returns a set of documents that are *relevant* to that query.
- 7) The following are the standard measures [12, 19] that are used to evaluate the performance of any IR tool in returning relevant documents:

➤ *Precision*: It is the ratio of number of relevant documents retrieved by the IR tool divided by the total number of documents retrieved:

$$Precision = \frac{\text{Number of relevant matches retrieved}}{\text{Number of extracted matches retrieved}} \quad 2.3$$

➤ *Recall*: It is the ratio of the number of relevant documents retrieved divided by the total number of relevant documents that exists in the collection:

$$Recall = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents in the collection}} \quad 2.4$$

As different users have different information needs, a document which might be relevant to one user may not be of much use to other. Therefore, the definition of relevance is difficult to quantify and the above metrics used for calculating are purely subjective. The information retrieval tool should not only target to achieve 100% precision but also maximize recall as the former is only possible if the system retrieves just a single document [18, 22]. Giving more weight to common documents is one method of improving precision [37]. Similarly, a 100% value of recall can only be obtained if the information retrieval tool returns all the documents. Thus, the system should also try to maximize *precision* as well [22, 24]. Recall can be improved by selecting those document collections that help in retrieving different relevant documents [37].

2.5. CLASSIFICATION OF WEB CRAWLERS

A crawler typically follows the links in the order in which they are encountered i.e. in a FIFO manner where the process begins by a particular web page and then explores all its linked web pages that are reachable by following a single hyperlink from the home page. After exploring all the web pages at the first level, it starts exploring the other web pages. Thus, a breadth-first crawl targets to maximize the coverage of the web content in the search engines index. The strategy works well in a situation if a small number of web pages exist over the Web and have to be traversed. However, the WWW has millions of pages linked to one another. The enormous size of WWW makes it almost impossible to crawl the contents of entire Web as the index of a search engine cannot accommodate all the pages. The ultimate goal of search engines is not only to cover WWW as much as possible in the minimum time period but also to download the quality web pages. Thus, practically, a crawler does not need to look into every corner of the Web if it is to acquire information on a particular topic of interest. Therefore, many types of Web Crawlers came into the picture. Few of the important web crawlers are discussed as follows:

2.5.1. FOCUSED CRAWLERS

WWW is a collection of hypertext documents from all possible domains. It is very difficult for any general purpose crawler to download such a large collection of documents from diversified areas. Focused crawlers restrict the crawling process on a specific set of topics that represent a narrow domain of the Web. A focused or a topical web crawler attempts to efficiently download web pages relevant to a set of pre-defined topics and guide the search based on content and link structure of the Web. It does not collect all the related and reachable web pages as the general purpose crawlers do, yet have higher efficiency as the URLs that point to low quality or irrelevant pages have been avoided. Crawlers taking this approach have more relevant documents, compared to their “store everything” counterpart.

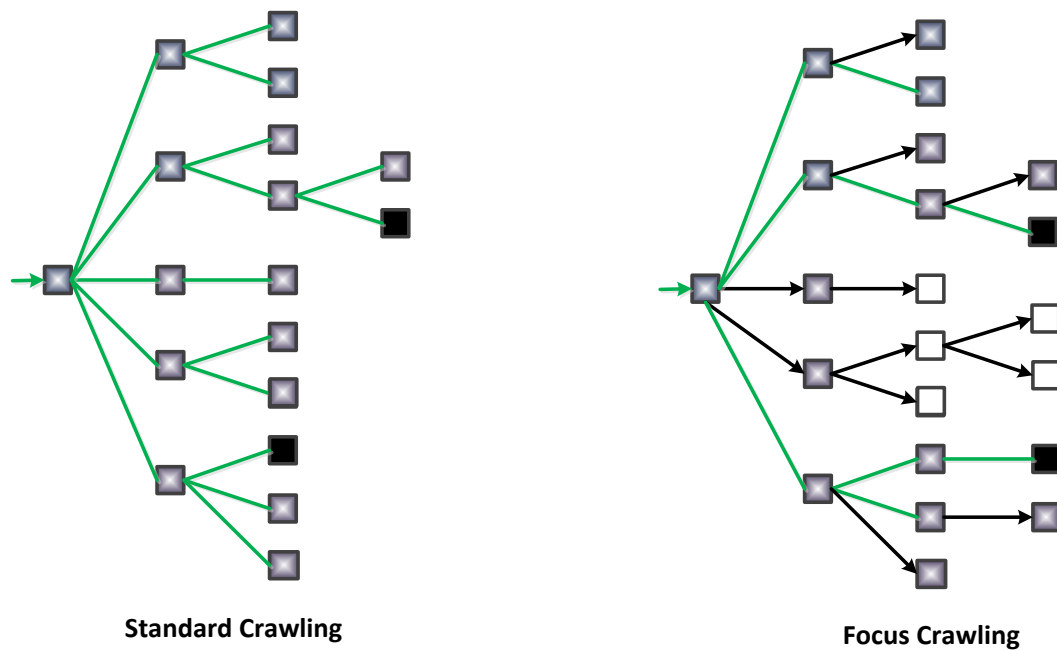


Figure 2.7: a) A standard crawler following each link. b) A focused crawler trying to identify the most promising links

Figure 2.7 graphically illustrates the difference between an exhaustive crawler and a typical focused crawler. A standard crawler follows each link, typically applying a breadth first strategy. If the crawler starts from a document which is i steps from a target document, all the documents that are up to $i-1$ steps from the starting document must be downloaded before the crawler hits the target. Whereas, a focused crawler tries to identify the most promising links, and ignores off-topic documents. If the crawler starts from a document which is i steps from a target document, it downloads a small subset of all the documents that are up to $i-1$ steps from the starting document. If the search strategy is optimal the crawler takes only i steps to discover the target.

The focused strategy is based on an assumption that a page under a certain topic (or region) is likely to be linked from another page with the same topic, thus link context forms an important part of web based information retrieval task. Before fetching the web document, focused crawlers try to predict whether the target URL is pointing to a relevant and high quality document or not.

Focused crawling method collects web pages following these steps. A set of keywords is defined prior to the crawling. Start pages are chosen, and the crawling starts. If a scanned page contains a word from the keyword set, then this page is saved in the database, and links from this page are added to the queue. Crawling goes on until the queue is empty or no page has been collected for over certain length of time.

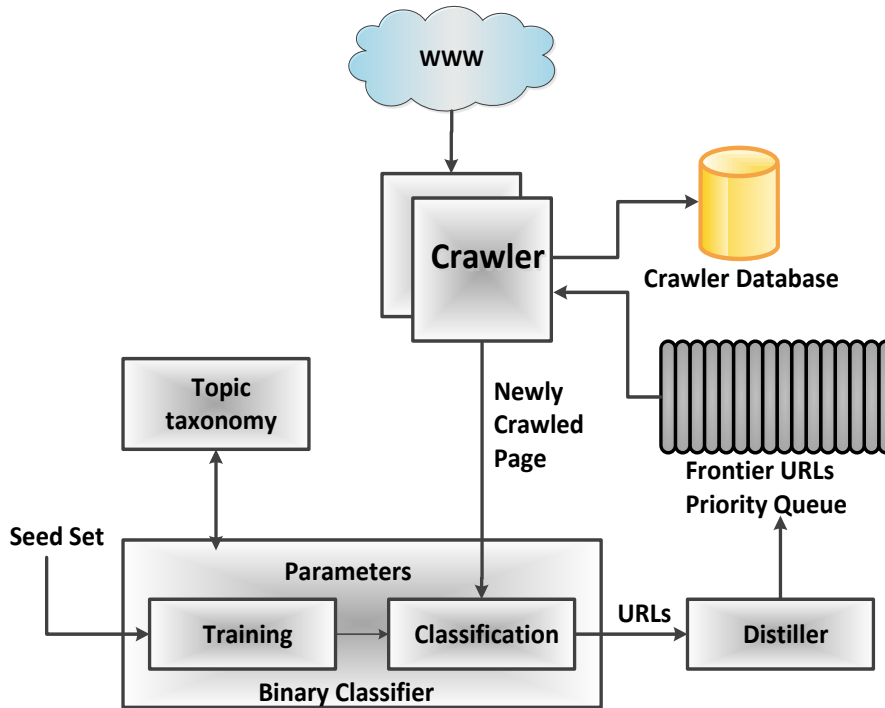


Figure 2.8: Focused Crawler Architecture

According to the Chakrabati [18], as shown in Figure 2.8, the key components of a focused crawler are a Document Classifier and a Document Distiller. The task performed by classifier is to evaluate the documents (that are linked to the current document thus henceforth called children) with respect to the focused topic and that of a distiller is to identify for the best URLs for the crawl frontier. The classifier is either provided by the user in the form of query terms, or can be built from a set of seed documents. Each link is assigned the score of the document to which it leads. More advanced crawlers adapt the classifier based on the retrieved documents, and also model the within-page context of each hyperlink. However, ensuring flexibility in the classifier without simultaneously corrupting it is difficult [22]. Precision and recall are two basic parameters to measure the performance of focused crawlers.

Designing a focused crawler poses a number of challenges like:

- **High Quality Training Data:** The best URLs are those that can serve as great access points to many relevant pages within a few links. Thus, maintaining the quality of the training data is of utmost importance to keep the crawl on focus which acts as a main bottleneck for the effective performance of the focused crawler.

- **Restricted Search:** Since the links on the Web are unidirectional, the search is restricted to ‘forward crawl’ or ‘top-down traversal’ for the focused crawlers. Moreover, the websites frequently have large components that are organized as trees, entering a website at a leaf can result in a serious barrier to finding closely related pages.

Once the challenges and issues related with their design mechanism are tackled, it offers several advantages that are mentioned as follows.

- **More Relevant Pages:** By limiting web pages to those containing predefined keywords, the resulting set of web pages is expected to have higher concentration of relevant pages, despite ambiguities in place names.
- **Fresh Content:** Focused crawlers can cover the specialized topics in depth and keep the crawl database fresher as they are limited to a specific topic or region of the WWW.
- **Efficiency in Crawling:** Because crawler does not collect all pages but those containing predefined keywords, it has higher efficiency in collecting desired pages, compared to “store everything” crawling system. This saves both hardware and network resources.
- **Avoidance of Ambiguity:** By tracing down links only when there was a keyword found in the page, only the pages that are linked by another relevant page are collected. This reduces ambiguity, since the page under consideration is guaranteed to be linked from the page related to the topic because of the semantic relationships represented in the links.

2.5.2. PARALLEL CRAWLERS

The amount of information presented over the WWW and the number of pages providing this information have reached to such a level that it is difficult, if not impossible, to crawl the entire web by a single processor. Thus, current search engines use multiple parallel processors to maximize the download rate of the crawler. When multiple processes are used to simultaneously download the web pages, the type of the crawler is referred to as a parallel crawler. Nevertheless, in order to design an efficient parallel crawler, major techniques applied in parallel crawlers and the challenges to be faced must be analyzed. This section thus presents the various

considerations to be taken in mind when designing the internal structure of parallel crawlers.

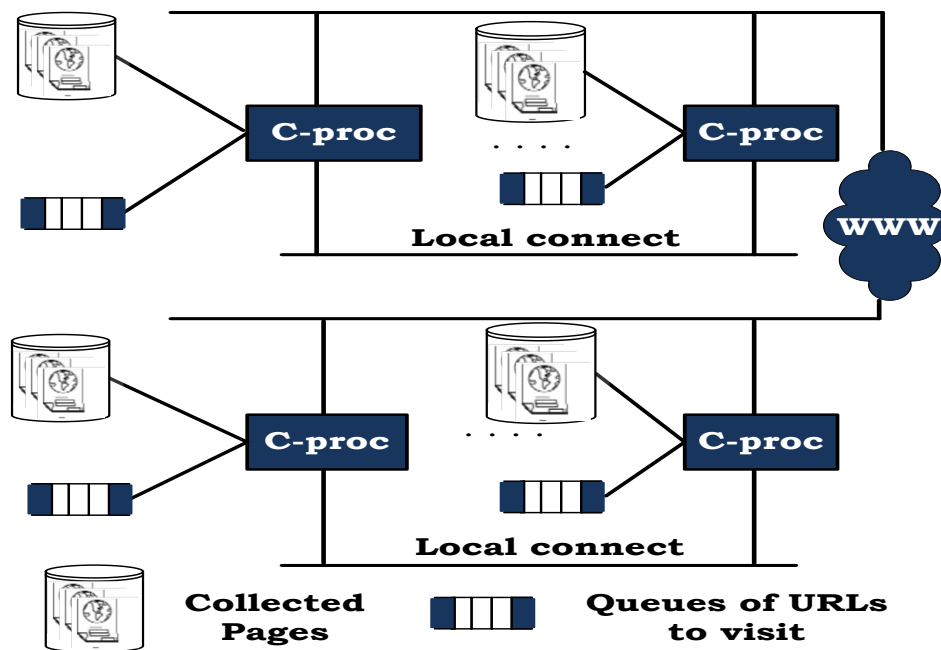


Figure 2.9: Generic Architecture of a Parallel Crawler.

The general architecture of a parallel crawler is as shown in the Figure 2.9. In a parallel crawler, multiple crawling processes (typically termed as the C-procs) working in parallel are used to download the web pages and perform all the tasks that are basically performed by any crawler. Each C-Proc maintains a local database of the web pages collected by it. The C-Proc also owns a queue of URLs that are yet to be visited by it. Once the C-Proc finishes its task, the pages collected by each C-Proc are added to a central or common repository maintained by the search engine. If the target download rate has not been achieved by the crawler and needs to be increased further, number of C-procs can be added to the system.

An augmentation to hypertext documents by including a Table of Links (TOL) for parallel crawling was developed by A.K.Sharma et al [35]. According to this method if the links contained within a document become available to the crawler before an instance of crawler starts downloading the documents itself, and then downloading of its linked documents can be carried out in parallel by other instances of the crawler. But in the traditional crawling scenario, the links become available to the crawler only after a document has been read by the downloader. Therefore, providing meta-information in the form Table of Links (TOL) consisting of the links contained in a

document was proposed. This TOL is stored external to the document such that it can be retrieved separately by the crawler. The storage may be in the form of a file with the same name as the document but with different extension (say as .TOL). Extraction of TOL is a one-time process that can be done at the time of creation of the document. In [36], the architecture of a scalable parallel crawler based on their extraction of TOL has been proposed. Their crawler partitions the task into two stages: At the first stage, they have divided the document retrieval system into two parts: the crawling system and the hypertext (augmented) documents system. The augmented hypertext documents provide a separate TOL for each document to be downloaded by the crawling process. Once the TOL of a document becomes available to the crawler, the linked documents, housed on external sites, can be downloaded in parallel by the other instances of the crawler. Moreover, the overlapped downloading of the main documents along with its linked documents on the same site also becomes possible. In the second stage, the crawling system has been divided into two parts: Mapping Process and Crawling Process. The Mapping process resolves IP addresses for a URL whereas the Crawling Process downloads and processes documents.

The main advantage of the Parallel crawling architecture is that it increases the efficiency of any search engine. But using several crawling processes that work in parallel raises certain problems or issues that need to be dealt with. These issues are described in the following section.

2.5.2.1. Challenges faced with Parallel Crawling Architectures

- **Risk of Overlapping Web pages:** The problem arises when multiple crawling processes running in parallel download the same web page multiple times due to the reason that a process may not be aware that another process has already downloaded the page. Also, to avoid overloading of a single Web server, many organizations put duplicate copies of their documents on many different servers. This increases the chances that many copies of the same web page are downloaded by the crawler. This necessitates that such multiple downloads be avoided or reduced to minimum in order to preserve network bandwidth for increasing the effectiveness of the crawler.
- **Quality of downloaded web pages:** To maximize and maintain the quality of the web page collection downloaded by the crawler, it must download “relevant” pages earlier during the process. However, in a parallel crawler

framework, to download such relevant or important pages first, multiple crawling processes running in parallel must be familiar with the portion of the Web that is stored in the common or centralized repository so that redundancy among the duplicate web pages may be eliminated.

- **Network or Communication bandwidth:** The crawling processes need to communicate among themselves for coordinating with each other. This coordination helps to reduce the overlap and thus improve the quality of web page collection. However, if the number of crawling processes working in parallel is too large, the traffic generated due to communication increases significantly and results into a huge consumption of network bandwidth.

Thus, implementation of a framework based on parallel crawling in a search engine is a big challenge.

2.5.2.2. Advantages of Parallel Crawler Architecture

Though the parallel crawling framework is challenging to implement but yields many important advantages [23] in comparison to the single process or sequential crawler. These advantages are mentioned below:

- **Scalability:** Due to the huge size of the Web, downloading the web pages in parallel by the crawling processes prove highly useful in achieving the target download rate and thus providing efficient coverage.
- **Network-load Dispersion:** Multiple crawling processes of a parallel crawler may run at geographically distant locations, each downloading “geographically-adjacent” pages. For example, a process in Sweden may download all European pages, while another one in India crawls all Asian pages. In this way, one can easily disperse the network load to multiple regions. In particular, this dispersion might be necessary if a single network cannot handle the heavy load from a large-scale crawl.
- **Reduction in Network-load:** In addition to the dispersing load, a parallel crawler may actually reduce the network load also. For example, assume that a crawler in India retrieves a page from Europe. To be downloaded by the crawler, the page first has to go through the network in Europe, then the Europe-to-Asia inter-continental network and finally the network in India. Instead, if a crawling process in Europe collects all European pages, and if

another process in Asia crawls all pages from Asia, the overall network load will be reduced, because pages go through only “local” networks. Thus, it can be concluded that Parallel Crawler Architecture is a better option as compared to single crawler architecture. Also the number of web pages around the globe is huge. Thus, in order to cover the whole web, only parallel crawlers can do this job in short span of time by keeping in consideration the issues listed above.

2.5.3. HIGH-PERFORMANCE CRAWLERS

Many search engines have implemented their own versions of high-performance crawlers to index the Web. They consist of a number of crawler threads, which run on distributed sites and interact in a peer-to-peer fashion. Each such thread entity has the knowledge of its URL subset, as well as mapping from URL subset to network address of corresponding peer crawler thread. Whenever any crawler thread encounters a URL from a different URL subset, it is forwarded to the appropriate peer thread based on URL look up table. Each crawler thread maintains its own database, which only stores the documents from the URL subset assigned to that particular thread. The databases are disjoint and can be combined offline when the crawling task is complete.

Ubicrawler [20] and Mercator [24] are the few popular examples of high-performance web crawlers. Scalability and fault tolerance are the primary features apart from other features such as balanced load distribution, politeness, fully distributed, high performance, and portability for any web crawler. A detailed discussion on the design of these crawlers is as follows:

- (a) Mercator[24]: It is a scalable and extensible crawler, that is used by the AltaVista search engine. Multiple crawler threads are used to accomplish the task of crawling where each thread repeatedly performs the steps needed to download and process a document. Figure 2.10 shows the major components of the Mercator crawler. In the first step, the crawler thread takes an absolute URL from the URL frontier for downloading.

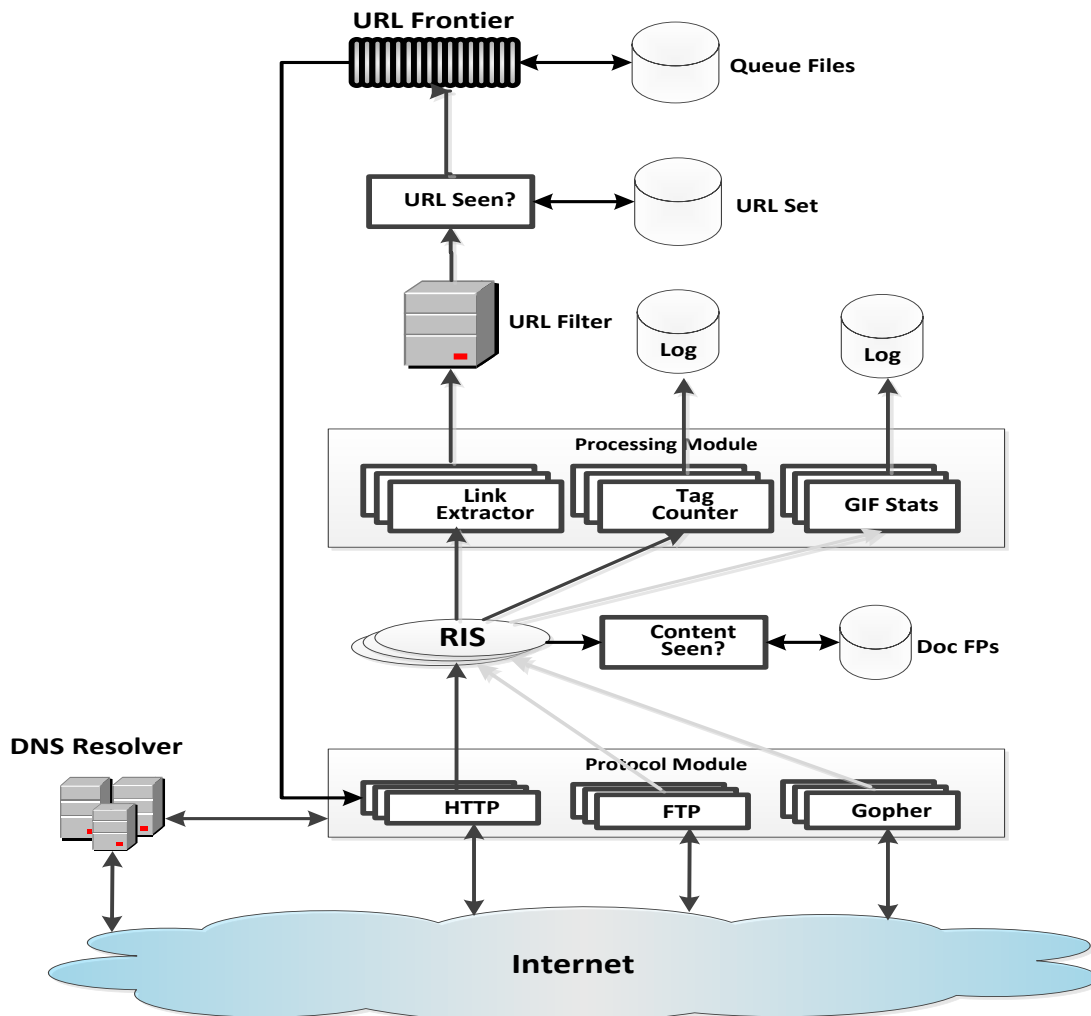


Figure 2.10: The architecture of Mercator Crawler

In Mercator, the network protocols to be used during the crawl are specified in terms of protocol modules. These specifications are listed in the configuration file that is supplied by the user at the start of the crawl. The default configuration includes protocol modules for HTTP, FTP, and Gopher. Also, there is a separate instance of each protocol module per thread, which allows each thread to access local data without any synchronization.

Based on the URL's scheme, the crawler instance selects the appropriate protocol module from the Mercator's configuration for downloading the document. It then invokes the protocol module's *fetch* method, which downloads the document from the Internet into a per-thread *RewindInputStream* (RIS). A RIS is an I/O abstraction that is initialized from an arbitrary input stream, and that subsequently allows that stream's contents to be re-read multiple times.

Once the document has been written to the RIS, the worker thread invokes the *content-seen test* to determine whether this document, even if associated with a different URL, has been seen before. If so, the document is not processed any further, and the worker thread takes the next URL from the frontier. Every downloaded document has an associated MIME type. The Mercator's configuration's file not only associates schemes with protocol modules but also the MIME types with one or more *processing modules*. Like protocol modules, there is a separate instance of each processing module per thread. For example, the Link Extractor and Tag Counter processing modules in Figure 2.10 are used for text/html documents, and the GIF Stats module is used for image/gif documents. By default, a processing module for extracting links is associated with the MIME type text/html.

A processing module is basically an abstraction for a typical parser that is used to process the downloaded documents. It does the tasks like extraction of links from HTML pages, counting the tags in HTML documents, or collects statistics about GIF images. Based on the downloaded document's MIME type, the crawler thread invokes the *process* method of each processing module that corresponds to the associated MIME type which then extracts all the links from the downloaded hypertext document. Each extracted link is converted into an absolute URL, and tested against a user-supplied *URL filter* to determine if it should be downloaded. If the URL passes the filter, the thread performs the *URL-seen test*, which checks if the URL has been seen before, i.e. if it is in the URL frontier and has already been downloaded. If the URL is new, it is added to the frontier else is simply discarded. In short, Mercator offers the following distinguishable features:

- (i) Mercator is designed to scale up to the entire Web so as to fetch the web of documents efficiently.
- (ii) It is designed in a modular way, keeping in mind the expectation for extensibility by adding new functionalities that might be specified by third parties.
- (iii) The architecture of Mercator suggests the use of pluggable components so that it can be reconfigured to use different versions of its various components like URL Frontier, URL Filter etc. This allows customizing the behavior by plugging in the various components dynamically.

(b) UbiCrawler[20]: It is a high performance fully distributed crawler in terms of the crawling processes with a prime focus on distribution and fault tolerance. Its design has been partitioned into two major components - *Crawling System* and *Crawling Application*. The *Crawling System* itself consists of several specialized components, in particular a crawl manager, one or more downloader's, and one or more DNS servers. All of these components, plus the crawling application, can run on different machines (and operating systems) and can be replicated to increase the system performance.

The crawl manager is responsible for receiving the URL input stream from the applications and forwarding it to the available downloader's and DNS resolvers while enforcing rules about robot exclusion and crawl speed. A downloader is a high-performance asynchronous HTTP client capable of downloading hundreds of web pages in parallel, while a DNS resolver is an optimized stub that forwards queries to local DNS servers. Figure 2.11 presents the architecture of the UbiCrawler with the main focus on how the data flows through such a distributed system.

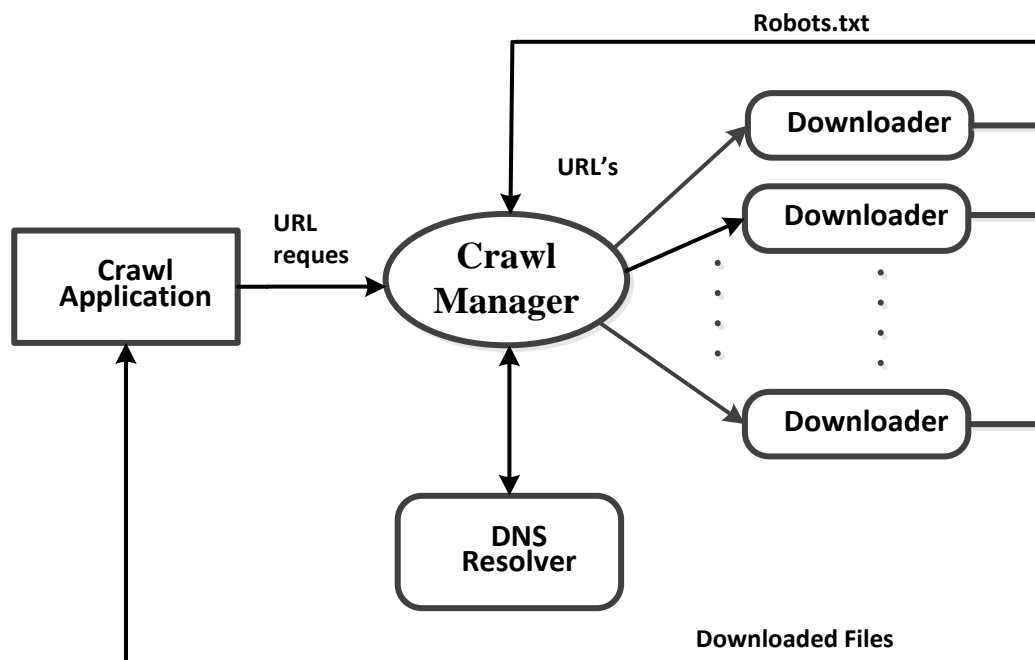


Figure 2.11: The UbiCrawler.

The main components of UbiCrawler are discussed below:

A. Crawl Manager

The crawl manager is the central and the first component of the system that is started up. Afterwards, other components start and register themselves with the manager to offer or request services. The manager is the only component visible to the other

components. It receives requests for URLs from the application, where each request has a priority level and a pointer to a file containing several hundred or thousand URLs and located on some disk accessible via NFS. The manager enqueues the request, and eventually loads the corresponding file in order to prepare for the download, though this is done as late as possible in order to limit the size of the internal data structures. In general, the goal of the manager is to download pages in approximately the order specified by the application, while reordering requests as needed to maintain high performance without putting too much load on any particular web server.

After loading the URLs of all request files, the manager queries the DNS resolvers for the IP addresses of the servers, unless a recent address is already cached. The manager then requests the file robots.txt in the web server's root directory, unless it already has a recent copy of the file. After parsing the robots files and removing excluded URLs, the requested URLs are sent in batches to the downloader's, making sure that a certain interval between requests to the same server is observed. The manager later notifies the application about the various web pages that have been downloaded and are available for processing. The manager is also in charge of limiting the overall speed of the crawl and balancing the load among Downloaders and DNS resolvers, through monitoring and adjusting the DNS resolver load along with the downloader speed as needed. The manager performs periodic tests of its data structures, and after a crash, a limited number of pages may have to be re-crawled. It is up to the application to detect these duplicate pages.

B. Downloaders and DNS resolver

The downloader component, implemented in Python, fetches files from the web by opening up to 1000 connections simultaneously to different servers, and polling these connections for Data signal. The downloaded or arrived data is then collected into files located in a directory determined by the crawling application. Since a downloader often receives more than a hundred pages per second, a large number of pages have to be written out in one disk operation. The way pages are assigned to these data files is unrelated to the structure of the request files sent by the application to the manager. Thus, it is up to the application to keep track of which of its URL requests have been completed. The crawl manager may adjust the speed of a downloader by changing the number of concurrent connections that are used.

C. Crawling Application

The crawling application is breadth-first crawl starting from a set of seed URLs, in this case the URLs of the main pages, which are initially sent to the crawl manager. The application then parses each downloaded page for hyperlinks, checks whether these URLs have already been encountered before, and if not, sends them to the manager in batches of a few hundred or thousand. The downloaded files are then forwarded to a storage manager for compression and storage in a repository. The following two important points were observed: First, since each page contains on average about 8 hyperlinks, the set of encountered URLs grows very quickly beyond the size of main memory, even after eliminating duplicates. Thus, after downloading 20 million pages, the size of this set reaches above 100 million. Second, at this point, any hyperlink parsed from a newly downloaded page and sent to the manager is only downloaded after several days. Thus, there is no reason for the manager to immediately insert new requests into its dynamic data structures.

2.5.3.1. Issues and Challenges with High-Performance Crawlers

The following challenges have been observed with high-performance crawlers:

1. Assignment of URL's among different agents: The major challenge in distributed crawler is the way URLs are assigned efficiently and dynamically to download among the crawler threads. The assignment must take into consideration the various constraints like minimum rate of requests to Web servers, appropriate location of the crawler threads on the network, and the effective exchange of URLs.
2. Priority in Crawling: The URL collection in the Frontier must be effectively partitioned in such a way that only those URLs that seem to be useful are processed for fetching answers to any query and not by processing all the URLs. Also, this chosen subset should be able to provide a high number of relevant documents. Considering the dynamic nature of the Web, a criterion that orders the URLs based on certain priorities and must be brought into existence. Hence, an important challenge is to find effective ways of partitioning the URL collection so as to process the smallest possible subset when answering any user query.

3. Load Balancing: The next challenge is to determine an effective way of balancing the load among the different index servers. There must be a good strategy to distribute the data in order to balance the load as much as possible.
4. Network bandwidth consumption: Network bandwidth is a scarce resource and is a big challenge. Therefore, when queries are resolved in a distributed fashion, the servers that should be contacted must be determined efficiently.
5. Efficient cache design: The next challenge is to design an effective cache that have high hit ratio and also overcomes the network constraints

2.6. PROBLEMS WITH THE SEARCH ENGINE LANDSCAPE

The Search Engines and their employed crawlers offer a lot of advantages as mentioned in the earlier sections. Different type of crawlers (focused crawlers, parallel crawlers etc.) are used by different search engines to serve the differing needs of the users. The most common and important of these advantages is the ease of user-search process. Likewise, the various search engines that are in practice typically suffer from a common set problem that get in their ways while providing service. Some of these are discussed as follows:

- 1) The most basic thing that is required to retrieve the information from the WWW is the URL. However, when accessing information via a search form using the HTTP GET request method or HTTP POST request method, certain optional parameters need to be specified in the URL. The parameters are included as part of the URL in the case of HTTP GET, but not in the case of HTTP POST. In such a case of HTTP POST when no parameters have been specified, it is not possible to publish the resulting page through a link or URL. Hence, those pages cannot be crawled in the conventional way of following hyperlinks.
- 2) The contents of a page are typically assumed to have changed if a request yields different contents upon re-issuing the same request. For example, in booking sites or shopping sites, the availability or number of items in stock may change rapidly. Also, new products are repeatedly added and old ones removed. In such cases of dynamic content, if it is to be included in the index of a search engine, then it should then also be crawled and updated frequently and regularly.

- 3) The recent studies have shown that the coverage of a single search engine is very limited [31], [32], [33], [34]. Therefore, it is not completely satisfactory to search for information on the Web by using only one search engine even if this search engine is most powerful. In addition, users do not know the capabilities of a search engine, due to lack of user interface support by these engines. On the WWW, there are a lot of search engines, but majority of the Web users, make use of only several famous ones like Google, Yahoo! etc. This makes users unable to judge and know where they can find the search engines that might provide the best answers for their queries. However, some special-purpose search engines can better answer users' specific information needs than the general-purpose ones can do.
- 4) The existing search engines typically define relevance of a document based on the count of links (forward links and back links) and words (term frequencies) with a little consideration to the semantics.
- 5) The crawlers working in parallel suffer from overlapping problem in the sense that multiple copies of the same web documents may be downloaded multiple times, leading to wastage of crawler's time, network bandwidth and other resources such as storage at the Search engine side.

Thus, in the light of above discussion, it may be noted that there is a need to modify the existing architecture of web crawlers, with a view to improve the quality of downloaded web documents, within time constraints.

The next chapter brings an insight into the Hidden Web and the various crawlers existing in the literature for the same.

CHAPTER 3.

HIDDEN WEB: A REVIEW

3.1. INTRODUCTION

A traditional web crawler simply crawls through the web pages by following hyperlinks but a huge amount of the data is contained inside the web databases behind interactive search forms, which is not easily ‘crawlable’. Based on this, the WWW has been categorized into the following two parts:

- 1) Surface Web [11, 28] includes the former category of web pages i.e. the portion of the Web that can be crawled by conventional web crawlers and indexed by general-purpose search engines.
- 2) Hidden Web [39, 81] refers to the second category that includes the abundant information hidden behind user-interactive search forms and is not directly accessible to crawlers. A search form typically consists of various control elements like text boxes, labels, buttons etc. An example of such a search forms that allows online search of books is as shown in Figure 3.1.

The image shows a screenshot of a web-based search interface for books. At the top, there is a navigation bar with links: [Advanced Search](#), [Browse](#), [Booksellers](#), [Community](#), [Sell Books](#), [Textbooks](#), and [Rare Books](#). Below this is a search bar with the text 'Search Books:' followed by a dropdown menu set to 'By Keyword' and a text input field. To the right of the input field is a red button labeled 'Find Book' and a link '> Advanced Search'. Below the search bar, there is a breadcrumb trail: [Home](#) > [Advanced Search](#). The main section is titled 'Advanced Search' and contains two columns of search criteria. The left column has input fields for 'Author:', 'Title:', 'ISBN:', 'Keywords:', 'Publisher:', 'Published Date:' (with 'min to' and 'max' labels), 'Price (US\$):' (with 'min to' and 'max' labels), and 'Bookseller Country:' (a dropdown menu set to '- All Countries -'). Below these is a 'Boolean Searching:' section with radio buttons for 'On' and 'Off' (selected), and a link '[More Info](#)'. The right column is titled 'Refinements' and includes 'Attributes:' with checkboxes for 'First Edition', 'Signed', 'Dust Jacket', and 'Not Print on Demand'. Below this is 'Books Listed Within:' with radio buttons for '24 Hours', '48 Hours', and 'All' (selected). Then there is 'Binding:' with a dropdown menu set to 'Any Binding'. Next is 'Sort Results By:' with a dropdown menu set to 'Lowest Total Price'. Finally, there is 'Results Per Page:' with a dropdown menu set to '30'. At the bottom of the form, there is a red button labeled 'Find Book' and a link '[Clear Fields](#)'.

Figure 3.1: An example of user-interactive search form that offers online search of books.

A user must fill in the values for some of the control elements to obtain the result pages. Based on the search criteria specified in the form, certain tuples are selected from the database for generating the response pages. Thus, the results are generated “on the fly” in response to the user-specified search conditions.

3.2. CHARACTERISTICS OF THE HIDDEN WEB

The Hidden Web is principally defined by its included content or resources which are not only authentic (up-to-date and accurate) but also highly relevant to every information requirement as more than half of its content, resides in topic-specific databases. Most of these databases are overflowing with massive interesting and valuable information [119]. The characteristics that primarily urge the need to access the content in the Hidden Web are as follows:

1. *Huge and Ever-increasing Size:* A lot of information is buried inside such online databases and the pages generated dynamically thereof. Few examples of those tremendous databases include information like patents, shopping catalogs, flight schedules, climate data, stock exchange data, data collected on space missions, academic databases filled with scientific papers [32, 40]. It is estimated to contain 7,500 terabytes of information compared to 19 terabytes of information in the Surface Web [28]. More than 200,000 Hidden web sites presently exist on the Web [11, 28]. Also, the size of the Hidden Web is expected to increase at a pace faster than the Web itself as more and more organizations put their valuable content online through an easy-to-use Web interface [41, 42].
2. *Content Quality:* The Hidden Web content is believed to be of very high quality as it is contained within authorized databases and is accessible only through the search tools purposely designed for that site [11, 29] allowing searching in real time and retrieving the information that is current, up-to-minute.
3. *Publicly accessible content:* Public information (like Media, News) on the Hidden Web is estimated 400 to 550 times larger than that commonly defined WWW. Around 95% of the content contained in the Hidden Web is publicly accessible and not subject to fees or subscription.
4. *User facilitated search process:* Users can perform Hidden Web searches as easily as surface web searches & with greater results as the search interfaces serving as front end are user- understandable.
5. *Dynamic Nature:* The dynamic nature of the Hidden Web has been witnessed from the fluid nature of the WWW itself which is an ever-evolving information source where web pages, web databases and even whole websites, appear and disappear, modified or restructured giving the associated

implication “a significant percentage of the Hidden Web databases have a small life span”.

All these characteristics not only make the study of Hidden Web interesting but also a challenging task.

3.3. USER-INTERACTION IN THE HIDDEN WEB

A search form typically consists of the HTML code and the CGI program. The HTML tags create the visual representation of the search form and the CGI program decodes the information contained within the form. A form usually starts with the `<FORM>` tag and begins with an `ACTION` attribute that specifies the URL of the CGI program that will process the form information. This form information usually comes in terms of the user data raised as a query by specifying the query terms in the search form. Once the form has been completed, the program's name and collected parameter values are sent to the server, as shown in Figure 3.2. The search form then specifies the CGI program that is to be executed at the server side, along with parameter values (that are filled in the form fields by the user). The server then passes the document to the user/client. The Hidden Web comprises of this dynamic content included in such a document.

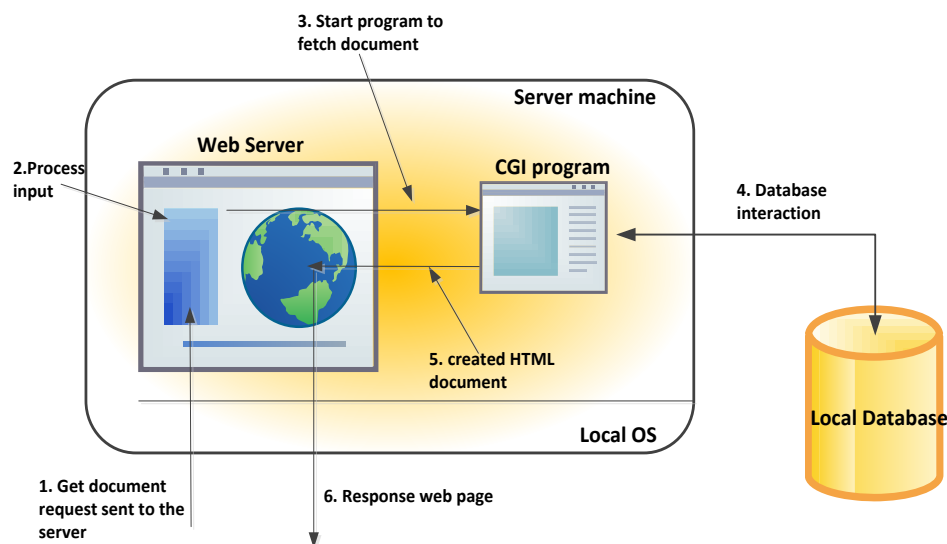


Figure 3.2: The principle of using server-side CGI Programs

The main task of a server is to handle the client requests by simply fetching documents. Basically the CGI defines a standard way by which a Web server can execute a program taking user data as input.

The data contained in the web database that hosts the search form is accessible to the user by filling the form with certain values and submitting it. The form submission returns a web page containing the list of links to relevant data.

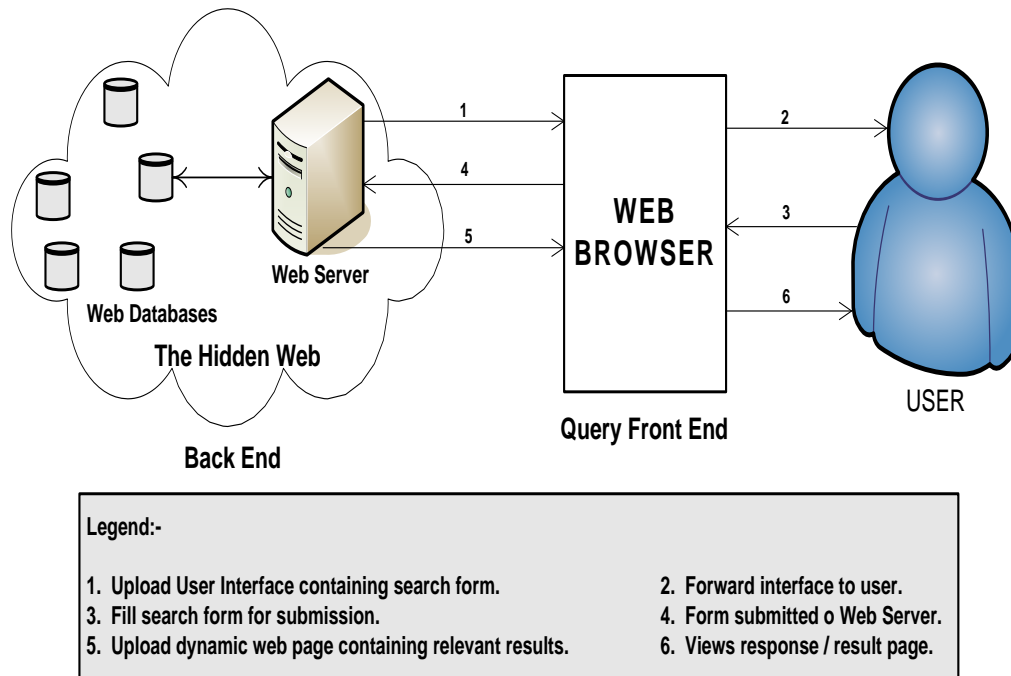


Figure 3.3: User interaction with a search form interface

Figure 3.3 illustrates the sequence of steps that take place when a user wants to access the contents in the Hidden Web. The user has to fill in the search form for retrieving the documents that are dynamically generated from the underlying database based on the specified query [39, 41]. But, with millions of databases and endless possible permutations of search criteria, it is not only difficult rather impossible to sift through every possible combination of the dynamically resulting tuples or pages. Hence, an automated approach to access the Hidden Web content is needed.

