AN ENHANCED WEB ROBOT FOR THE CINDI SYSTEM

RUI CHEN

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE & SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE AT
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

APRIL 2008

Your file Votre référence
ISBN: 978-0-494-40937-4
Our file Notre référence
ISBN: 978-0-494-40937-4

ABSTRACT

An Enhanced Web Robot for the CINDI System

Rui Chen

With the explosion of the Web, traditional general purpose web crawlers are not sufficient for many web traversing and mining applications. Consequently, focused web crawlers are gaining attention. Focused web crawlers aim at finding web pages only related to the pre-defined topic at much less storage and computing cost. It is inherently suitable for the construction of digital libraries. As an essential part of Concordia INdexing and DIscovering system (CINDI) digital library project, CINDI Robot is a focused web crawler digging and collecting online academic and scientific documents in computer science and software engineering field.

In this thesis, we discuss the details of building a multi-threaded, large-scale, intelligence-based focused web crawler, CINDI Robot. To enhance CINDI Robot, some state-of-the-arts techniques are exploited or modified to accommodate our task. The naïve Bayes classifier and the Support Vector Machine classifier are utilized to contribute to the classification; a revised context graph algorithm and a special tunneling strategy are employed to increase recall; URL ordering policies are set up to sort all crawling web pages. Other heuristics obtained during the experimental stage are also incorporated into the final version of the CINDI Robot. Finally we form a multi-level inspection infrastructure to efficiently traverse the Web. Through this multi-level inspection scheme, text features of web page contents, URL patterns and anchor texts are considered together to guide crawling processes. Our experiments demonstrate that the final version of our CINDI Robot outperforms traditional web crawlers in terms of precision, recall and crawling speed.

Acknowledgements

I would like to express my gratitude to all those who gave me warm care and support, which gave me the possibility to finish this thesis.

I am greatly indebted to my supervisor, Dr. Bipin C. Desai, for giving me the opportunity to get into CINDI Robot in which I am deeply interested, for his constant support, care and patience and for his inspirational ideas, suggestions and comments. His understanding and supporting are the most important factors that help me complete my thesis.

I owe my most sincere gratitude to my wife, Yuya, who gave me selfless love, understanding and patience. She gave me the most courage and encouragement that she could ever offer. During my master period, she made her bravest decision to come to Montreal to be with me.

I am deeply grateful to my colleagues who worked on various subsystems of CINDI system. Tao gave me many ideas of CINDI's implementation. Krishma gave me solid support by running the DFS subsystem. Especially, I would like to thank Cong Zhou who made substantial research on CINDI Robot. Without his efforts, I cannot successfully complete my thesis.

I am also grateful to my parents for their insights and their unselfishness. Their decision of allowing their only child to Canada was the very first step of this thesis.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ADR	Accepted Document Rate
API	Application Programming Interface
ASHG	Automatic Semantic Header Generator
BFS	Breadth-First Search
CINDI	Concordia INdexing and DIscovering system
CSE	Computer Science and software Engineering
CSV	Comma-Separated Values
DFS	Document Filtering Subsystem
DRR	Document Download Rate
DTD	Document Type Definition
FCS	File Conversion Subsystem
FIFO	First In First Out
HTML	Hyper Text Markup Language
ODP	Open Directory Project
PRR	Page Relevancy Rate
RDF	Revised Document Frequency
RDVT	Representative Document Vector Table
RKF	Revised Keyword Frequency
SQL	Structured Query Language
SVM	Support Vector Machine
TF×IDF	Term Frequency and Inverse Document Frequency
URL	Uniform Directory Project