3.3.1. AUTOMATIC APPROACH TO ACCESS THE HIDDEN WEB

To automatically access the contents in the Hidden Web, the crawler must imitate the sequence of steps (as in Figure 3.3) that are being followed by the human. Figure 3.4 illustrates the difference in the sequence of steps undertaken by any crawler to access the Hidden Web's informational content. This approach has the main advantage of best fit with the conventional search engine technology.

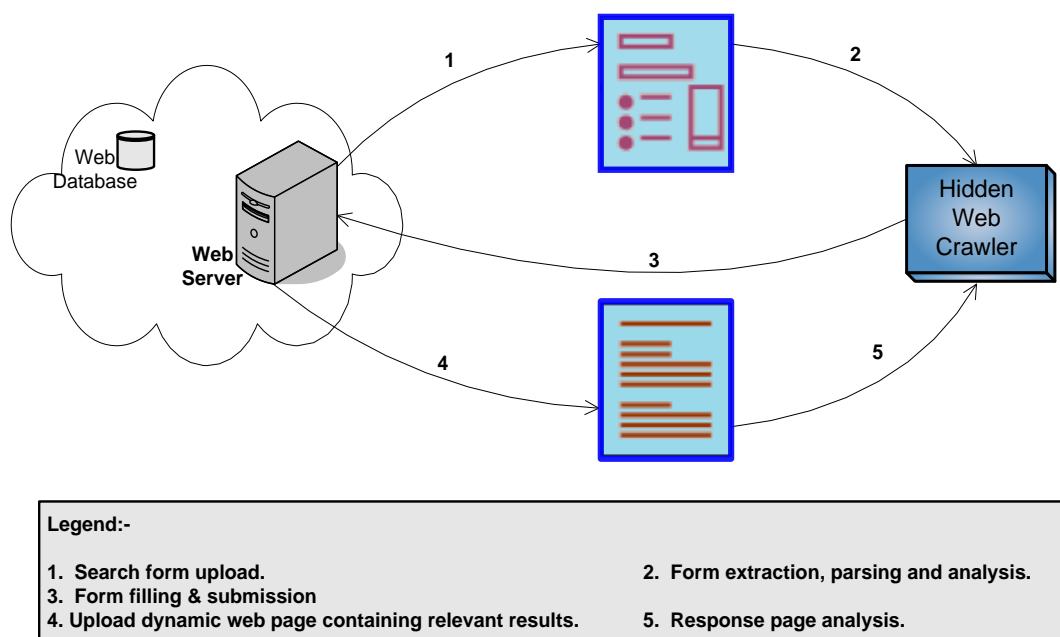


Figure 3.4: A crawler interacting with the search form interface

Though the approach is straightforward and is easily applicable, but pre-computing valid and relevant form submissions for all search forms is a challenging issue because of the following factors:

- 1) Many different kinds of databases exist on the Web [11]. The databases may contain content that is either free form text (unstructured database) or data records with attribute-value pairs (structured database) [41, 44]. In order to search a database, its underlying design must be analyzed for effective retrieval of contents.
- 2) Another major reason is the interface or the search form that is offered by the database. Every search form has its own structure which may not resemble with the others as the search form used to query depends on the structure of the underlying database. The search form used to query the content contain either a single query box accepting a free-form string (unstructured or single attribute forms) or multiple query boxes pertaining to different attributes of the content (structured or multi attribute forms) [41, 44].

The unstructured databases usually contain plain-text documents which are not well structured and provide a simple keyword-based search interface having an input control (text type) where users type a list of keywords to fill it in. An example of such a search interface is shown in Figure 3.4.



Figure 3.5: Keyword-based Search Interface

In contrast, structured databases provide multi-attribute search interfaces that have multiple query boxes pertaining to different aspects of the content. For example, Figure 3.5 shows a multi-attribute search form interface for an online book store offering structured content (title, author, publisher, price, ISBN, number of pages) coupled with a structured query interface (typically a subset of the content attributes like title, author, ISBN, publisher). But these search forms serve as the only entry point to the Hidden Web and thus, must be modeled and processed.

Figure 3.6: A multi-attribute search form interface for an online book store

Though pre-computing the most relevant form submissions for all interesting HTML forms is a challenging issue but is a passive task which can be carried out by the

crawler in the background when active, irrespective of the structure and design of the underlying hidden web databases.

3.3.2. DIFFERENCE BETWEEN A CONVENTIONAL CRAWLER AND HIDDEN WEB CRAWLER

A general web crawler starts by taking a URL from the seed set stored in the URL Frontier and downloading the associated web page. The downloaded web page may contain a search form as the pages containing the search form are simply designed like other webpages using HTML tags. But these crawlers do not possess the information and intelligence needed to fill the search forms as can be done by humans. The flowchart depicting the working of the traditional web crawlers for the Surface Web is shown in Figure 3.6

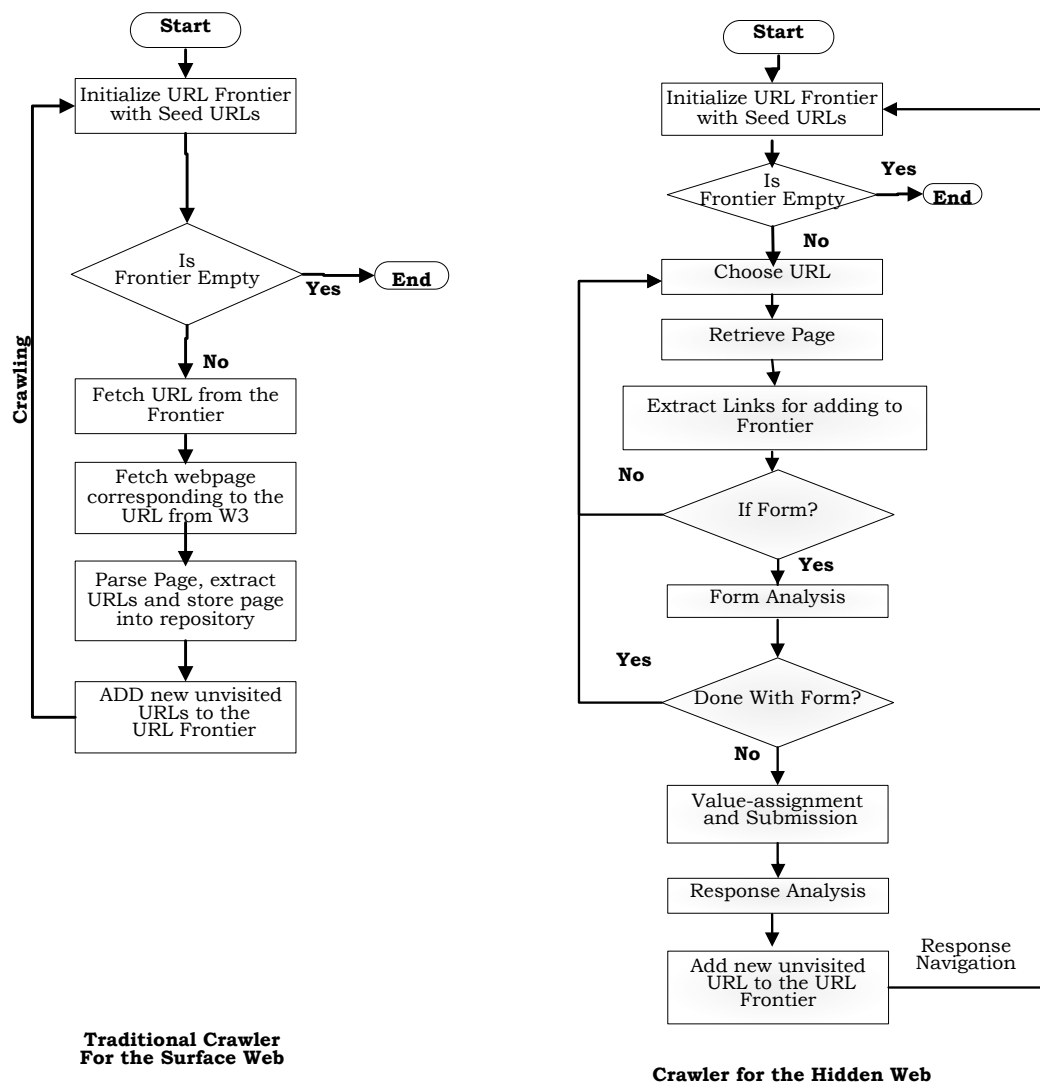


Figure 3.7: Principle behind a traditional crawler and the Hidden Web Crawler.

On the other hand, the Hidden Web crawlers starts in the same way as the traditional crawlers by taking a set of seed URLs but are trained to process the search forms by providing them the information necessarily needed to fill the forms. Once the hidden web crawler downloads the webpage, it checks whether the web page contains any search form. If the downloaded web page contains a search form, then the form is extracted for analysis and processing. For analyzing the extracted form, it is parsed for the underlying structure (the various control elements) and schema (the type of values that can be taken by the control). This aids the Hidden Web Crawler in effectively processing the search forms. As a next step, the crawler tries to predict the set of values that can be filled in each form field for valid submission. The filled form is then submitted to the designated server for downloading the response pages. The flowcharts in Figure 3.6 illustrates the difference in the working of crawlers for the Surface and Hidden Web

In brief, the traditional crawlers can record the address of the search front pages but cannot input the search criteria and request data. The difference is fundamental and implies that traditional crawling techniques that have been successfully applied to access the Surface Web content are inappropriate for the Hidden Web. Hence, pops up the non-trivial problem of “training” the crawler for processing the search forms that are restricted for use by humans.

3.4. CRAWLING THE HIDDEN WEB

The Hidden web is exceptionally huge in size and diversity of data. The databases on the Web comprise of data from numerous domains like Books, Travel, Automobiles, Health, Sports etc. This further complicates the task of any Hidden Web crawler. To minimize the cost of crawling and extracting the hidden web content, the hidden web crawler must avoid: processing of irrelevant databases and search forms; use of invalid values for filling or processing the search forms. unnecessary processing of unnecessary resources [18, 45, 46, 47]. This drives the focus of the Hidden Web crawler to the following two activities:

- A. Resource Discovery/ Filtering irrelevant sources:** In order to overcome the problem of Scale, the crawler must be trained to carry a crawl of only the relevant sources (effective crawling) rather than carrying a comprehensive crawl of the Hidden Web (comprehensive or exhaustive crawling) [39, 41, 48, 49]. In other words, the crawler must selectively seek out hidden database

sources that are relevant to user's information need at hand. This requires the crawler to first locate the sites containing search form interfaces and then select the relevant subset from it. Most of the current day search engines try to possess a comprehensive crawl of the Surface Web, which is likely to include abundant Hidden Web search pages but the main challenge lies in evaluating the source for relevance. (This involves identifying relevant sections of the HTML page that contains the form, identifying relevant page attributes like the various HTML tags say title tag, meta tags etc.). And as the Hidden Web data sources are growing continuously at a high rate, selecting the subset of relevant sources proves not only cost-effective but also effective in time and makes the crawler less prone to errors.

B. Content extraction: The task of harvesting or extracting information lying behind the search form interfaces of the selected Hidden Web sources depends largely on the way, it deals with the those forms[5, 39, 41, 42, 44, 52, 81]. The crawler has to be able to understand and model the search form and come up with meaningful queries to issue to the search form and probe the database behind it. Finally the crawler must be able to extract the data instances from the retrieved result pages. This problem of Content Extraction poses significant challenges and the solution lies in the three steps namely: Search interface understanding, automatically filling search interfaces and Information extraction. The three steps together typically comprise of the following four modules of the system:

- Form Analyzer analyzes each and every downloaded page to see if it can be used as a search page to retrieve information or not. It basically checks whether a web page is query able, has some form fields or not and.
- Form Parser extracts the fields from the search form and passing them on to the form processor for filling.
- Form Processor fills in the various form fields by assigning appropriate values and finally submits the form for retrieving result pages.
- Result Analyzer will analyze all the result pages obtained by the crawler after form processing and execution, in order to get the required information.

3.5. RELATED WORK IN HIDDEN WEB CRAWLING

In this section, we discuss relevant research work in the area grouped by the problem domain and the various strategies have been discussed. Accessing the Hidden Web content is difficult, chunking the problem domain into that of resource discovery and content extraction and presenting the related works as per the categorization helps us to better explore the research already being done in the area. Circumscribed by the crawler's limitation of resources and the huge size of the Hidden Web, a Hidden Web crawler typically adopt to one of the following strategies for extracting the content residing in web databases:

1. ***Breadth-Oriented crawling:*** As the hidden Web contains tens of millions of databases and search forms, a breadth oriented hidden Web crawler focuses on covering more and more data sources rather than exhaustively crawling the content inside one specific data source. Thus, the major challenge in this kind of crawling seems to be locating the hidden Web resources and analysing the returned results for learning and understanding the interface required to automate the process of content extraction.
2. ***Depth-Oriented crawling:*** It focuses on extracting the contents from a designated hidden web resource i.e. the goal is to acquire most of the data from the given data source. Now, the crucial challenge for the crawler is to actively issue queries at the interface of the designated database in order to uncover the database contents while incurring minimal cost. However, the crawler must automatically generate promising queries so as to carry out efficient crawling which is an exigent task. The problem is termed as query selection.

The above approaches are equally facilitated by gaining an insight into the type of information being contained in any web database which may be either unstructured or structured.

3.5.1. BREADTH-ORIENTED HIDDEN WEB CRAWLERS FOR RESOURCE DISCOVERY

A focused crawler targets to select and download web pages associated with only those links that lead to documents that seem relevant to the domain or topic of interest

and hence addresses the resource discovery problem, The work on focused crawling [18, 48, 49] mentioned in section 2.3.1 describes the design of focused crawlers for the Surface Web. The work appears to complement the mentioned problem of scale and size of the Hidden Web as the techniques used to discover the relevant documents can be used to identify the relevant web databases for the Hidden Web crawler.

The web directories like dmoz, brightplanet's searchable directory etc. that provides links to online databases in a topic hierarchy can be used as seed points for the crawl.

The searchable forms can typically be identified using any one of the following two approaches [5, 7, 45, 53, 64]:

- 1) A *pre-query approach* that analyzes the features of the search forms.
- 2) A *post-query approach* that is based on the analysis of the result pages retrieved in response to a query submission through the filled search form..

3.5.1.1. Automated Discovery of Search Interfaces on the Web

Cope et. Al. [53] first presented the pre-query method to identify the searchable forms. The approach relies on representing each search forms by automatically generating the unique features. For each search form that is given as input, the following features or parameters were extracted: the name off each control element, the values that can be taken by each control on the search form; the number of control elements that take free form text as input, the number of control elements taking passwords as inputs etc. On the basis of these generated features, the search forms were classified with the help of a machine-learning algorithm based on Decision Trees. The Classifier C4.5 that was introduced by Quinlan [57]. This classification algorithm C4.5 suggested the development of classification models or decision trees from the input data set in accordance with the set of features generated in the first step. To prune the decision trees, the models were induced to handle various possible types of training data like data having continuous attribute value ranges; data with missing attribute values, continuous attribute value ranges.

Their approach achieved a precision of 87 percent and a recall of 85 percent when the method was experimentally evaluated by the authors on an academic web site and random web sites.

3.5.1.2. Crawling For domain Specific Hidden Web Resources

Bergholz and Chidlovskii [45] presented an example of the post-query approach for automatically discovering the searchable forms. Their approach uses a domain-

specific crawler that starts from the web pages on the Surface Web or Publicly Indexable Web for detecting the relevant search forms in a domain. To initialize the seed URLs for the domain-specific crawler, the Directory structure of Google is used in their work. The hidden web crawler presented in their work comprises of four main components:

1. *Seed Starter* is used to set the seed URLs that act as starting points for the crawler,
2. *Local Crawler* to find the pages containing the search forms and are relevant in the domain of consideration,
3. *Form analyzer* that analyzes the search forms discovered by crawling and separates the searchable forms from all others ,
4. *Query Prober* issues the queries to the separated searchable forms and extracts the data hidden behind these forms to find its relevance to a given topic or domain.

They have presented two different modes of working for the Hidden Web Crawler: *a domain-specific* and *a random mode*. In the domain-specific mode, Google Web Directory [10] is used as a reference to the hierarchy specifying the categories of interest. The hierarchy comprises of a collection of web sites that are manually selected and classified by experts in the domain. Also, with each category C_i in the hierarchy, a set of keywords D_i that seem relevant to that category is associated. The approach has considered the top five levels of the category hierarchy from the Directory and for each of the selected category from the hierarchy, the set $S = \{s_1, \dots, s_N\}$ of top N pages has been considered relevant for use by the system. The top N pages have been identified by ranking them for relevance and importance by using the Google's PageRank method [3, 43]. As a next step, the set of relevant keywords D_i , is used to fill the separated searchable forms.

In the random mode, the crawler randomly selects a set of M web pages, $R = \{s_1, \dots, s_M\}$ by using Yahoo! Random Web site generator to find the relevant resources. When working in the random-mode, the relevant keywords for filling the search forms is chosen by selecting the characteristic keywords from the random pages using Xerox XeLDA server.

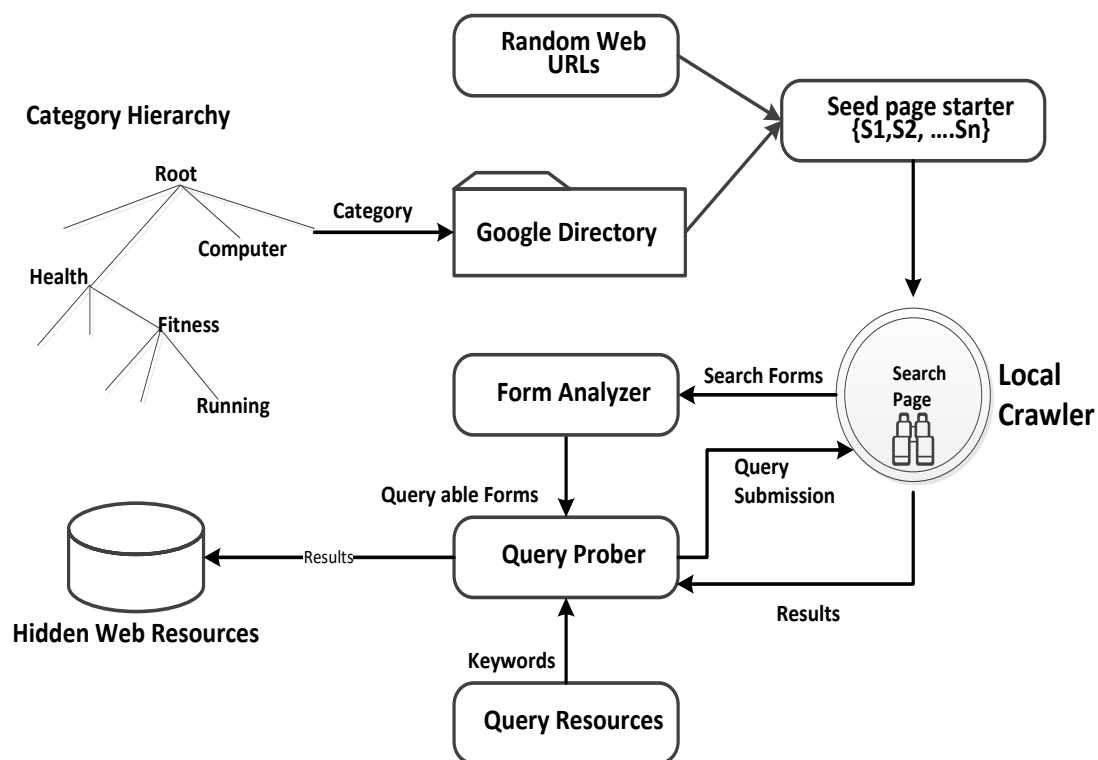


Figure 3.8: Architecture of Hidden Web Crawler

Figure 3.8 presents the detailed architecture of the hidden web crawler. It can be explained with the help of its various components as follows:

The *local crawler* identifies the pages that work as entry points to the Hidden Web. The starting points for the local crawler are selected by using either the set S of top N pages from the category hierarchy (in domain-specific mode) or the set R of M randomly selected webpages (in random-mode). It then traverses the web by follows the hyper-links in the standard breadth-first manner and keeping a record of all the webpages that contains an HTML form. The crawler has been termed ‘local’ in the sense that it never leaves the site of the seed s_i and ignores all the links that forces quitting the site.

The search forms are extracted from all the web pages that have been traversed by the local crawler and are passed to the form analyzer. The *form analyzer* parses each HTML page downloaded by the crawler, eliminates the duplicate forms and creates the definitions for all HTML forms. The analyzer then filter out forms that require pre-registration such as login, sign in etc. from the queryable forms. Four different types of forms have been defined as shown in Figure 3.9. The largest class contains all forms. Textual forms are the ones that contain at least one free-form input or a

textual control and form a proper subclass of all extracted forms. The textual forms however, still include some forms that are irrelevant (user registrations with no password protection) to search over the Hidden Web.

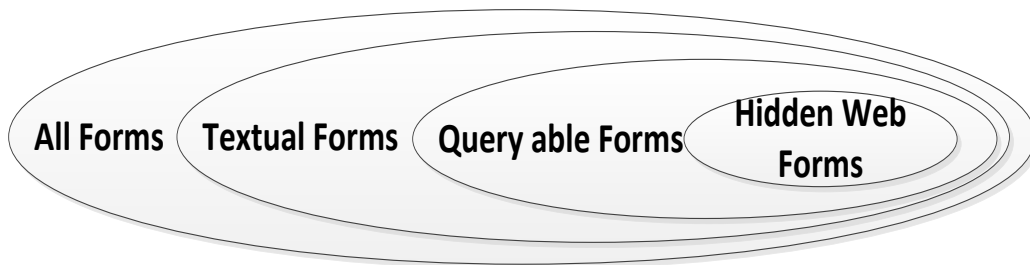


Figure 3.9: Classes of HTML forms

All the forms that need a prior registration or login and/or password protection controls are discarded from the inclusion in the class of searchable forms. The smallest subclass of the forms represents the Hidden Web forms, which serve as entry points [61] to the web databases. These forms can be used for submitting a query to the collection of Hidden Web resources and reporting the result of query execution on the collection. The Hidden Web forms cannot be syntactically recognized like the textual and queryable forms [41] and need further analysis. These search forms are sent to the Query Prober to assess whether they are interfaces to searchable web databases or not.

The query prober is then used to analyze and validate the syntax and the description generated by the form analyzer for each search form as an incomplete or incorrect description does not allow a valid form submission. The most important task performed by the Query prober is automatically filling the search forms to obtain valuable response pages. In order to automatically fill the search forms, the Query prober uses the default values of non-text controls and some domain-specific phrases for the textual controls on the search form. These domain-specific phrases form the set of ‘positive’ queries whereas a set of ‘non-sense’ words is used to form the ‘negative’ queries to be raised by the query prober to the candidate forms. The Query Prober then compares the resulting pages obtained by issuing the positive and negative queries at the search form to assess and find whether the page has a searchable form. Another task of the query prober is to decide the relevance of the resource to a category hierarchy based on the obtained response pages.

3.5.1.3. ACHE : An adaptive crawler for locating hidden Web Entry points

The authors in [54] have proposed a way to automatically locate the web databases of interest. The architecture of the form focused crawler (FFC) given uses three different classifiers for locating the web pages: Page classifier, link classifier and form classifier. The architecture of FFC has been presented in Figure 3.10. The crawler combines the use of a page classifier and a link classifier for focusing its crawl on a particular topic. Both the classifiers are trained by taking into account the contents of web pages and by observing the patterns in & around the hyperlinks embedded within the web page.

The authors first make use of a backward search strategy to analyze and prioritize links leading to searchable forms. The frontier manager is another major component of the FFC framework and is used to select the next target link for crawling based on their reward values decided by the current status of the crawler and the priority of the link in the current crawling step. The FFC also uses a form classifier to filter out useless forms. If a form is found searchable by the form classifier, it is added to the form database if not already present in it. The form classifier is based on the usage of decision trees to determine whether the candidate form is searchable or not. The forms that could not be considered for search include forms with login, registration, discussion groups, mailing subscriptions etc.

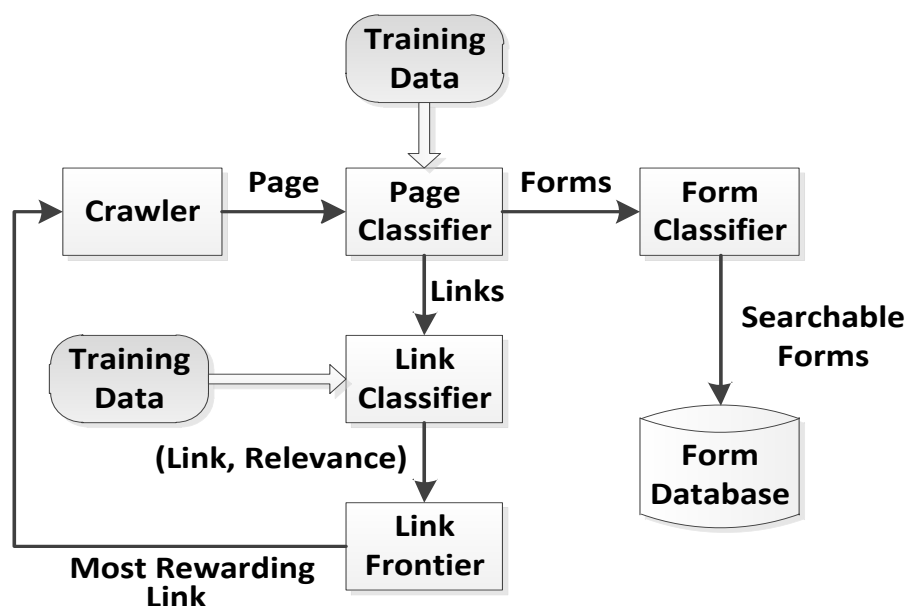


Figure 3.10: The Form Focused crawler (FFC)

The authors again in [56] addressed the limitations of the FFC by presenting a new framework ACHE (Adaptive Crawler for Hidden-Web Entries). ACHE tries to improve its behavior in the future runs by learning from its previous executions and adapting to the environment accordingly. Given a set of Web forms that are entry points to online databases, ACHE aims to efficiently and automatically locate other forms in the same domain

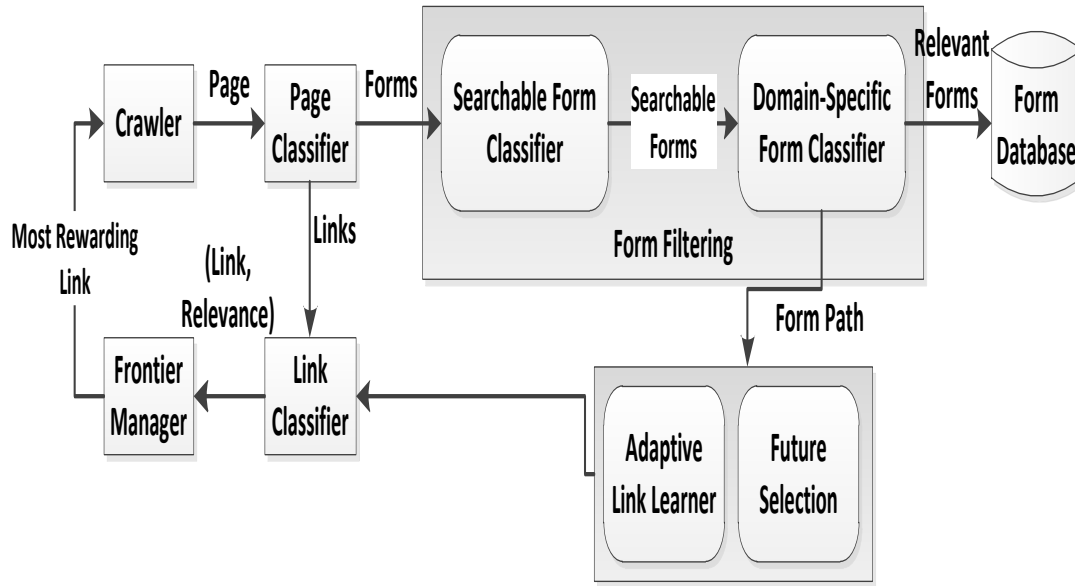


Figure 3.11: ACHE Architecture

In addition to FFC, the ACHE comprises of two more classifiers: the searchable form classifier (SFC) which classifies the retrieved form as searchable or non-searchable and the domain-specific form classifier (DSFC) which checks whether the form belongs to the target domain. ACHE also employs a component called the adaptive link learner that dynamically learns features automatically extracted from successful paths by the feature selection component and updates the link classifier.

3.5.2. DEPTH-ORIENTED HIDDEN WEB CRAWLERS FOR CONTENT EXTRACTION

Most approaches to information retrieval in the Hidden Web are focused on leveraging high-quality information available in online databases[5, 39, 41, 42, 44, 52], which needs understanding the various semantics associated with the form elements and automatically filling them as they are the only entry points to the Hidden Web.

3.5.2.1. HiWE

Raghavan and Garcia Molina [39] introduced the problem by proposing an operational model for extending the crawler beyond the Surface Web. The target of their model is to add to the crawlers, the capability of automatically filling forms. Their model shown in Figure 3.11 serves as a basis for the prototype hidden Web Crawler called the HiWE (Hidden Web Exposer).

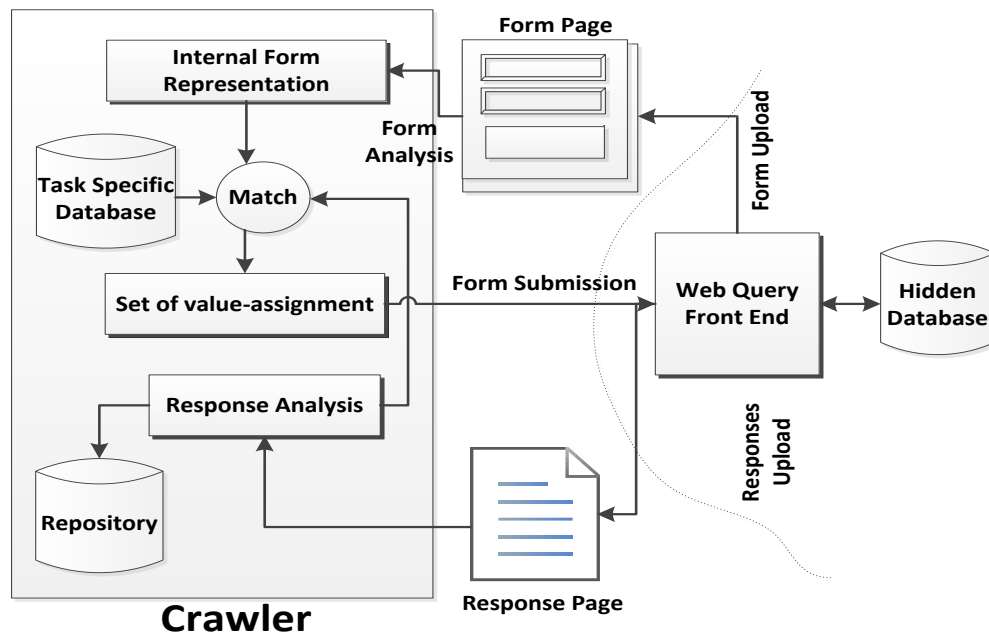


Figure 3.12: Crawler Form Interaction

An outline of the architecture of the model is given in Figure 3.13. This model of a hidden Web crawler consists of two internal data structures and six functional modules.

The most basic data structure of HiWE is the URL list that contains all the URLs that have been discovered by the crawler so far in its process. When the crawler is started, the URL list is initialized to a seed set of URLs. Another important data structure is the task-specific database. The task-specific database D is presented in the form of a table called Label Value Set (LVS) table. D contains all the information that is necessary for the crawler to formulate search queries relevant to the particular task. For example, the 'Book Domain' D could contain lists of author names and title of books. The actual format, structure, and organization of D are specific to the crawler implementation at hand.

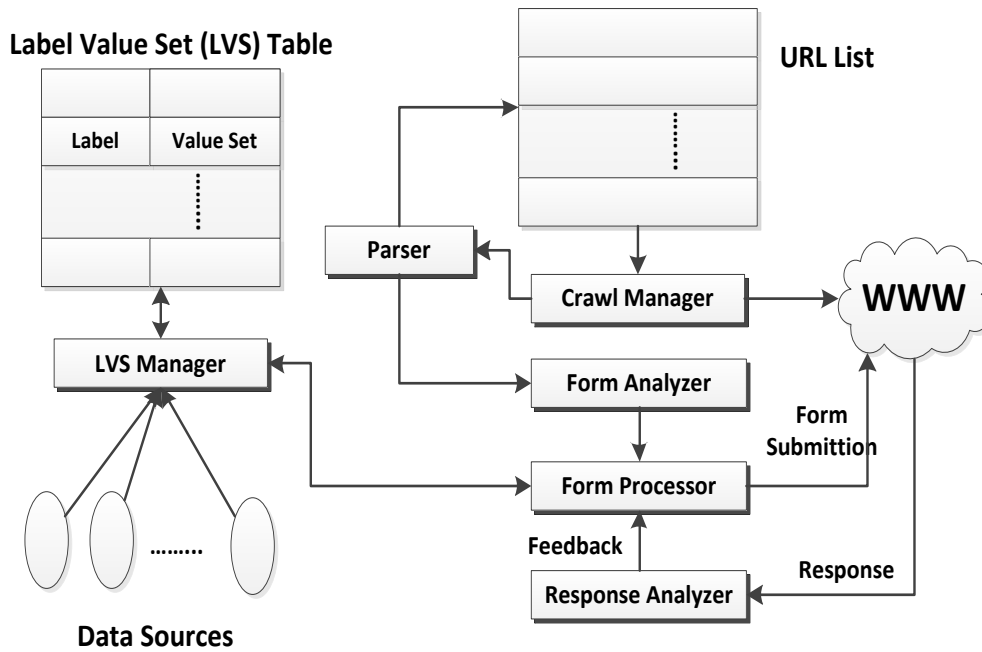


Figure 3.13: Architecture of HiWE.

The Crawl Manager component is responsible for controlling the entire process of crawling. It decides the following:

- 1) the links to be included and excluded from crawling by the HiWE,
- 2) the links to visit next,
- 3) the network connection to be used for retrieving the web page.

The Crawl Manager passes the downloaded page to the Parser module that in turn extracts the embedded hyperlinks from the web page and adds them to the URL List data structure mentioned above. The LVS Manager is responsible for automatically filling the search forms by accessing the data stored in the LVS table. It computes a set of value assignments based on the internal representation of the search form with the help of a matching function. These values will be used for filling the search forms. The LVS Manager repeatedly fills and submits the search forms until all the value assignments in the generated set are exhausted. The LVS Manager is also responsible for maintaining and populating the contents of the LVS table by providing an interface for various application-specific data sources. The major challenge of their approach is dealing with the form elements with infinite domain. Also, the HiWE is not able to recognize and respond to simple dependencies between the control elements on the form (e.g., given two control elements corresponding to states and cities, the values assigned to the “city” element must be cities that are located in the state assigned to the “state” element).

The Response Analyzer takes the result pages obtained after each form submission and stores it in the search engine's index. It also distinguishes between the pages containing search results and pages containing error messages.

HiWE uses LITE (Layout-based Information Extraction) to extract information from the search forms and the response pages. It considers both the textual content as well as the physical layout of the web pages while extracting content. LITE is based on the observation that the physical layout of the different elements of a web page contains significant amount of information.

3.5.2.2. Hidden Web crawler for the Hidden Seek

Ntoulas and Junghoo Cho [41] have provided a theoretical framework for analyzing the process of generating queries for a document collection that support single-attribute queries by examining the obtained results. They have proposed a generic algorithm for Hidden Web Crawler which is given in Figure 3.14.

Algorithm: Hidden_web_crawl()

Step1: While (Resources available)

do

2. $q_i = \text{SelectTerm}()$

//select a term to send to the site

3. $R(q_i) = \text{QueryWebSite}(q_i)$

/*where q_i is the selected query & $R(q_i)$ is the result page for the Query q_i .*/

4. Download ($R(q_i)$);

5. End;

Figure 3.14: Algorithm for crawling Hidden Web Site.

As in the algorithm, their crawler only considers single terms as queries thus has to select the query term, use it for issuing the query, and retrieves the result index page (step 3). It then downloads the Hidden Web pages from the site on the basis of the links that are found on the result index page (step 4). The process is repeated until all the available resources are exhausted (step 1). The main challenge their approach tackled was making the choice of the keyword for the query for which their approach defined three query-generation policies: a policy that picks queries at *random* from a list of keywords, a policy that picks queries based on their *frequency* in a generic text collection, and a policy which *adaptively* picks a good query based on the content of

the pages downloaded so far from the Hidden-Web site. The process starts by learning a global picture starting with a random query, downloading the matched documents, and learning the next query from the current documents. This process is repeated until all the documents are downloaded.

All the three approaches shared the goal of finding the queries that return the maximum number of web documents with the minimum cost. The work defined $P(q_i)$ as the fraction of pages that are returned by issuing query q_i to the site and $Cost(q_i)$ to represent the cost of issuing the query q_i . Where depending on the scenario, the cost can be measured either in terms of time, network bandwidth, the number of interactions with the site, or it can be a function of all of these. The query cost consists of a number of factors like the cost for submitting the query to the site, retrieving the result index page and downloading the actual pages. Cost for submitting a query (C_q) was assumed to be fixed. The cost for downloading the result index page is proportional to the number of matching documents to the query, while the cost (C_d) for downloading a matching document is also fixed. Then the overall cost of query q_i is:

$$Cost(q_i) = C_q + C_r P(q_i) + C_d P(q_i) \quad 3.1$$

To download the maximum number of pages, this Hidden web Crawler considered two main factors:

- The number of new documents that can be obtained from the query q_i and
- The cost of issuing the query q_i .

For example, if two queries, q_i and q_j , incur the same cost, but q_i returns more number of new pages than q_j , q_i is more desirable than q_j . Similarly, if q_i and q_j return the same number of new documents even then q_i is more desirable if q_i incurs less cost than q_j . Based on this observation, the Hidden-Web crawler uses the following efficiency metric to compute the popularity of the query q_i :

$$Efficiency = P_{new}(q_i) / Cost(q_i) \quad 3.2$$

The efficiency of q_i measures how many new documents have been retrieved per unit cost. They compared their adaptive method with two other query selection methods: the random method (queries are randomly selected from a dictionary), and the generic-frequency method (queries are selected from a 5.5-million-web-page corpus based on their decreasing frequencies). The experimental result shows that the adaptive method performs remarkably well in all cases.

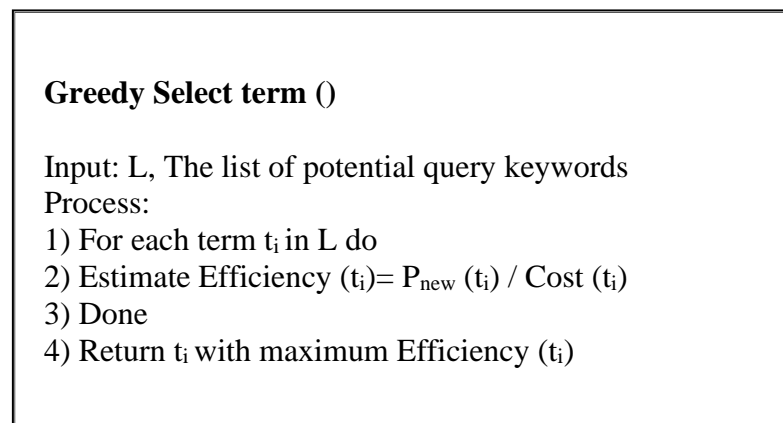


Figure 3.15: Algorithm for selecting the next query term

The best of our policies, the *adaptive* policy could download more than 90% of a Hidden-Web site after issuing approximately 100 queries.

3.5.2.3. AKSHR: A Domain-specific Hidden web crawler.

AKSHR is a domain specific Hidden Web crawler which provides fully automatic techniques to download the search interfaces and matches them by using the DSIM framework.

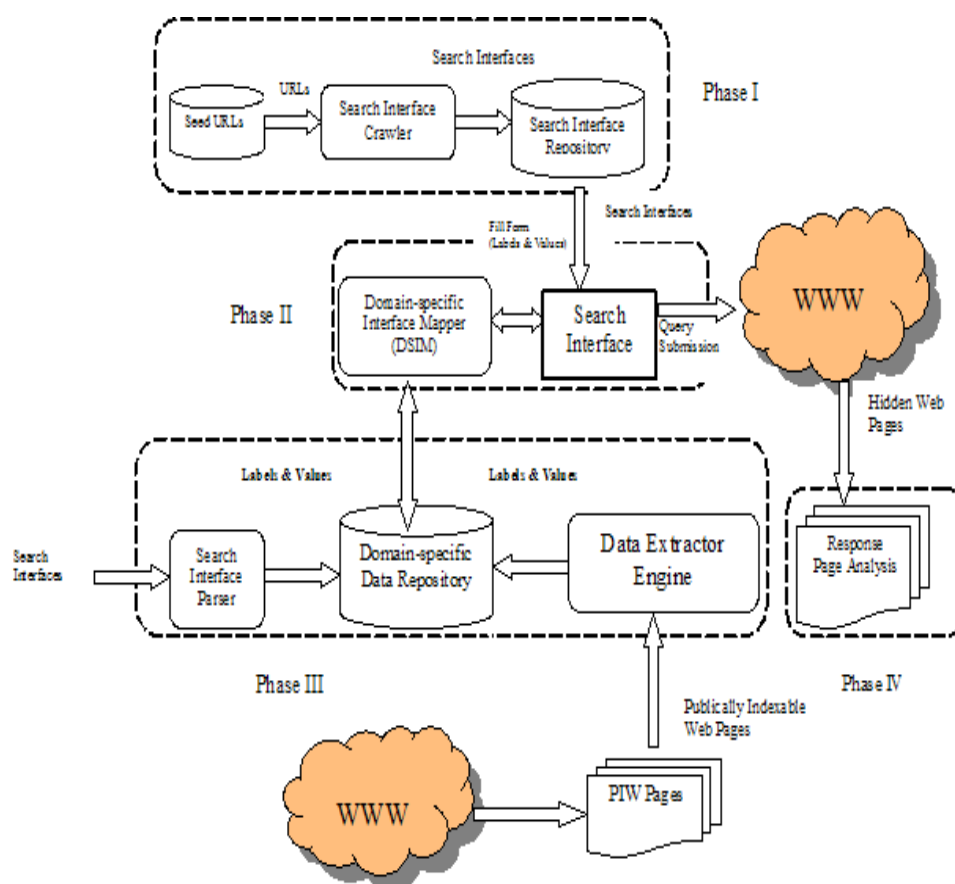


Figure 3.16: Architecture of a Domain-specific Hidden Web Crawler (AKSHR)

The architecture of AKSHR as presented in Figure 3.16, employs a suite of algorithms distributed into the following four phases:

The Phase I of AKSHR uses a Search Interface Crawler [4] which provides a mechanism for automatic extraction of domain-specific search interface by adopting domain-specific-assisted approach for crawling the hidden web. The identified search interfaces are then stored in a search interface repository.

In its phase II, AKSHR uses a component DSIM that identifies the semantic mappings between the attributes of different search interfaces of the same domain i.e. all the interfaces belong to the same domain such as books domain are candidates for mappings. The main inputs to this Interface mapping system are two interfaces A and B comprising of a number of components i.e. $\{n_1, n_2 \dots n_p\}$ and $\{n'_1, n'_2 \dots n'_q\}$ respectively.

In Phase III, The Search Interface Parser extracts the interfaces from the Search Interface Repository and parses them to obtain the structure of a query interface that has been represented as a hierarchical schema. Figure 3.17 shows a typical example of two query interfaces in the books domain and its corresponding hierarchical representation.

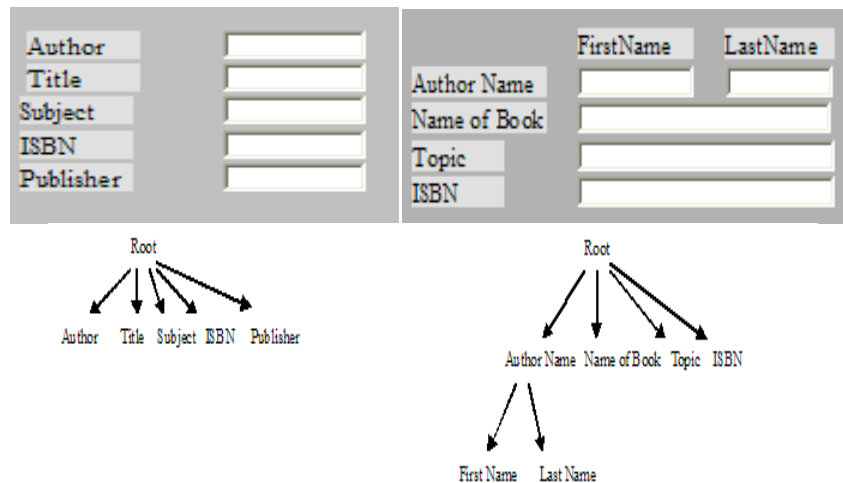


Figure 3.17: Two examples Query Interfaces from Books domain and their Hierarchical representation

DSIM uses a *Search Interface Repository* to store domain-specific search interfaces. It also provides an extensible domain-specific matcher library to support multi-strategy match approach. The multi-strategy match [5] approach uses different matching strategies like *fuzzy matching*, *domain-specific thesaurus* etc that are executed independently. The DSIM also uses a *Mapping Knowledgebase* that stores the

important semantic mappings so that they can be used further when after sometime the search interface repository would be updated.

The AKSHR also attempts to automatically fill forms by extracting labels and their corresponding values with the help of a Data Extractor Engine. The extracted labels and values are used to generate Domain-specific Data Repository for further use.

In its last phase AKSHR uses a response analyzer to distinguish between the response pages containing search results, and pages containing error messages. The pages containing error messages show that no matches were found for the submitted queries whereas the pages containing search results shows that information was found against the submitted queries.

3.6. COMPARISON OF THE VARIOUS HIDDEN WEB CRAWLERS

To achieve a notable progress in this fragment of Hidden Web crawling requires additional efforts for extending the current crawlers. In the next section we provide a comparison of the above discussed crawlers.

Table 3.1: Comparison of various Hidden Web Crawler

Descriptive criteria	Year	Focused Perspective	Database type	Technique	Strength	Limitation
Raghavan et.al.[39]	2001	Depth-Oriented crawler for content extraction	Multi-attribute or structured	1) Text similarity to match fields and domain attributes. 2) Partial page layout and visual adjacency for identifying form elements 3) Hash of visually important parts of the page to detect errors	1) Significant contribution to label matching process 2) Updates the user provided task description by learning information from the successful extracts of crawling.	1) ignores forms with fewer than 3 attributes 2) Require significant human input thus performance highly depends on the quality of input data 3) not scalable to hidden web databases in diversified domains.
Liddle et.al. [44]	2002	Depth-Oriented crawler for content extraction	Multi-attribute or structured	1) Stratified Sampling Method (avoid queries biased toward certain fields) 2) Fields with finite set of values, ignores automatic filling of text field 3) Concatenation of pages connected through navigational elements	1) domain-independent approach 2) accounts for duplicate results identified by computing hash values	1) Do not consider detection of forms inside result pages. 2) Detection of record boundaries and computes hash values for each sentence poses huge resource requirements.
Garvan et.al.	2002	Depth-Oriented	document based	1) use of topically focused queries	1) facilitates design of meta-search engines 2)	1) Query chosen only by using

[61]		crawler for content extraction	or unstructured	2) adaptive query probing	used to categorize hidden web databases	hierarchical categories as in Yahoo! and does not consider flat classification
Bergholz et.al. [45]	2003	Breadth-oriented crawler for resource discovery	unstructured database in a domain	1) domain specific crawling 2) Query probe to recognize and assess the usefulness of the HW resource.	1) Efficient at discovering unstructured hidden web resources as uses the combination of syntactic elements of HTML forms and query probing technique.	1) Only deal with full text search forms. 2) Initialized with pre-classified documents and relevant keywords
Barbosa et.al. [42]	2004	Depth-Oriented crawler for content extraction	document based or unstructured	1) Considers candidate query based on its frequency of appearance in each round	1) Simple and completely automated strategy 2) Automatically creates sufficiently accurate description of document therefore, can be used in other resource discovery systems. 3) Leads to high coverage.	1) No assurance of acquiring new pages 2) ineffective for search interfaces that fix the number of returned results 3) simple approach therefore raises security issues
Ntoullas et.al. [41]	2005	Depth-Oriented crawler for content extraction	document based or unstructured	1) Incremental adaptive method 2) frequency estimation based on already downloaded documents 3) greedy algorithm that tries to maximize the 'potential gain' in every step.	1) Combination of policies (random, generic and adaptive) for choosing appropriate queries. 2) use of multiple frequency estimators - independent and zipf's law based	1) Query distribution does not make sure to adapt to the attribute values set of the database. 2) Memory requirements for calculating potential gain are huge. 3) Assumed constant cost for every query which does not hold in real situations.
Barbosa et.al. [54]	2005	Breadth-oriented crawler for resource discovery	structured & unstructured databases	1) Link classifier to focus search on a specific topic 2) use of a stopping criteria to avoid unproductive searches	1) Highly efficient in retrieving searchable forms focused for a particular topic	1) Manually selecting a representative training set is difficult so creating the link classifier is time consuming
Alvarez et.al. [19]	2006			1) set of domain definitions each one of which describes a data-collection task 2) use of heuristics to automatically identify relevant query forms	1) System can be extended for discovering relevant resources. 2) Handles client side as well as server side hidden Web 3) Experimentally proved effective for collecting data.	1) No defined threshold for associating form elements and attributes in the domain definitions 2) hypothetical assumption of having at least one label associated with every form element which does not hold true for most of the bounded form

						elements (drop down boxes)
Ping Wu et.al. [64]	2006	Depth-Oriented crawler for content extraction	Multi-attribute or structured	1) Models each structured database as a distinct attribute - value graph 2) Set the graph to crawl the database (set-covering problem)	1) issues only meaningful queries as tuned with domain knowledge 2) overcomes limitation of greedy methods	1) Query results in each round must be added to the graph thus involves huge cost of resources
Barbosa et.al. [55]	2007	Breadth-oriented crawler for resource discovery	unstructured databases	1) Greedy algo derived by the weights associated to keywords in the collected data 2) Issue queries using dummy words to detect error pages	1) Improved harvest rates as crawl progresses 2) retrieves homogeneous set of forms 3) Automated and adaptive thus eliminates any bias arising out of learning process.	1) configuring the crawler to start initially needs more effort than manually configured crawlers 2) works only for Single keyword-based queries
Madhavan et.al. [43]	2008	Depth-Oriented crawler for content extraction	Multi-attribute or structured	1) Evaluate the query templates by defining the informativeness test.	1) efficiently navigates the search space of possible input combinations	1) No consideration to the efficiency of deep web crawling
Komal Bhatia et.al. [60]	2010	Depth-Oriented crawler for content extraction	Multi-attribute or structured	1) Domain Specific Interface Mapper to create unified query interfaces for a domain 2) calculation of re-visit frequency based on probability of change of web page	1) Multi-strategy interface matching 2) use of mapping knowledge base to avoid repetition for minimizing the mapping effort 3) Enhances the scope of developing a specialized search engine for the Hidden Web.	1) Indexing technique was not specified for storing pages in the repository 2) Defined the performance only for crawling while the efficiency of schema matching and merging procedures over variety of query interfaces has not been quantified.

3.7. PROBLEM STATEMENT

A critical look at the techniques exploited by the available crawlers for the Surface and the Hidden Web indicate the following issues that need to be addressed.

- **Scale of the Hidden Web:** Research conducted in March 2000 by brightplanet.com shows that the hidden Web contains a much bigger amount of data than the Surface Web. The Hidden Web is big and getting bigger and as the volume of information in the hidden-web grows, there is increased importance to use parallel crawlers. A hidden-web crawler needs to be developed within view to resolve the problem faced by a single process crawler.

- Identification of Relevant Search Forms:** The tremendous size and heterogeneity of the Hidden Web makes comprehensive coverage very difficult and possibly less useful than domain specific crawling. Searching irrelevant web pages is a major cause for the users to waste time on the Web. Thus, the system must identify the few databases out of the huge number available that seem to be the most relevant so that the search can be directed to only those databases. Hence, the search forms must first be pruned for relevance prior to accessing their contents so that the crawler can benefit from the knowledge of the different application domains.
- Automatic processing of search forms:** Most of the Hidden Web is made up of the content of hundreds of thousands of specialized searchable databases. Information stored in these databases is accessible only by filling out a form on a web page and submitting it to the databases. Conventional search engines do not attempt to fill out forms and index the resulting pages. Because of the lack of knowledge of the underlying database schema, it is difficult to generate form assignments that are guaranteed to yield information-rich resulting pages;
- Classification of Web Pages:** The information in the Hidden Web is available in heterogeneous databases from different domains. Also, the search forms to these hidden web databases, even those belonging to the same domain, are very different, and therefore it is required to design an internal representation of these searchable forms that can help in identifying the relevant domain of the search forms and the available information needed to automatically fill forms is a real challenge.
- Synchronizing Parallel tasks/processes:** Overlap problem occurs when multiple crawlers running in parallel download the same web document multiple times due to the reason that one web crawler may not be aware of another having already downloaded the page. Also many organizations mirror their documents on multiple servers to avoid arbitrary server corruption. In such a situation, crawlers may also unnecessarily download many copies of the same document. Hence, it would be very important to minimize such multiple downloads to save network bandwidth and increase the crawler's effectiveness. Thus, the processes must be coordinated to minimize overlap.

- **Reduced Network Bandwidth:** In order to minimize overlap and maintain the quality of downloaded web pages, the coordination between individual crawling processes needs communication which consumes network bandwidth. So, an important objective is to minimize communication overhead and thus the network bandwidth consumption while maintaining the quality of crawling.
- **Scalability:** There exists a variety of Hidden web sources that provide information on the multitude of topics/domains. The continuous growth of information on the WWW and hence the domain specific information with ever increasing number of domain areas pose a challenge to crawler's performance. The crawl of the portion of the web for a particular domain must be completed within the expected time. This download rate of the crawler is limited by the underlying resources such as the number of crawling processes. There is a need to design a crawler that scales its performance according to the increase in the information on the WWW and number of domains.

The above mentioned objectives have been addressed and resolved in the subsequent chapters.

CHAPTER 4.

DESIGN OF A PARALLEL HIDDEN WEB CRAWLER

4.1. INTRODUCTION

Much of the Web's usability depends on the efficiency of the search engines and their crawlers. The traditional crawler works by taking a URL from the URL frontier, downloading the web page associated with that URL, extracting hyperlinks that are embedded in the downloaded web page and adding extracted hyperlinks to the URL frontier to keep the process going. It traverses the web by following the hyperlinks from page to page and downloading the documents. But this results in gathering only those web pages that are interconnected via the link structure, ignoring the Hidden web contents as no link is available for referring to its contents. Thus, it is not possible for the general purpose crawler to visit the entire web by following hyperlinks.

The contents in the Hidden Web reside in searchable databases and thus can only be accessed by raising the queries at the interface offered by the database. The pages in the Hidden web do not exist until they are created dynamically in response as a result to some direct query by the user. These dynamic web pages could not be efficiently retrieved and collected by the traditional crawlers for inclusion in the search engines' index leading to a large volume of undiscovered Hidden Web contents. Hence, there is a need to design and develop a crawler that can find and crawl the Hidden Web contents.

Moreover, with an ever increasing size of the Hidden Web there is a need to design a scalable and extensible Hidden Web Crawler. The hidden web crawler must be able to handle the continuous growth of contents in the Hidden Web and must be extensible in the sense that third party modules can be added as and when required.

Therefore, in this thesis a design of a Parallel Hidden Web Crawler has been proposed that solves the above mentioned issues. The proposed work makes the following contribution:

- (i) It proposes a novel approach to crawl the Hidden Web resources by employing a number of parallel crawling processes (as shown in Figure 4.1). Crawling the Hidden Web contents in parallel, offers a scalable solution.

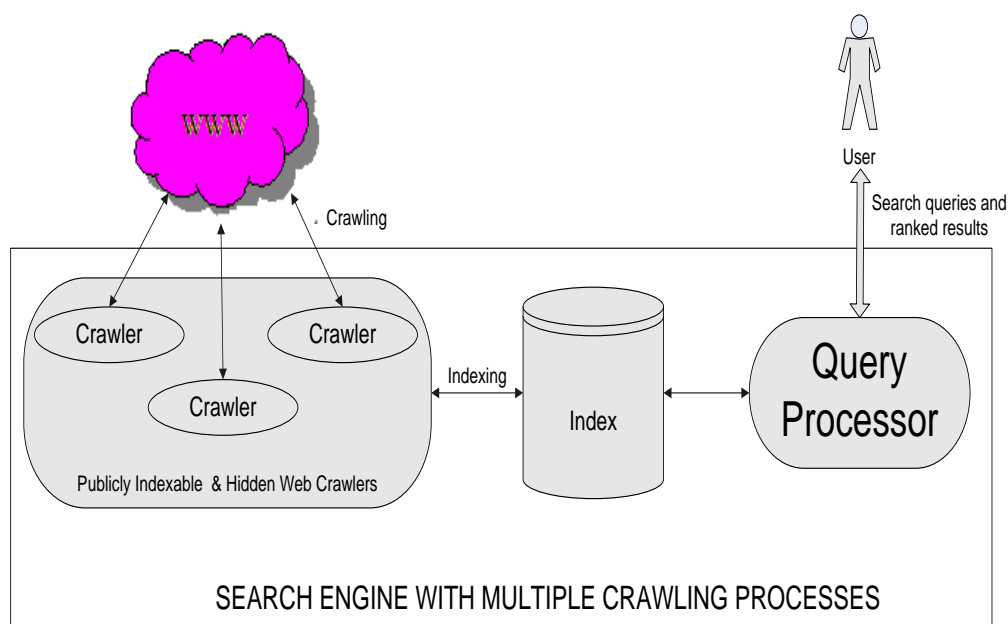


Figure 4.1: A Search Engine with several crawling threads to achieve parallelism.

- (ii) A URL Scheduler has been used to organize the seed URLs into several Domain-Specific Priority Queues for efficient crawling by the parallel processes.
- (iii) To improve the quality of the collection downloaded by the hidden web crawler, a URL Ranker has been used to rank the various URLs (that will be discovered during the process) according to their relevance in each domain.
- (iv) To automatically process the form pages (by the parallel form processing elements), the surface web pages are segregated from the downloaded web pages and classified according to their domains to create various domain-definitions and Domain-Specific Data Repositories that helps in filling the search forms that acts as an entry point for the Hidden Web resources.
- (v) For the proposed Hidden Web crawler to work equally well in both the generic and domain-specific modes, the Form Analyzer analyzes each search form for organizing them into different Domain-Specific Search Interface Repositories. When behaving as a generic crawler, the search forms from these repositories are processed in parallel by the various Form Processing Elements.
- (vi) The proposed crawler enforces parallelism at two different levels to

- a) At the level of domains, to reduce the overhead of synchronizing the various parallel threads.
 - b) At the level of processing search forms, to minimize the overlap among the downloaded resources.
- (vii) To minimize the consumption of the network bandwidth, the Query Ranker suggests optimal queries by ranking them according to their behavior in the previous crawls. An optimal query helps to reduce the chance of retrieving error pages in response to a valid search form submission to get a faster coverage of the Hidden Web databases. Since, only optimal queries will be chosen for filling the search forms, significant reduction is achieved in the number of queries that might otherwise be used by any arbitrary approach for filling the search forms. A metric *Reduction Efficiency* has been proposed to justify the proposed approach.

The next section discusses the proposed design of the Parallel Hidden Web Crawler in detail.

4.1.1. PROPOSED DESIGN OF THE PARALLEL HIDDEN WEB CRAWLER

In order to maximize the benefits, the working of the proposed Parallel Hidden Web Crawler has been divided into six phases where each phase is structurally well defined and functionally supports the others. The detailed design of the proposed Hidden Web Crawler has been shown in Figure 4.2. The brief functionality of each phase is as follows:

1. The first phase is used to initialize the crawler by choosing a seed set of URLs in each domain. These seeds have been stored in the URL pools respective to its domain. In order to maximize the benefits achievable from a parallel crawler, these Domain-Specific URL Pools must be processed in parallel by the crawler. Moreover, the crawler should also aim to download high quality pages first. Therefore, a URL Scheduler is used that is responsible for creating prioritized URL queues corresponding to each Domain-Specific Seed URL Pool. The URLs from each of these priority queues acts as input to the next phase of the crawler.

Figure 4.2: Architecture of the proposed Parallel crawler for the Hidden Web.

2. The second phase of the crawler takes the URLs from each of the Domain-Specific Prioritized URL Queues, downloads the associated web pages. The phase also analyzes and distinguishes between the two types of web pages, the ones that can be publicly indexed by crawlers (typically called Publicly Indexable Web pages) and those that contain a search form (called form page or a hidden web page). The second phase also helps the crawler to rank the newly discovered URLs as per their relevance in each domain so that important pages get downloaded earlier during the crawl.
3. Third phase uses the PIWP given by the second phase to create the Domain-Specific Data Repositories that are further used by Phase fifth to fill the search forms in order to download the Hidden Web resources. So, a novel approach to organize and classify the downloaded collection of web pages according to their domains like Auto, Books, Food, Travel, etc. is being proposed.
4. The form pages discovered in second Phase are passed onto the fourth Phase where they are analyzed by the Form Analyzer and stored in various Domain-Specific Search Interface Repositories that further facilitate the process of automatically filling the search forms in phase fifth.
5. The fifth phase of the crawler is responsible for an even distribution of search forms among the parallel processes for processing them efficiently. It uses a Search Interface Distributor that is responsible for filling search forms of each domain with the help of the various Form Processing Elements.
6. In the last phase, the various response pages thus retrieved by the crawler in its previous phases are analyzed to get the useful pages and sift away the pages that are containing the error messages. In this phase a novel technique for discovering optimal queries to get an optimal outcome has been introduced.

The following sections describe the functional detail of each phase of the proposed Parallel Hidden Web Crawler.

4.2. PHASE I: CREATING A URL FRONTIER FOR PARALLEL ACCESS.

This phase is the most basic and important to the entire process and is used to initialize the seed URLs to be further used by phase II of the proposed crawler. In a parallel crawler, the seed URLs must be overseen by all the parallel processes to achieve scalability and efficiency. This is done by partitioning and organizing the seed URLs into different Domain-Specific URL Pools that can be accessed in parallel by the various parallel components used by the crawler [19, 20]. For example, the URLs like www.makemytrip.com , <https://placetoseeindiahi.wordpress.com> etc.

which provide information in Travel domain are included in the URL pool meant to store the URLs for Travel domain. Each such group of URLs that belongs to a common domain is therefore referred to as Domain-Specific URL Pool.

In other words, the proposed crawler follows a domain specific approach to initialize the seed URLs for crawling where each URL pool stores the URLs from a different domain like Books, Travel, Auto, Real Estate, Food etc. This helps the proposed crawler to work well in the domain specific mode as per requirement.

To gather the seed URLs for the Domain-Specific URL Pools, two different methods are used by the proposed crawler. Initially, the proposed crawler takes advantage of the classification hierarchy offered by DMOZ since such a directory includes a collection of Websites selected and manually classified by Open Directory volunteer editors. For each category in the hierarchy the system supports the retrieval of top N relevant URLs. Those top N URLs will serve as starting points for the crawler. Later as the crawler progresses, all the URLs that are gathered during crawling are added to these different Domain-Specific URL pools with the help of other components of the crawler.

Also, once each of these Domain-Specific URL pools is initialized with some URLs, the URLs in these pools need to be prioritized so as to enable the crawler to always populate its document collection with some finite number of top relevant pages in that domain [24, 73]. So, the URL Scheduler constructs a prioritized URL queue for each Domain-Specific URL pool based on the relevance score of the URLs.

The structure of the Prioritized URL queues that has been used in the proposed approach is shown in Figure 4.3. The domain-specific prioritized URL queue contains the relevance scores (R-Score) in the decreasing order, computed by the URL Ranker

(used in phase II) and with each R-Score a list of URLs having that score has been appended.

Figure 4.3: Structure of the Prioritized URL Queue for a domain.

The priorities have been indicated by the positional markers 1 to m in the queue. The highest priority URLs are contained in the URL list appended at position 1 whereas the least priority URLs are the ones in the URL list at position m . The URLs with the maximum value of R-score is assigned the highest priority. So, for constructing the prioritized URL queues, the URLs in each of the domain- specific URL pools are sorted in accordance with the m priority scores.

As initially, the URLs have been taken from a classification hierarchy, so the R-Score value for the seed URLs is same as the order of their occurrence in the listing provided by the hierarchy and thus a different R-Score value is used for each URL. Thus, the URL Scheduler is responsible for maintaining as many prioritized URL frontier queues as there are Domain-Specific URL pools.

4.3. PHASE II: WEB PAGE COLLECTION AND ANALYSIS

This phase of the proposed system is responsible for extracting the URLs from the domain-specific Prioritized URL Queues according to their domain and allocates them to the Multi-threaded Document Downloader to download the web page associated to that URL. This web page is further analyzed by the Web Page Analyzer that separates the PIWP from the web pages containing search forms. Besides this, it also extracts the new URLs from the downloaded web pages for adding them to the URL pool after a R-Score is being assigned to the URL by the URL Ranker. The following functional components have been used in this phase:

4.3.1. URL ALLOCATOR

Phase I generates multiple Prioritized URL queues, one for each domain of consideration. The URL Allocator is responsible for extracting the URLs from these multiple Prioritized URL Queues and assigning them to the multiple threads of the Document Downloader for downloading the associated web pages.

The URL Allocator extracts URLs at a time from each of the Domain-Specific Prioritized URL Queue and allocates them to the various threads of the Multi-threaded Document Downloader to enable parallel downloading of web pages in each domain. In order to choose a URL from a Domain-Specific Prioritized URL Queue, the URL Allocator selects the URL from the highest priority URL list in the prioritized URL queue. The URL list with the highest priority includes all those URLs that have the maximum value of R-Score for that domain.

The first URL in a domain is the one which occurs in the URL list stored at the first position in the Prioritized URL Queue respective to that domain. It processes all the URLs from the URL list having the highest priority (at first position) in a sequential manner by following a first come first serve approach.

Figure 4.4: Example of a domain-specific priority queue

For example, consider the prioritized URL Queue as in Figure 4.4. Where m different priority lists of URLs have been formed by the URL Scheduler for a domain. These m URL list are formed by using the different values of R-Score computed by the URL Ranker for each of the different URLs included in the respective Domain-Specific URL Pool. The URL Allocator starts by taking the URL list at the first position headed by the relevance score $R\text{-Score}_1$ as shown in Figure 4.4. It extracts the first URL, denoted by URL_{11} in this list, allocates it to a thread of the Document Downloader for downloading the web page associated with this URL_{11} , takes the next URL_{12} in the same list allocates it to a thread of the Multi-threaded Document Downloader and proceeds in a similar fashion for all the k URLs included in this list. After allocating all the URLs in a URL list, the URL Allocator extracts the URLs from the list occupying the next position $R\text{-Score}_2$ in a similar way. Thus, the URL Allocator extracts the other URLs from the other lists (those having lesser priority) stored in the Domain-Specific Priority Queue in the same manner. Thus, it performs the same task for each URL list at the various m positions in the decreasing order of their priorities denoted as $R\text{-Score}_1 > R\text{-Score}_2 > R\text{-Score}_3 > R\text{-Score}_4 > \dots > R\text{-Score}_m$. The working algorithm of the URL Allocator is given in Figure 4.5.

Figure 4.5: URL Allocator Algorithm

This dynamic assignment of URLs avoids dedicating all the threads to download web pages from a common domain in case when other prioritized URL queues contain URLs for processing. Moreover, it also allows sharing the bandwidth when all the URLs from all the Domain-Specific Prioritized URL Queues except one have been processed.

4.3.2. A MULTI-THREADED DOCUMENT DOWNLOADER

The Multi-threaded Document Downloader is a high performance asynchronous HTTP client capable of downloading several web pages in parallel, initiates a number of downloader instances equal to the number of URLs received (for downloading) from the URL Allocator by processing the different prioritized URL queues. The instances download the pages in parallel from the multiple web servers and pass them to the Web Page Analyzer for further processing. The functionality of the multithreaded document loader can be explained with the help of the algorithm in Figure 4.6.

Figure 4.6: Multi-threaded Document downloader Algorithm

If any information about the domain of the just downloaded Web page has been made available in the due course, the Multi-threaded Document Downloader also passes it to the Web page Analyzer along with the webpage. This has been indicated by the arrow labeled as an ordered pair (Webpage, domain) in the architecture provided in Figure 4.2. For example: If the web page W is a page from TRAVEL domain, then (W, Travel) is passed to the Web page analyzer. But, if no information about the domain of the webpage is acquired by the loader, then a NIL values is simply passed for the domain of the web page which can be represented by (W, NIL).

4.3.3. WEB PAGE ANALYZER

The web pages downloaded by the Multi-threaded Document Downloader are passed to the Web Page Analyzer. These web pages may contain search forms. Since these search forms act as entry points for the vast information hidden behind them, therefore, the proposed system must scan the downloaded web pages to differentiate between the web pages having forms and those not containing them. Thus, each such downloaded Web page is given as input to one of the various threads of the Web page

Analyzer, which consists of two components: a Parser and a Page Type identifier. The Parser extracts the useful content from the HTML tag structure and the Page Type Identifier identifies and differentiates between the Publicly Indexable Web pages (PIWP) and the Form pages (FP).

Figure 4.7: Example of a web page having new or child URLs

The information that is stored in the URL Database corresponding to this downloaded web page and the new URL is represented by the following data structure in the URL Database.

Now, in order to identify the form pages, the Page Type Identifier searches the <FORM> tag in the HTML code of the downloaded web page. If the page does not contain the <FORM> tag, then it is assumed that the page can't act as an entry point for the Hidden Web. Thus, these types of web pages are categorized as the Publicly Indexable Web pages or the PIWP. However, if the page contains the <FORM> tag, then this may be treated as entry point for the Hidden Web. This can be represented by the algorithm in Figure 4.8.

Figure 4.8: The Page Type Identifier Algorithm.

Thus, the Page Type Identifier helps in segregating the Hidden Web Pages from the PIWPs.

4.3.4. URL RANKER

The URL Ranker is an important component of the proposed parallel hidden web crawler as it ranks the URLs that would be downloaded by the Multi-threaded Document Downloader. Ranking the URLs according to their relevance is necessary as it will be helpful for the Multi-threaded Document Downloader to download good and important pages first.

The URL Ranker basically takes the URLs from the URL Database and ranks them according to their relevance. Ranking the URLs is important so that relevant URLs can be considered on priority basis than the other documents.

The URL Ranker thus performs the ranking of each URL to predict its relevance and importance:

1. Among all the URLs of a single domain in a Domain-Specific URL pool.
2. And across the URLs of all domains in different Domain-Specific URL pools.

Before calculating the relevance score of any URL, the URL Ranker first restores

www.ricksteves.com then an absolute URL has been generated. For example:

```
<a href="/europe">Explore Europe</a>
```

Points to the URL **[http://www.ricksteves .com/Europe](http://www.ricksteves.com/Europe)**.

The URL Ranker fetches the URLs from the URL Database and computes their relevance score so that URLs can be prioritized for further processing. This relevance score of a URL in domain D, has been represented by R-Score (URL, D).

It is implicit that the relevance of the new URL is not known thus, the relevance score of the new URL has been computed by the URL Ranker prior to fetching the web page associated with that URL. This task of computing the R-Score is simplified by assuming the hyperlinked structure of the Web and the fact that web pages are significantly more likely to link to pages that are related to the domain of the containing web page.

In order to compute the R-Score for a new URL, the URL Ranker considers the relevance score of the parent page of that URL, where the parent page is said to be the page that contains the new URL. Also, in that case the new URL is termed as the child URL. This has been done based on the assumption that it has a higher probability that if a page belongs to a particular domain then the links embedded in

that page or in the running text / paragraph of that page may belong to the same domain as of the domain of its parent page.

The above assumption leads to the calculation of the relevance score by including two values: *a domain score* that is calculated based on the terms that surround the URL in the containing paragraph and a Link Score that accounts for the relevance of the discovered URL to a domain based on the domain of its parent pages. Thus, the relevance score of the URL in domain D denoted as R-Score (URL, D), can be computed by:

$$\text{Type equation here.} \quad 4.1$$

The domain_score (URL, D) for a particular URL belonging to a domain D is computed with the help of the domain definitions (given in Appendix A) which are generated by the Page Classifier in phase III whereas the link_score (URL, D) for a URL belonging to domain D is computed with the help of domain information of its parent URL. The following sub-sections discuss the calculation of the domain and Link Scores in detail.

4.3.4.1. Domain Score

The Domain Score is used to quantify the effect of the terms contained in the running text or the paragraph of that new URL. So, its value is computed with the help of the various terms that surround the new URL and the domain definitions created by the Page Classifier in phase III.

In order to compute the domain_score for a particular URL, the contents of its parent page (corresponding to the parent URL) has been checked and the paragraph in which the new URL exist has been analyzed. Thus, for computing the value of the domain_score, the following steps have been followed:

- 1) *Running text identification:* For appropriate computation of the domain_score, the parent page has been first segmented into blocks or paragraphs so that the URLs embedded as hyperlinks in any of these blocks can be analyzed and evaluated independent of the other hyperlinks (child URLs) that exist in the parent web page. Each such block will have features like text, images, applets, tables that help in providing the best possible description of the URL. The

paragraphs can easily be identified based on <p> tags associated with them in the HTML code of the web document.

- 2) Tokenization: As a next step, for each such paragraph that comprises a child URL, the various tokens or terms occurring in that paragraph has been identified.
- 3) Stop-word Removal: Based on a list of stopwords (refer Appendix B), the terms or tokens that are having no meaning are eliminated from the list generated in step 2 above. For example: is, are, of, to, for etc. are some of the stopwords.
- 4) Domain Score computation: The domain_score is computed on the basis of probability computation. From the remaining set of terms extracted from the paragraph, the number of terms occurring from each of the domains is identified. This is done by matching each of the extracted term against the key terms included in the Domain Definitions of each domain. The counts are then further used to compute the domain_score of the new/child URL which is given by the probability of the new/child URL belonging to a domain D. Thus,

$$Domain_score (URL, D) = \frac{\text{Number of terms from domain D}}{\text{Total terms extracted pages}} \quad 4.2$$

This domain_score is further used to compute the rank of the new/child URL on the basis of its R-Score.

Consider the web page (source: www.placetoseeindelihi.wordpress.com) shown in Figure 4.9. The page provides a brief detail on the various places that one can visit in Delhi and belongs to the Travel domain.

Figure 4.9: Example web page containing hyperlinks (Source: www.placetoseeindelhi.wordpress.com)

The source code view of this parent web page is shown in Figure 4.10 and refers to a hyperlink places to visit near Delhi (illustrated by bold white text in the paragraph in black). The parser extracts the child URL “<http://www.theweekendleader.com/Travel/1913/must-in-delhi.html>” embedded as this hyperlink in the parent page.

Figure 4.10: HTML tag structure of the web page in Figure 4.9.

Now, in order to calculate the domain-score of this new or child URL “<http://www.theweekendleader.com/Travel/1913/must-in-delhi.html>” the running paragraph or textual content is separated for tokenization and identification of the terms in the comprising paragraph. Stopwords are then removed from this obtained list of tokens to extract the terms that seem useful and relevant for finding the domain_score.

Thus, the following terms have been extracted: *Delhi, loaded, number, tourist, attractions, sightseeing, location, luxurious, hotels, amenities, delight, travelers, Qutub, Minar, Jama, Masjid, India, Gate, Salimgarh, fort, places, visit, near, Agra, Chandigarh, Mathura, Bharatpur, Corbett, Mandawa, magical, land, destination, North, vibrant, culture, bustling, stress, shopping, centers, ancient, monuments, perfect, combination, modernity, ethnicity, Trip, excellent, way, spend, vacation, prime, enjoying, modern.*

As a next step, these terms have been matched against the various domain definitions (generated in Phase III and given in refer appendix A) to find the probability in each domain that the new URL might belong to. For computing the probability and thus the Domain Score of the new URL in each domain, a count of terms occurring as keywords in each of the domain is recorded. It has been found that the following seven terms tourist, attraction, hotel, travel, visit, trip, culture (out of total 53 terms) occur in the domain definition of the Travel domain giving a value of

$$\text{domain_score}(\text{URL}, \text{Travel}) = \frac{7}{53} = 0.13 \quad 4.3$$

If a term occurs only once in the paragraph but is available in the domain definitions of more than one domain, then it adds a value to the score calculated for each such domain. For example: the term ‘travel’ also occurs in the domain definition for the Entertainment domain along with other terms ‘trip’, ‘adventure’, ‘stress’, thus giving a value of

$$\text{domain_score}(\text{URL}, \text{Entertainment}) = \frac{4}{53} = .07 \quad 4.4$$

for the Entertainment domain.

The terms like Agra, ancient, number, loaded etc. that do not occur in a domain definition, adds a zero value to the count and thus do not modify the value of Domain Score for the URL in that domain. Thus the value of Domain Score of the new URL in all other domains equals zero i.e

$$\text{domain_score}(\text{URL}, \text{Food}) = \text{domain_score}(\text{URL}, \text{Sports}) = 0 \quad 4.5$$

Similarly, if a term occurs more than once but belongs to only one domain definition, then it adds a value that equals the number of repeated occurrences of the term to the score of the URL in that domain. Also, if a term occurs more than once in the paragraph and also belongs to many domain definitions stored in the Domain-Specific Data Repositories, then it adds a value that equals the number of repeated occurrences of the term to the score of the URL in each of the domains. Thus, the sum of the counts of the occurrences of the terms that belongs to a domain and surround a given discovered URL specify the score of the URL in that domain or the Domain Score of the URL.

For the cases where the same URL appears in more than one paragraph in the parent page, then the Domain Score of the URL is calculated for each of the paragraph block separately and respective Domain Scores obtained for each paragraph are summed up to find the overall relevance of the URL in each of the various domains.

4.3.4.2. Link Score

The Link Score to predict the relevance of the new URL in the domain has been computed with the help of the domain information of its parent pages. As already discussed earlier, the URL Database contains the new/child URL that is extracted from the downloaded web page, the URL of its parent page and the domain of that parent URL. Thus, it has been interpreted that the probability of the new URL as belonging to a domain of its immediate back-link is higher than the probability of it belonging to other domains.

Consider for example, the new URL that is included in the URL Database and when the URL is searched for its parent pages, six different parent pages P_1 , P_2 , P_3 , P_4 , P_5 and P_6 have been identified. So, the new URL has a total of 6 back-links P_1 , P_2 , P_3 , P_4 , P_5 and P_6 and the domain information that is stored in the URL Database for each of these six web pages is as Travel, Entertainment, Sports, Travel, Travel & Entertainment respectively. This data has been summarized as in Table 4.1.

Table 4.1: Extracted Back-links for the child URL and their associated domains.

New/child URL	Immediate back-link →	P_1	P_2	P_3	P_4	P_5	P_6

	Domain of the immediate back-link →	Travel	Enter- tainment	Sports	Travel	Travel	Enter- tainment
--	--	--------	--------------------	--------	--------	--------	--------------------

The link_score of the new URL in each of the domains like Travel, Sports, Entertainment is then calculated by finding the ratio of the number of parent pages in that domain and the total number of parent pages. Thus, the link_score of the new URL in TRAVEL domain is calculated as:

$$\text{Link_score (URL, Travel)} = \frac{\text{Number of parent pages in travel domain}}{\text{Total number of parent pages}} = \frac{3}{6} = 0.5 \quad 4.6$$

Similarly,

$$\text{Link_score (URL, Entertainment)} = \frac{2}{6} = 0.33 \quad 4.7$$

$$\text{Link_score (URL, Sports)} = \frac{1}{6} = 0.16 \text{ \& Link_score (URL, Food)} = 0 \quad 4.8$$

The domain_score and link_score for the new URL are used to calculate the R-Score of the URL for different domains as per the equation 4.1 the calculation of the R-score for the new/child URL is shown as an example in Table 4.2.

Table 4.2: Relevance Score Calculation for new/child URL.

It may be observed that the Link Score considers the relevance of all the parent pages to the URL as the hyperlink is a reference to a child web page from all the parent pages that contain it.

Thus the discovered URL will be added to the URL pool for Travel, Sports and Entertainment domains with the respective R-Score of 0.63, 0.16 and 0.40.

Calculating the rank of each URL within a domain will help in focusing the crawl to the most popular & relevant links in a domain.

URL Ranker after calculating the relevance scores adds the discovered URLs to its respective Domain-Specific URL pool. Thus, the URL Ranker finds the relevance of a new URL in each domain based on its Domain Scores and Link Scores. The next phase discusses the creation of the various Domain-Specific Data Repositories that is further used for filling the search forms later.

4.4. PHASE III: CREATION AND MAINTENANCE OF DOMAIN-SPECIFIC DATA REPOSITORIES

The aim of the Phase III is to create the domain-specific data repositories that are further used to fill the search interfaces in order to download the Hidden Web contents. This phase takes the PIWP and the domain of the PIWP if any, in the (PIWP, Domain) format and generates the Domain-Specific Data Repository with the help of the Page Classifier and the Page Content Extractor.

The second phase of the crawler makes a distinction between the PIWP and the form pages based on the presence of the <FORM> tag in the HTML web page. It then passes the PIWPs to the Phase III that analyzes each PIWP to classify them into different domains (Books, Entertainment, Food, Real Estate, Sports & Travel) by using the Page Classifier. After classification, domain-specific databases for each specified domain is created with the help of the Page Content Extractor.

The **Page Classifier (PC)** scans the PIWPs to create various *Domain-Specific page repositories (DSPR)* where the pages have been grouped and organized as per their domains. The **Page Content Extractor (PCE)** then fetches the pages from these repositories to facilitate the creation of the *domain-specific data repositories (DSDR)*. The Page Classifier and the Page Content Extractor collectively are responsible for storing the web pages and other useful data organized according to their domains. The framework in Figure 4.11 shows the functionality of Phase III used to generate the Domain-Specific Data Repository. integrated work of the page classifier and the page content extractor in creating the Domain-Specific Data Repositories.

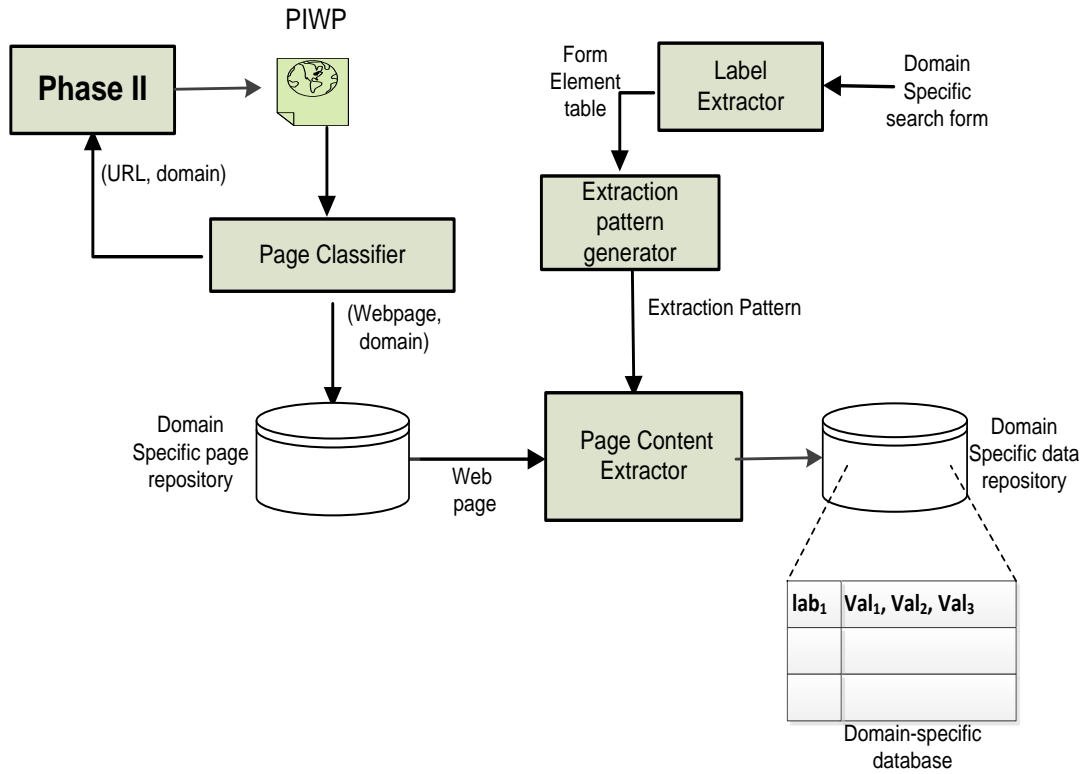


Figure 4.11: Integration of the Page classifier and the Page content Extractor for the creation of Domain-Specific data repository.

The Page Classifier also sends this information about the domain of a web page to Phase II for ranking the URLs in domain. The information is passed to the URL Ranker of Phase II in the form of pairs $\langle \text{URL of the web page}, \text{Domain name} \rangle$.

The working of the Page Classifier and the Page content extractor is discussed below.

4.4.1. PAGE CLASSIFIER

Page Classifier is a very important component of the Phase III as it is responsible for creating and maintaining the various Domain-Specific Page Repositories. These repositories help the proposed Hidden Web crawler (in the later stage) to fill the search forms to retrieve the hidden-web databases. In order to create these different repositories, the Page Classifier must find the domain of each and every web page it comes across. Though this information may be made implicitly available to the Page classifier at certain times through phase II of the crawler, the case may not be always true. In cases where the domain of the web page is not made available to the page classifier, it processes the web page to predict its domain. The architecture of the Page Classifier is shown in Figure 4.12.