Chapter 1

Introduction

1.1 Problem Statement

As an essential part of Concordia INdexing and DIscovery System (CINDI) [1] , CINDI Robot digs and collects online academic and scientific documents (research papers, technical notes, FAQs and so on) in the computer science and software engineering field. However, the size of the Internet is growing exponentially. According to the Internet Systems Consortium Domain Survey [2], the number of Internet hosts was only 313,000 at the nascent phase of the World Wide Web, then rose to 93,047,785 at the turn of the century and exploded to 433,193,199 in January, 2007. This volume of data poses a huge challenge on traditional web crawlers which simply follow the hyper linked network structure and aim at collecting as many web pages as they can, for example, the Google Crawler. For the construction of a digital library using traditional web crawlers, we have to traverse the whole Web before we can identify all target documents. It is impossible in terms of computing and storage resources. Even the Google crawler only crawls and fully indexes 20% to 30% web pages of the whole Web [3]. Consequently, focused web crawlers are gaining attention. A focused web crawler only crawls relevant web pages limited to a specific topic, domain or format in traversing the Web at affordable costs. CINDI Robot is a typical focused web crawler which only traverses relevant regions of the whole Web to obtain desirable documents. The previous version of CINDI Robot [4] works at reasonable computing and storage costs, but fails to satisfy the requirements in terms of precision, recall and crawling speed.

1.2 Proposed Solution

Designing a scalable Web crawler suitable for large-scale web page collections is a complex task. Shkapenyuk and Suel [5] noted that: "While it is fairly easy to build a slow crawler that downloads a few pages per second for a short period of time, building a high-performance system that can download hundreds of millions of pages over several weeks presents a number of challenges in system design, I/O and network efficiency, and robustness and manageability." Thus major modifications are needed to enhance the previous CINDI Robot. An intelligence core based on Naïve Bayes classifier and Support Vector Machine classifier is added for guiding the CINDI Robot to confine the crawling process to merely relevant regions. By parsing web page text contents, the CINDI robot can analyze the relevance of a web page and of even a web site, which helps the CINDI Robot promote precision of the crawling course. URL patterns and anchor texts are also utilized to form a multi-level inspection infrastructure. A revised context graph algorithm and a tunneling strategy are created to increase the recall. And more sophisticated and real-time statistical feedbacks and heuristics are utilized to improve the CINDI Robot's overall performance.

1.3 Organization of the Thesis

This thesis is organized as follows. Chapter 2 introduces the existing Web utilization approaches, presents related works of focused web crawling and gives an overview of the whole CINDI system. Chapter 3 presents the overall system architecture of CINDI Robot and elaborates the five CINDI Robot components. Chapter 4 describes the implementation of the CINDI Robot and elaborates various heuristics used in the CINDI Robot. Chapter 5 discusses the experiments performed on the CINDI Robot. In Chapter 6, we draw our conclusion and present the suggestions for future work.

Chapter 2

Background

2.1 Web Utilization Approaches

The size of the Web is growing exponentially and it is estimated that the number of searchable web pages on the Internet has exceeded 8.9 billion [6]. This volume of data makes it difficult to get targeted information from the Web. Consequently, two families of approaches have been proposed to better utilize the Web, namely web directories and web crawlers.

2.1.1 The World Wide Web

The World Wide Web is distinct from the Internet. "The World Wide Web (commonly shortened to the Web) is a system of interlinked, hypertext documents accessed via the Internet [7]". The World Wide Web was created around 1990 by Tim Berners-Lee and Robert Cailiau in order to facilitate users to share information allocated on computers across the whole Internet. The Web can be viewed as a graph whose nodes correspond to web pages on the Web, and whose edges correspond to the hyperlinks between these pages [8].

After its creation, it became a major stimulation for the accelerated growth of the Internet. The expansion and popularity of the Web are incredible. Though the exact numbers of unique sites and web pages may vary from study to study, the general tendency is the same that the Web is growing exponentially. According to Netcraft's data, Boutell estimates there are over 29.7 billion pages on the World Wide Web as of February 2007 [9]. And it is reasonable to deduce the actual number would be even higher. From another study of World Internet Stats, it indicates there are approximate 1262 million Web users over the World [10].

The dynamic nature of the World Wide Web and the ever growing volume of time-sensitive information demand certain techniques to manage and organize the whole Web. As a result, two families of approaches have been proposed: web directories and web crawlers.

2.1.2 Web directories

Web directories organize web sites into categories and subcategories by subject from broad to specific. The categorization is accomplished in the unit of a whole web site, rather than single web page, which is quite different from search engines. Usually, one web site will be limited to inclusion in only one or two categories. Web directories are often generated by web masters or directory staffs, who are usually experts in particular categories. Editors either discover web sites they feel suitable for a category or inspect submissions from site owners for fitness. The qualities of web directories are guaranteed by human editors.