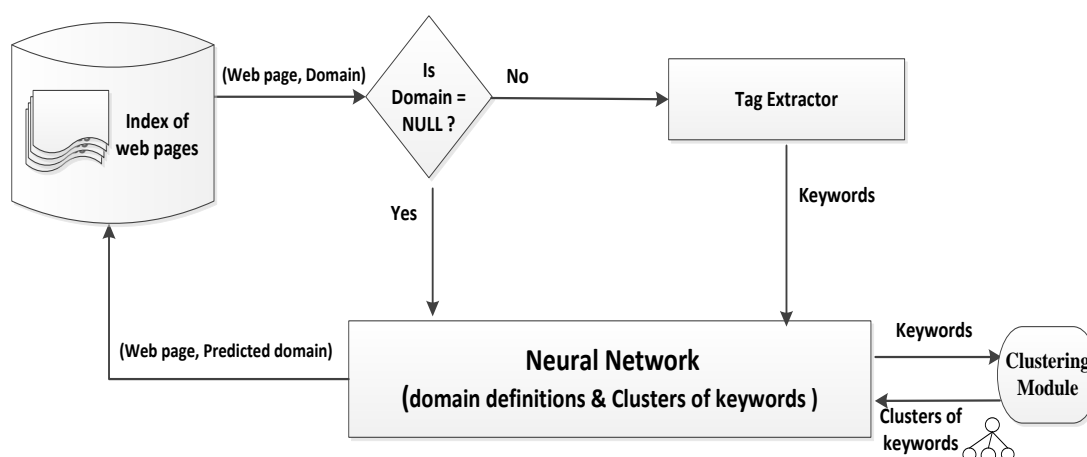


Figure 4.12: The proposed system for domain identification and page classification.

The architecture of the Page Classifier consists of the following components:

- Index of PIWPs
- Tag Extractor
- Clustering Module
- Domain-specific Repositories
- Neural Network model classifier

The Page Classifier is basically used to identify the domain of all the web pages that are stored in Index of PIWPs. As the domain of every web page is not known, therefore, first the domain of the web page is checked. If it is available then the Tag Extractor is used to extract the keywords from that page and these keywords are used to train the Neural Network and creating the Domain Definitions. But if no information is associated with the web page the trained Neural Network is used to find out the domain of the web page. After identifying the domain, the web page is again stored in the index of PIWP. In this manner, after some time the domain of most of (all) the web pages is available and these pages are further stored in their corresponding Domain-Specific Page Repositories. For example, if the domain of the web page is identified as Travel domain then this page is stored in the Domain-Specific Page Repository related to Travel domain. The trained neural Network is used to identify or predict the domain of a web page.

The proposed approach classifies the web pages by using an artificial neural network (ANN) which works on two-step framework composed of many simple elements operating in parallel so that a particular input leads to a specific target output. In order to predict the relationship between inputs and outputs, the artificial neural networks are initially trained by observing the behavior in some exemplary input data set. As a next step, the trained neural network can then be used later for predicting the behavior of any new data set that is given as input to it. The typical layout of any neural network is as given below in Figure 4.13

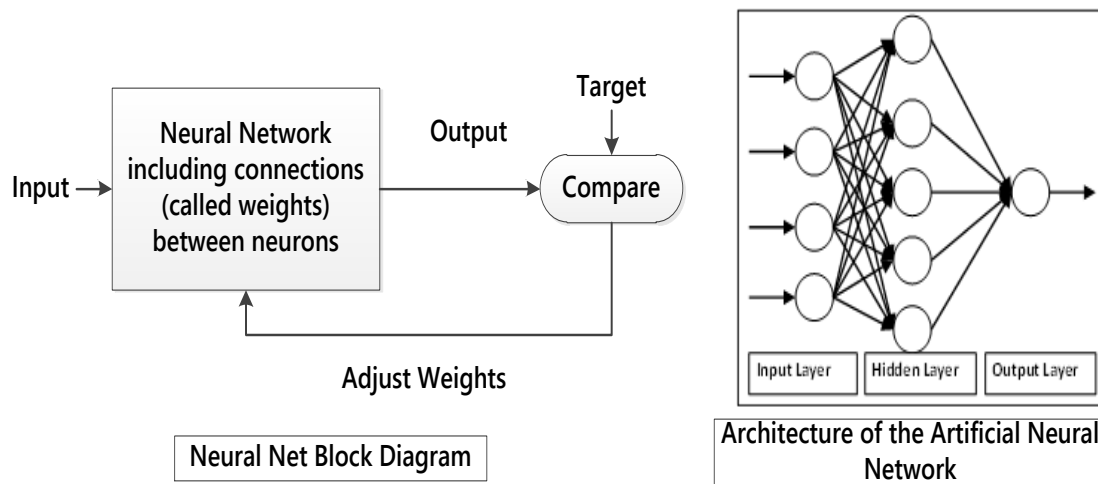


Figure 4.13: Block Diagram of a basic Neural Network (back propagation) [79] and the abstract architecture of the same as used by the proposed system.

While training any such neural network, the connections between the elements must be adjusted for their weight values [99, 103]. If the network is adjusted to suitable values for weights, it is likely that the network output matches the target when compared. So, usually many such input/ output pairs will be used, in the learning process to train a network.

In the proposed work, the Neural Network (NN) is given a set of web pages (and their URLs) with known domains for training the neural network. This set of web pages and URLs can be obtained in either way by using any Web directory like DMOZ or the result listing of any search engine. Thereafter, an index is created to store the web pages along with their domain information.

In order to train the neural network, a web page is taken from this index. If the information about the domain of the web page is already available to the page

Classifier, the web page is given as input to the tag extractor. The tag extractor extracts keywords from the <META> and <TITLE> tags.

These extracted keywords are used to train the neural network by generating suitable domain definitions for each specified domain. These domain definitions have been generated by selecting important keywords for that domain. Consider, for example, the same web page as in Figure 4.9 from the site www.placetoseeindelhi.wordpress.com that provides certain details on the various places of visit in Delhi. The Tag Extractor when used on the HTML source, extracts the following Keywords from the <META> and <TITLE> tags: 'City', 'Delhi', 'Place', 'Travel'. These keywords are judged for importance by keeping a record of the frequency of its occurrence in the tags and in the web page as a whole. Based on the frequency of occurrence of each keyword, the training module of the NN computes a weight value for each keyword and sets a threshold value for the domain. All the extracted keywords from web pages in a specified domain that have the weight value above than the threshold value of that domain will be included in the domain definitions for the respective domain.

These various domain-definitions generated for the different domains are further used by the trained neural network for classifying the web pages. Also, for each domain, the clustering module creates clusters of similar keywords that will be used by the Page Content Extractor to help in the creation of Domain-Specific Databases.

In this work, the keywords with similar context and sharing the same dictionary meaning have been clustered together. For example, a cluster in the *Food* domain may be {carbohydrate, protein, vitamin} signifying the common context '*all the nutrients included in different types of eatables*' and sharing the meaning '*Organic Compound*' whereas another cluster in the same domain might include {burger, snack, cake} based on the common context of 'bread' while another cluster might be {restaurant, cuisine} based on the meaning of 'place to eat'. Similarly the various clusters of keywords generated in TRAVEL domain include {city, place, town, destination}, {tour, travel}, {culture, tradition} based on their respective contexts of location; journey; customs & behavior respectively.

The neural network is now trained by providing the respective keywords, clusters and weights to find the unique identifiers for each domain of consideration. Thus, the proposed algorithm *INITIAL_NN()* is used to initially assign weights to the extracted key words that has been used for training the neural network for uniquely characterizing & identify a domain has been summarized in the following Figure 4.14

Algorithm: INITIAL_NN()**Input: A set of domains D, An index of web pages****Output: Different Domain Definitions and clusters of keywords in a domain**

1. For each domain, $d \in D$ do
2. For all web pages $W=\{w_1, w_2, w_3, \dots, w_n\}$ in domain d do
3. if (<META> & <TITLE> tags exist in a page)
 - 3a. Extracts the keywords from these tags, say m keywords are extracted
 - 3b. For each extracted keyword, k do
 - (i) Record the number of occurrences of k in the <Meta> tag of all web pages, MF_k ,
 - (ii) Record the number of occurrences of k in the <Title> tag of all web pages, TF_k
 - (iii) Record the total number of occurrences of k in all the web pages, TOT_k
 - 3c. Calculate the weight of each keyword in domain, d using the formula

$$Weight(k, d) = \frac{MF_k + TF_k}{TOT_k}$$
 - 3d. Calculate the threshold frequency of domain d .

$$TH_Freq(d) = \frac{\sum_{k=1}^m (weight(k, d))}{m}$$
 - 3e. For each page do
 - select the keywords k , where $Weight(k, d) > TH_Freq(d)$
 - 3f. For each domain do
 - i) generate the domain definition by taking top t weighted keywords from the selected set
 - ii) obtain the clusters of similar keywords from the domain definitions
- else
 - Discard the web page for use by NN;
 - Continue;

Figure 4.14: Algorithm for selecting initial input and their weights for the neural network

The *INITIAL_NN()* algorithm takes the web pages $w_1, w_2, w_3, \dots, w_n$ whose domains already known for each domain d as input and extracts m keywords from the <META> and <TITLE> tags of each web page w_i . Based on their frequency of occurrence in the tags and the web page as a whole, weights are assigned to each keyword k . Initially, Top twenty keywords are selected for composing the four domain-definitions one each for Auto, Books, Food & Travel domains based on a computed threshold frequency $TH_Freq(d)$ for the keywords of each domain.

Based on the generated unique Domain-Definitions, the trained neural network is now used to predict the domain of a new webpage. The classified web pages thereafter are stored in the various Domain-Specific Page Repositories as shown in the Figure 4.2. This process is used for a number of web pages over the specified set of domains. The

performance of the Page classifier depends on the Domain-Definitions generated through the extraction of the keywords as discussed above.

4.4.2. PAGE CONTENT EXTRACTOR

The Page Content Extractor (PCE) takes each Domain-Specific Page Repository as the input and generates a corresponding Domain-Specific Data Repository (DSDR) as the output. These Domain-Specific Data Repositories consist of Domain-Specific Databases that are further used by the Hidden Web Crawler to fill the search forms. Thus, the Page Content Extractor is responsible for populating and maintaining the DSDRs. For each domain, a separate Domain-Specific Data Repository has been created that will facilitate the Hidden Web crawler to fill the corresponding search forms to obtain the Hidden Web data. Thus, a number of Domain-Specific Databases each defining the attributes and values from a different domain have been created. For example, in this work four domains (Travel, Books, Auto, Food) have been used. Therefore, Page Content Extractor generates four different Domain-Specific Data Repositories.

Also, these Domain-Specific Databases must be suitable enough to enable correct and valid submissions for the filled search forms. Therefore, each such Domain-Specific Database consists of attributes and their corresponding values that can be used to fill in the search forms belonging to that particular domain by the Hidden Web Crawler. Initially, these Domain-Specific Databases have been manually equipped with instances provided by the domain experts but are later populated automatically with the help of the Page Content Extractor. As an example, consider the Domain-Specific Database in Table for the Travel domain consisting of attributes and values.

Table 4.3: Sample Domain-Specific databases.

Attributes	Values
Source; Departure City; From	Delhi; Mumbai
Destination; Arrival City; To	Delhi; Mumbai, Chennai

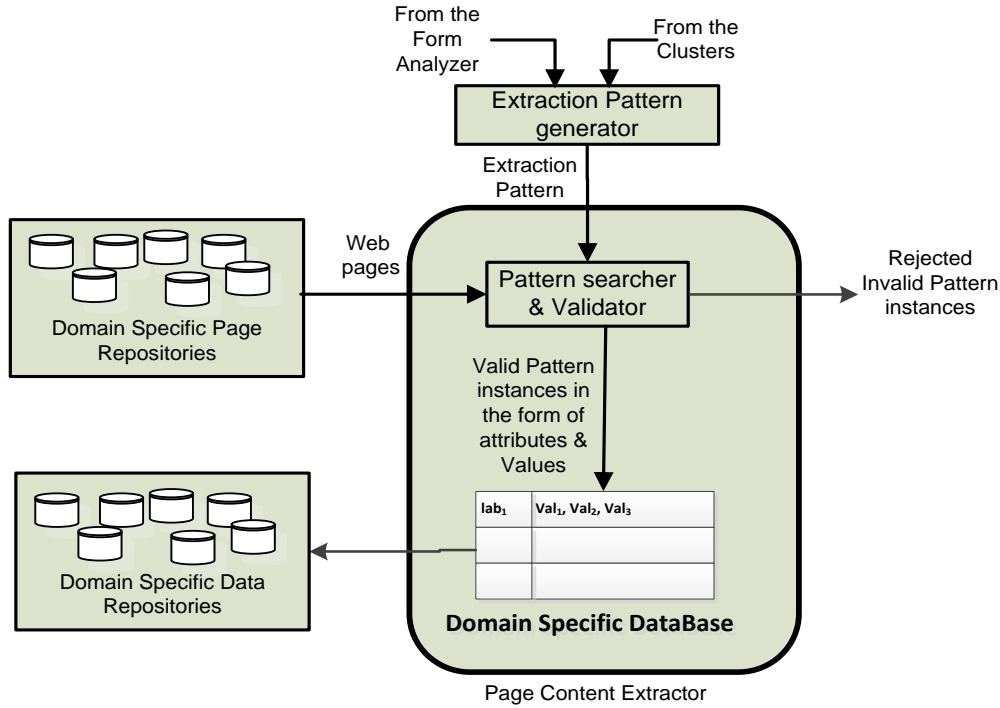


Figure 4.15: The Page Content Extractor

The working of the Page Content Extractor is based on a set of extraction patterns that are generated by the Extraction Pattern Generator (EP Generator) in the proposed approach as shown in Figure 4.15. The Extraction Pattern Generator takes two inputs:

- 1) **From search Forms:** Since the search forms acts as the entry point for Hidden Web , the labels/attributes in the search forms are helpful to generate the instances/values that can be used by the Hidden Web Crawler to fill the forms. Therefore, the EP Generator takes the labels from the search forms to generate the extraction patterns.
- 2) **From the clusters:** Since, the same attribute in a domain can be represented by many different names or labels on different search forms, using all such associated labels will be helpful in generating the values for that attribute. Therefore, the cluster whose keywords share the same dictionary meaning as that of the attribute also acts as input labels to be used by the EP generator. These clusters have already been created by the Page Classifier.

The EP Generator takes the above mentioned two inputs and generates forms the extraction patterns which further helps to find the values/instances that are used to fill in the search forms to retrieve the Hidden Web. The six type of the extraction patterns that have been used by the EP Generator in this work are listed as follows:

Extraction patterns:

EP1: Ls like NP1, ..., NPn

EP2: Ls for example NP1, ..., NPn

EP3: Ls such as NP1, ..., NPn

EP4: Ls comprise of NP1, ..., NPn

EP5: Ls including NP1, ..., NPn,

EP6: Ls in contrast to NP1,...NPn

Where EP_i is any extraction pattern.

And, in the patterns *Ls* indicate the attribute/label like Source, Destination etc, and *NP1, ..., NPn* indicate the values / instances that can be taken by that attribute/ label in the pattern.

For example, consider the sample form from ‘Travel’ domain as shown in Figure 4.16(a). The search form consists of various control elements: a textbox control labeled *Search by Airline* that takes a free-form input, a radio button labeled *Flight trip* giving the option for selecting a One-Way travel or a Round Trip and two combo boxes labeled *Departure City & Arrival City* for selecting the departure and arrival cities of travel respectively.

Figure 4.16: A sample search form

The labels are extracted from the search form by the Form Analyzer to create a Form Element Table corresponding to each search form. The Form Element Table generated by the Form Analyzer corresponding to the search form in Figure 4.16 is given in Table 4.4.

Table 4.4: A sample of the Form Element Table as generated by the Form Analyzer for Figure 4.16

Control Element (E)	Label (L)	Values/Dom(E)
---------------------	-----------	---------------

Select	Departure City	Delhi, Mumbai
Select	Arrival City	Delhi, Mumbai
Radio	Flight trip	One-way, Round trip
Text	Search by airline	String of characters
Submit	Search	Submit

Next, these extracted labels are taken from the Form Element table for forming the patterns such as “*departure city like*”, “*departure city for example*”, “*departure city such as*” etc. These extraction patterns can easily be formed based on the various labels used in the form.

For each label in the supplied Form Element Table, the matching clusters are used to generate the other inputs or *Ls* instances for the EP Generator. As an example, for the label ‘*departure city*’, the clusters in the page repository for ‘TRAVEL’ domain are used to find the matching clusters. The clusters {area, city, capital, destination, place, town, state} and {departure, migration} from the set of clusters were found to match with the label ‘*departure city*’. These clusters were also given as input to the EP Generator. Now, for a pattern of type EP₃, the EP Generator not only forms an extraction pattern like ‘departure city such as’ but also ‘area such as’, ‘city such as’, ‘capital such as’ etc. Similarly for EP₁, it generates ‘departure city like’, ‘area like’, ‘city like’, ‘capital like’ etc.. These set of extraction patterns are then used to extract the values from the PIW pages respective to the various labels.

To extract the values, the Pattern Searcher and Validator raises these extraction patterns as queries on any general purpose search engine. The retrieved instances are tokenized to extract the various candidate values. The candidate values are the ones that can be possibly associated with that Label or attribute.

For example, consider the webpage as in Figure 4.17. The web page is scanned for the occurrences of the extraction pattern, say “Departure city like” by the Pattern Searcher and Validator.

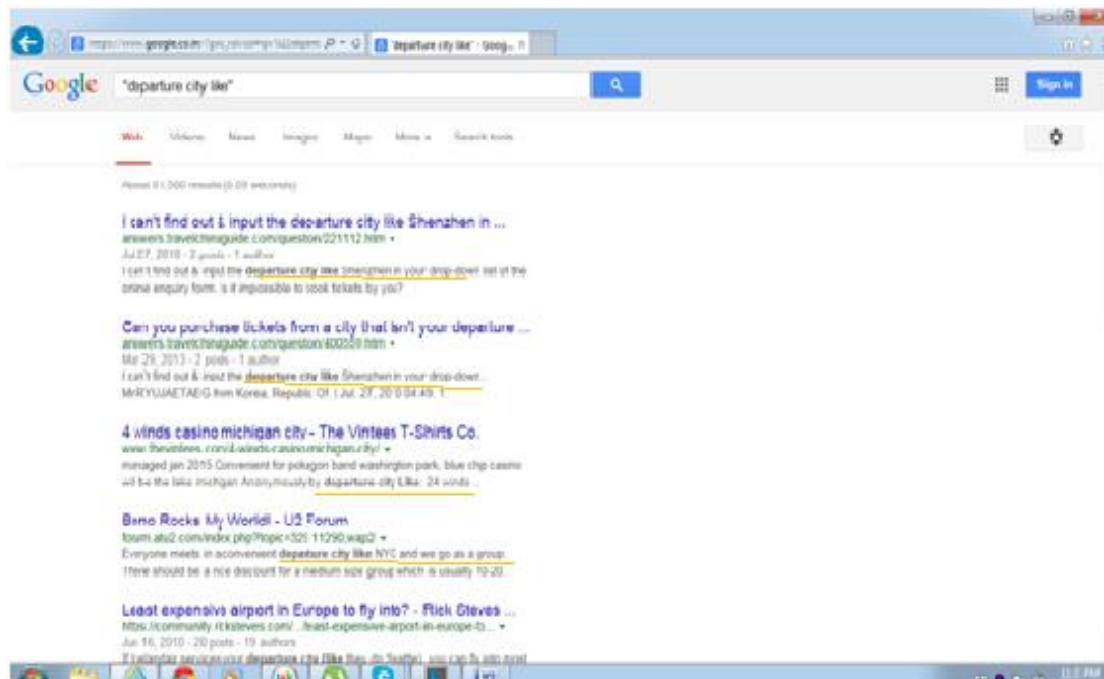


Figure 4.17: Google web page for the query “departure city like”

As can be observed, the specified pattern occurs at many locations in the web page (marked yellow) giving instances like:

- Departure city like *Shenzhen in your dropdown*.
- Departure city like *NYC and we go as a group*.
- Departure city like *Tokyo, Kyoto, Hiroshima and Osaka*.
- Departure city like *Rome, London, Frankfurt, Milan etc*.
- Departure city like *Amsterdam heading through Germany to Switzerland*.

In order to find the values for different attributes or labels, each such instance is tokenized by the Pattern Searcher & Validator. Tokenization leads to the extraction of following values for the labels : *Shenzhen, in, your, dropdown, and, we, go, as, a, group, NYC, Tokyo, Kyoto, Hiroshima, Osaka, Rome, London, Frankfurt, Milan, Amsterdam, heading, through, Germany, to and Switzerland*.

From this set of extracted tokens, the various stopwords like in, your, and, we, go, as, a, to etc. are eliminated by the PSV to form the candidate values that can be attained by the underlying attribute or label Ls. The candidate set of values that was thus obtained for the attribute or Label Ls = ‘departure city’ are: *{Shenzhen, NYC, dropdown, group, Tokyo, Kyoto, Hiroshima, Osaka, Rome, London, Frankfurt, Milan, Amsterdam, heading, through, Germany, Switzerland }*

Now, it is often the case that not all such extracted attribute value pairs are valid. Thus, the attribute value pairs with valid values must be separated out from the set

containing the invalid values. To find out the valid values, the Pattern Searcher & Validator assigns a score to each extracted value for finding its relevance in filling forms from that domain. The score for a candidate value is calculated based on its frequency of occurrence in the PIW pages stored in the DSPR respective to the domain of the search form from which the label L_s was extracted.

A threshold value has been defined for values of each label and all the values that have gained a score above this threshold has been included in the Domain-Specific Database as valid values for that label. This threshold value has been computed by taking an average of the number of occurrences gained by all the values over all the pages in that DSPR.

For the above example, when the candidate values were examined for their number of occurrences, the irrelevant values like heading, through, group, dropdown were filtered by using the set threshold for the label 'Departure City'. A detailed explanation for finding the valid values/ instances on the basis of threshold computation is given in Section 6.3.2. Thus, all the values that have a score more than this value of threshold were considered as valid values to be included in the Domain-Specific Database for the Travel domain. So, the retrieved valid values NYC, London, Rome, Tokyo, Frankfurt, Milan, Amsterdam was added in the Domain-Specific Database corresponding to the labels included in the cluster matching to the label 'departure city',.

Thus, by using the various extraction patterns and the pages in the DSPRs, the Domain-Specific Database is populated automatically by the Page Content Extractor during the process. Also, the same value might be extracted for more than one label in the same Domain-Specific Database and if these labels form parts of different clusters, the values have been added for all such constituting clusters.

The working of the Pattern Searcher and Validator is explained with the help of the algorithm PSV() in Figure 4.18 where EP_i is any extraction pattern formed by using the extracted label. The occurrences of the extraction pattern EP_i is searched over the pages on the WWW to find the initial values V_1, V_2, \dots, V_m . As a next step, the stopwords are removed from this initial set to get the candidate values cv_i 's. The final valid values for the label are then selected by computing a threshold value that is set for the label using the formula given in the algorithm.

Algorithm: PSV()**Input: Extraction Patterns EP_i , DSPRs****Output: Domain-Specific Database having Attribute/Label and values**

1. For each extraction pattern EP_i , do
 - i) Search the EP_i over the pages on the WWW
 - ii) Tokenize the retrieved instances to form the set $V = \{V_1, V_2, \dots, V_m\}$
 - iii) Remove the stopwords from the set V to get the set of candidate values $CV = \{cv_1, cv_2, \dots, cv_i\}$
2. For each candidate value, $cv_k \in CV$ do
find its frequency of occurrence, $F(cv_k)$ in each page of the DSPR

3. Calculate the threshold value for the label L_s in EP_i

$$TH_Freq(L_s) = \frac{\sum_{k=1}^i F(cv_k)}{i}$$

4. For each $cv_k \in CV$ do
 - if ($F(cv_k) > TH_Freq(L_s)$)
Include the candidate value val cv_k in the Domain-Specific database corresponding to L_s
 - Else
Discard the web page for use by NN
 - Continue;

Figure 4.18: Algorithm for Pattern Search and Validator.

The next chapter discusses the other three phases of the Proposed Design of a Parallel Hidden Web Crawler. The chapter analyzes the pages containing search forms and stores them in the Domain-Specific Search Interface Repositories. Also, the task of form filling and response analysis has been discussed in the next chapter.

CHAPTER 5.

PHASE IV: DISCOVERING THE HIDDEN WEB RESOURCES IN A DOMAIN

5.1. INTRODUCTION

The phase II divides the web pages into two broader categories i.e. the web pages containing the search forms and Publically Indexable Web Pages (PIWP). This phase takes the web page containing the search forms that acts as the entry points for the hidden web as the input and creates the Domain-Specific Search Interface Repository containing the search forms as the outcome. Each Domain-Specific Search Interface Repository contains search forms for a specified domain. For example Domain-Specific Search Interface Repository for Books domain contains the search forms for only Books domain. The working of this phase depends on the Form Analyzer which comprises of the following components:

- 1) Form Extractor
- 2) Label Extractor
- 3) Match Value Generator

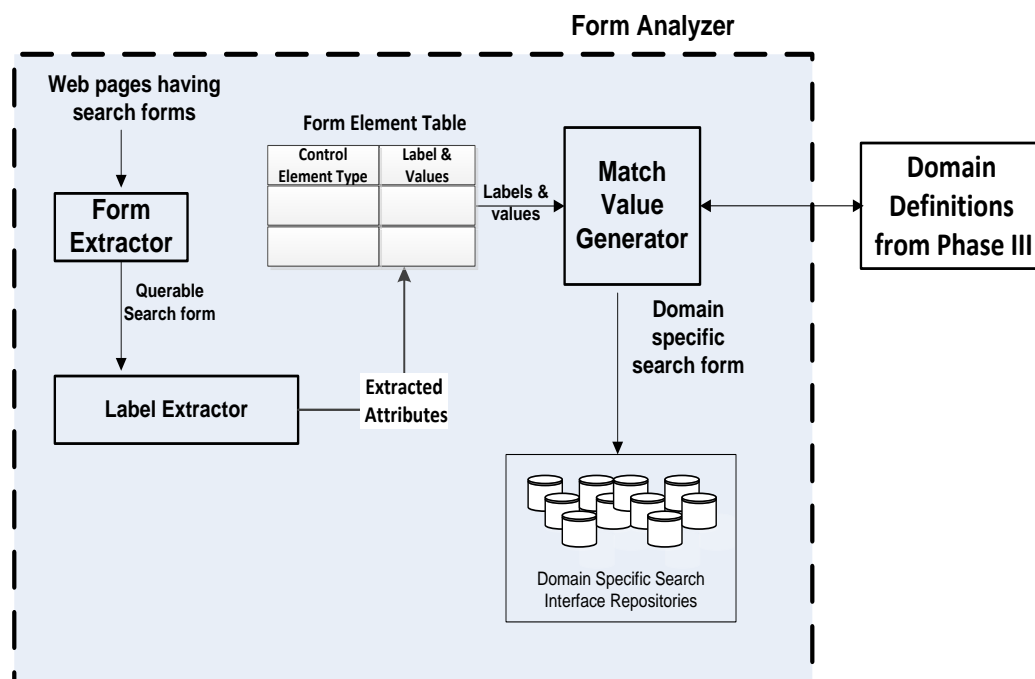


Figure 5.1: The Form Analyzer

The functionality of each component of Form Analyzer is discussed as follows:

5.1.1. FORM EXTRACTOR

This component extracts the search form within that web page and checks whether that search form is a queryable form or any registration form. A queryable form is a form that upon filling with valid values returns a web page that has been dynamically generated from the web database. If a web page contains a queryable form then it is forwarded to the label extractor otherwise it is discarded. Discarding of such search forms (that require pre-registration) improves the efficiency of the Hidden Web crawler. The working of the Form Extractor can be explained with the help of the algorithm in Figure 5.2.

Figure 5.2: Algorithm for Form Extractor

Separating the queryable forms from all the other forms that require registration has been done based on the observation that the forms that are designed for the search functionality have the submit buttons which are typically found to be named as “GO”

Label Extractor

Label Extractor is another important component of the Form Analyzer that takes the given queryable form as input and returns a parsed representation of the form. It extracts the labels of the various control elements, the type of the control element and any corresponding values that are present on the form and store this information in a Form-Element Table (FET). For example, Figure 5.3 shows an example of a search

form for which labels and values are to be extracted and Table 5.1 shows its corresponding FET.

Figure 5.3: Example search form

Table 5.1: Parsed Representation for the above form as Form Element Table (FET)

Control Element (E)	Label (L)	Values/Dom(E)
Select	From	Delhi, Mumbai
Select	To	Delhi, Mumbai
Radio	Flight trip	One-way, Round trip
Text	Search by airline	String of characters
Submit	Search	Submit

Some control elements like the ones labeled as From, To, flight trip offer a finite list of possible values such as select-option, checkboxes or radio buttons which are embedded in the webpage itself. Such elements are termed as bounded elements. Other elements like ‘Search by Airline’ offer free-form input, such as text boxes, have infinite domains (e.g., set of all text strings) are termed unbounded. In this work, forms having certain bounded controls have been considered. In general, if E is any control element, then Dom (E) is the set of values that are valid as input to E. For example: for label ‘From’ and element type *Select* the $Dom(E) = \{\text{Delhi, Mumbai}\}$

For some domains like the Real Estate, numerous search forms exist on the Web with almost a different set of labels for every other form. Also, these search forms contain very few labels but support a large number of values like “apartment”, “villa”, “colony”, “building” etc. associated in the form of substrings or options to the control element. This makes it more difficult to analyze and process the search forms. So, to better analyze the search forms, if any values exist for some control element of the form, they are also extracted by the Label Extractor.

5.1.2. MATCH VALUE GENERATOR

Another important task of *Form Analyzer* is to find the domain (Auto, Books, Food, Travel) of the candidate search form. Identifying the domain of the search form leads to the discovery of the Hidden Web resources that provides information about that specified domain. This is done with the help of the Match Value Generator component of the Form analyzer. The working of the Match Value Generator depends on a Match Logic that makes uses of the Domain Definitions generated by the Page Classifier in the phase III. The Match Value Generator finds the semantic mappings between the following two components as shown in Figure 5.4:

- (a) The extracted labels that are stored in the Form Element Table for the candidate search form
- (b) The Domain Definitions generated by the Page Classifier in Phase III.

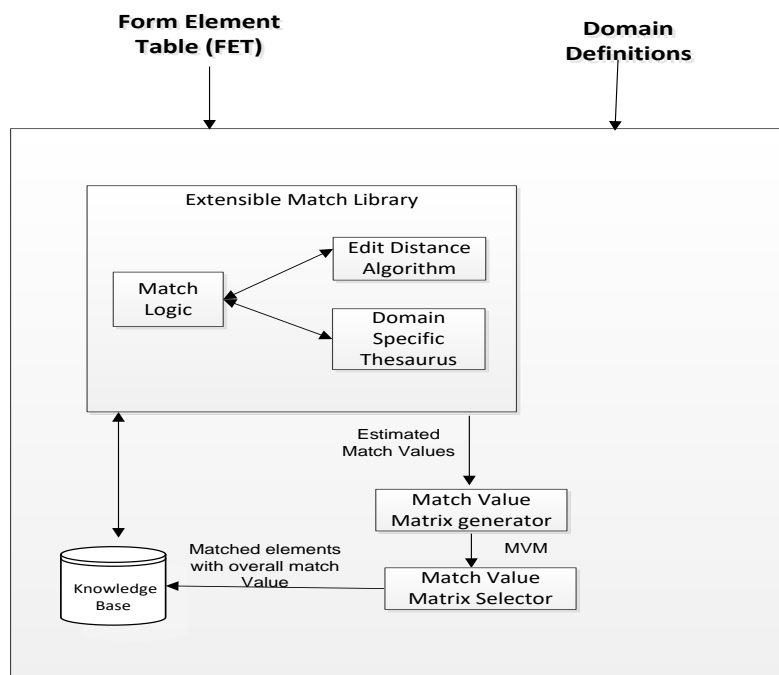


Figure 5.4: Match Value Generator

Thus, the Match Value Generator takes two main inputs: the FET and the Domain definitions and outputs all the matches among the two inputs. The Match Logic used by the Match Value Generator supports multi-strategy matching i.e. uses multiple strategies like Edit Distance Algorithm, Domain-Specific Thesaurus etc. for matching the two inputs. Each matching strategy can be executed independent of the other and new strategies can be added to the Match Logic and used as and when required. In

the proposed work, this extensible Match logic uses two types of matching strategies as follows:

A. Edit Distance Matching

As the same attribute is available under different names in different search forms and the Domain Definitions, matching the attributes to the key terms in the Domain Definition has been done on the basis of the Edit Distance Algorithm [4]. Matching the labels and attributes based on imprecise strings or tokens is much easier by computing the Edit Distance between the strings. The Edit Distance is defined as the minimal number of characters that have to be replaced, inserted and deleted to transform string S_1 into string S_2 . If the distance is small, then the two strings are assumed to be substantially the same. Thus, if the strings are identical then the edit distance is *zero*. To calculate the generalized Edit Distance between the label and the key term in one of the domain definitions, the label or attribute (acts as S_1) must be converted into the key term of the Domain Definitions (acts as S_2) by changing one character at a time. For example: suppose $S_1 = \text{cat}$ and $S_2 = \text{fast}$, then the Edit Distance between S_1 and S_2 is equal to 3 as it involves 3 operations of replacing the character c in S_1 by f of S_2 , replacing the character t in S_1 by s of S_2 , and inserting the character a in S_1 at the end. The computed value of the Edit Distance is divided with the length of the longer of the two strings i.e. the label and the key term for calculating the match value. This normalizes the Edit Distance to lie in the range $[0, 1]$, the result thus obtained is called the Match Value (MV) of the label and the keyterm.

Table 5.2: Match value computation using edit distance method for the label ‘Name’ and other key terms included in different Domain Definitions.

Label (L)	Key term (K)	Edit distance, E_1	Edit distance with reversed strings, E_2	Match Value = $\frac{\min(E_1, E_2)}{\max(L , K)}$
Name	Naming	3	6	$3/6=0.50$
Name	Company	6	6	$6/7=0.85$
Name	FirstName	9	5	$5/9=0.55$

Name	Time	2	2	2/4=0.50
------	------	---	---	----------

The comparison between the extracted labels and the keyterms included in the domain definitions has been represented in the Table 5.2. The edit distance algorithm gives more significance to a mismatch at the beginning of a string than to a mismatch at the end. Therefore, Edit Distance algorithm is used to compare not only the original strings, but also their reverse strings, i.e., “emaN” and “emaNtsriF” as shown in row 4 of the table. The 3rd and the 4th column of the table indicate the comparison between the two inputs when available in their original and reversed forms respectively. The smaller of the two similarity values becomes the final edit distance. The fifth column indicates the Match Value computed between the label (Name) and the keyterms included in the various domain definitions by dividing the edit distance with the length of the larger of the two strings. In the same way, the match values are tabulated for each of the keyterms in the Domain Definitions and each label of the given FET.

B. Domain-Specific Thesaurus

-
-
-
-
-
-
- is hypernym of *car*.
- *Meronymy*: String S_1 is meronym of string S_2 if S_1 is a part of S_2 .

For example, *First Name* is meronym of Author Name.

If there exists any relationship between the label and keyterm then Match Value is assigned as 1 otherwise it is 0. An example of the Match value computed using the Domain-Specific Thesaurus has been shown in Table 5.3.

Table 5.3: Match Value Computation based on relationship in Domain-Specific Thesaurus

Label (L)	Key term (K)	Relationship	Match value
First Name	Name	Meronym (L, K)	1

Last Name	Name	Meronym (L, K)	1
Book Title	Book Name	Synonymy(L, K)	1
Hardcover	Format	Hypernym (K, L)	1

For example, If the FET contain the label '*Hard cover*' and the keyterm '*Format*' can take the value hardcover, then the key term *Format* becomes the hypernym of *hard cover* as shown in column 3 of the table .Similarly, the 'part of ' relationships may be used to discover meronyms. If the extracted label consists of two parts *First name* & *Last name* as in column 1 and column 2, then they can be identified as part of *Name* and thus a meronym of *Name*.

While using the Domain-Specific Thesaurus, a match value of *zero* indicates that the match logic does not identify any relationship between the two inputs (label and keyterm). However, if any relationship (Synonymy, Hypernymy and Meronymy) exists between the two input strings, a match value of 1 is returned by the Match Value Generator.

Now for each mapping, the overall Match Value (also called Estimated Similarity Score) is computed as an average of the Match values obtained by using the two strategies. This estimated similarity score for the labels of the target FET is represented in the form of a Match Value Matrix (MVM). A number of such matrices are generated for the target FET, one corresponding to the key terms of each domain definition. Consider as an example, a FET containing A, B, C as labels and the domain definition containing the key terms X,Y,Z are given as inputs to the Match logic, the Match Vale Matrix is a cross-product similarity matrix having the combinations for A:X,A:Y,A:Z, B:X, B:Y, B:Z, C:X, C:Y, C:Z . Each cell of this MVM contains the overall match value between the stated label and the specified key term from the Domain Definition. A schematic representation of the MVM using the labels of FET and the keyterms of Domain Definition is given below in Figure 5.5.

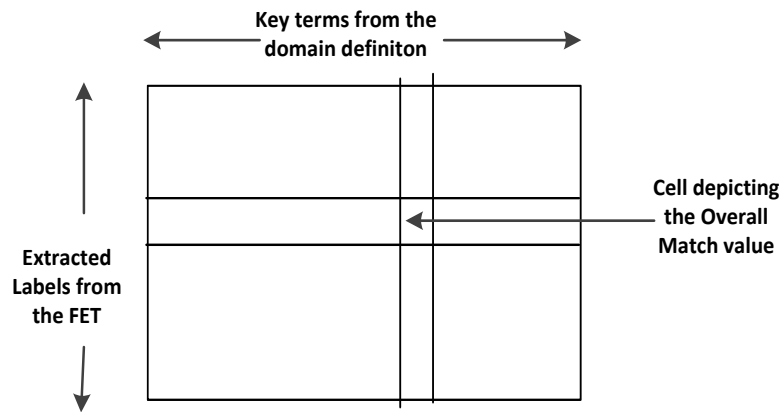


Figure 5.5: Schematic diagram of Match Value Matrix

Table 5.4 shows an MVM for a search form with FET f and keyterms from the Domain definition of Books:

Table 5.4: An example of a Match Value Matrix

Key Terms → Labels	Author	Title	Subject	ISBN	publisher
Author	1.00	0.20	0.35	0.20	0.30
name of the book	0.30	0.80	0.45	0.30	0.10
ISBN	0.10	0.15	0.12	1.00	0.15
Topic	0.10	0.50	0.15	0.20	0.25

Practically, some of the generated matches in the MVM may be irrelevant and therefore of no importance. Thus, the Match Logic also employs a Selector Function that checks all the MVM's and their overall Match Values to find the matches that are more valuable, instead of using all that are generated by the Match Logic, thereby improving the performance of the Match Logic of the Form Analyzer.

To find the valuable matches, the Matrix Selector uses a threshold value as the selection parameter. This threshold value is compared with the overall Match Value for each MVM to find the valuable matches. The matches having the overall Match Values greater than the threshold values are treated as important and thus stored in a Knowledge Base for future reference. The matches having overall Match Value below the threshold value would be ignored. Now if in future a label 'author' of the FET is

matched with the keyterms 'Author' and 'subject' in the Domain Definition, the matcher returns overall Match Value for both the matches i.e. for *author* and *Author* , *author* and *subject*.

In order to avoid needless match effort and redundancy of storage, before starting the next

eyterms of the Domain Definition, it associates a Boolean variable termed 'MATCH' with each label. The Match logic sets the value of the variable 'MATCH' to TRUE if the label matches with a keyterm in some Domain Definition whereas a "MATCH" value of FALSE for the label indicates that there does not exist any match for that label in any of the Domain Definitions. Besides indicating a true/false match, the Match Logic also associates the domain of the matched keyterm with that label. This will ease the task of identifying the domain of the search form. If the number of labels of a FET that match with the keyterms from a common Domain Definition are more than a set cut-off, then it is qualified as a search form in that domain.

[Advanced Search](#)
[Browse](#)
[Booksellers](#)
[Community](#)
[Sell Books](#)
[Textbooks](#)
[Rare Books](#)

Search Books:
By Keyword

[Advanced Search](#)

[Home](#) > [Advanced Search](#)

Advanced Search

[Search Preferences](#) | [Top 5 Search Tips](#)

Author:
Title:
ISBN:
Keywords:
Publisher:
Published Date: min to max
Price (US\$): min to max
Bookseller Country: - All Countries -
Boolean Searching: ☐ On ☒ Off [More Info](#)

Refinements

Attributes:

☐ First Edition

☐ Signed

☐ Dust Jacket

☐ Not Print on Demand

Books Listed Within:

☐ 24 Hours

☐ 48 Hours

☒ All

Binding:

Any Binding

Sort Results By:

Lowest Total Price

Results Per Page:

30

[Clear Fields](#)

Figure 5.6: A sample search form from Books domain

As an example consider the search form in Figure. 5.6 above. In this scenario when the Match logic generates the matches between the labels of the FET and the key terms of the various domain definitions, the “MATCH” field is always assigned a TRUE value. Moreover, all the labels get associated with the key terms of a common domain definition of Books as shown in the 4th column of the table 5.5.

Table 5.5: Labels and type of input associated with them for the Search Interface Form in Figure 5.6

[illegible]

All seven labels match to the key terms in the definition of Books domain i.e. the search form exhibits a 100% match with the Books domain, thereby predicting the Hidden Web resource as relevant in Books domain. Although, the label *price* also adopts definition from auto & Travel domains other than that of Books, the match percentage is less than 15% (only 1 out of the 7 label match) and hence the resource does not qualify for relevance in any other domain. The various hidden web resources after examination have been stored in the various Domain-Specific Search Interface repositories. Also, these repositories are open for updating whenever any new relevant hidden web resource or a search form is discovered in a domain.

5.2. PHASE V: PARALLEL CONTENT EXTRACTION FROM THE HIDDEN WEB

This phase of the crawler takes the various Domain-Specific Search Interface Repositories as input and processes the search forms in parallel to efficiently retrieve the contents from the Hidden Web. The working of this phase has been illustrated in Figure 5.7.

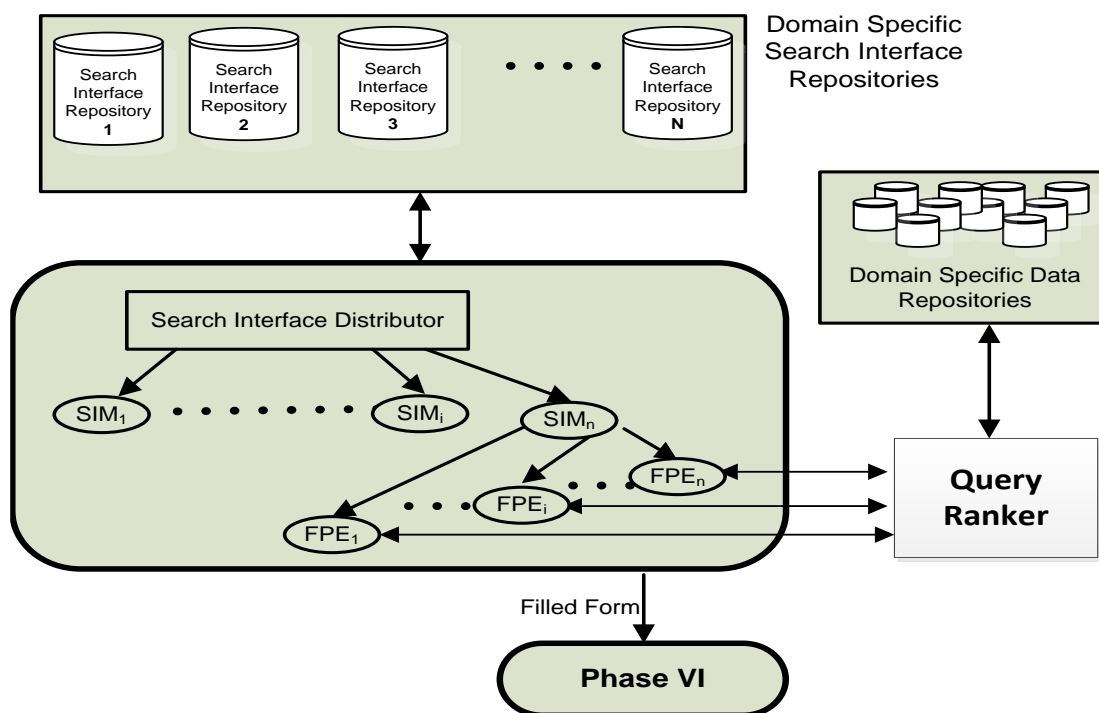


Figure 5.7: Working of Phase V of the crawler

In order to exploit parallelism, the crawler should be able to process and submit the search forms in parallel. Therefore, in this phase parallelism has been incorporated in following two levels.

- 1) *At the level of Search Interface Managers (SIM):* The Search Interface Distributor (SID) acts as the coordinator at this level. It is responsible for distributing the search interface repositories among the different search interface managers or SIMs according to their domains. For example, the search interface repository having numerous search forms from the *Books* domain is assigned to the search interface manager that is held responsible for processing the forms from the Books domain. Similarly, all other SIMs are assigned the responsibility for processing the search forms that are contained in the search interface repository respective to their assigned domain. Distributing the search forms domain-wise helps in avoiding overlap and redundancy.
- 2) *At the level of Form Processing Elements (FPE):* At this level, SIM, that is containing a repository of search forms for a particular domain, distributes the search forms to every form processing element (FPE) is responsible to fill that form in future. For example, consider the sample of the search form repository from Books domain shown in Figure 5.8 that contains only those search forms which allow online searching of books.

Figure 5.8: Sample of the Domain-Specific search interface repository in Books domain

The SIM is responsible for fetching the hidden web data from the Books domain and distribute these m forms to m Form Processing Elements (FPE). This enables parallel harvesting of hidden web data in the *Books* domain through the parallel processing of search forms by the various Form Processing Elements created by the SIM responsible for it.

In other terms, each Search Interface Repository is assigned to a Search Interface Manager (SIM) that becomes responsible for harvesting the corresponding search forms. Hence, the search forms from different domains are assigned to different

Search Interface Managers. For example, in order to perform a crawl of the portion of the web databases related to the “Auto” world, the Search Interface Distributor assigns the “Auto” domain Search Interface Repository to one of the SIM which evenly distributes the search interfaces among its Form processing elements.

Now, as each SIM is assigned only the search forms from a fixed, unique domain, the number of SIMs is constant in a single crawl and equals the number of distinct domains of crawler consideration. This guarantees that all the search interface repositories have been harvested. But as all the search interface repositories are initially mapped to the set of SIMs, the crawler cannot increase the number of Search Interface Managers to accelerate processing of search forms (the crawling process). Therefore, to overcome this limitation on the crawl rate, each SIMs is made responsible for a random distribution of its load among the parallel *Form Processing Elements* (FPE) to facilitate load balancing. Also, all the FPEs of a Search Interface Manager need to register themselves with their representative SIM at that instant before a request for processing the list of search form is made by it. Each FPE can be assigned a maximum of n search forms for processing by it. If the search interface repository of a SIM contains more than n forms, then the SIM creates a new instance of the FPE and assigns another n forms from its repository to the newly created FPE. This assures an even distribution of workload among all the form processing elements that were registered with a particular SIM. Thus, the number of FPEs must be created and destroyed dynamically by the respective SIM as per the requirement in its domain.

Figure 5.9: Algorithm to distribute load by the SIMs to the associated FPEs

After distribution, each form processing element is held responsible for harvesting exclusively the set of search forms received from its SIM without communicating directly with each other. Whenever a FPE starts, it is initialized with a list of parameters from its Search Interface Manager. These parameters include: the parsed representation (generated in phase IV) of the search form to be processed, content filter, list of URLs to be included and excluded from crawling etc.

A FPE analyzes the various parameters to judge the type of the contents that is required to fill up the search form. The Domain-Specific Database and the Page Statistics Repository which are stored together in the Domain-Specific Data Repository act as filling resources for the form processing elements. The working of the Form Processing Element is the guided by the Query Ranker that is used to generate ranking among the queries listed in the Domain-Specific database based on the statistics that is collected by the crawler during its execution. A detailed working of the Query Ranker is provided in the Section 5.4. The FPE then fills the form by raising queries as per their ranks and associating a suitable value with each control, the value being chosen from the domain of the respective control element.

Figure 5.10: Algorithm of a Form Processing Element (FPE).

The FPE, after filling the search form, submits the request to the corresponding Web Server and obtains valuable responses in the form of dynamically generated web pages.

The working of an individual form processing element can be explained with the help of the following algorithm in Figure 5.10.

Once the FPE is over with the set of search form assigned for crawling, it makes a request for new search forms to be assigned to it. The Search Interface Manager does not need to permanently monitor the system because the FPEs demand work on a need basis. The FPEs does not impose any overhead on the Search Interface Manager because the SIM simply responds to requests for unprocessed search forms and all the SIMs and the FPEs work in a recursive fashion till the specified time or the exhaustion of other available resources.

The interaction between the SID, SIM, FPE and the hidden web database can be illustrated with the help of a sequence diagram as shown in Figure 4.15:

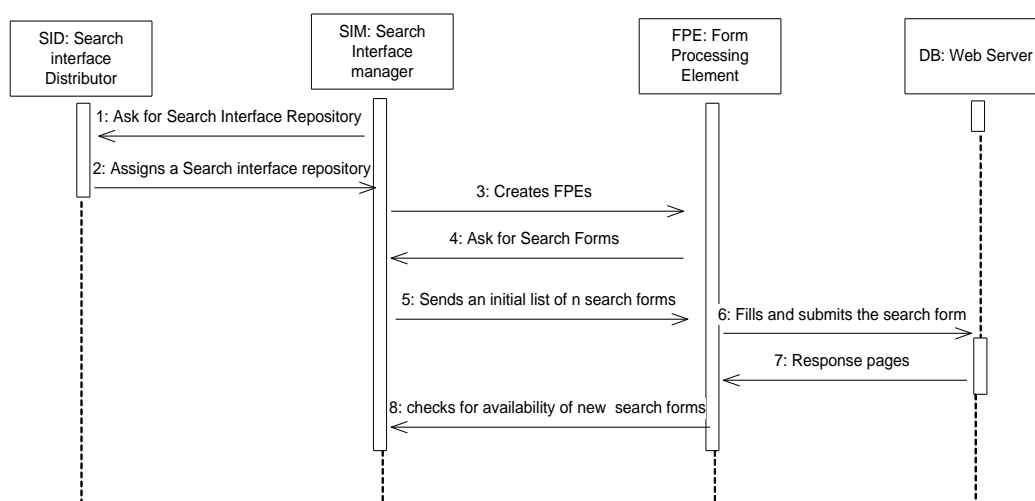


Figure 5.11: A sequence diagram showing interaction between the SID, SIM, FPEs and the Web server.

The sequence diagram consists of four objects, one each of a SID, SIM, FPE and the database server. It consists of the following steps:

- 1) The SIM created for a domain requests the SID for the Search Interface Repository associated with the domain of its consideration.
- 2) The SID provides the corresponding Search Interface Repository to the SIM.
- 3) The SIM creates a FPE for processing the provided Search Interface Repository.
- 4) The FPE then asks its controlling SIM for the n search forms to be processed by it.
- 5) The SIM sends the requested number of search forms to the FPE.

- 6) The FPE processes each search form by filling with appropriate values and submits the filled search forms to the Web Server.
- 7) The Web Server generates dynamic web pages in response which are then passed to next phase VI of the crawler.
- 8) After the FPE has processed the n search forms assigned to it, it asks the controlling SIM for more search forms if available in the Search Interface Repository to continue harvesting the Hidden Web data.

The SID object remains active for steps 1 and 2; the SIM object remains active for steps 1,2,3,4,5,8; the FPE object remains active for 3,4,5,6,7,8 and the Web Server remains activated during steps 6 and 7.

The distribution of the search forms from the Search Interface Repository assigned to the SIM is done dynamically during the crawl by the SIM itself through its support to variable number of form processing elements.

The design of this phase helps to combat the problem of:

- 1) *Overlap*: For the proposed hidden web crawler, overlap is said to exist if and only if the same search form is considered for processing by more than one SIM as this will lead to multiple fetches of the same webpage and thus a redundancy in the extracted hidden web content. However, the proposed approach avoids overlap by assigning the search form to a unique SIM respective to its domain.
- 2) *Synchronization*: The Search Forms have to be partitioned to enable parallel crawling but this involves an overhead of synchronizing the parallel crawling processes to minimize overlap. The proposed approach does not involve the overhead of synchronizing the parallel threads as the Search Interface Distributor acts as the centralized coordinator of the operation of all these parallel threads or elements and distributes the work as per the domains.
- 3) *Communication Bandwidth*: Although, each SIM need to coordinate with its associated FPEs but neither the various SIMs nor the FPEs need to communicate among themselves to coordinate with each other. This helps to minimize the communication bandwidth needed for synchronizing the work of the various parallel threads. Moreover, avoiding the overlap among the search forms during processing will be useful to eliminate repeated harvesting of the same hidden web database which further helps the proposed Parallel Hidden

Generally, two types of response pages are typically retrieved for a search in the web database: a multi-match page consisting of a list of result records and a no match page; Figure 5.13 and 5.14 respectively show such pages for a hidden database offered by querying the website makemytrip.com in the ‘travel’ domain. The response result page

in Figure 5.13 has multiple records so it can be called as a multi-match page. Herein, each retrieved record describes the schedule of a particular flight and the fare incurred. Such a multi-match page has been termed in this work as a Valid Response page (VRP) whereas a page in Figure 5.14 that contains an error message reporting that either no matches were found for the submitted query or page not found (HTTP 404 error) has been termed as Dead Response page (DRP).

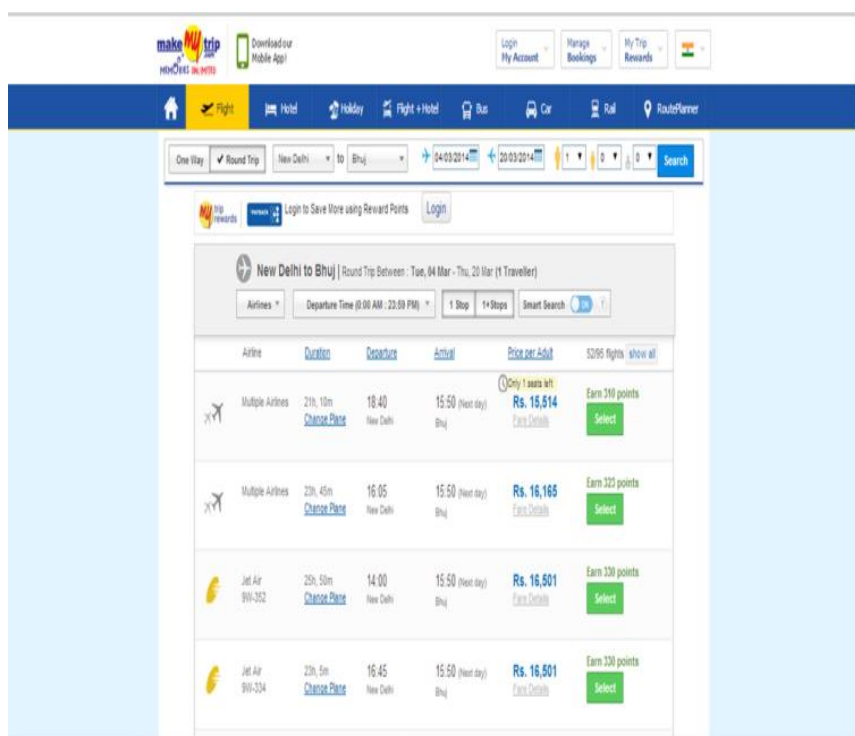


Figure 5.13: A valid response page or a multi-match page from a hidden database in ‘travel’ domain

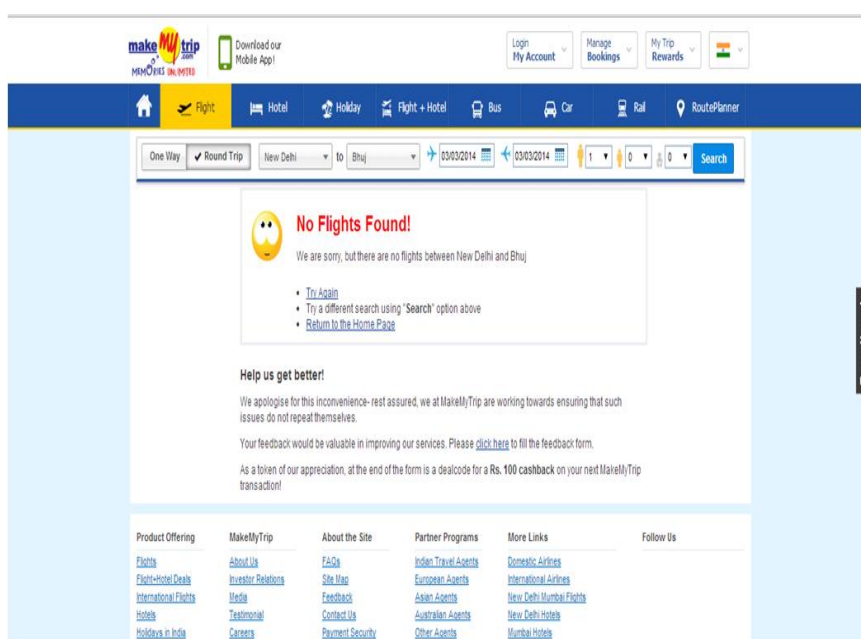


Figure 5.14: A dead Response page or a no-match page from the same hidden database of ‘travel’ domain.

The aim of the Response Analyzer is to automatically distinguish between a VRP and a DRP and provide the feedback which acts as a huge source of information for the Form Processing Element to tune the crawler for suitable and appropriate value assignments in the next run of the crawler. The feedback includes items as follows:

- 1) the size of the response page that has been dynamically generated;
- 2) the content or data items extracted from the response pages in the form of snippets
- 3) the hyperlinks embedded in the response pages.

These three features are helpful to identify the optimal queries to be supplied for filling the search forms. These resources of information are needed by the crawler to efficiently fill the search forms and are collectively stored in the Domain-Specific Data Repository (created during phase III). Also, these resources are adaptive in nature while the crawler proceeds towards its target, thus are self-governed by the crawler. After the first run of the crawler is over, the obtained set of response pages and web pages is analyzed to collect data for these repositories. However, for the ease of implementation of the response analyzer, immediate navigation of the response pages further have not been chosen. The next sub-sections discuss this process of analysis in detail by providing a description of the various functional components of the Response Page Analyzer which are as follows.

5.3.1. SIZE EXTRACTOR

The dynamically generated response pages are usually structured in a similar fashion, be it an answer page with a single record or a long list of matching records. In usual, a response page containing a long list of matching records is of a bigger size as compared to the one that contains fewer number of matching records which in turn is bigger in size to a dead response page. It has been assumed here that the size of a ‘*no match page*’ lies in the range 1-3KB. The Size extractor calculates the size of the data retrieved by a query in terms of the size of the response pages, based on which it later guides the crawler through the wise suggestion of optimal queries for better valid form submissions. Such queries are hereafter termed as ‘*optimal*’ queries. The size of the data retrieved by a query or in short, the size of a query is assumed to be the size of its

retrieved response pages for that query submission. Therefore, a few terms have been defined to be used for the purpose mentioned above.

- **Definition 1: Valid query:** A query q_i submitted to the web database DB, is a valid query if it returns m search result records that are forked into n response pages with each page accommodating at most M result records. In other words, a valid query is one that returns a valid response page.
- **Definition 2: Dead Query:** A query q_i submitted to the web database DB, is a dead query if it retrieves a dead response page i.e. the page containing no results or no match page.

Therefore,

For a valid query, q_i :

$$m >= 1, n >= 1$$

For any dead query, q_i :

$$m = 0, n = 1$$

Where n is the number of response pages retrieved in response to q_i & m is the total number of records that are retrieved when q_i was fired on the search form.

- **Definition 3: Invalid Query:** A query q_i to the web database DB is termed as invalid if does not even allow the submission of the filled form to the web site.
- **Definition 4: Query response Size, QRS():** The Query Response Size of a query q_i , represented as $QRS(q_i)$ has been computed as the size of response page that has been retrieved by issuing the query q_i to database DB.

For a valid query q_i , Query Response Size (in KB) is defined as the size of the response page R_i , i.e.

$$QRS(q_i) = \text{Size}(R_i)$$

It is obvious that the QRS of a dead query is the size of the DRP that has been retrieved. It has been assumed here that Query response Size of a dead query is between 1KB to 3 KB. Thus,

For a dead query q_i ,

$$1\text{KB} \leq QRS(q_i) \leq 3\text{KB}$$

The computation of the size of the Response Page not only helps the proposed crawler to distinguish between the valid, invalid and dead queries in its future runs but also in estimating the cost of issuing the specified query which depends directly on the huge cost of downloading from the Web. The queries that have once been marked *invalid*

remains in the same status always but the queries which might have been marked as *dead* in a particular run of the crawler may not necessarily behave the same always. Thus, elimination of invalid queries from the query space reduces the time of execution of unwise or non-optimal queries and the query space occupied by these queries but eliminating the dead queries may debar certain promising queries from consideration in the subsequent runs. Thus, in this work, the dead queries have not been eliminated from the consideration of being optimal queries. Therefore, among all queries- valid or dead, the crawler wisely predicts the ‘optimal’ query by ranking them based on the Query Response Sizes.

5.3.2. THE PAGE CONTENT EXTRACTOR

The Page Content Extractor used in this phase of the crawler has been used to extract the content from the retrieved Response Pages in order to populate the contents of the Domain-Specific Databases that are stored in the Domain Specific Data Repositories. The Domain-Specific Databases have been created in terms of labels/attributes and an associated set of values and are initialized with instances provided by the domain experts based on their knowledge in phase III of the proposed crawler. But, in order to update these Domain-Specific Databases, new entries are extracted from the Response Pages for populating their contents. This otherwise may lead to the situation of “insufficient data” for the crawler when trying to automatically process the search form through filling. Populating the contents of these Domain Specific Databases helps to improve the crawler’s ability to more effectively fill the forms during its subsequent runs.

5.4. QUERY RANKER

Query Ranker generates the rank of the candidate queries listed in the Domain-Specific Database based on the page or query statistics collected during crawling. For ranking the queries, consider $\text{Dom}(E_i)$ to indicate the set of values in the domain of control element E_i for each $i \in [1, d]$ where d is the number of controls on the form; then, the Cartesian product of $\text{Dom}(E_1)$, $\text{Dom}(E_2)$, $\text{Dom}(E_3)$, ..., $\text{Dom}(E_d)$ form the query space for the proposed Parallel Hidden Web crawler.

We refer to each element of the Cartesian product as a query q_i in the query space i.e. $Q = \{q_1, q_2, q_3, \dots, q_m\}$ i.e. a query is one of the possible combinations from the values of all the elements. Alternatively, each q_i is a list of (label, value) pair where label of

the control element is an attribute of the form and value is one of the instances from the domain of the associated control element.

Let CD_i denote the number of choices for each control field, thus

$$CD_i = |\text{Dom}(E_i)| \text{ for each } i \in [1, d] \quad 5.1$$

Then the query space Q , comprises of a maximum of m queries that can be sent correctly for filling a form

$$\text{Where } m = CD_1 * CD_2 * CD_3 * \dots * CD_d \quad 5.2$$

Or

$$m = \prod_{i=1}^d CD_i = \prod_{i=1}^d |\text{Dom}(E_i)| \quad 5.3$$

Thus, m is the total number of possible query combinations for a search form.

Figure 5.15: Example of a search form to a structured database

Consider the multi-attribute search form as in Figure 5.15 and a sample of the Domain-Specific Database for the same in Table 5.6 that is used as running example to explain the working of the query ranker while filling out the designated search form. The first column of the domain-specific database contains the labels of the various control attributes of the referred search form and the second column contains the possible values for the corresponding attribute.

Table 5.6: Sample Domain-Specific Database showing the 3 attributes or controls of the search form and the domain of each such attribute.

Control Element (Ei)	Domain(Ei)
From	Delhi, Mumbai
To	Delhi, Mumbai
Search flights	One-Way, Round-trip

For the referred case, the control element say ‘From’ can have either ‘Delhi’ or ‘Mumbai’ as a valid value, ‘To’ can take either ‘Delhi’ or ‘Mumbai’ & the control ‘Search Flights’ can take one-way or round-trip as its possible values, thus each control element offers a choice of two values giving the values of $m=2*2*2=8$. This value of m specifies the number of queries that can be possibly raised for the form. The possible set of queries has been listed in table 5.7.

Table 5.7: Set of 8 possible query combination

S.No.	From	To	Search flights
1	Mumbai	Delhi	One-way
2	Mumbai	Delhi	Round-trip
3			
4			
5			
6	Delhi	Delhi	Round-trip
7	Delhi	Mumbai	One-way
8	Delhi	Mumbai	Round-trip

When the proposed crawler starts initially, its working is based on the contents of the Domain-Specific Database where all the query combinations have equal probabilities of selection. But, practically, dependencies exist among the attributes of the hidden database, because of which the Query Ranker excludes certain combination of values from forming a candidate query. In the above example, with proper external knowledge of the dependency between the Source of departure and the Destination of arrival many combinations of queries can be discarded, so as to form the query space Q as in Table 5.8. Thus, the crawler need not explore queries having values for From=

‘Delhi’ and To= ‘Delhi’. Such excluded queries (at s. no. 3, 4, 5, 6 in Table 5.7) form the set of invalid queries which should also be debarred from all the future runs of the crawler. The query Space, Q , thus formed consists of four queries which have been shown in table 5.8. The Query Ranker now assigns a query ID to each query in Q so as to uniquely identify them during the ranking process.

Table 5.8: The query space Q and the assigned query id’s

When the crawler initially starts, it raises the query in the same sequential order as per their Query id’s. After all the queries have been executed, the retrieved set of response pages has to be analyzed for updating the Domain-Specific Database and initializing the page statistics repository. Thus, the completion of the first run of the crawler is responsible for initializing the contents of the Page Statistics Repository. After the first run, these queries are further ranked by the proposed Query Ranker module to get the optimal sequence of all the possible queries.

The Query Ranker performs a random ranking of the queries in the Domain-Specific Database based on the size of response pages in the Page Statistics Repository. Also, the proposed Random Ranking mechanism considers the following:

- 1) Not all the candidate queries from the query space Q that are listed in the domain-specific database can retrieve Valid Response Pages (VRP) from the hidden database. Certain queries appear to be valid but when fired on the search form does not retrieve valid response page. The reason might be because the hidden web database does not offer search for the criteria that has been stated in the query.
- 2) A query that once retrieved Dead Response Pages and thus was termed as ‘Dead’ may not necessarily be ‘dead’ always when raised in the next crawls of

the proposed crawler. The possible reason being that the hidden web databases might have been updated.

Intuitively, the proposed ranking approach defines a random ranking function, `random_rank()` based on two factors: a static and a dynamic factor. The static factor considers the behaviour the query exhibited in all the previous executions of the crawler whereas the dynamic factor considers the behavior of the query in the immediate previous crawl. Therefore, the rank of the query q_i when the crawler runs for the k^{th} time depends on a value of the static rank which would be computed based on the statistics from all the previous crawls and the dynamic rank whose value is computed based on the sizes of the response pages obtained in the immediate last crawl i.e. the $(k-1)^{\text{th}}$ crawl. Thus, the rank of a query q_i to be executed when the crawler runs for the k^{th} time is the cumulative value of static and dynamic rank generated for the k^{th} crawl :

$$\text{Random_rank}(q_i, k) = \alpha \cdot \text{static_rank}(q_i, k) + (1 - \alpha) \text{dynamic_rank}(q_i, k)$$

5.4

Where,

For $k=1$

$\text{static_rank}(q_i, k)=0$

$\text{dynamic_rank}(q_i, k) = i$

For $k \geq 2$

$\text{static_rank}(q_i, k) = \text{random_rank}(q_i, k-1)$

$\text{dynamic_rank}(q_i, k) = \text{pos}(q_i, \text{desc_sort}(\text{QRS}(Q)))$

Here, `desc_sort()` is a function that produces a listing in the decreasing order of the `QRS(qi)` for all the queries in the query space, Q . And, `pos()` gives the position of the query q_i in the sorted list.

The static ranking function has been termed as static because its contribution to the value of `random_rank` for the k^{th} crawl is independent of the various statistics generated during the $(k-1)^{\text{th}}$ execution of the crawler thereby remaining constant in that particular run of the crawler. Thus, it is a value that is assigned to the query prior to the execution in the current crawl. The value of `random_rank` for the first run of the crawler thus has been computed by using the formula stated in Equation 5.4.

$$\text{Random_rank}(q_i, 1) = \alpha \cdot \text{static_rank}(q_i, 1) + (1 - \alpha) \cdot \text{dynamic_rank}(q_i, 1) \quad 5.5$$

Also, the parameter α lies in the range $[0, 1]$ and determines when random ranking should be done. If $\alpha=1$, the `random_rank` of the query computed for the immediate previous crawl is returned to form the rank of the queries for the next run of the crawler; which means no statistics need to be collected during the crawler's execution and no re-ranking is performed. But this always issues the queries in the same order repeatedly which may always project the same response from the server, and might not help to obtain the other records present in the hidden database.

If $\alpha=0$, the initial static rank is excluded from consideration and only the effect of the statistics collected from immediate previous crawl is examined to predict the 'optimal' query i.e. only the dynamic factor is taken into consideration.

As an example, the value of $\alpha = 0.25$ has been considered. This assigns a higher weight to the `dynamic_rank` so as to give more importance to the behavior of the query in the most recently executed crawl. The size needs consideration as some queries may bring a detailed description of certain records and thus more of the data that is resident in the hidden database. Thus, by taking the value of α as 0.25.

$$\text{Random_rank}(q_i, 1) = 0.25 * \text{static_rank}(q_i, 1) + 0.75 * \text{dynamic_rank}(q_i, 1) \quad 5.6$$

As mentioned earlier, for the initial start of the crawler, the values of `static_rank` ($q_i, 1$) = 0 for all q_i and the value of `dynamic_rank` ($q_i, 1$) = i as the queries were arranged in some random order by the crawler. Table 5.9 shows the computation of random rank values that would be used in the first run of the crawler.

Table 5.9: Computation of random rank values using equation 5.6 for the first run of the crawler

Query ID	static_rank($q_i, 1$)	dynamic_rank ($q_i, 1$)	random_rank($q_i, 1$)
			0.75
			1.5
			2.25
q_4	0	4	3.0

The `random_rank` values for the queries are used to decide the order in which the queries will be raised at the search forms. More is the value of the `random_rank`, less is the importance of the query. Alternatively, the higher is the value of random rank for the query, much later it will issued at the interface by the FPE. Thus the queries would be raised in the order q_1, q_2, q_3, q_4 for the first run as can be depicted by the values of `random_rank` ($q_i, 1$) in Table 5.9. Now, when the queries were raised by the FPE in this order, different response pages with varying sizes were retrieved.

Table 5.10 provided a description of each such response page that has been retrieved by issuing the queries q_1, q_2, q_3 and q_4 . For the example under consideration, the page statistics repository has been initialized as in Table 5.10 which contains the URL and sizes of the response pages retrieved by issuing the queries in Table 5.8 but as per their order in Table 5.9:

Table 5.10: The contents of the page statistics repository for the queries

Query ID	Response page	Query Response Size (KB)
q_1		
q_2		
q_3		
q_4		

Now, the ranking of the queries for the next run of the crawler depends on the output or the size of response pages obtained in the current run. The static rank of queries for the next run of the crawler is computed based on the random rank values that were calculated for the immediate previous crawl.

Thus, the static rank of the queries for the second run would be given by the `random_rank` values obtained for the first run which is same as the order in which the queries were issued during the first run of the crawler. This has been shown in Table 5.11. This way the first query q_1 having the value 0.75 for its `random_rank` gets a rank

static_rank ($q_1, 2$)=1, while the query q_4 having the random_rank value as 3.0 is assigned the last rank in the ordered list i.e. static_rank ($q_4, 2$)=4.

$$\text{Random_rank}(q_i, 2) = 0.25 * \text{static_rank}(q_i, 2) + 0.75 * \text{dynamic_rank}(q_i, 2) \quad 5.7$$

where,

$$\text{static_rank}(q_i, 2) = \text{random_rank}(q_i, 1)$$

and

$$\text{dynamic_rank}(q_i, 2) = \text{pos}(q_i, \text{desc_sort}(\text{QRS}(Q)))$$

Table 5.11: The static rank of the queries for the second run of the crawler

Query ID	Static_rank ($q_i, 2$)
q_1	1
q_2	2
q_3	3
q_4	4

The dynamic rank for the queries for the second run have been obtained by ranking the queries in the decreasing order of their Query Response Sizes (QRS) values obtained for each query in the first run. The QRS and the dynamic rank values have been shown in Table 5.12.

Table 5.12: Dynamic rank values for the second run using equation 5.7

Queries	Response page size	Dynamic_rank ($q_i, 2$)
q_1	100	2
q_2	96	3
q_3	108	1
q_4	96	3

By considering the values of $\alpha=0.25$ which gives more weight age to the recent behavior, the computation of the random_rank values as per the equation (5.7) for the second run has been shown in Table 5.13.

Table 5.13: Computation of random_rank for the queries for the second run using equation 5.7

Queries	static-rank (q _i ,2)	dynamic_rank (q _i ,2)	Random_rank (q _i ,2)
q ₁	1	2	2.5
q ₂	2	3	2.75
q ₃	3	1	1.5
q ₄	4	3	3.25

As can be seen from the values of random rank in Table 5.13, the proposed approach issued the queries as per the order q₃, q₁, q₂, q₄. When the queries were issued as per their ascending random_rank values obtained in Table 5.13, a total of 229 unique records were retrieved, though each query independently retrieved 142, 87, 177, 150 records from the hidden database. This has been shown in Table 5.14.

Table 5.14: Number of records retrieved by each query

Queries	Number of records retrieved
q ₃	142
q ₁	87
q ₂	177
q ₄	150

The proposed random ranking approach ensures that by issuing the queries as per their random_rank values, some minimum number of queries will be required to exhaustively retrieve the contents of the target database. Thus, the component, Overlap Statistics Miner has been used to analyze the obtained records retrieved by each query. When the Miner extracted the duplicates and unique records retrieved by each query, it was found that just q₃ and q₁ would suffice to retrieve all the 229 records as the two queries q₂ and q₄ retrieved only duplicates. The Query q₂ retrieved a total of 177 records of which 39 records have already been retrieved by q₁ and 138 records have already been retrieved by q₃. Similarly, the query q₄ fetched 150 records of which 23 records overlapped with q₁ and another 127 were duplicates with q₃. Thus, the proposed approach leads to an optimal solution as per which only two queries would be sufficient for retrieving the same amount of data from the hidden database.

For another next run of the crawler, the static_rank values to be used for the queries will be based on the random_rank values used for the respective query in this run of the crawler. However, to obtain the values of dynamic_rank for the various queries, the QRS that would be obtained from the response pages retrieved by this run would be used. The QRS for the various queries used in the second run are as shown in Table 5.15.

Table 5.15: QRS and the dynamic_rank values for the next run of the crawler

Queries	Response page size	Dynamic_rank ($q_i,3$)
q ₁	92	3
q ₂	98	2
q ₃	88	4
q ₄	102	1

So, the random_rank for the next run would be computed by using equation 5.4 again. These computed values of random_rank to be used for the next run are shown in Table 5.16.

Table 5.16: Computation of random_rank for the queries for the second run using equation 5.7

Queries	static-rank ($q_i,3$)	dynamic_rank ($q_i,3$)	Random_rank ($q_i,3$)
q ₁	2	3	2.75
q ₂	3	2	2.25
q ₃	1	4	3.25
q ₄	4	1	1.75

Hence, the order of queries as suggested by the Query Ranker for the next run would be q₄, q₁, q₂, q₃. Thus, the queries will be raised by the FPE as per the values of

computed `random_rank`. Similarly, the rank among the queries for the subsequent runs of the crawler would be obtained.

The details of implementation and the results of the various experiments that were conducted to evaluate the proposed work of parallel hidden web crawler are discussed in the next chapter of this thesis.

CHAPTER 6.

IMPLEMENTATION & RESULT ANALYSIS OF PARALLEL HIDDEN WEB CRAWLER

6.1. GENERAL

With the rise in the number of Hidden Web databases accessed through search forms, finding efficient ways of exploring contents from these hidden databases is of supreme and increasing importance. One of the methods to access the Hidden Web employs an approach similar to ‘traditional’ crawling but aims at extracting the data residing in databases behind the search interfaces or forms. Moreover, the hidden Web is big and getting bigger. As the volume of information in the hidden-web database grows, there was a need to design a crawler that scales its performance according to the increase in the information on the Hidden Web. The proposed Parallel Hidden Web Crawler addresses these issues by automatically discovering relevant resources in different domains like Books, travel, Auto etc through the analysis of each web page. It then tries to automatically process each search form by raising appropriate queries through an analysis of the response pages and submitting it to the respective web server. The proposed work has been divided to work in six phases:

- 1) The first phase is used to initialize the crawler by choosing a seed set of URLs in each domain. These seeds have been stored in the URL pools respective to its domain which are then scheduled by the URL Scheduler for crawling. This is done by creating a priority queue for each Domain-Specific URL Pool. The URLs from each of these priority queues will act as input to the next phase of the crawler.
- 2) The second phase is responsible for taking the URL from each Domain-Specific Prioritized URL Queues to download the associated web pages. The phase also finds whether the downloaded web pages belong to the Publicly Indexable Web or to the Hidden Web with pages containing search forms. The second phase also helps the crawler to rank the newly discovered URLs as per their relevance in each domain so that important pages get downloaded earlier during the crawl.
- 3) Third phase uses the PIWP given by phase II to create the Domain-Specific

Data Repositories that are further used by Phase V to fill the search forms in order to download the Hidden Web resources. So, a novel approach to organize and classify the downloaded collection of web pages according to their domains like Auto, Books, Food, Travel, etc. is being proposed.

- 4) The form pages discovered in Phase II are passed onto Phase IV where they are analyzed by the Form Analyzer and stored in various Domain-Specific Search Interface Repositories. This facilitates the process of automatically filling the search forms in phase V.
- 5) The fifth phase of the crawler employs a Search Interface Distributor (SID) that is responsible for an even distribution of search forms among the parallel processes for efficiently processing them. The SID is also responsible for filling search forms of each domain with the help of the various Form Processing Elements.
- 6) In the last phase, a novel technique for discovering optimal queries (by analyzing the response pages) to get an optimal outcome has been introduced queries are generated by analyzing the pages retrieved in response to earlier query submissions. This significantly reduces the load on the web servers as only valid queries will be used to fill search forms and any invalid queries would be eliminated. This helps the crawler in getting useful pages and sifting away the pages that contain error messages.

The proposed Parallel Hidden Web Crawler has been implemented using .NET framework 3.0 and SQL Server 7.0. It was mainly written in Java using JDK 1.4.2 but it also includes software components implemented in native code i.e. in other programming languages, such as C, C++ but can be inter-operated with java code using the JAVA Native Interface (JNI) .To check the performance of the proposed work, various metrics have been used that are discussed in the next section.

6.2. PERFORMANCE METRICS

The standard measures of performance in the area of Information Retrieval are: *Precision*, *Recall*, and *F-measure*. So, the performance of the proposed parallel hidden web crawler is measured via these three metrics. For the purpose of analyzing the performance, consider the following terms:

- C, the number of valid or correct search form submissions.
- I, the number of invalid or incorrect search form submissions and

- N, the number of search forms that could not be filled and submitted by the proposed processed parallel hidden web crawler.

With the help of above defined terms C, I and N, Precision, recall and F-Measure can be defined as:

- 1) *Precision* is defined as a fraction of correct form submissions over all the form submissions by the Parallel Hidden Web Crawler. Mathematically, the *Precision* is given by

$$P = C / (C + I) \quad 6.1$$

- 2) *Recall* is defined as a fraction of correct form submissions over all the search forms given to

- 3) the system for processing by the crawler, then the *Recall* of the proposed parallel hidden web crawler crawling system is given by the expression given in equation 6.2

$$R = C / (C + I + N) \quad 6.2$$

- 4) *F-measure* incorporates both precision and recall. *F-measure* is given by

$$F = 2PR / (P + R) \quad 6.3$$

where Precision P and Recall R are equally weighted.

6.2.1. DATA SETS

For experimental evaluation of the proposed work, the following four domains have been considered: Auto, Books, Food and Travel. The proposed crawler is initialized by taking about 140-150 URLs for each domain from the list provided under the DMOZ open dir

ectory that were stored in the various Domain-Specific URL Pools.

In phase I, each initial list of URLs that is stored in the various Domain-Specific URL Pools is given as input to the URL Scheduler for deciding the order in which the URLs must be fetched from the WWW. The order is decided by the URL Scheduler based on the rank assigned by the URL Ranker (phase II) to each URL. During the initial run of the crawler, the rank of any URL is considered to be the same as in the directory listing. The URL appearing at the topmost position in the directory list takes the top rank for scheduling also. Based on these rank values, the URL Scheduler creates a prioritized URL queue for the URLs in each domain. A sample of the initial URL queue for the Food domain contained in the text file is as shown in Figure 6.1.

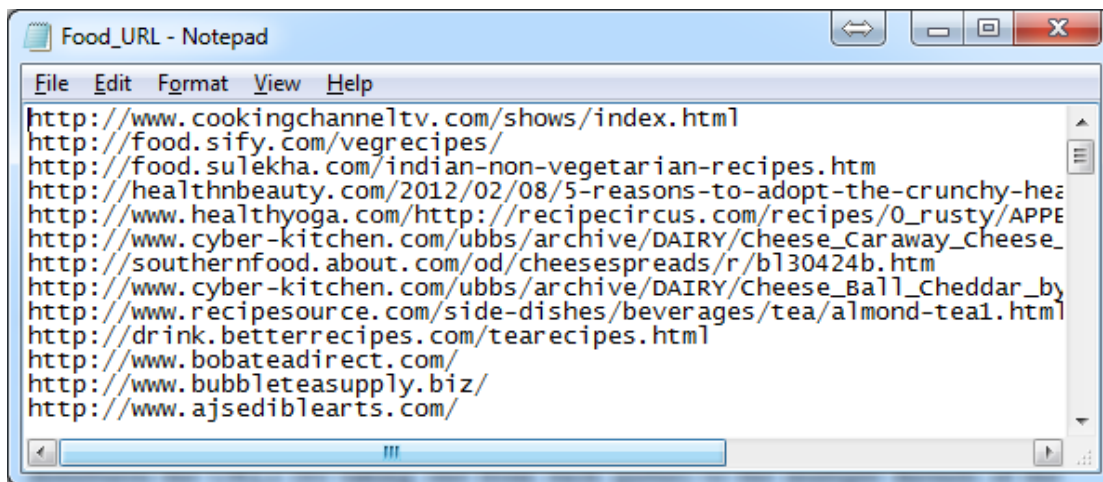


Figure 6.1: Initial list of URLs for 'Food' domain

Similarly, URLs were taken for other domains and arranged in order in the respective URL Queues (.txt files). These text files or queues were given as inputs in each domain as shown in the interface of the crawler in Figure 6.2.

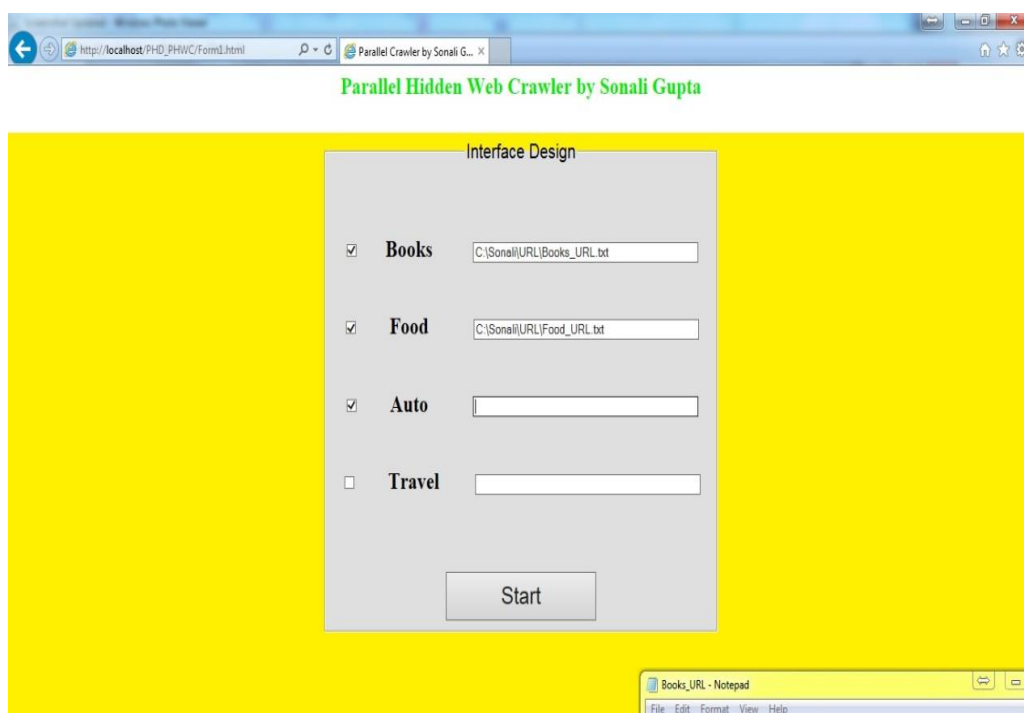


Figure 6.2: Interface of the Parallel Hidden Web Crawler

Based on these URLs, web page have been retrieved in each domain which was then analyzed to measure the performance of the proposed system by using the above-defined metrics, *Precision*, *Recall* and *F-measure*. A detailed discussion on the performance of the various components is given in the following sections.

6.3. EXPERIMENTS & RESULTS

6.3.1. FINDING THE FORM PAGES AND THE PIW PAGES

In the next phase, the URLs that were ready for downloading in the various Domain-Specific Priority Queues Pools are allocated to the URL Allocator that uniformly distributes the URLs (by taking one from each queue) to the multiple threads of the document loader. The Multi-threaded Document Downloader then downloads the web pages by establishing the http connections to the designated web servers.

As soon as the web page was downloaded, it was given as input to the Web page Analyzer that employs a parser and a page identifier. The parser parses the web page to extract the useful content like *<anchor>*, *<form>*, *<meta>*, *<title>* etc. from the HTML tag structure and the page identifier finds and separates the form pages from the PIW pages.

All the functionality of phase II has been incorporated in the Web Data Extractor module of the crawler implementation which has been implemented by using the .NET framework with Java technology. The interface of the web data extractor of phase II is shown in Figure 6.3. Multiple threads of the Web Data Extractor can be executed simultaneously. This can be done by increasing the thread count in the "New Session - Other" tab on the interface.