The Open Directory Project (ODP) [11], which is maintained by a team of volunteers all over the world, is the largest, most comprehensive human-edited directory of the Web. It is hosted and administered by Netscape Communication Corporation. One big challenge of a web directory is the astounding expansion rate of the Web. A small staff can provide high quality web sites, but cannot scale to the growth of the Web and maintain the quality at the same time. ODP handles this problem in an elegant way. Having realized that as the web grows, so does the number of the Web users, ODP forms a self-regulating community of Web users that basically runs itself. Through the system of self-governance, ODP volunteer editors manage the directory's growth and development, and through a system of checks-and-balances, ensure the directory is of superior quality [12]. Now the ODP categorizes over 5 million web sites into 21 main categories and over 600,000 subcategories and 31 languages [13].

Due to the involvement of human-beings, Web directories may provide high-quality categorizations. However, also due to too much human involvement, web directories may face a serious problem of scalability. Human-maintained directories may cover most popular topics effectively but are expensive to build and maintain, slow to update, and cannot cover all existing topics. Long delays in approving submissions or for rigid organizational structures and disputes among volunteer editors are still obstacles for the development of web directories [14].

2.1.3 Web crawlers

Compared to web directories, web crawlers provide a more automatic method to explore the whole Web and are more widely used [15, 16, 17, 18]. A web crawler, also known as a *web robot*, a *web spider*, a *web worm* or a *web wanderer*, is an automated program that explores new pages by traversing the hyper linked Web structure and discovers updates of web pages it already knows about. Web crawlers are essential for many Web applications, such as search engines, digital libraries, investment portals and so on. Although the web crawling algorithm is straightforward, the construction of a scalable, efficient and extensible web crawler is a complex endeavor. Most of the challenges to build a high-performance web crawler can ultimately be ascribed to the scale of the Web.

Generally, a web crawler starts the crawling process with a list of seed URLs. As the crawler visits these URLs, it identifies all the hyperlinks in the web pages and adds them to the crawl frontier which is the repository of URLs to visit. URLs from the frontier are visited according to a set of policies. This set of policies divides web crawlers into two main branches: general purpose web crawlers and focused web crawlers. This set of policies also guides the web crawler to crawl brand new web pages and already known web pages in different ways. Usually, for re-crawling a web page, the web crawler can perform a more efficient crawling algorithm. After the crawling process, desirable web pages are full text indexed for further user queries. This is another feature that distinguishes web crawlers from web directories.

2.1.3.1 General purpose web crawlers

The general purpose web crawlers try to construct a general index of the Web by covering any existing topic, try to follow every URL it can find and try to answer every query made by users. Thus, scalability is the main challenge faced by general purpose Web crawlers. As the Web grows, it poses increasing difficulties on network bandwidth to download web pages, memory to maintain private data structures in support of their algorithms, CPU to evaluate and select URLs, and disk storage to store the text and links of fetched pages as well as other persistent data [19]. Typical examples of general purpose web crawlers include Google crawler and Mercator.

The Google Crawler is the best representative of general purpose web crawlers, which is designed to scale to the entire Web. Since the creation of Google, the Google Crawler is a central part of the entire Google search engine system. It collects web pages of all types including documents, technical reports and so on from the Web and thereafter builds a searchable index for the Google search engine. The web crawling of the Google Crawler is done by distributed crawlers running on hundreds of individual computers in multiple locations. There are five functional components of the Google Crawler. A *URL server* sends lists of URLs to multiple crawlers to crawl. The *crawlers* download web pages and then send them to a single *StoreServer*. Each *crawler* keeps roughly 300 connections open at once. The *StoreServer* compresses and stores the web pages into disk. These web pages are later retrieved from disk by an *indexer*, which extracts hyperlinks from HTML pages and puts them into an anchors file. A *URL resolver* process reads the anchors file, converts the URLs contained to absolute URLs and stores them in the URLs repository; these URLs are read by the *URL server*. The Google Crawler has two versions, namely deepbot and freshbot according to if it is going to crawl new web pages or web pages it already knows. Deepbot, the deep crawler, tries to follow every hyperlink it can find and download as many pages as it can and passes them to the *indexer*. Freshbot, the re-crawling crawler, crawls only the web pages it already knows about for updates. It visits web sites that change frequently to keep its index up-to-date. The deep crawl is performed once a month while the fresh crawl is done nearly every day.