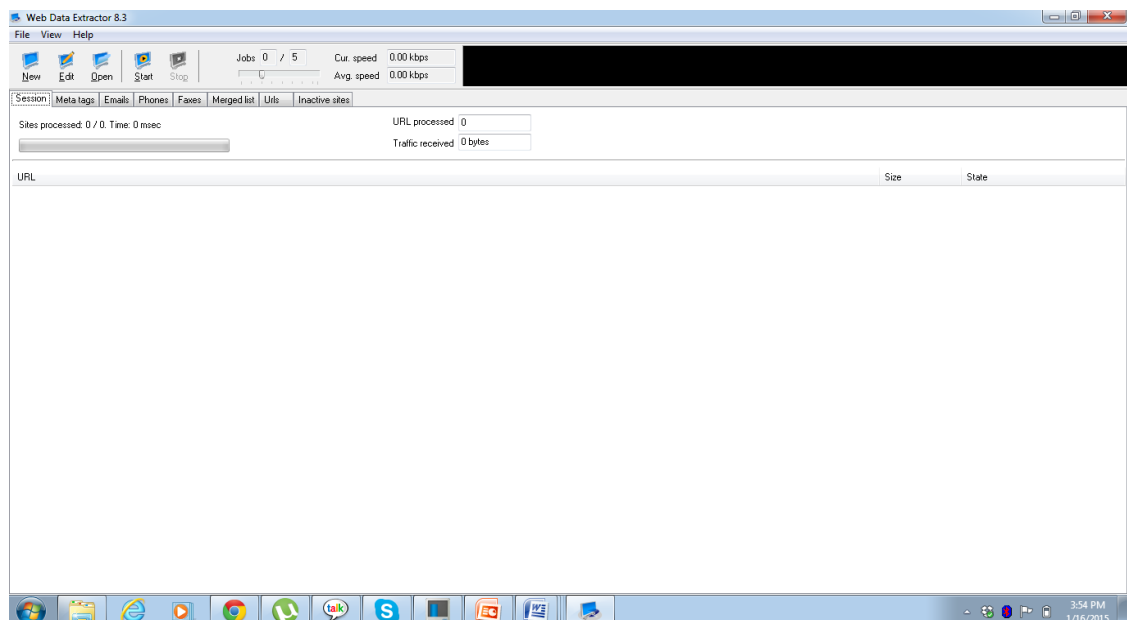
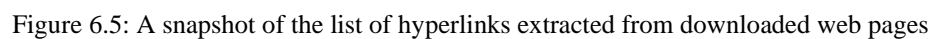


Figure 6.3: Interface of the Web Data Extractor employing URL Allocator, Multi-Threaded Document Loader and the Web Page Analyzer.

A session is started by clicking “New” by which a window pops-up asking for the prioritized URL queues created by the URL scheduler as input. To extract the URLs



During crawling, it is not necessary to add the newly found URLs to the Domain-Specific URL Pools each and every time

<META> tag. A total of 565 pages were downloaded by the Multi-threaded Document Downloader and were given to the Page Type Identifier (after parsing) that classified them as per the data in Figure 6.6:

Total Pages downloaded	PIW Pages	Form Pages
565	182	383

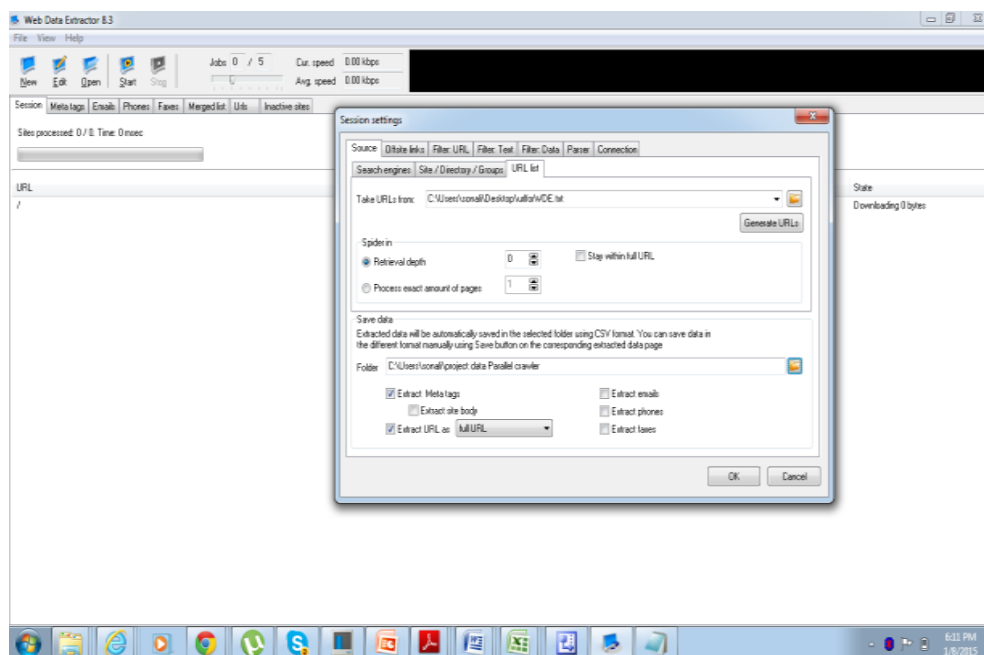
Figure 6.6: The number of pages of each type as identified by the Page Type Identifier

Each of the PIWP and the Form page, immediately after analysis by the Web page identifier is passed respectively to the Page Classifier used in Phase III and the Form Analyzer used in Phase IV for further functioning of the crawler

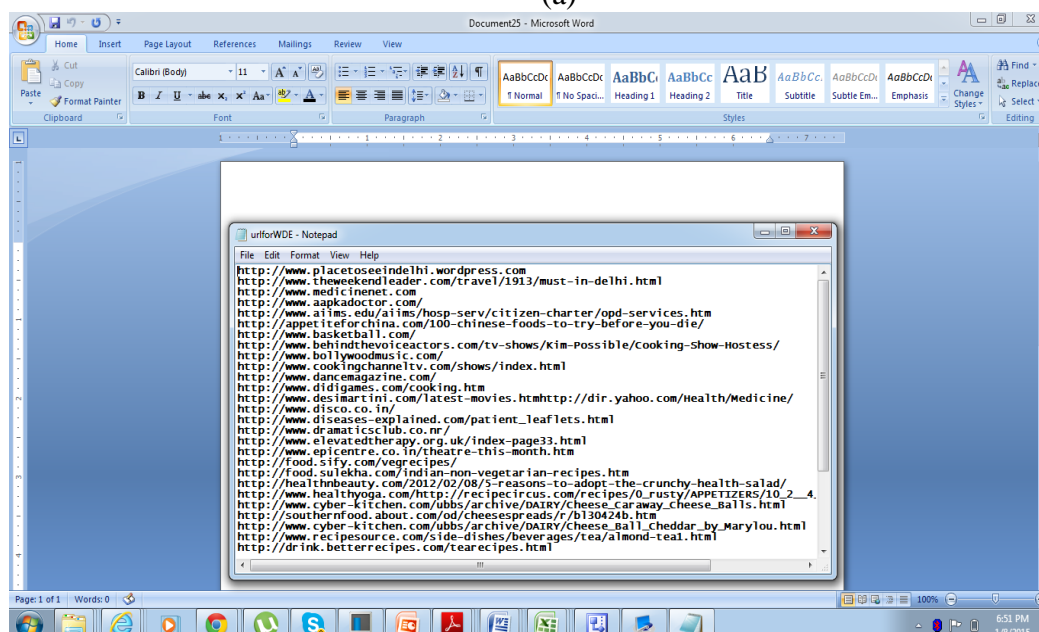
6.3.2. THE PAGE CLASSIFIER AND THE PAGE CONTENT EXTRACTOR

All the pages that were identified as PIWPs by the Web Page Analyzer in Phase II are passed onto the Page Classifier for classifying and organizing them according to their domains, in Phase III. The Page Classifier is designed with the help of a back-propagation neural network that is configured and trained by extracting key terms in each domain with the help of a Tag Extractor. The open source software *Neural Network Toolbox* that uses MATLAB environment is used to implement the functionality of the used neural network.

The keyterms for training are those that are extracted from the <meta> keywords, <meta> description and <title> of the web pages whose domain information is already available. The URLs of such web pages are passed in the form of a plain text file with one URL starting at each line. This file has been generated by the Web Data Extractor module in the implementation and is referred by the name “urlforWDE.txt”. The interface of the Web Data Extractor and a sample of the generated file to be used as input for extracting key terms is shown in Figure 6.7.



(a)



(b)

Figure 6.7: Initialization of the key term extraction process in (a) with a sample of the input file urlforwde.txt in (b).

To start with a session, the file “urlforwde.txt” is given as input, a snapshot of key term extraction after the session has started is shown in Figure 6.8.

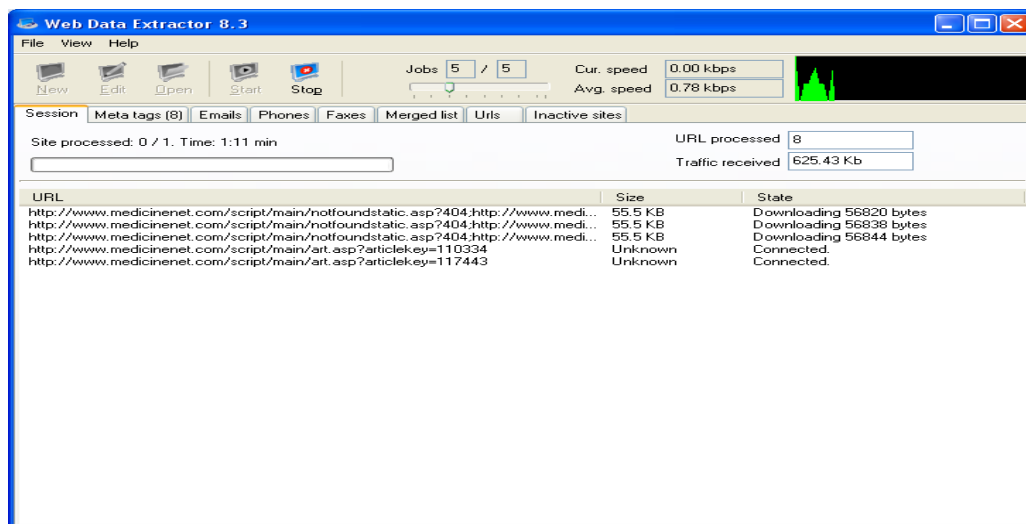


Figure 6.8: A snapshot during term extraction

The extracted keyterms using the <meta> keywords, <meta> description and <title> are as shown in Figure 6.9.

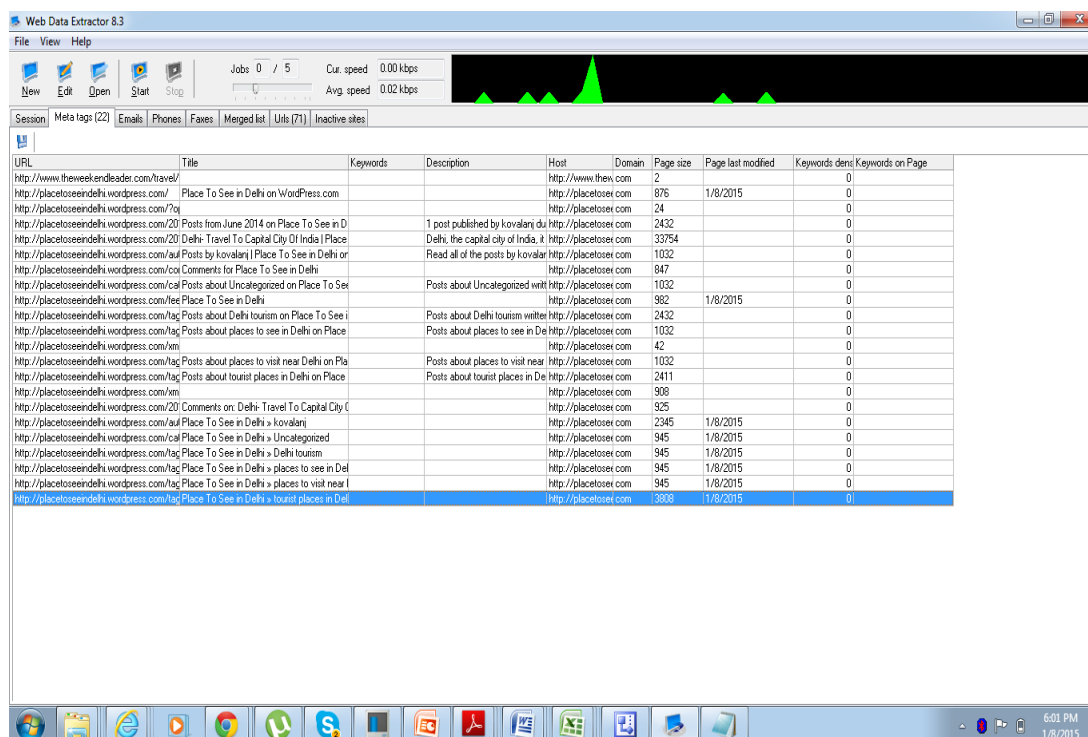


Figure 6.9: Snapshot of extracted title and keywords from the META tag.

These extracted keyterms from all the web pages whose domains were available act as input for training the neural network. The proposed system uses a back propagation neural network model as a basic network that has been designed with three layers, one input layer, one output layer and one hidden layer as shown in Figure 4.7. Table 6.1 illustrates the biases used at each of the three layers and illustrates the various activation functions that have been used by the proposed system.

Table 6.1: Biases and Activation Functions Used

Biases		Activation Functions	
Number of Neurons in Input layer	4	For Input layer	Piece-wise linear
Number of Neurons in Hidden layer	5	For Hidden layer	Sigmoid
Number of Neurons in Output layer	1	For Output layer	Sigmoid
		Error criteria used	Mean Squared Error
		Target accuracy	0.00000015

The network was trained to zero error in sixteen epochs, as shown in Figure 6.10.

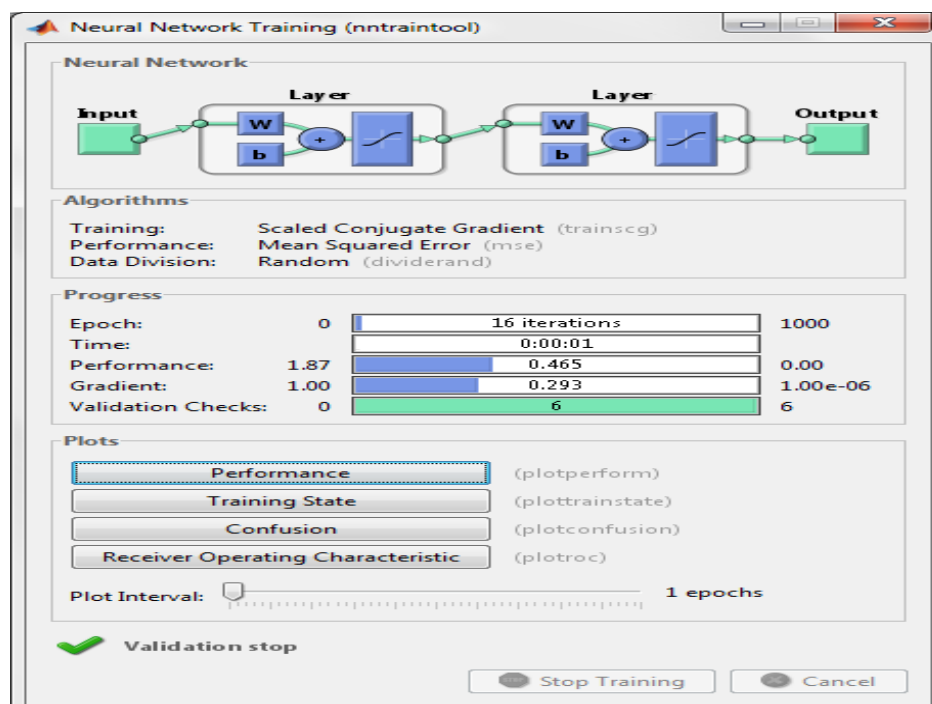


Figure 6.10: Training the neural network

By Clicking on 'Performance' plot button, the mean error squared can be found. This is depicted in the graph shown in Figure 6.11. The graph shows the value of the

performance function versus the iteration number. It can be seen from the graph that the mean squared error of the network starts at a large value but keeps on decreasing showing that the network is learning. The plot has three lines where blue is used to depict the training set, green to validate how well the network generalized and red represent the network's generalization to data that it has never seen (test set). Training continues as long the network's error keeps on reducing for the validation data. For the purpose of implementation this validation data (with minimum error) has been referred to as Domain Definitions which include a set of key terms for each domain so as to uniquely define that domain. These Domain Definitions helps in actually identifying the documents belonging to each such domain.

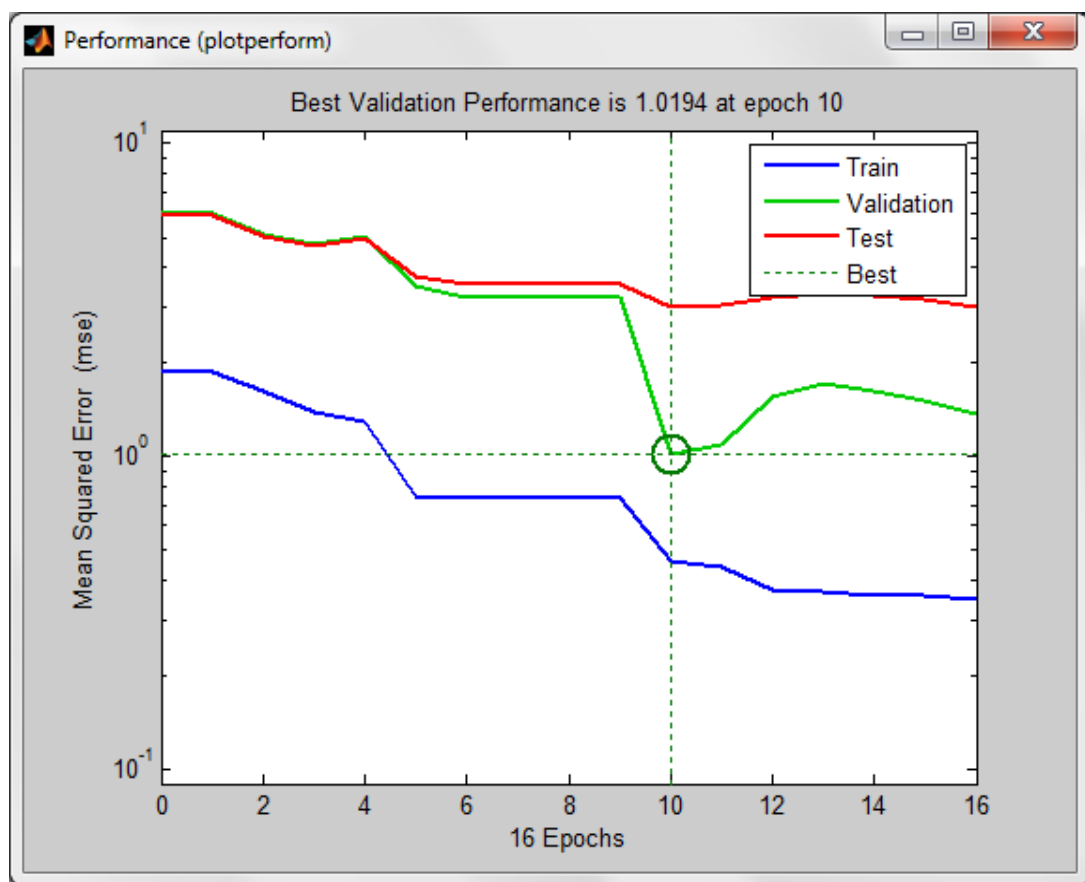


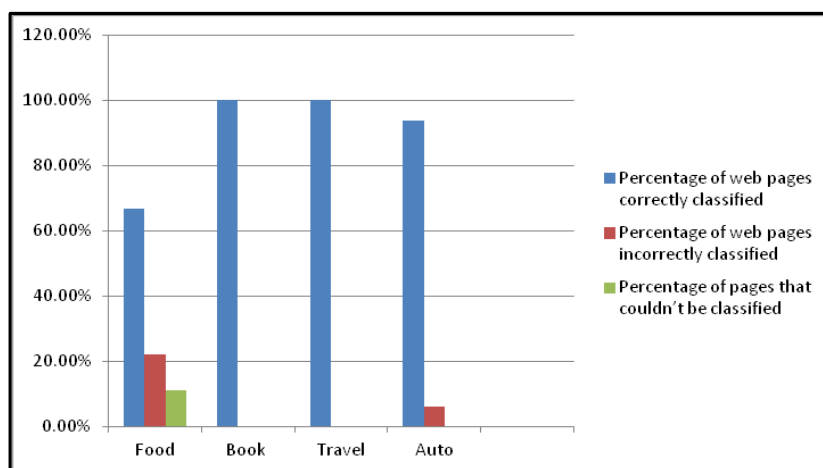
Figure 6.11: Performance of the Neural Network (error vs iteration)

s that are assigned to those key terms.

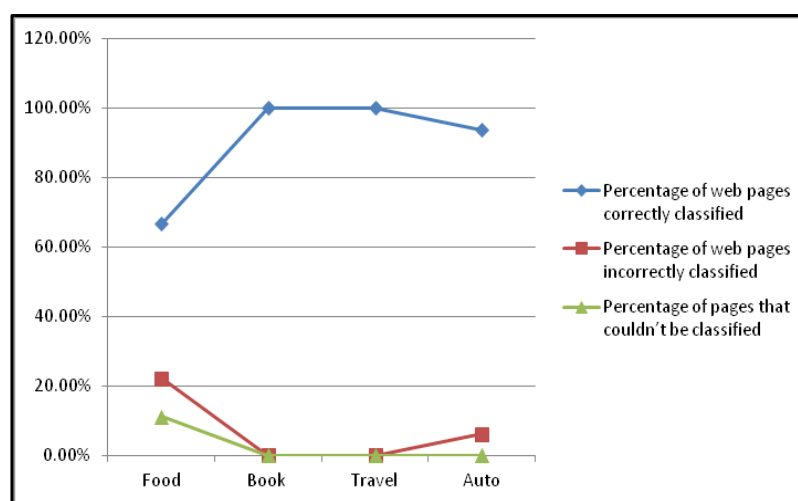
Table 6.2: Experimental Results Obtained for the Page Classifier

Domain of web pages	Percentage of web pages correctly classified	Percentage of web pages incorrectly classified	Percentage of pages that couldn't be classified
Food	66.67%	22.22%	11.11%
Books	100%	0%	0%
Travel	100%	0%	0%
Auto	93.75%	6.25%	0%
Average	89.75%	7.69%	2.56%

The graphs in Figure 6.12 (a) and (b), represent the performance of the proposed page classifier. On the X-axis are labeled the various domains of consideration by the proposed system whereas the Y-Axis represents the percentage of web pages that are classified either correctly or incorrectly or could not be classified by the Page Classifier.



(a)



(b)

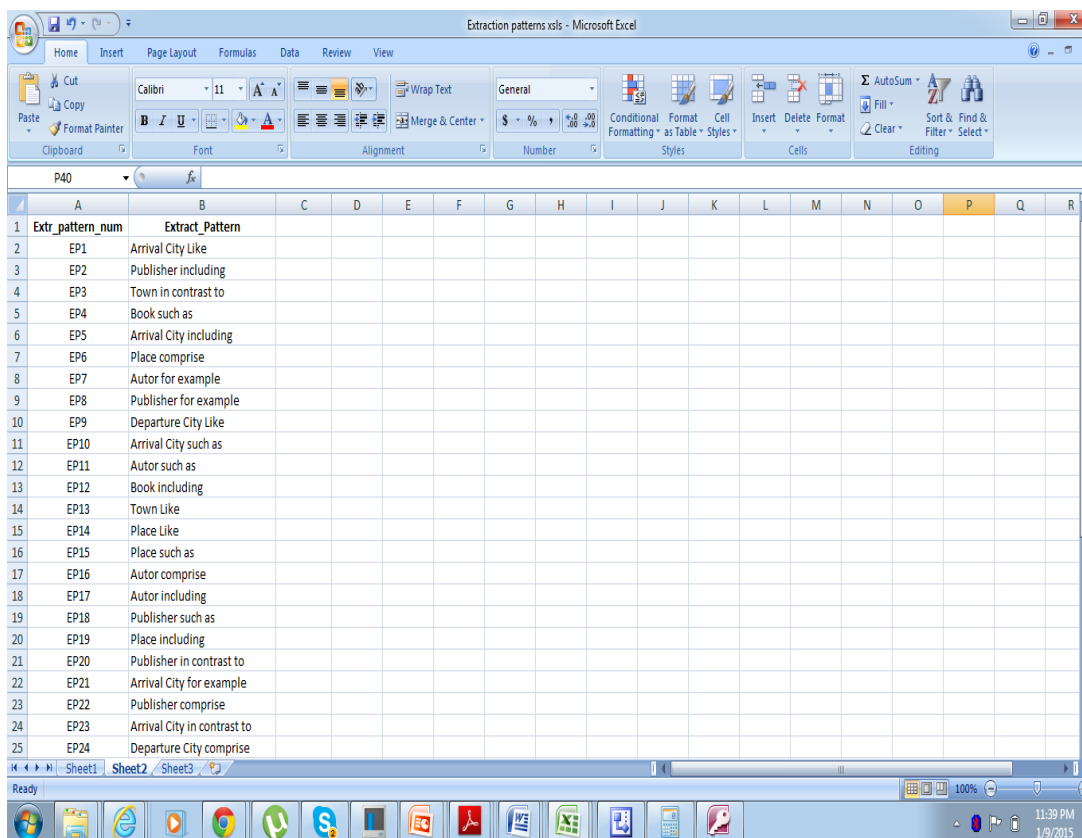
Figure 6.12: Performance of the Page Classifier

The performance of the Page Classifier is low in the 'Food' domain, the reason behind which is as follows: Though web pages contain useful features as discussed above but, these features are sometimes missing, misleading, or unrecognizable for various reasons in some particular web pages. For example, web pages that belong to the Food domain typically contain large images or flash objects but little textual content. In such cases, it becomes difficult for classifiers to make reasonable judgments based on features available on the web page.

To store the classified collection of web pages as per their domains, the Page Classifier creates the various Domain-Specific Page Repositories. i.e. each web page is stored in the Domain-Specific Page Repositories (DSPRs) respective to its domain. The web pages contained in each such repository were then passed as input to the Page Content Extractor (PCE) that will extract the labels and values for creating the Domain-

Specific Databases for filling the search forms in Phase V. The Page Content Extractor works by extracting values from the various instances of the different extraction patterns occurring in the repositories.

These extraction patterns have been generated in the proposed system with the help of labels extracted from the search forms and the various clusters generated by the clustering tool used with the neural network. For an extracted label ‘departure city’ six patterns were formed such as “*departure city like*”, “*departure city for example*”, “*departure city such as*” etc. based on the type *EP1*, *EP2*, *EP3* respectively. As departure city forms a part of the cluster {city, place, town, destination} in Travel domain, thus a total of 24 extraction patterns were formed by using a single label ‘departure city’ and the cluster {city, place, town, destination} i.e. six patterns for each cluster element based on the defined six type of extraction patterns. More than 100 extraction patterns were generated for each domain by the Extraction Pattern Generator in the form of an excel file, a small sample of which is shown in the snapshot in Figure 6.13.



Extr_pattern_num	Extract_Pattern
EP1	Arrival City Like
EP2	Publisher including
EP3	Town in contrast to
EP4	Book such as
EP5	Arrival City including
EP6	Place comprise
EP7	Author for example
EP8	Publisher for example
EP9	Departure City Like
EP10	Arrival City such as
EP11	Author such as
EP12	Book including
EP13	Town Like
EP14	Place Like
EP15	Place such as
EP16	Author comprise
EP17	Author including
EP18	Publisher such as
EP19	Place including
EP20	Publisher in contrast to
EP21	Arrival City for example
EP22	Publisher comprise
EP23	Arrival City in contrast to
EP24	Departure City comprise

Figure 6.13: The generated Extraction Patterns in the Travel Domain

When these extraction patterns were raised as queries on general purpose search engine like Google [117], various candidate instances were retrieved for the extracted labels which were validated against their occurrences in the web pages of the Domain-Specific Page Repository to find the set of valid values for the label. Figure 6.14 shows a sample page from the PIW when the extraction pattern “departure city like” was raised as query on Google [117] by Page Content Extractor.

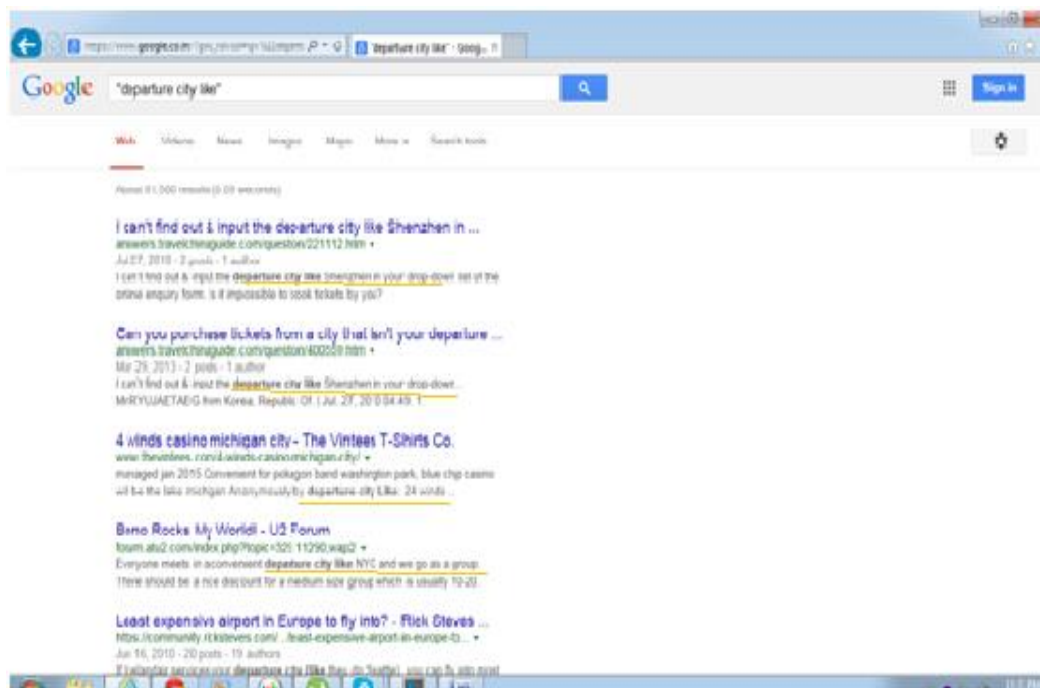


Figure 6.14: Result page when the extraction pattern ‘departure city like’ was raised on Google.

Similarly, all the extraction patterns were raised as queries for extracting the candidate instances from the PIW pages. But as not all the candidate values are of use practically, the set of valid values from the candidate set need to be found. This is done by assigning a score to each candidate value based on its frequency of occurrence in the PIW pages stored in the DSPR respective to the domain of the search form from which the label L_s was extracted.

Figure 6.15 show a snapshot when the retrieved values were analyzed for their occurrences in the PIW pages in the DSPRs.

WordAnalysis : Database (Access 2007) - Microsoft Access

Security Warning: Certain content in the database has been disabled. Options...

All Access Objects: Tables (Pattern_Agg_Count, Pattern_Count, Pattern_Word_Ind_Count, Reject_Word, WebPageInformation), Queries (Final_Analysis)

Webpage	Departure City like	Word	Count Patten	Count Indivi
Webpage1	Departure City like	"cruise	1	1
Webpage1	Departure City like	"Instant	1	1
Webpage1	Departure City like	...	2	16
Webpage1	Departure City like	160	1	1
Webpage1	Departure City like	9.CruiseCritic.c	1	1
Webpage1	Departure City like	about.	1	1
Webpage1	Departure City like	Airways?	1	1
Webpage1	Departure City like	Amsterdam	1	1
Webpage1	Departure City like	answers.travel	1	1
Webpage1	Departure City like	city	1	18
Webpage1	Departure City like	complain	1	1
Webpage1	Departure City like	departure	1	18
Webpage1	Departure City like	drop-down	1	2
Webpage1	Departure City like	flights	1	1
Webpage1	Departure City like	Frankfurt,	1	1
Webpage1	Departure City like	Germany	1	1
Webpage1	Departure City like	group.	1	1
Webpage1	Departure City like	heading	1	1
Webpage1	Departure City like	Hiroshima,	1	1
Webpage1	Departure City like	Jul	1	2
Webpage1	Departure City like	know	1	2
Webpage1	Departure City like	Kyoto,	1	1
Webpage1	Departure City like	line,"	1	1

Datasheet View: Record: 1 of 51, No Filter, Search

Num Lock, 11:08 PM, 1/9/2015

Figure 6.15: Snapshot of .mdb file when analyzing the instances of candidate values in PIW pages stored in the DSPRs.

The set of valid values was generated by using the algorithm devised for the Pattern Searcher

candidate value in a small sample of web pages from the page repository of Travel domain.

Table 6.3: A sample of candidate values with their occurrences in the web pages in the DSPR

Web page#	Ls= Departure City	NYC	London	Rome	Through	Heading	Tokyo	Frankfurt	Milan	Group	Amsterdam	Kyoto	Osaka	Shen zhen	Switzerland
1	8	2	1	0	1	0	0	0	0	0	0	0	0	0	0
2	0	1	4	2	0	0	0	0	0	1	0	0	0	0	0
3	0	1	3	2	0	0	0	0	0	1	0	0	0	0	0
4	6	3	8		2	0	0	0	0	0	0	0	0	0	1
5	42	35	63	16	3	0	21	17	15	4	29	0	1	1	0
6	6	0	2	10	0	0	0	1	9	0	1	0	0	0	1
7	7	0	0	0	3	1	0	0	2	0	0	0	0	0	0
8	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
9	11	0	0	1	0	2	0	0	0	2	0	0	0	0	0
10	4	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Total Occurrences	86	44	82	31	9	3	21	18	26	8	30	0	1	1	2

The threshold frequency for the values of a label would be then computed by

$$TH_Freq(Ls) = \frac{\sum_{k=1}^i F(cv_k)}{i} \quad 6.4$$

Using the above formula as stated in the PSV Algorithm given in Figure 4.18, where cv_k is denotes any candidate value and $F(cv_k)$ denotes its frequency of occurrence in each page of DSPR. Therefore,

$$\begin{aligned}
 TH_Freq(\text{departure city}) &= \frac{44 + 82 + 31 + 9 + 3 + 21 + 18 + 26 + 8 + 30 + 0 + 1 + 1 + 2}{17} \\
 &= \frac{326}{17} = 19.714
 \end{aligned}$$

Thus, the threshold value of 19.7 is used to filter the useless values from the candidate set of values for the label ‘departure city’. All the values having the number of occurrences more than this value of threshold was considered for inclusion in the task-specific database for the Travel domain. Thus, the following nine values were added in the task-specific database: *NYC, London, Rome, though, Tokyo, Frankfurt, Milan,*

group, *Amsterdam*. For the same label and the various extraction patterns generated by the EP generator, candidate values from all the web pages in the same domain are extracted similarly by the Page Content Extractor. Thus, the Task-specific database is populated automatically during the process through repeated extraction of values from the web documents by the PCE.

The label and its corresponding set of valid values were added into the respective Domain-Specific Database. The performance of the Page Content Extractor is evaluated in terms of the number of valid value generated from the set of candidate values extracted by searching each extraction pattern. Table 6.4 shows the number of candidate and valid values that were obtained for the label ‘departure city’ by raising the six extraction patterns of type EP1, EP2,...EP6.

Table 6.4: Number of candidate values and valid values generated for the label ‘departure city’ using the six extraction patterns.

Extract_Pattern	Number of Candidate values	Number of Valid Values	%age of Valid Values
Departure City Like	77	47	61
Departure City for example	17	8	47
Departure City such as	23	7	30.4
Departure City comprise	8	5	62.5
Departure City including	32	4	12.5
Departure City in contrast to	6	1	16.6
Total	163	73	44.7

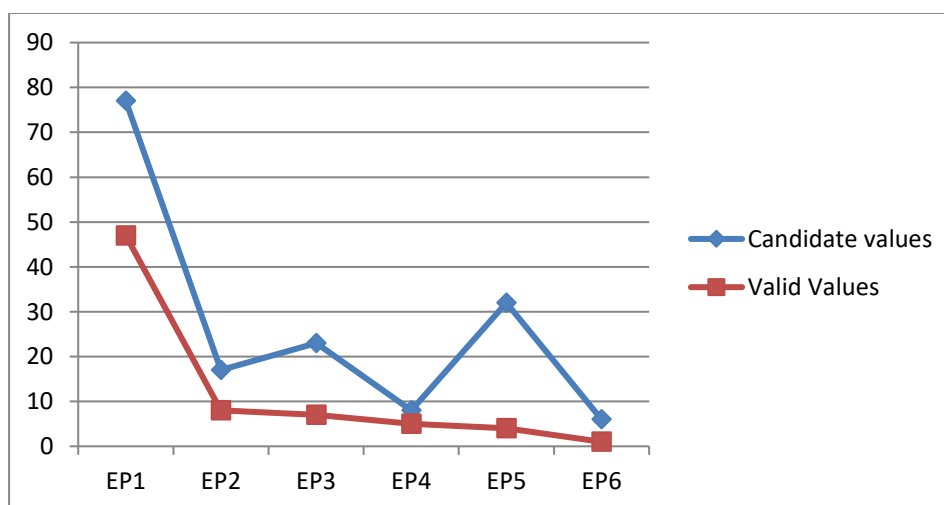


Figure 6.16: Graphical representation of the number of candidate and valid instances by the PSV

So, the Pattern Searcher and Validator helps in significantly improving the performance of the crawler by suggesting a set of valid values only. This cuts on the number of unnecessary requests that would be otherwise made to the web server if all the extracted candidate values were used to fill the search forms. For the label ‘departure city’, a total of 163 candidate values were extracted, only 73 of which were found to be the actual valid instances i.e. 91 or 55.3% irrelevant values were filtered from being included in the Domain-Specific Database under the label ‘departure city’ for the Travel domain. This is represented by the graph in Figure 6.16.

A similar set of valid values is generated for each of the various labels that would be extracted by the label extractor of the Form Analyzer. Each extracted label with its set of corresponding valid values is stored in the respective Domain-Specific Database. Hence, this phase helps in creating the various Domain-Specific Databases that are required to fill in the search forms when the crawler reaches its phase V during execution.

6.3.3. FORM ANALYZER

For all the form pages received from phase II, the Form analyzer first extracts the labels and values from the given search form. And, then performs matching among the extracted labels/ values and the key terms of each domain definitions to predict the domain of relevance of the form page or hidden web resource.

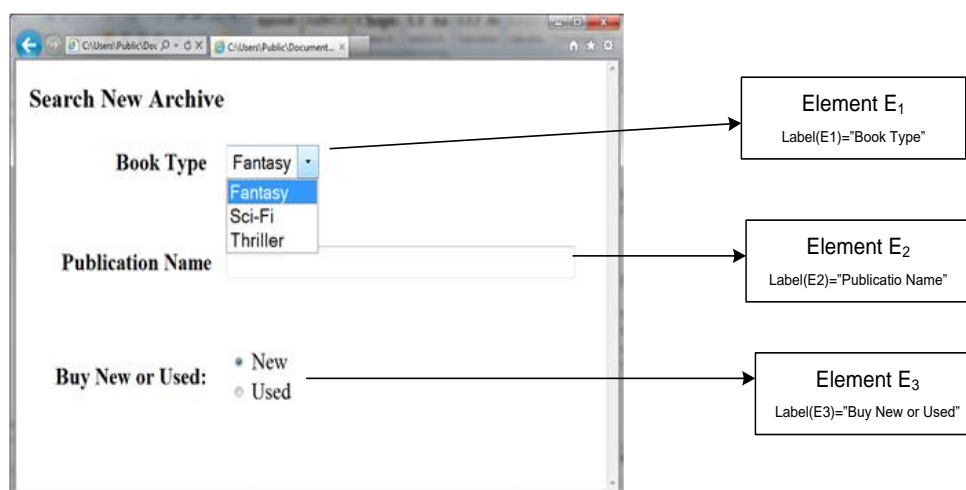


Figure 6.17: A Sample Search Interface with Labels and Values

An example of a typical search form with its control elements and labels is shown in Figure 6.17. The form in the Figure also displays certain values like ‘Fantasy’, ‘Sci-Fi’, ‘New’ and ‘Used’ for the control element E_1 . Such controls are called as bounded

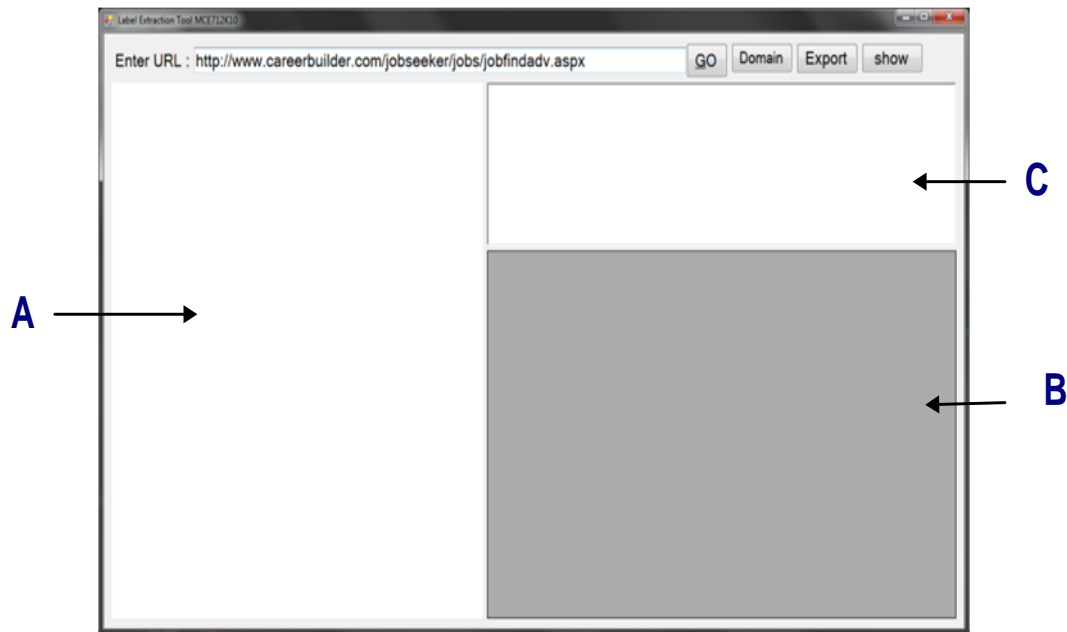


Figure 6.19: The Interface of the LET.

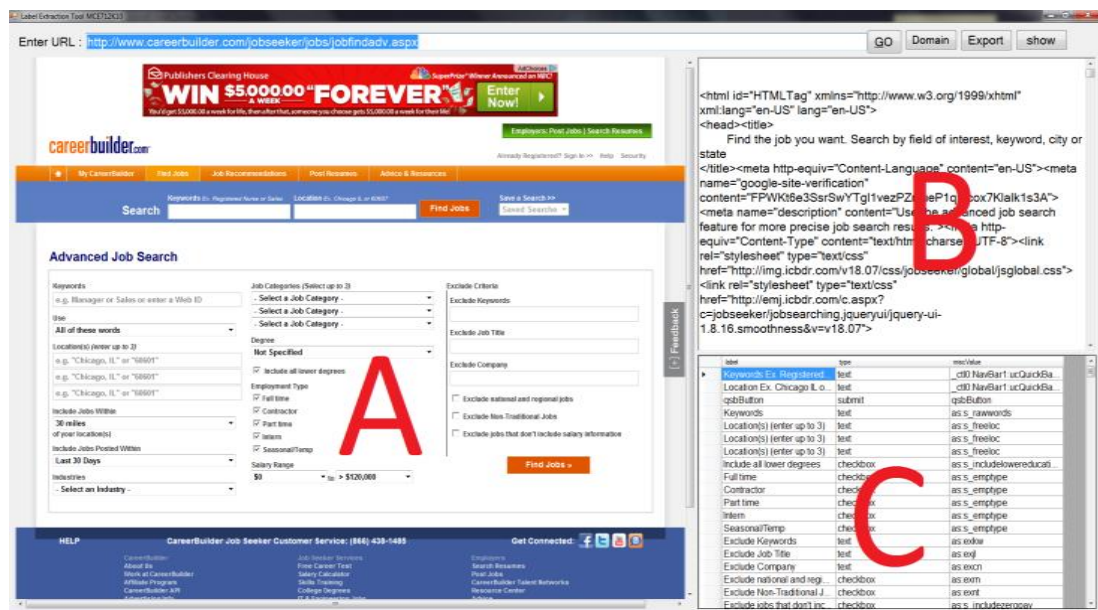


Figure 6.20: Layout of the Form Analyzer during the process of label extraction.

For all the search forms that were given as input to the Form Analyzer in each domain (Auto, Books, Food, Travel), labels were extracted for the control elements. The Form Analyzer was able to extract 65.13% of labels from the search forms in Auto Domain. Similarly, 54.54%, 50.09% and 64.59 % of labels were extracted from the search forms in Books, Food and Travel domain respectively as shown in Figure 6.21.

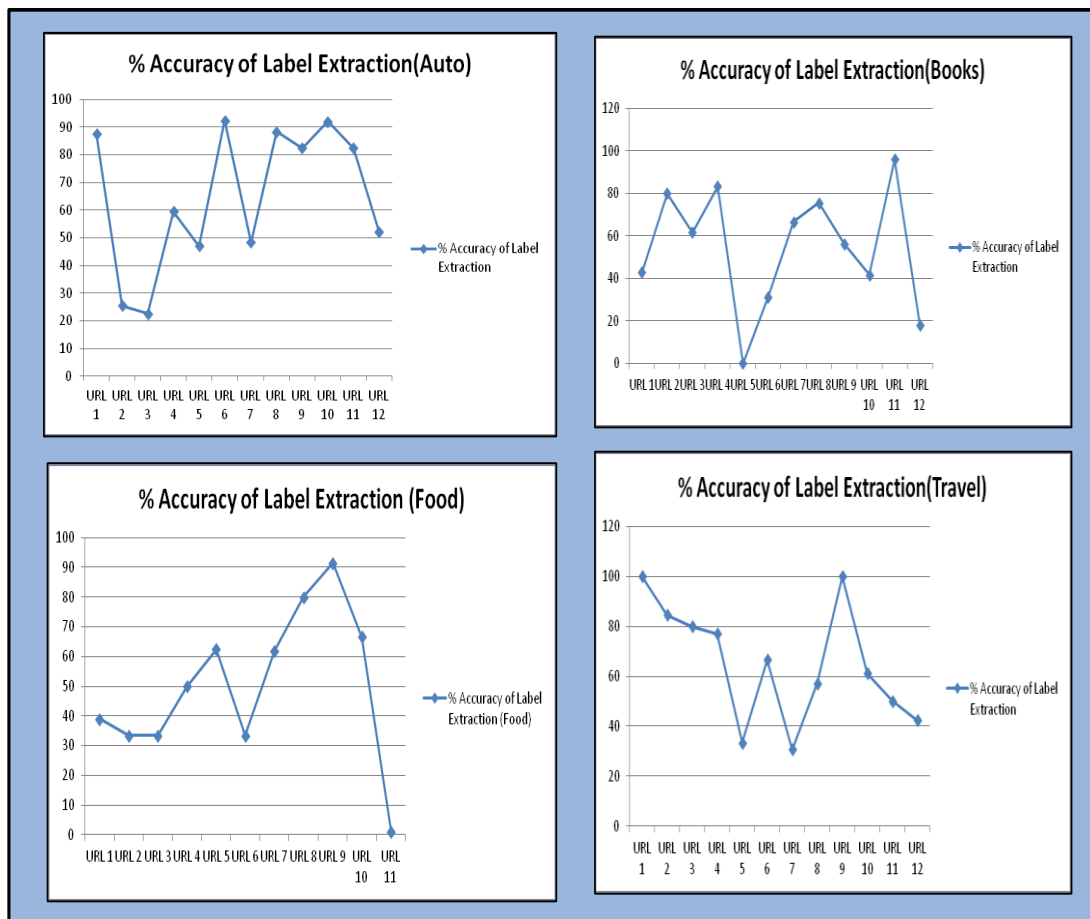


Figure 6.21: Performance of Form Analyzer when extracting labels.

The various labels that were extracted from a given search form are temporarily stored for analysis in an excel spreadsheet as shown in Figure 6.22.

	A	B	C	D	E	F	G	H	I	J	K
1	http://www.abebooks.com/	Actual Label on SI	Label Extracted	Input Type	Misc.Values	Default	Matching				
2	Books(AUTHOR)	Author:	Author:	text	an		TRUE				Actual No Of Labels
3	Books(TITLE)	Title:	Title:	text	tn		TRUE				5
4	Books(ISBN)	Keyword:	Keyword:	text	kn		TRUE				Labels extracted Matching
5	Books(BOOK)	what's this? ISBN:	what's this? ISBN:	text	isbn		TRUE				4
6	Books(BOOK)			ComboBox			FALSE				Accuracy: Labels extracted/ Actual Labels present
7		Select Another List:	Select Another List:	Option	default		TRUE				80
8		AbeBooks' Bestsellin	AbeBooks' Bestsellin	Option	abe-bestsell		TRUE				Domain Identified
9		AbeBooks' Bestsellin	AbeBooks' Bestsellin	Option	abe-signed-k		TRUE				Books
10											Number of Values Entrained
11											8
12											Number of Values on FORM Match
13											7
14											
15											Accuracy with labels
16											87.5
17											

Figure 6.22: Snapshot of the temporary sheet created during label extraction

The Form Analyzer shows the best performance for label extraction in the Auto domain as the search forms in this domain had only few controls like Make, Model etc. The total number of labels extracted and the total number of labels that existed on search forms in each domain is shown in the graph in Figure 6.23(a) whereas the percentage accuracy of extracted labels against the actual labels that exist on the search form in each domain is shown in Figure 6.23(b).

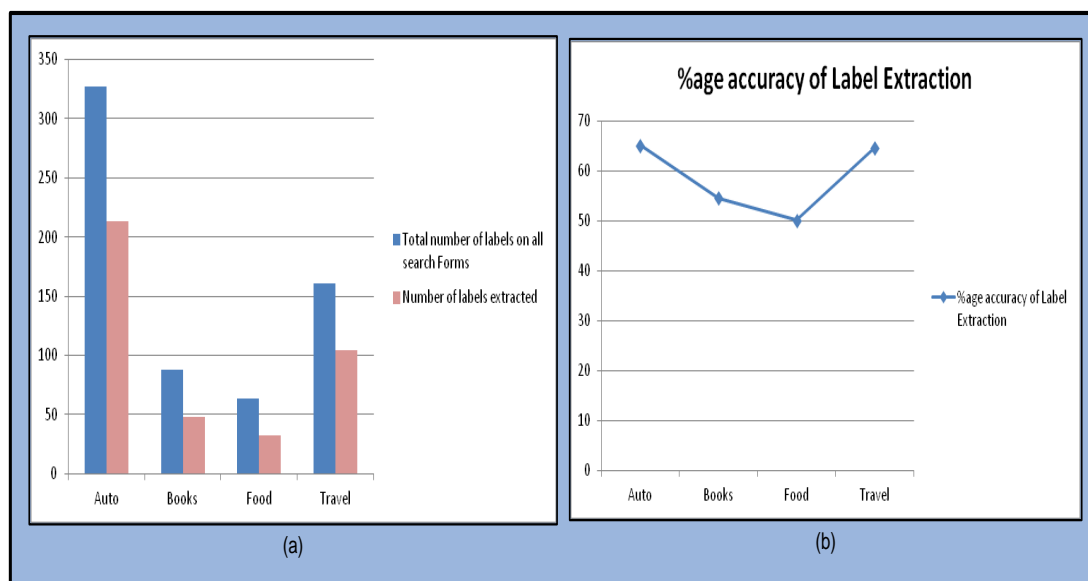


Figure 6.23: The number and Percentage of Labels Extracted in each domain

To further improve the performance of the Form Analyzer, values for controls were also extracted. The method for label extraction used by the form analyzer was tuned to extract the values of the control elements with finite domain, as their values can be depicted from within the search form itself, like the SELECT tag that has OPTIONS and INPUT, RADIO tag that has values which are selected while filling the form manually/automatically. The INPUT defines areas that can be edited in the form, the Attribute TYPE of the INPUT control further describes the type as text, checkbox, radio, submit etc. The control SUBMIT of the INPUT type is generally used to submit the values filled in the form. Every control has a name attribute which is used to appoint control's label. SELECT is used to create a drop-down list box or a multi-choice list box. TEXTAREA is used to create a text box which can input multiple lines of words.

Figure 6.24 shows the domain-wise performance when just values (and not labels) were extracted by the Form Analyzer from the various search forms. The Form

Analyzer was able to extract 88.97 % of the values in Auto domain, 28.04 % of values in Books domain, 65.95% in Food domain and 94.29% in Travel domain.

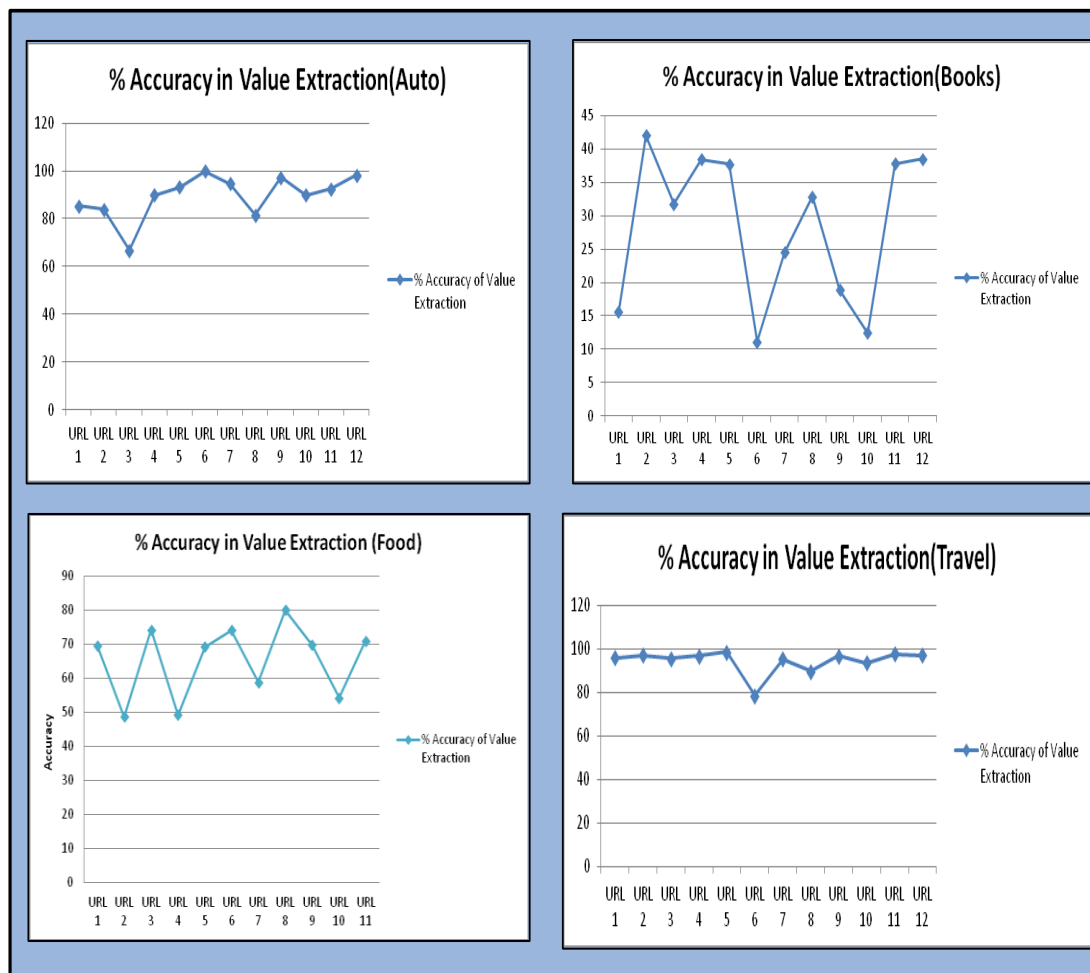


Figure 6.24: Performance of Form Analyzer when extracting values.

As shown in Figure 6.25(a), of all the search forms in Travel domain, a total of 314 values were extracted from a total of 333 values that existed on those search forms. Moreover, the Form Analyzer was able to extract 121 values from all the 136 values that were present on the search forms in Auto domain giving an accuracy of 88.97%. For other domains Books and Food, the Form Analyzer was able to extract 23 and 31 values from the respective total of 82 and 47 values that were present on the search forms in these domains.

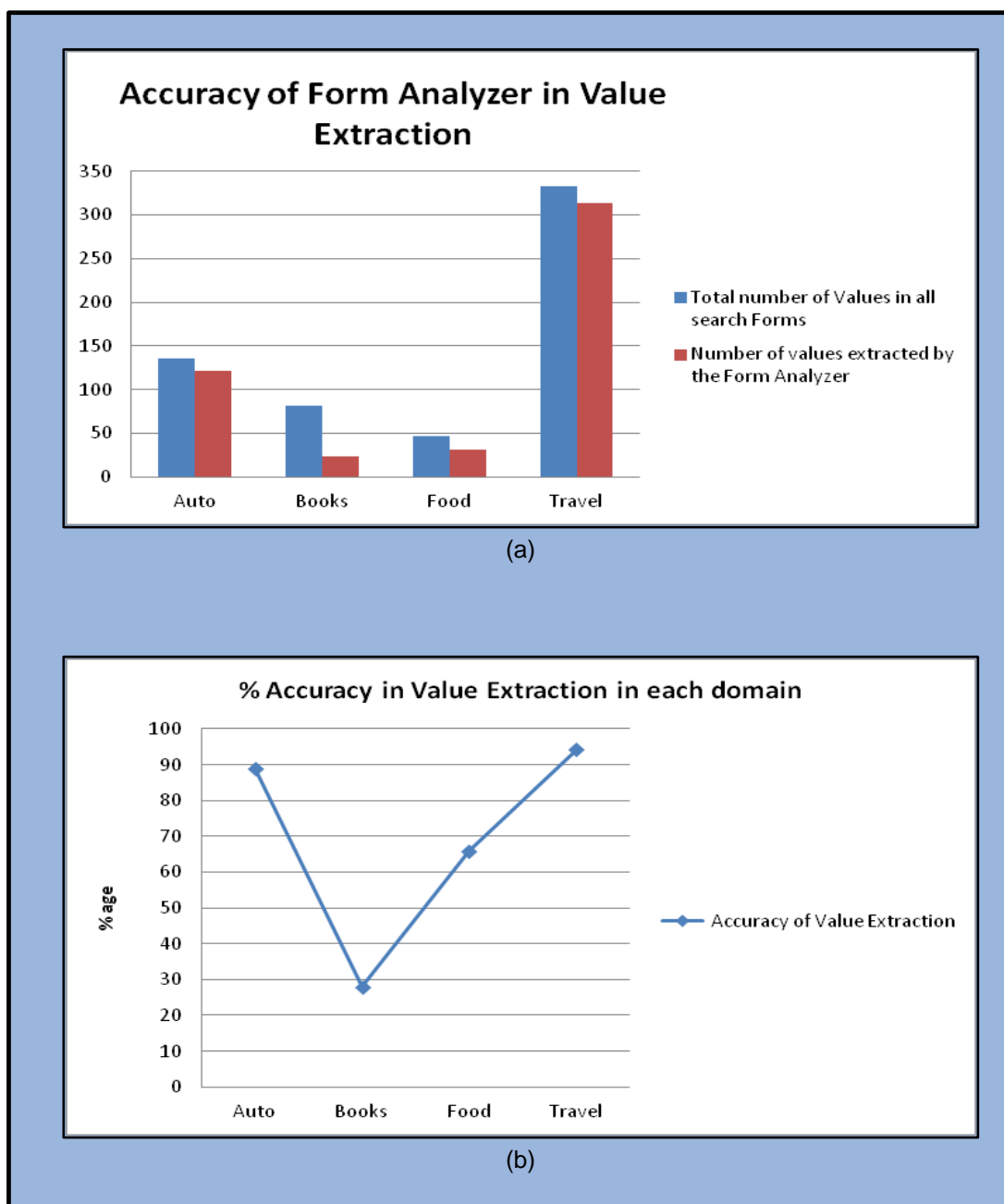


Figure 6.25: Performance of the Form Analyzer when extracting values of control elements.

The Form Analyzer performed its best when extracting the values/instances from search forms in Travel domain (as shown in Figure 6.25 (b)) as control elements on the search forms from this domain typically consisted of drop-down boxes listing all possible values that can be taken by that control while filling the search form.

The average accuracy of the Form Analyzer for the four domains Auto, books, Food and Travel when both labels and values were extracted from the search forms is shown in Figure 6.26.

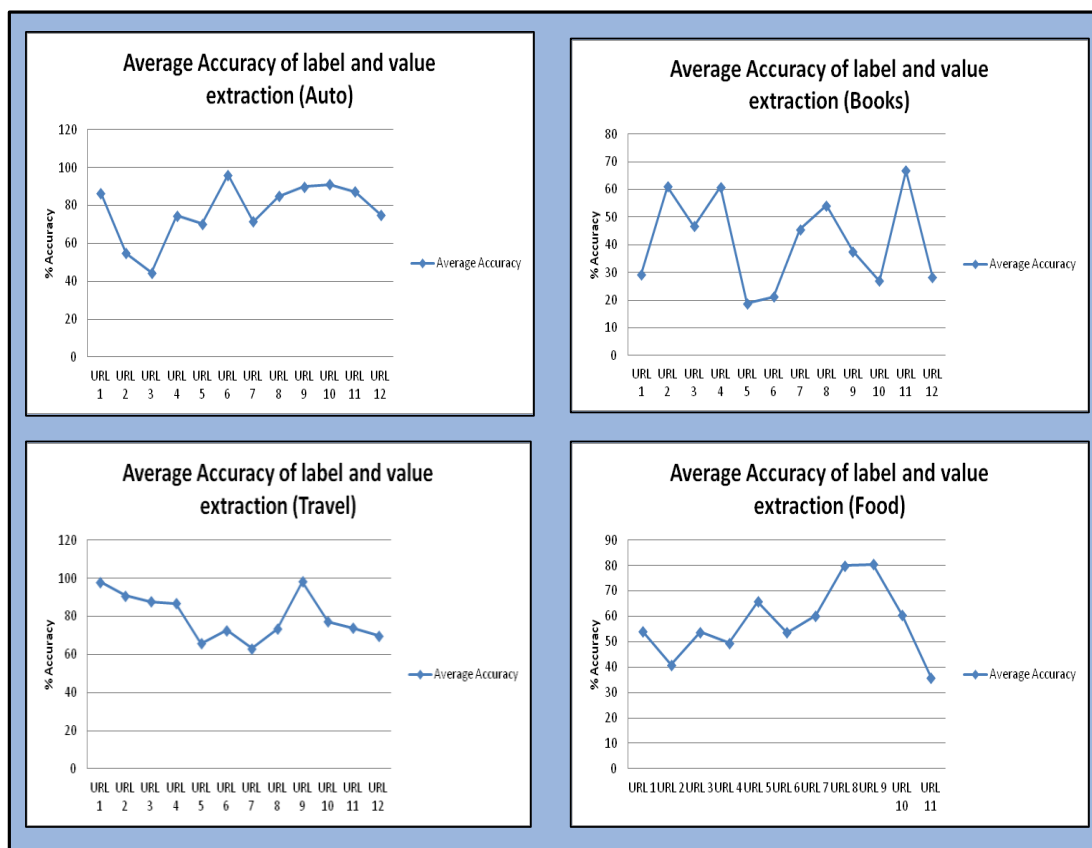


Figure 6.26: Domain Wise performance of Form Analyzer when extracting labels and values both.

The overall percentage accuracy of the Form Analyzer in each of the four domains for extracting the labels and values together is shown Figure 6.27.

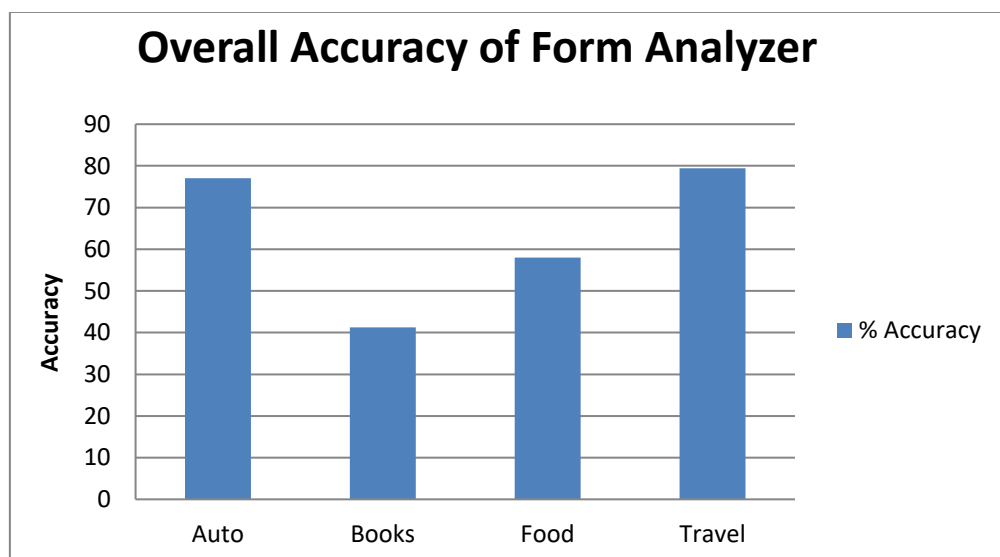


Figure 6.27: Overall %age Accuracy of Form Analyzer when extracting labels and values.

As shown in Figure 6.27, the Form Analyzer shows the best performance for the search forms in Travel domain.

When the various search forms were given as input to the Form Analyzer in phase IV, the extracted labels were matched against the domain definitions generated by the page Classifier (in Phase III) to predict the relevant domain of the hidden web resource.

Based on the label extraction process and the Match logic, the Form Analyzer was able to discover the various search forms in each domain. Of the total 383 search forms received by the crawler in its phase IV, 369 were found to be queryable. When these 369 forms were analyzed, 322 were discovered as relevant search forms in one or the other domain but could not predict the relevant domain of 27 search forms. The observed data is shown in table 6.5.

Table 6.5: Results of the Form Analyzer

Total # of search forms in the Hidden Web	369
# correctly discovered relevant search forms	299
# incorrectly discovered search forms	43
# search forms that could not be classified	27

Also, among the 342 that were discovered as relevant search forms, almost an equal number of resources belonged to each domain like Auto, Books, Food and Travel. But when the results were analyzed, a total of 43 search forms (from 342) were incorrectly organized in a Domain-Specific Repository. The graph in Figure 6.28 shows the accuracy of the Form Analyzer in discovering relevant search forms.

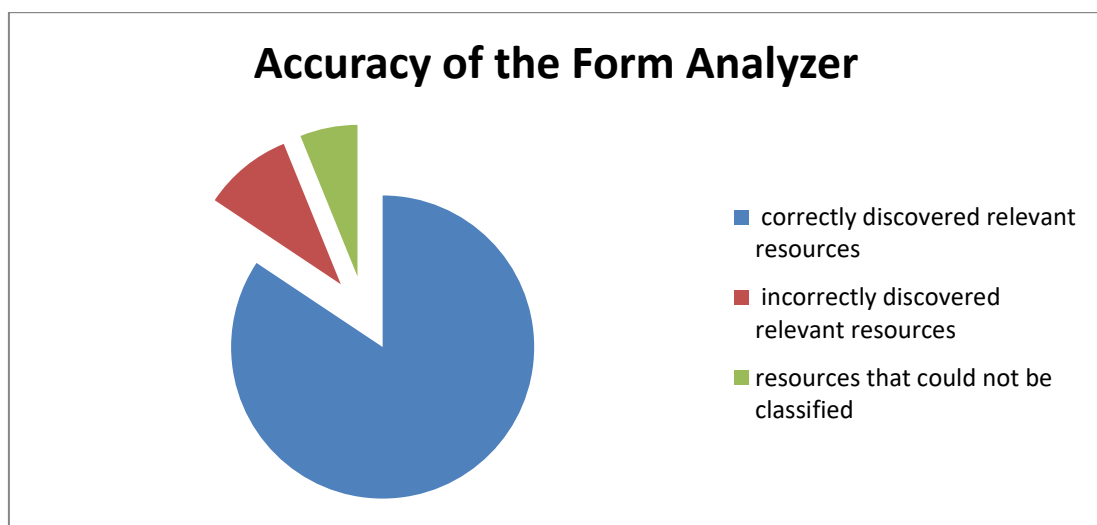


Figure 6.28: Accuracy of the Form Analyzer while discovering search forms in various domains

The reason behind this incorrect classification might be the match logic that depends on the use of a Domain-Specific Thesaurus in the Matching library which might have been built manually or automatically. So, using the Match logic, there are likely to be some search forms which cannot be definitely classified for their relevant domains.

Based on the extracted labels, the Extraction Pattern Generator is used to create the various extraction patterns needed by the Page Content Extractor for finding the valid values of the extracted labels as done by the crawler in phase III.

The analysis of the various search forms by the Form Analyzer helps the proposed crawler to discover the hidden web resources for each domain. Also, each search form is stored in its respective Domain-Specific Search Interface Repository from where it is assigned to one of the parallel form processing element by the Search Interface Distributor in Phase V.

6.3.4. PARALLEL PROCESSING OF SEARCH FORMS

After these various Domain-Specific Databases and the search interface repositories were created in Phase III and Phase IV, the next step of the proposed crawler is to process in parallel, the search forms stored in each of these domain-specific repositories. In other words, the search forms were processed in parallel within and among domains. The search forms must be processed in parallel within and across different domains.

The top ranked query in the Domain-Specific Database (as suggested by the query ranker) is used to fill in all the search forms respective to that domain. Figure 6.29(a) shows a search form from Travel domain that is processed by the query From= Delhi, To= Mumbai and Search Flight= Round-trip as suggested by the Query Ranker. Figure 6.29(b) shows the corresponding dynamic web page retrieved in response of this form submission.

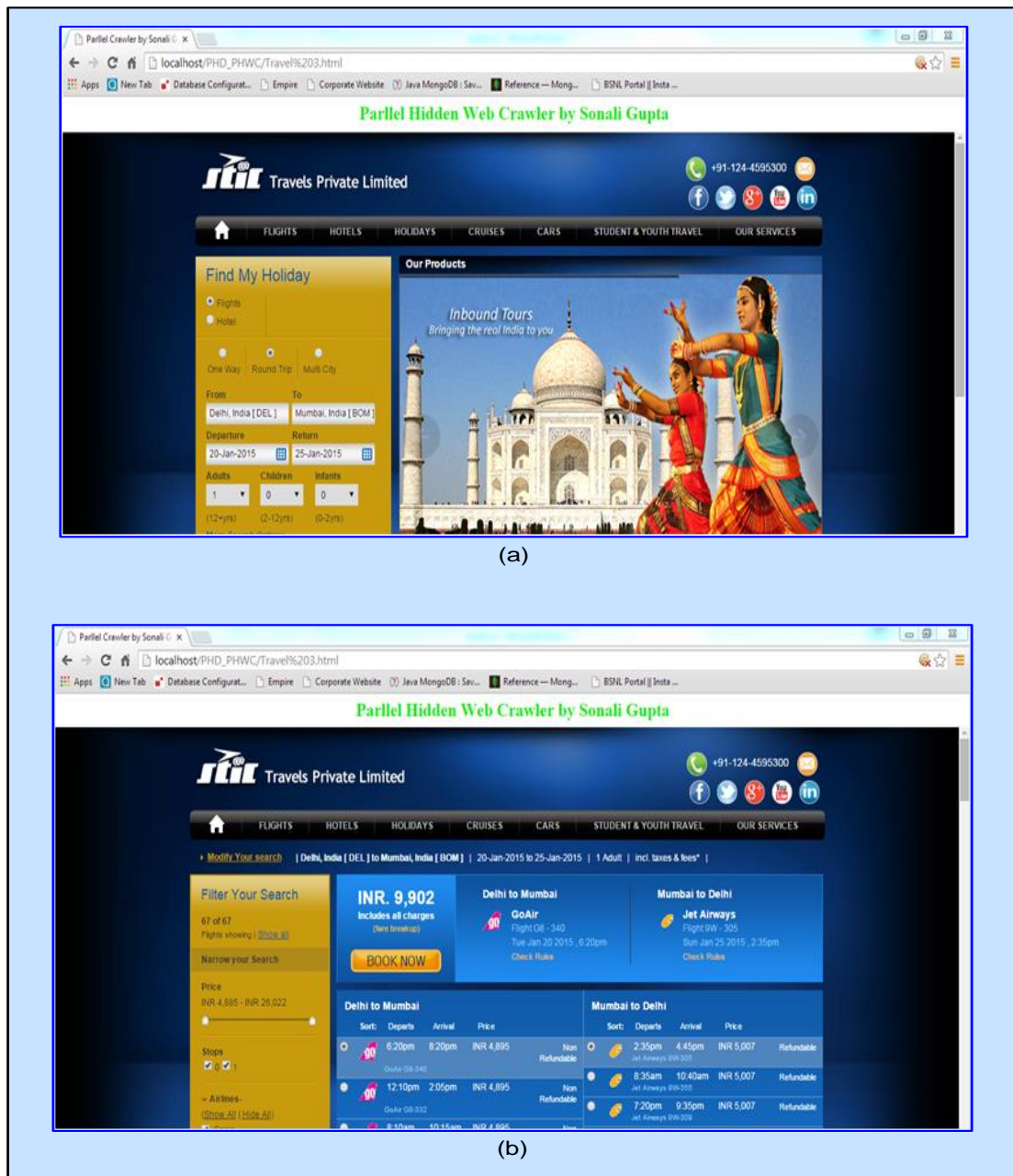


Figure 6.29: Sample search form from travel domain that is processed and the corresponding response page retrieved

Figure 6.30 and 6.31 shows a snapshot of the crawler's task of filling the search forms in parallel along different domains. Figure 6.30 shows that all the forms from the Travel domain when processed in parallel, are filled using the same query (flights making a round-trip from Delhi to Mumbai on the specified dates) and Figure 6.31 shows the response pages that were downloaded when the corresponding search forms were processed.

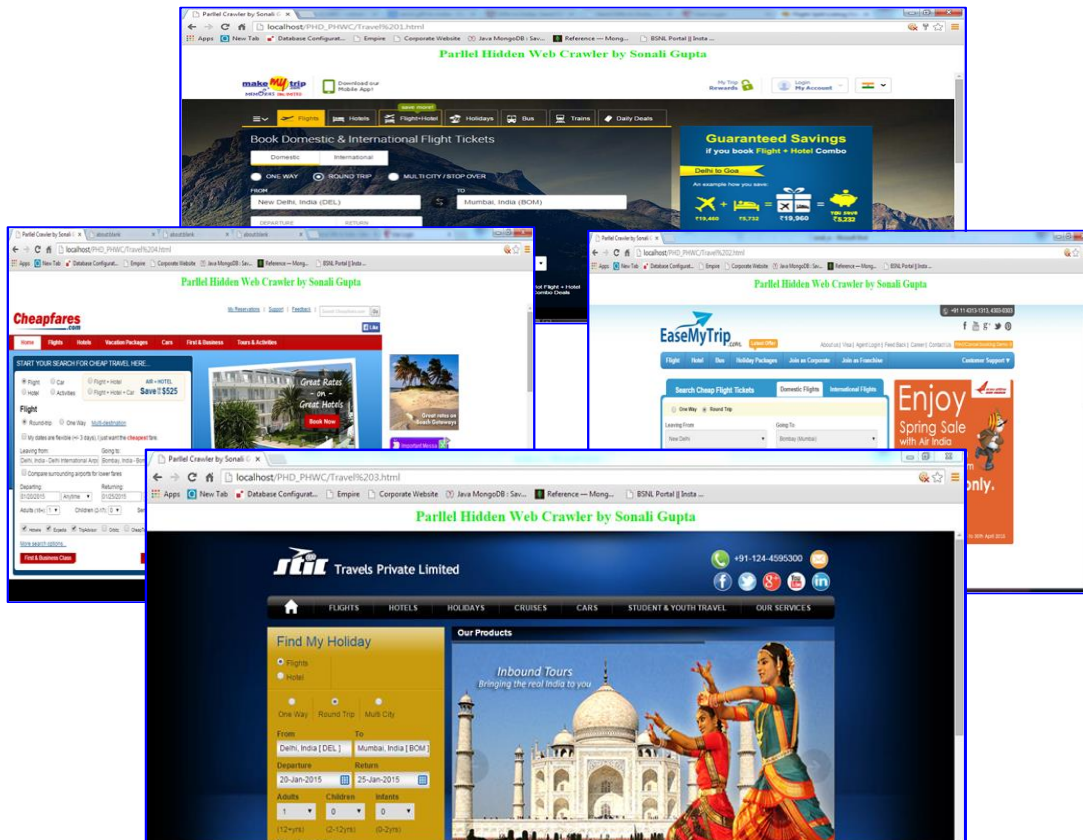


Figure 6.30: Parallel Processing of search forms in Travel domain with the query round-trip flights from Delhi to Mumbai on specified dates.

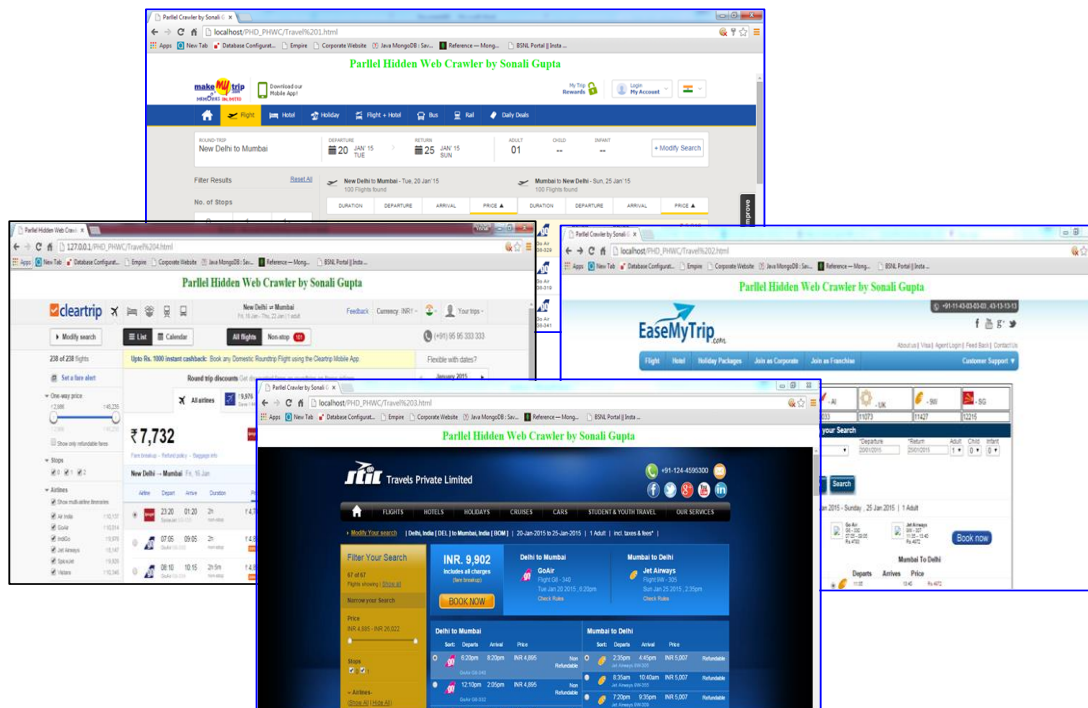


Figure 6.31: Parallel downloading of response pages from the Hidden web databases behind the search forms in Figure 6.30.

In the same way, the search forms from all the other three domains (Auto, Books and Food) are processed with the help of the FPEs by using the optimal queries as suggested by the Query Ranker in each domain.

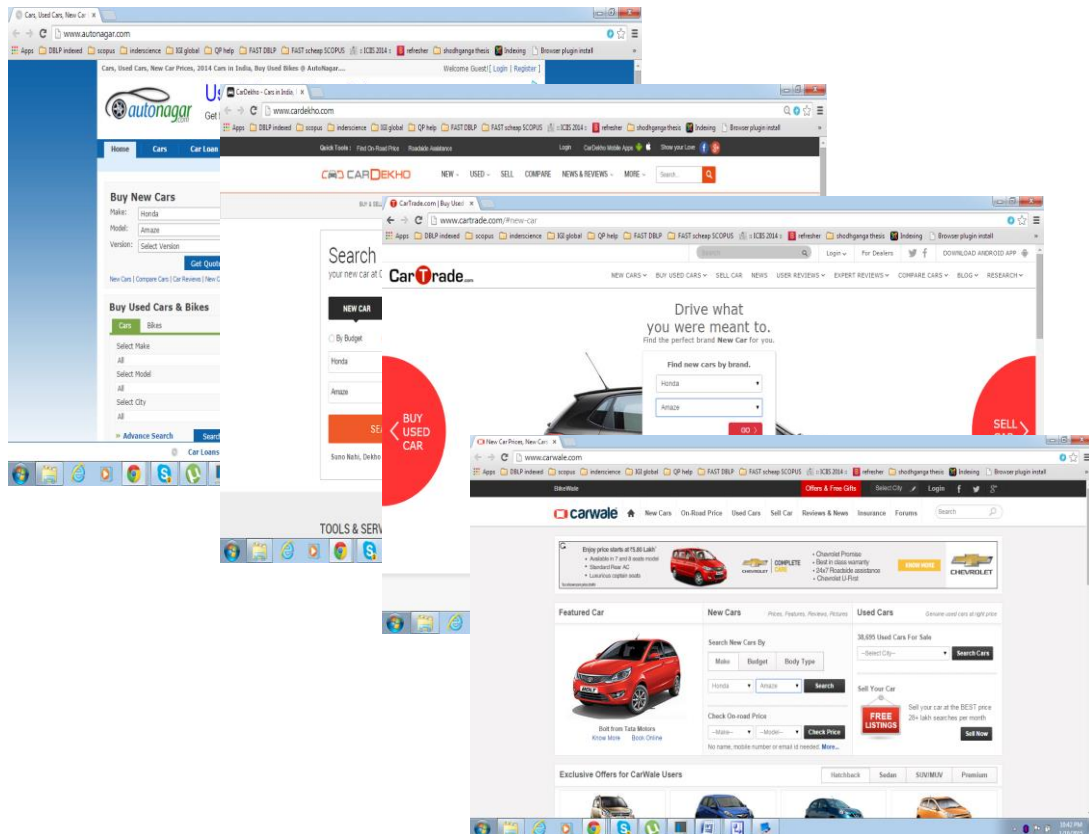


Figure 6.32: Parallel Processing of search forms in Auto domain.

Figure 6.32 shows the filled forms from the Auto domain when the search was made for the car Honda Amaze whereas Figure 6.33 shows the downloading of response pages in parallel when the search forms in Figure 6.32 were processed. Similarly, search forms from other domain were also processed in parallel.

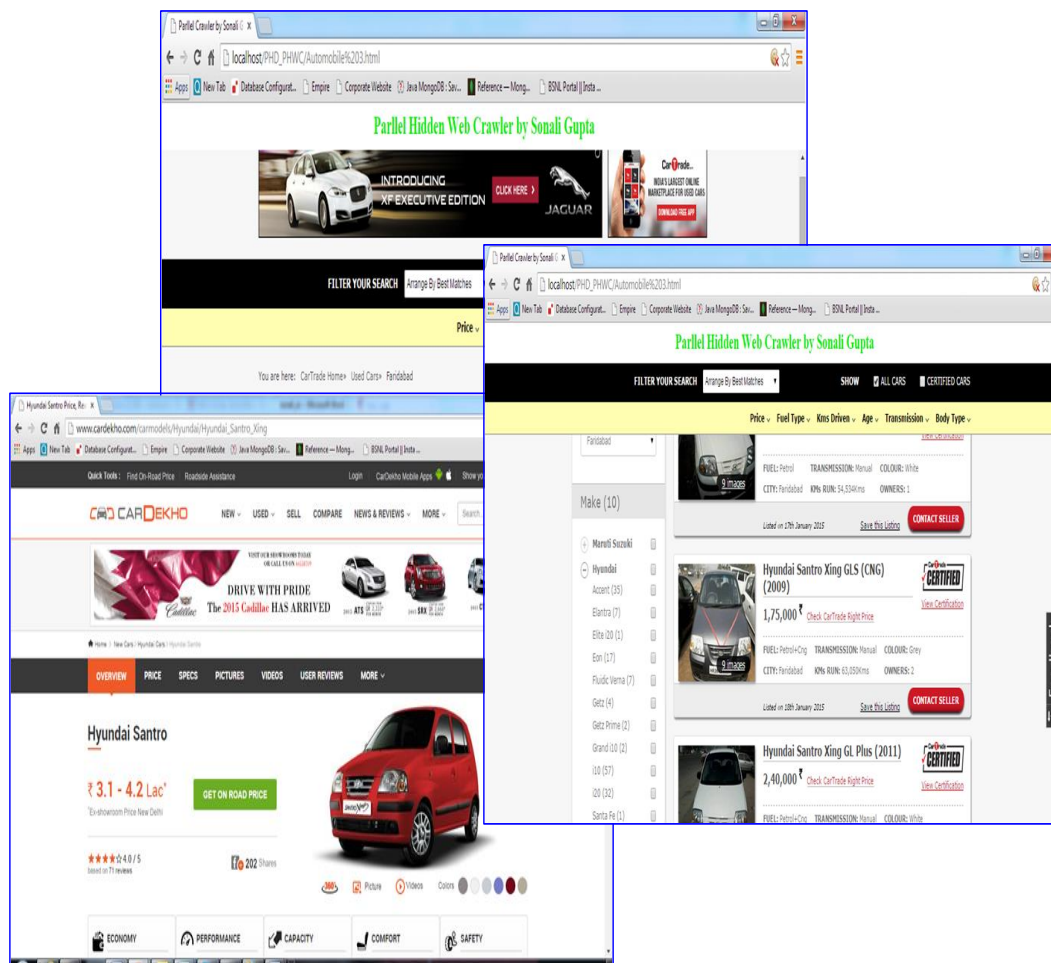


Figure 6.33: Parallel downloading of response pages in Auto domain

The Query Ranker initially suggests the queries stored in the Domain-Specific Databases based on any arbitrary ordering among them but the crawler simultaneously keeps a record of the percentage content that has been retrieved by issuing each query. The crawler stops issuing queries as soon as this percentage reaches approximately hundred. The crawler next fires the queries according to the random rank predicted by the proposed mechanism. Table 6.6 presents the results for evaluating the proposed Random Ranking Approach in terms of a comparison between the number of queries required in each approach i.e. the arbitrary and the proposed approach when a fixed percentage of the contents of the hidden database has been retrieved.

Table 6.6: Experimental Evaluation of the Query Ranker

%age of database contents fetched	Number of queries required		%age Reduction of Queries
	Arbitrary Ordering	Proposed random Ranking	
20	4	2	50
40	20	8	60
60	40	18	55
80	50	22	56
100	60	28	53.3

The proposed system measures the progress of the FPE based on the metric *Reduction Efficiency, R*. The metric has been defined as the ratio of the number of queries that has been cut off to the number of queries required when any arbitrary ordering among the queries was specified.

Thus,

$$R = \frac{\text{Reduction in the number of queries}}{\text{the number of queries required in the arbitrary approach}} \quad 6.1$$

The numerator can be computed by finding the difference between the number of queries required for arbitrary ordering and the proposed random ranking mechanism. For example when 40% of the database content has been retrieved, the value of R can be calculated as:

$$R = \frac{20-8}{20} * 100 = \frac{12}{20} * 100 = 60 \quad 6.2$$

As depicted from the last column of table 6.6, the proposed Random Ranking approach cut shorts the number of queries to approximately 50% of the total queries that were

needed when the queries have been raised in any arbitrary order. This nearly stable 50% ratio has been possibly achieved by including the factor of dynamic rank which is a query-dependent factor that predicts the behaviour of the query for the next run according to the query size statistics generated in the immediate previous run of the crawler.

The percentage of the obtained records (the x-axis) against the number of queries issued (the y-axis) has been shown in Figure 6.34. The crawler initially progresses rapidly with the increase in the number of queries justifying the target of the proposed approach which was to choose the wise queries (based on the analysis of the response pages) in such a way that most of the data from the database can be retrieved as early as possible rather than retrieving the entire contents in the end.

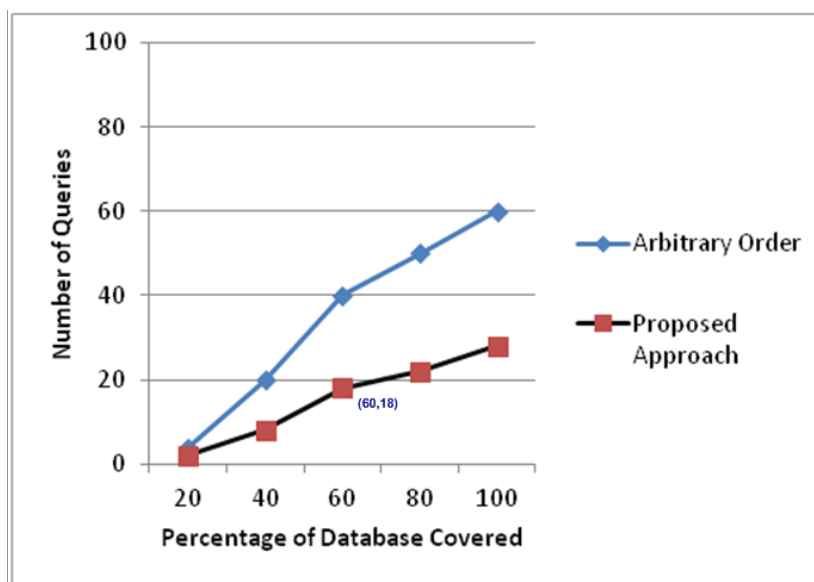


Figure 6.34: Comparison of different Query ranking approaches

For example, for the proposed approach a point (60, 18) (see Figure 6.34) means that the crawler was able to discover 60 % records from the database when it had issued just 18 queries. The same amount of data has been extracted by issuing 40 queries using the arbitrary approach earlier. Thus, the proposed algorithm is asymptotically optimal (in terms of number of queries and database coverage) in the sense that it retrieves the same size of Hidden web database by issuing less number of queries as compared to any arbitrary approach for firing the queries.

The search forms from all the other domains were also processed in the similar way in parallel. For each domain, the data in Table 6.7 shows the number of search forms correctly submitted and processed (C); the number of search forms incorrectly

submitted and processed (I); the number of search forms that could not be submitted and processed (N) by the proposed parallel Hidden web crawler.

After the search form were filled and submitted to the respective web servers, the web pages that were retrieved in response were forwarded further to the Response Analyzer. Each thread of the Response Analyzer employs an instance of the Page content extractor (as in Phase III) that helps in maintaining and updating the various domain-specific databases created in Phase III.

6.4. COMPARASON OF ALL DOMAINS

It may be observed from the data in the Table 6.7 that the number of search forms that could not be processed by the system (which was 46) is approximately the same as the number of hidden web resources that were assigned to a domain irrelevant for the search form or hidden web resource during the process of discovering the hidden web resources. Thus, a domain-specific database that matches the predicted domain of relevance of the hidden web resource or the search form which is in fact different than the actual domain of the search form, might have been used to process the search form, making the crawler incapable of processing it.

Table 6.7: Number of Form submissions: Valid, Invalid and Failures

Domain of search forms	Average no. of Forms successfully submitted & processed		Average no. of failures in form submission and processing, N
	Valid form submissions C	Invalid form submissions I	
Food	44	15	14
Books	62	12	13
Travel	79	10	8
Auto	65	13	11

Based on the data in Table 6.7, the precision, recall and f-measure values are computed in each of the domains for the proposed crawler. The same has been represented by the data in table 6.8

Table 6.8: Precision, Recall and F-measure values averaged over all the search forms processed in a domain.

Domain	Precision	Recall	F-Measure
Food	0.75	0.76	0.75
Books	0.84	0.83	0.83
Travel	0.89	0.91	0.90
Auto	0.83	0.86	0.84
Average	0.83	0.84	0.83

It may be observed from Table 6.8 and the corresponding graph in Figure 6.35 that on an *average* while processing search forms in any domain, the *Precision* is high i.e. ranges from 75.21% to 89.18%, range of *Recall* is also high i.e. from 76.66% to 91.2% and *average F-measure* of is also quite high i.e. varies from 75.12% to 90.03%.

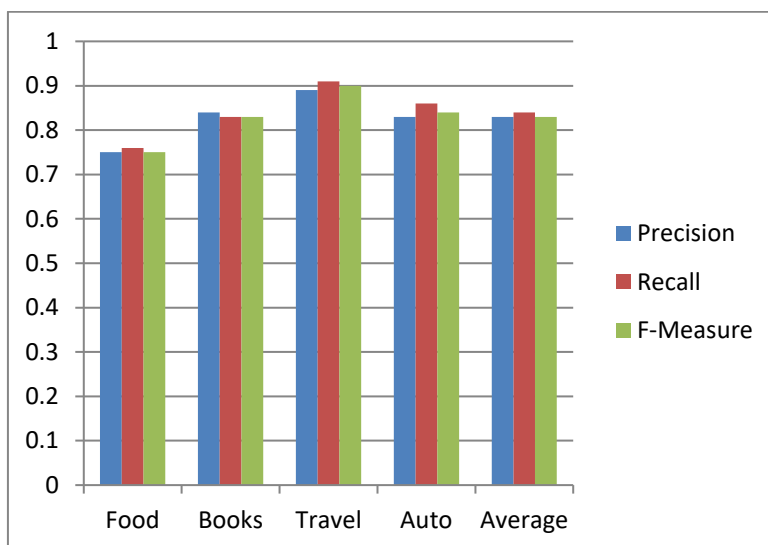


Figure 6.35: Precision, Recall and F-Measure values in each domain and average over all domains.

It may be noted that the maximum value of precision, recall and f-measure is observed for the web search forms in travel domain. The reason behind this is the performance of the Page Classifier and the Page Content Extractor that helps in creating a domain-specific database that is highly accurate.

A slightly low value of *Precision*, *Recall* and *F-measure* is observed in the *Food* domain occur due to the high number of cuisines that exist based on the geography of the region. Moreover, the same ingredient used for different food items in cooking has been assigned different names depending on the language used by the people in that

region. However, during the experiment, it was found that the values of *Precision*, *Recall* and *F-measure* for the Books, Auto and Travel domains remain fairly consistent. Thus, the experiments conducted over all the domains indicate that the framework proposed in this work is both consistent and efficient.

6.5. COMPARASION OF THE PROPOSED PARALLEL HIDDEN WEB CRAWLER WITH EXISTING WEB CRAWLERS

In this work, a design of the architecture for parallel hidden web crawler has been proposed and implemented. The experimental results show that the proposed crawler is able to effectively crawl the Hidden Web with high accuracy.

In [45], Bergholz and Chidlovskii implemented a domain-specific crawler that starts on the Publicly Indexable Web and detects search forms relevant to a given domain. Their crawler shows excellent results on single-attribute search forms that support the search of textual content residing in unstructured document databases while simply ignoring the multi-attribute search forms for structured databases.

The hidden web crawler in [39] finds candidate assignments for submitting a search form. The response analyzer has been used either to distinguish between pages containing search results and pages containing error messages or to simply store the resulting web page in a repository for supporting the user queries in future. The response analyzer did not analyse response pages for retrieving any other useful content that can facilitate the crawler's act in future. Also, another challenge of their approach lies in dealing with the form elements with infinite domain.

The authors in [46, 75] have developed domain-specific hidden web crawlers that are capable of automatically downloading and processing the search forms but did not consider the parallelization of the tasks in and across domains. A single search form is being processed at a time in their work.

In comparison, the hidden web crawler proposed in this work targets to download and process the various search forms in parallel within and across domains that enables efficient crawling. Moreover, the proposed approach also leads to the following advantages:

- 1) Each parallel instance that is responsible for processing the search forms is dedicated to tackle the search forms from a single domain in a particular run of the crawler. For example: the snapshot captured in Figure 6.31 shows the parallel filling and processing of search forms from *Travel* domain whereas

another snapshot of Figure 6.33 that was captured at the same instance shows, the parallel processing of search forms from a different domain, *Auto*. This eliminated the overhead of synchronizing the various parallel threads in execution.

- 2) Minimizing the synchronization helped in minimizing the communication bandwidth required to coordinate the parallel threads or instances.
- 3) Filling the search forms with only valid and wise queries as suggested by the query ranker helps in reducing the unnecessary requests that will otherwise be submitted to the web database server. Thus the proposed crawler obeys the property of politeness with respect to web servers.
- 4) The proposed Parallel Hidden Web crawler is also scalable and extensible [90, 92].

Table 6.9 compares the proposed Parallel Hidden Web Crawler over certain parameters with some of the existing crawlers for the Hidden Web.

Table 6.9: Comparison of proposed parallel hidden web crawler with the existing crawlers

Characteristics	Deep Web Crawler [39]	Hidden Web Crawler[45]	DSHWC[47]	AKSHR[65]	Proposed Parallel Hidden Web Crawler
Description	Processes the form and fills it by using LVS table	Analyze the form, classify it and fills the form	Downloads the forms, merges them into USI and fills it automatically	Downloads the form and fills them automatically	Downloads forms from the WWW & then processes them using the Query Ranker
Search Form Collection	Doesn't download forms	Doesn't download forms	Downloads the forms automatically using Search Interface Crawler	Downloads the forms automatically using Search Interface Crawler	Uses Form Extractor to download forms
Selection of Candidate Forms	No	No	No	No	Yes Uses Form Analyzer module for the purpose
Form Filling	Not Fully Automatic	Not Fully Automatic	Fully Automatic using Domain-specific Data Repository	Fully Automatic using Domain-specific Data Repository	Fully automated as depends on the automatic creation of <i>Filling Resources</i> comprising of a <i>domain-Specific Database and Page Statistics Repository</i>

Query Optimization	No	No	No	No	Yes
Polite	No	No	No	No	Yes
Precision	Low	Low	High	High	High
Recall	Low	Low	High	High	High
F-measure	Low	Low	High	High	High
Extensible	*	*	Yes	Yes	Yes
Scalable	*	*	Yes	Yes	Yes
Network Load	Uses Large Band Width	Uses Large Band Width	Uses optimal Band Width (due to Batch Mode)	Uses optimal Band Width (due to Batch Mode)	Uses optimal Bandwidth due to the reduced communication between the parallel FPEs

* not claimed.

The performance of the proposed Parallel Hidden Web Crawler, measured in term of *Precision*, *Recall* and *F-measure*, is found to be higher as compared to the existing Hidden Web Crawlers [45, 73]. The proposed Parallel Hidden Web Crawler also makes use of optimal network bandwidth by reducing the communication between the parallel Form Processing Elements.

CHAPTER 7.

CONCLUSION & FUTURE SCOPE

7.1. CONCLUSION

An effective and efficient technique to crawl and extract the content in the Hidden web databases has been designed and implemented in this thesis. More specifically, the main challenges involved in developing a parallel crawler that downloads pages from the Hidden Web have been addressed and resolved.

As the size of the hidden web contents are very large and it would continue to grow with the time, the main objective of this work was to resolve the problems faced by a single process crawler for the Hidden web. Thus, during this time, crawling has been studied at many different levels. Various crawling strategies for the Hidden Web and parallel crawlers were studied and analyzed to build a parallel Hidden Web crawler that tackles the following challenges:

- **Scale of the Hidden Web.** The tremendous size and heterogeneity of the hidden web makes comprehensive / exhaustive coverage very difficult and possibly less useful than domain specific crawling. Thus, in this work, a parallel approach that provides effective coverage by following a domain-specific approach and efficiently crawling the huge contents in the Hidden web has been proposed and implemented in this work.
- **Identification of Relevant Search Forms.** As the search forms act as the only entry point to the content residing in hidden web databases, a mechanism that automatically discovers the relevant search forms in a domain has been developed. A match logic has been designed for this purpose and various domain-specific search interface repositories have been created to store the relevant forms in each domain.
- **Automatic processing of search forms:** To extend the crawl beyond the surface web, the crawler must be capable of automatically filling the search forms. But as these forms are developed to be used by humans, automatically submitting the form by filling with suitable values is not only difficult rather impossible. Therefore, a framework that helps the crawler to automatically process the search forms has been designed. Domain-specific databases have been created and used for storing the labels and the values needed to fill in the search forms.

- **Classification of web pages:** As the information on the web has been drawn out of many sources and domains, automatically identifying the relevant domain of the available information is a real challenge. Thus, in this work an approach that automatically identifies the domain of web pages for their classification has been developed. Different domain-specific page repositories are used to store the web pages as per their domain after classification.
- **Automatic Query Generation.** The hidden web crawler should not only be capable of automatically processing the search forms but also of making an optimal choice among the candidate queries to be raised by it .In this work, the Query Ranker that ranks the queries in the domain-specific databases to be used by the crawler for filling the search forms has been used to suggest the most optimal query at that time.
- **Reduced Network Bandwidth:** In order to minimize overlap and maintain the quality of downloaded web pages, the coordination between individual crawling processes needs communication that consumes network bandwidth. Following the domain-specific approach helps the crawler in minimizing the communication overhead and save the network bandwidth consumption while maintaining the quality of crawling.
- **Extensible and Scalable:** The proposed crawler is extensible in the sense that third party components or modules can be added as per the requirements. In addition, the proposed crawler offers scalability in design as new modules may be incorporated in the system as per the requirements.

The proposed parallel Hidden Web Crawler has been implemented using .NET technology and SQL Server. For the conducted experiments, high values of Precision, Recall and F-measure were obtained which indicates that the proposed crawler efficiently crawls the hidden web pages. The classification of the proposed work into different phases not only improves the performance of each phase but also renders a modular and extensive framework to crawling.

7.2. FUTURE WORK

The work in this thesis extensively explore the problems that are faced while cawling the Hidden Web. Some of the possible extensions and issues that could be further explored or extended in the near future are as follows:

- **Designing of a Search Engine based on parallel hidden web crawler.** As the size of the hidden web contents are very large and it would continue to grow with the time. Therefore, work can be done to design a search engine that could be able to crawl, extract and index the content of these hidden databases.
- **Indexing the Hidden Web Pages.** Search engines typically allows to search the information to the users by maintaining large-scale inverted indexes which are often replicated for the purposes of scalability. Hence, to reduce the cost of operation, they must be pruned. Thus, future work can be done towards obtaining efficiency in maintaining the index for Hidden Web in general and searching the required information in the indexed documents in particular.
- **Updating the Hidden-Web pages.** The information on the Web today is constantly evolving. Once the parallel hidden web crawler has downloaded the information from the Hidden Web, it needs to periodically refresh its local copy in order to enable users to search for up-to-date information. Therefore, work can be done in this direction to crawl the Hidden Web information incrementally.

BIBLIOGRAPHY

- [1] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, Arthur Secret. The World Wide Web in the Communications of the ACM August 1994/ Vol. 37, No. 8, p.p 76-82.
- [2] Kwong Bor Ng and Paul P. Kantor. An investigation of the preconditions for effective data fusion in information retrieval: A pilot study, 1998.
- [3] http://www.googleguide.com/google_works.html
- [4] C. Manning, P. Raghavan and H. Schütze Introduction to Information Retrieval [http://nlp.stanford.edu/IR-book/pdf/ chapter20-crawling.pdf](http://nlp.stanford.edu/IR-book/pdf/chapter20-crawling.pdf)
- [5] Ipeirotis, P., G.; Gravano, L.; Sahami., M.; Probe, count, and classify: Categorizing hidden-web databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 67–78, Santa Barbara, CA, USA, May 2001.
- [6] Search engines directory.
<http://www.searchengineguide.com/searchengines.html>.
- [7] L. Gravano, P. G. Ipeirotis, and M. Sahami, “QProber: A system for automatic classification of hidden-Web databases” , ACM TOIS, 21(1):1–41, 2003.
- [8] <http://www.w3.org/WWW>.
- [9] Arnaud Le Hors, P. Le Hagaret, L. Wood, G. Nicol, J. Robie, M.Champion, S, Byrne. Document Object Model (DOM) Level 2 Core Specification. Technical Report 13 November 2000. Available at <http://www.w3.org/TR/DOM-Level-2/>
- [10] Profusion’s search engine directory. <http://www.profusion.com/nav>.
- [11] Sherman, C.; Price, G.: The invisible Web: Uncovering information sources search engines can't see. CyberAge Books ,Medford, NJ: (2001)
- [12] Baeza-Yates, R., ; Ribeiro-Neto, B.: Modern information retrieval (2nd ed.). Addison-Wesley-Longman (1999).
- [13] <http://www.tripadvisor.com/>
- [14] <http://www.medhunt.com/>
- [15] www.metacrawler.com
- [16] Jayant Madhavan, J., Cohen, S., Dong, X. L., Halevy, A. Y., Jeffery, S. R., Ko, D., and Yu, C. 2007. Web-Scale Data Integration: You can afford to Pay as You Go. In CIDR. 342–350.

- [17] Menczer, F., Pant, G. and Srinivasan, P., “Topical Web Crawlers: Evaluating Adaptive Algorithms”, *ACM Transactions on Internet Technology*, Vol. 4, No. 4, pp. 378–419, November 2004.
- [18] Chakrabarti, S.; Berg, M.; V., Dom. B. : Focused Crawling: A New Approach to topic-specific Web Resource Discovery. *Computer Networks*, 31(11-16), pp. 1623–1640,(1999)
- [19] Álvarez, Manuel, Raposo ,Juan, Pan, Alberto, Cacheda, Fidel, Bellas, Fernando, Carneiro, Víctor, “DeepBot: a focused crawler for accessing hidden web content”, *Proceedings of the 3rd international workshop on Data engineering issues in E-commerce and services: In conjunction with ACM Conference on Electronic Commerce*, pp. 18-25, June 2007.
- [20] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna, “UbiCrawler: A scalable fully distributed Web crawler”, In *Proceeding of The Eighth Australian World Wide Web Conference*, 2002. Or in *Software Pract. Exper.*, 34(8):711–726, 2004.
- [21] Najork, Marc, and Allan Heydon, “High-performance web crawling”, *Technical Report 173*, Compaq Systems Research Center. 2001.
- [22] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused Crawling Using Context Graphs. In *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 2000, pages 527–534,.
- [23] Junghoo Cho, Hector Garcia-Molina: “Parallel Crawlers”, 7–11 May 2002, Honolulu, Hawaii, USA.
- [24] A. Heydon, M. Najork. Mercator: a scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, December 1999
- [25] M. Koster. A standard for robot exclusion. <http://www.robotstxt.org/wc/norobots.html>, June 1994.
- [26] The Web Robots Pages. Html author’s guide to the robots meta tag. [Http://www.robotstxt.org/wc/meta-user.html](http://www.robotstxt.org/wc/meta-user.html), March 2005.
- [27] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web: a survey. *SIGMOD Record*, 27(3):59–74, Sept. 1998.
- [28] Bergman M.K.: The deep web: Surfacing hidden value. *The Journal Of Electronic Publishing* 7(1) (2001). brightplanet

- [29] Chang K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: observations and implications. *SIGMOD Record*, 33(3):61–70, 2004a.
- [30] K. C.-C. Chang, B. He, and Z. Zhang. Metaquerier over the deepweb: Shallow integration across holistic sources. In *Proceedings of the VLDB Workshop on Information Integration on the Web (VLDB-IIWeb'04)*, 2004b
- [31] Krishna Bharat and Andrei Broder. A technique for Measuring the Relative Size and Overlap of Public Web Search engines. In *Proceedings of the 7th World Wide Web Conference*, Pages 379-388, Brisbane, Australia, April 14-18, 1998.
- [32] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science* (280), Pages 98-100, April 1998.
- [33] Erik W. Selberg and Oren Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th International World Wide Web Conference*, Pages 195-208, Boston, MA, USA, December 1995.
- [34] David Brake. Lost in Cyberspace, *New Scientist*, June 28, 1997.
- [35] A.K.Sharma, J. P. Gupta, D. P. Agarwal, “Augment Hypertext Documents suitable for parallel crawlers”, *Proc. of WITSA-2003*, a National workshop on Information Technology Services and Applications, Feb'2003, New Delhi.
- [36] A.K.Sharma , J.P.Gupta, D.P.Agarwal. PARCAHYD: An architecture of a parallel Crawler based on Augmented Hypertext Documents. In *International Journal of Advancements in technology* Vol.1, No.2 October 2010
- [37] Gerard Salton , Christopher Buckley, Term-weighting approaches in automatic text retrieval, *Information Processing and Management: an International Journal*, v.24 n.5, p.513-523, 1988
- [38] Lawrence, S.; Giles, C.,L.: Accessibility of information on the web. *Nature*, 400:107–109, 1999.
- [39] Raghavan, S.;Garcia-Molina, H.: Crawling the Hidden Web. In: 27th International Conference on Very large databases (Rome, Italy, September 11-14, 2001) *VLDB'01*, 129-138, Morgan Kaufmann Publishers Inc., San Francisco, CA
- [40] Wright, A.: Searching the Deep Web. In *CACM*,51(10) pp. 14-15 (October 2008)

- [41] Ntoulas, A.; Zerfos, P.; Cho, J.: Downloading Textual Hidden Web Content Through Keyword Queries. In: 5th ACM/IEEE Joint Conference on Digital Libraries (Denver, USA, Jun 2005)JCDL05, pp. 100-109 (2005)
- [42] Barbosa, L.; Freire J.: Siphoning hidden-web data through keyword-based interfaces. In: SBBD, 2004, Brasilia, Brazil, pp. 309-321 (2004).
- [43] Madhavan, J.; Ko, D.; Kot, L.; Ganapathy, V.; Rasmussen, A.; Halevy, A.: Google's Deep-Web Crawl. In Very large data bases VLDB endow., 1(2), 1241-1252.(2008)'
- [44] Liddle, S.,W.; Embley, D,W.; Scott, D.T.; Yau, S.,H.: Extracting Data Behind Web Forms. In: 28th VLDB Conference2002 , pp. 2-11, HongKong, China, pp. 38-49 (2002)
- [45] Bergholz, A.; Chidlovskii, B.: Crawling for Domain-Specific Hidden Web Resources. In: 4th International Conf. on Web Information Systems Engineering pp.125-133 IEEE Press, 2003
- [46] Lage, J.;Silva A.;Golgher P.;Laender, A.:Automatic generation of agents for collecting hidden web pages for data extraction In: Data & Knowledge Engineering IEEE 2(49) (2004)
- [47] Sharma, A.,K.; Bhatia, K.,K.: A Framework for Domain-Specific Interface Mapper (DSIM). International . Journal of Computer. Science & Network Security,12(8), December 2008.
- [48] McCallum,A.; Nigam,K.; Rennie,J.; Seymore.K.: Building domain-specific search engines with machine learning techniques. In Proc. of the AAAI Spring Symposium on Intelligent Agents in Cyberspace, 1999.
- [49] Chakrabarti, S.; Punera,K.; Subramanyam,M.: Accelerated focused crawling through online relevance feedback. In Proc. of WWW, pages 148–159, 2002.
- [50] Rennie, J.; McCallum, A.: Using Reinforcement Learning to Spider the Web Efficiently. In Proc. of ICML, pages 335–343, 1999.
- [51] Wang,W.; Meng, W.; Yu, C.: Concept hierarchy based text database categorization In proc. Intern. WISE Conf., pp.283-290, China, June 2000
- [52] Zhang, Z., He, B., C.-C. Chang, K.: Light-weight Domain-based Form Assistant: Querying Web Databases On the Fly. In Proceedings of the 31st Very Large Data Bases Conference, 2005
- [53] Cope, J., Craswell, N. and Hawking, D. (2003), “Automated discovery of search interfaces on the web”, Proceedings of the 14th Australasian Database

- Conference, Adelaide, pp. 181-9. Jared Cope, Nick Craswell, and David Hawking. Automated discovery of search interfaces on the web. In Proceedings of the 14th Australasian database conference, pages 181–189, 2003.
- [54] Luciano Barbosa and Juliana Freire. Searching for hidden-web databases. In Proceedings of WebDB’05, pages 1–6, 2005.
 - [55] Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-web entry points. In Proceedings of the 16th international conference on World Wide Web, pages 441–450, 2007.
 - [56] Luciano Barbosa and Juliana Freire. Combining classifiers to identify online databases. In Proceedings of WWW’07, pages 431–440, 2007.
 - [57] Quinlan, R. (1996), “Boosting, bagging, and C4.5”, Proceedings of AAAI-96 14th National Conference on Artificial Intelligence, Portland, OR, pp. 725-30.
 - [58] Bergholz, A. and Chidlovskii, B. (2004), “Learning query languages of web interfaces”, Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, pp. 1114-21.
 - [59] A. K. Sharma, Komal Kumar Bhatia: “Automated Discovery of Task Oriented Search Interfaces through Augmented Hypertext Documents” Proc. First International Conference on Web Engineering & Application (ICWA2006).
 - [60] A. K. Sharma, Komal Kumar Bhatia. Merging Query interfaces in Domain-Specific Hidden Web Databases accepted in International Journal of Computer Science, 2008.
 - [61] A. de Carvalho Fontes and F. Soares Silva, “SmartCrawl: A New Strategy for the Exploration of the Hidden Web”, ACM Transaction on WIDM’04, Washington, DC, USA, November 2004.
 - [62] P.Ipeirotis and L. Gravano, Distributed search over the hidden web: Hierarchical database sampling and selection," in VLDB, 2002.
 - [63] Manuel Álvarez, Juan Raposo, Alberto Pan, Fidel Cacheda, Fernando Bellas, Víctor Carneiro: Crawling the Content Hidden Behind Web Forms. In Proceedings of the 2007 International conference on Computational Science and its applications, Published by Springer-Verlag Berlin, Heidelberg, 2007.
 - [64] Ping Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query Selection Techniques for Efficient Crawling of Structured Web Sources. In ICDE, 2006.

- [65] Komal kumar Bhatia, A.K.Sharma, Rosy Madaan: AKSHR: A Novel Framework for a Domain-specific Hidden web crawler. In Proceedings of the first international Conference on Parallel, Distributed and Grid Computing, 2010.
- [66] H. He, W. Meng, C. T. Yu and Z. Wu, "Automatic extraction of web search interfaces for interface schema integration.," in WWW (Alternate Track Papers & Posters), ACM, 2004, pp. 414-415.
- [67] "AbeBooks Official Site - New & Used Books, New & Used Textbooks, Rare & Out of Print Books," abebooks.com, 2012. [Online]. Available: <http://www.abebooks.com/>.
- [68] H. Nguyen, E. Y. Kang and J. Freire, "Automatically Extracting Form Labels," ICDE - IEEE, pp. 1498-1500, 2008.
- [69] K. C.-C. Chang, B. He and Z. Zhang, "Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web.," CIDR, pp. 44-45, 2005.
- [70] M.Marin, R. Paredes,C.Bonacic.High Performance Priority Queues for Parllel crawlers in proceedings of the ACM conference WIDM '08 , California, USA
- [71] D. H. Chau, S. Pandit, S. Wang, C.Faloutsos. Parallel crawling for online social networks. In Proceedings of the 16th international Conference on World Wide Web (Banff, Alberta, Canada, May 08 - 12, 2007). WWW '07. ACM, New York, NY, 1283-1284.
- [72] Shoubin Dong, Xiaofeng Lu and Ling Zhang, A Parallel Crawling Schema Using Dynamic Partition Lecture Notes in Computer Science Volume 3036/2004, pp. 287-294.
- [73] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. Computer Networks and ISDN Systems, 30(1–7):161–172, 1998.
- [74] D Cai S, Yu J wen. “Extracting Content Structure for Web Pages Based on Visual Representation”. In the International Conferences on Asia-Pacific Web Conference(APWeb), 2003.
- [75] Golub, K. and A. Ardo (2005, September). Importance of HTML structural elements and metadata in automated subject classification. In Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Volume 3652 of LNCS, Berlin, pp. 368–378. Springer.

- [76] X. Qi and B. D. Davison. "Knowing a web page by the company it keeps". In International conference on Information and knowledge management (CIKM), pages 228-237, 2006.
- [77] Xiaoguang Qi and Brian D. Davison "Web Page Classification: Features and Algorithms", Department of Computer Science & Engineering, Lehigh University, June 2007.
- [78] Bing Liu. "Web Data Mining, Exploring Hyperlinks, Contents, and Usage Data."Springer. 2007.
- [79] Arul Prakash Asirvatham and Kranti Kumar Ravi. "Web Page Categorization based on Document Structure", International Institute of Information Technology, Hyderabad, India 500019.
- [80] Sini Shibu, Aishwarya Vishwakarma and Niket Bhargava, "A combination approach for Web Page Classification using Page Rank and Feature Selection Technique" , International Journal of Computer Theory and Engineering, Vol.2, No.6, December, 2010
- [81] Sonali Gupta, Komal Kumar Bhatia: Exploring 'Hidden' parts of the Web: the Hidden Web, in 4th International Conference on Advances in recent technologies in communication and computing, ARTCom 2012 organized by the Association of Computer Electronics and Electrical Engineers ACEEE, proceedings in Lecture Notes in Electrical Engineering , Springer Verlag Berlin Heidelberg 2012 p.p. 508-515. ISSN 1876-1100.
- [82] Pikakashi Manchanda, Sonali Gupta, Komal Kumar Bhatia: On the automated classification of Web pages using Artificial Neural Networks, IOSR Journal of Computer Engineering (IOSRJCE) ISSN: 2278-0661 Volume 4, Issue 1 (Sep-Oct. 2012), PP 20-25.
- [83] Sonali Gupta, Komal Kumar Bhatia: A system's approach towards Domain Identification of Web pages, in Second IEEE international conference on Parallel, distributed and Grid computing , dec 06-08, 2012 organized by Jaypee University of Information Technology, Solan, Shimla H.P, indexed in Scopus.
- [84] Sonali Gupta, Komal Kumar Bhatia: Deep Questions in the 'Deep or Hidden'Web. In International Conference on Soft Computing for Problem Solving SocPros-Dec 28-30 , 2012, Jaipur, rajasthan, proceedings by springer., indexed in DBLP.

- [85] Sonali Gupta, Komal Kumar Bhatia: Domain Identification and Classification of Web pages using Artificial Neural Networks, in 3rd International Conference on Advances in computing, communication and Control ICAC3, Jan 18-19, 2013 organized by Fr. Conceicao Rodrigues College of Engg. Mumbai, proceedings by Springer Verlag Berlin Heidelberg. Indexed in Scopus.(pp 215-226).
- [86] Sonali Gupta, Komal Kumar Bhatia: Crawl Part :Creating Crawl Partitions in Parallel Crawlers in IEEE International Symposium on Computing and Business Intelligence, ISCBI August- 2013, held at delhi organized by Cambridge Institute of Technology.IEEE Xplore, Indexed in Scopus
- [87] Sonali Gupta, Komal Kumar Bhatia:HiCrawl: A Hidden Web crawler for Medical Domain in proceedings of 2013 IEEE International Symposium on Computing and Business Intelligence, ISCBI, August18-18, 2013 Delhi , India .indexed in scopus
- [88] Sonali Gupta, Komal Kumar Bhatia, Pikakshi Manchanda: WebParF: A Web partitioning framework for Parallel Crawlers. In International Journal of computer Science and Engineering (IJCSE)published by engineering journals indexed by Google Scholar, CiteseerX, Index Copernicus , DOAJ vol.5 No.8 August 2013 pp 718-725.
- [89] Sonali Gupta, Komal Kumar Bhatia: A Novel Term Weighing Scheme (VARDF) Towards Efficient crawl of Textual Databases. In International Journal of (IJCEA) , ISSN: 2321-3469 (Impact Factor: 2.84), Volume-IV, Issue-I/III indexed by DBLP.
- [90] Sonali Gupta, Komal Kumar Bhatia: Hidden Web Resource Discovery through semantic understanding of Search Form interfaces in IEEE international Conference on Advanced Computing and Communication technologies , ICACCT-2014 proceedings by IEEE Xplore.
- [91] Sonali Gupta, Komal Kumar Bhatia: A Comparative Study of Hidden Web Crawlers in International Journal of Computer Trends and Technology (IJCTT) – volume 12 number 3 – Jun 2014, pp 111- 118, ISSN: 2231-2803
- [92] Sonali Gupta, Komal Kumar Bhatia: Optimal Processing of Search Forms for Hidden Web Extraction through a Novel Random Ranking Mechanism, communicated in International Journal of Information Retrieval Research (IJIRR),

- [93] Meta tags, Frontware International.
- [94] Daniele Riboni “Feature Selection for Web Page Classification”, Università degli Studi di Milano, Italy.
- [95] J. Yi and N. Sudershesan, “A classifier for semi structured documents”, In KDD, Boston, MA USA, 2000.
- [96] Wai-Chiu Wong, A. Wai.C. Fu, “Incremental Document Clustering for Web Page Classification”, Chinese University of Hong Kong, July 2000.
- [97] Gerry McGovern. "A step to step approach to web page categorization". www.gerrymcgovern.com.
- [98] H. Yu, J. Han, K.C.Chang. PEBL: positive example based learning for web page classification using SVM. In KDD’02 : proceedings of the 8th ACM SIGKDD international conference on Knowledge Discovery and Data mining, pages 239-248, New York, NY, USA, 2002.
- [99] Daniela XHEMALI, Christopher J. HINDE and Roger G. STONE,” Naïve Bayes vs. Decision Trees vs. Neural Networks in the Classification of Training Web Pages” IJCSI International Journal of Computer Science Issues, Vol. 4, No 1, 2009.
- [100] S. M. Kamruzzaman, “Web Page Categorization Using Artificial Neural Networks”, Proceedings of the 4th International Conference on Electrical Engineering & 2nd Annual Paper Meet 26-28 January, 2006.
- [101] TheMathsWork, <http://www.mathworks.com/products/neuralnet/>.
- [102] U. Schonfeld, Z. Bar-Yossef, I. Keidar. Do not crawl in the DUST: different URLs with similar text. In Proceedings of the 15th International World Wide Web Conference, pages 1015–1016, New York, NY, USA, 2006.
- [103] Russell, S. & Norvig, P. Artificial Intelligence: A Modern Approach, London: Prentice Hall, 2003
- [104] MLADENIC, D. 1998. Turning Yahoo into an automatic Web-page classifier. In Proceedings of the European Conference on Artificial Intelligence (ECAI). 473–474.
- [105] Kwon, O. W. ; Lee, J.H. Web Page Classification based on k-nearest neighbour approach. In proceedings of 5th International Workshop on Information Retrieval with Asian Languages. ACM Press, new TOrk, NY, 9-15, 2000

- [106] Kwon, O. W. ; Lee, J.H. Text Categorization based on k-nearest neighbour approach for Web site classification. *Information Process Management* 29, 1(Jan.) 25-44, 2003.
- [107] On the Automated Classification of Web sites, John M. Pierre, Linkoping University Electronic Press, Sweden.
- [108] Do, H.H., E. Rahm: COMA – A System for Flexible Combination of Match Algorithms. *VLDB* 2002
- [109] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD*, 2004.
- [110] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Mappings between Database Schemas. In *SIGMOD*, 2004.
- [111] Jayant Madhavan, Philip A. Bernstein, Erhard Rahm, Generic Schema Matching with Cupid, *VLDB* 2001
- [112] Jyoti Gautam, Ela Kumar, and Mehjabin Khatoon Semantic Web Improved with IDF Feature of the TFIDF Algorithm in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2014 Vol I, IMECS 2014, March 12 - 14, 2014, Hong Kong*
- [113] Shaikh, F. and Siddiqui, U.A. and Shahzadi, I.(2012) Semantic Web based Intelligent Search Engine, in *proc. of International Conference on Information and Emerging Technologies*, pp. 1-5..
- [114] Lu, C. and Hu, X. and Park, J. (Sep. 2011) Exploiting the Social Tagging Network for Web Clustering, (*Systems, Man, and Cybernetics – Part A: Systems and Humans*), vol. 41, pp. 840-852.
- [115] Jomsri,P. and Sanguansintukul, S. and Choochaiwattana, W. (2010) A Framework for Tag-Based Research Paper Recommender System: An IR Approach, in *proc. of the 24th International Conference on Advanced Networking and Applications Workshops, IEEE*, pages 103-108.
- [116] Vagelis Hristidis, Yuheng Hu, Panagiotis G. Ipeirotis:Relevance-Based Retrieval on Hidden-Web Text Databases Without Ranking Support. *IEEE Trans. Knowl. Data Eng.* 23(10): 1555-1568 (2011)
- [117] www.google.com
- [118] www.makemytrip.com

APPENDIX A

Domain Definitions considered for the various domains of consideration:

1) Travel:

Domain-Definitions: {Tour, Travel, visit, country, hotel, street, road, local, attraction, city, capital, metropolitan, world, nation, route, migration, price, vacation, railway, roadway, coming, leaving, source, arrival, economy, culture, heritage, airline, airway, reservation, age, passengers, adults, children, natural, return, largest, Asia, Africa, America}

Clusters: {city, capital, place, state, town, area, destination}; {departure, migration}; {arrival}, {Travel, journey, trip, tour, excursion}, {visit}, {country, nation, capital, world}, {road, street, route, roadway, }, {local, area}, {vacation, leaving}, {culture, heritage}.

2) Food:

Domain-Definitions: {Fats, proteins, carbohydrates, minerals, vitamins, calcium, food, oil, butter, heat, dry, boil, fry, steam, burger, snacks, fast food, junk food, nutrients, nutrition, cuisine, restaurant, bake, oven, grill, microwave, grind}

Clusters: {Fats, proteins, carbohydrates, minerals, vitamins, calcium} ; { burger, snacks, fast, junk, food, restaurant, grill}; {nutrients, nutrition}; {cook, grill, cuisine, fry, steam, dry, bake, oil, butter}; {boil}; {grind}; {microwave, oven}.

3) Books:

Domain-Definitions: {Author, Name, title, ISBN, price, Book, Publisher, Edition, Subject, Table of Contents, Index, University, Literature copyright, Higher, education, graduate, Fiction, novel, biography, writer}

Clusters: {Author, writer, name, publisher, subject, graduate}; {name, title, book, table of contents}; {ISBN, publish, book, edition}; {book, name, subject, index}; {education, university, higher}; {publish, copyright}; {Literature, Fiction, Novel, Biography}

4) Auto:

Domain-Definitions: {vehicle, car, taxi, price, transport, Private, commercial, wheel, machine, motor, Make, model, manufacturer, sedan, Hatchback, body; MRF, SUV, XUV, MUV, tyre, mortgage}

Clusters: {vehicle, tyre sedan, car, hatchback, taxi, motor, transport};{ Private, commercial, price, MRF };{ Make, manufacturer};{ model, body};{wheel, tyre, machine, motor}; {mortgage};{SUV, XUV, MUV}

APPENDIX B

Table of stopwords accounted in the research work:

is	he	another	such	if
am	she	must	thus	usually
are	it	all	then	I
the	you	any	than	The
this	they	not	only	was
that	there	nor	also	were
for	here	or	find	has
of	which	either	up	have
see	it	neither	down	had
it	be	each	even	to
with	in	every	like	above
and	a	but	still	over
by	an	although	new	so
we	other	once	on	hence

BRIEF PROFILE OF THE RESEARCH SCHOLAR

Name: Ms. Sonali Gupta

Designation: Assistant Professor, Department of Computer Engineering
YMCAUST, Faridabad

Qualification:

Research Interests:

- Internet and Web technologies
- Information Retrieval
- Data & Text Mining

Work Experience:

- Assistant Professor, Department of Computer Engineering, YMCAUST, Faridabad (August 2009 to till date)
- August 2004 to July 2009)
- (August 2003 to July 2004)

LIST OF PUBLICATIONS

List of Published Papers

S.No	Title of the paper along with volume, Issue No, year of publication	Publisher	Impact Factor	Referred / Non-Referred	Whether you paid any money or not for publication	Remarks
1.	Optimal Processing of Search Forms for Hidden Web Extraction through a Novel Random Ranking Mechanism Volume IV-Issue –II, 2014	International Journal of Information Retrieval Research IGI Global		Referred	No	Indexed by ACM Digital Library, INSPEC, JournalTOCs, Bacon's Media Directory, Cabell's Directories,
2.	WebParF: A Web partitioning framework for Parallel Crawlers Volume-5 Issue No.8 August 2013	International Journal of computer Science and Engineering	-	Referred	No	Indexed By DBLP, Google Scholar, Scirus, CiteseerX, DOAJ
3.	A Comparative Study of Hidden Web Crawlers volume 12 number 3 – June 2014	International Journal of Computer Trends and Technology (IJCTT)	Journal Impact Factor = 2.358	Referred	yes	Indexed by DBLP, Google Scholar,
4.	A Novel Term Weighing Scheme (Vardf) Towards Efficient Crawl Of Textual Databases Volume-IV, Issue-I/III	International Journal of Computer Engineering & Applications (IJCEA)	Impact Factor: 2.84	Referred	yes	Indexed by DBLP, Google Scholar
5.	On the automated classification of Web pages using Artificial Neural Networks <i>Volume 4, Issue 1 , 2012</i>	IOSR Journal of Computer Engineering (<i>JCE</i>)		Referred	yes	Indexed by DBLP, Google Scholar

6.	Exploring ‘Hidden’ parts of the Web: the Hidden Web 2012 IET Conference publications	Springer 4th Int. Conf. on Advances in recent technologies in Comm & comp. ARTCom		Referred		Indexed by Scopus
7.	A system’s approach towards Domain Identification of Web pages 2012	Second IEEE , international conference on Parallel, distributed and Grid computing (PDGC)		Referred		Indexed by Scopus
8.	Deep Questions in the ‘Deep or Hidden’ Web	Springer , International Conference on Soft Computing for Problem Solving SocPros-2012		Referred		Indexed in DBLP
9.	Domain Identification and Classification of Web pages using Artificial Neural Networks, 2013	Springer 3 rd International Conference on Advances in computing, communication and Control ICAC3,		Referred		Indexed in Scopus
10.	Crawl Part :Creating Crawl Partitions in Parallel Crawlers, 2013	IEEE International Symposium Computing and Business Intelligence, ISCBI		Referred		Indexed in Scopus
11.	HiCrawl: A Hidden Web crawler for Medical Domain	IEEE International Symposium Computing and Business Intelligence, ISCBI		Referred		Indexed in Scopus

12.	Hidden Web Resource Discovery through semantic understanding of Search Form interfaces	IEEE international Conference on Advanced Computing and Communication technologies ICACCT-2014		Referred		Indexed by Google Scholar, Research Gate, Academia
------------	--	---	--	----------	--	---