Currently, the Google Crawler follows only HREF links and SRC links, and indexes and caches 14 file types, including HTML, PDF, Word documents, Excel spreadsheets, Flash SWF, plain text files, among others. Up to 2006, the Google crawler has indexed over 25 billion web pages, 1.3 billion images, and over one billion Usenet messages [20]. In addition, the Robot Exclusion Protocol [21] is respected by the Google Crawler.

2.1.3.2 Focused web crawlers

The concepts of topical and focused crawling were first introduced by Menczer [22] and by Chakrabarti et al. [23] respectively. Focused web crawlers selectively seek out pages that are relevant to a pre-defined set of topics. A focused crawler analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the Web [23]. This effort brings significant savings in computing, storage and network resources, and makes the crawl more up-to-date. Usually, the topics of interest are defined not by keyword, but by a set of exemplary documents. The major challenge of a focused web crawler is the capability of predicting the relevance of a given page before actually crawling it. Certain intelligence is required for a focused web crawler to achieve this goal. Focused Web crawlers use this kind of intelligence to avoid irrelevant regions of the Web in order to make the task manageable. In addition to the ability to test the relevance of a web page, a focused web crawler should also pay attention to the ability to discover relevant regions which are separated by groups of irrelevant web regions in order to achieve desirable web coverage. A well-design focused web crawler should be able to stay in pre-defined topics as long as possible, while covering the Web as much as possible.

CiteSeer Crawler, Panorama, is a good example of focused web crawlers. CiteSeer was one of the early systems that demonstrated the use of the Web to establish a digital library. Now, CiteSeer has become the most popular web-based scientific literature digital library and search engine that focuses primarily on the field of computer and information science. Since its appearance in 1998, CiteSeer has grown into a collection of over 730,000 documents with over 8 million citations [24]. CiteSeer consists of three basic components: a focused crawler or harvester called Panorama, the document archive and specialized index, and the query interface. Currently, Panorama is implemented as multi-threaded objects in Java in which the Robot Exclusion Protocol is also respected [25].

Panorama traverses the Web for relevant documents in PDF and Postscript formats in computer science field. The topic of interest is constructed in an efficient way in

Panorama. Panorama submits relevant papers' main titles and the titles of references within the papers as separate exact phrase queries to Google Web APIs [26]. The authors deem the results returned by Google would be very relevant. Thus these URLs form a positive example set. Examples from unrelated papers form a negative example set. Both two sets are used to train a Naïve Bayes classifier, which is used to guide the crawling process. Panorama keeps a single synchronized URLs frontier. New discovered URLs will be added and sorted according to the classifier, which guarantees most relevant pages get crawled first. In other words, it ensures Panorama stays in relevant regions as long as there is still any relevant web site in the URL frontier. Once the crawler has downloaded a collection of the Web pages, clustering is performed to split the collection into meaningful sub parts. After clustering, Web pages are then indexed using autonomous citation indexing, which automatically links references in research papers to facilitate navigation and evaluation. In conclusion, CiteSeer is a full text search engine based on a focused web crawler with an interface that permits search by document or by numbers of citations or searching based on values of attributes, not currently possible on general purpose web crawlers.

2.2 Robot Exclusion Protocol

Web crawlers traverse many pages in the World Wide Web by recursively retrieving linked web pages. However, web crawlers are not always welcome by all Web servers, or at least for all regions of all Web servers. In 1993 and 1994, there had been occasions where web crawlers were not welcome for various reasons [21]. Web crawlers may access the same server too frequently and open too many connections so that the whole server can get congested; web crawlers may also unconsciously crawl some confidential areas and so on. Thus the Robot Exclusion Protocol [21] was established for web servers to indicate which parts should not be crawled. It is respected by both general purpose web crawlers and focused web crawlers. However it is not mandatory.

The protocol suggests a server to guide web crawlers by creating a file named "robots.txt" in its root directory. This method is easy for web servers to implement and also easy for web crawlers to figure out the crawling policies. The "robots.txt" consists of two parts, *User-agent* lines and *Disallow* lines. The *User-agent* lines indicate the names of web robots for which this file is set. The *Disallow* lines specify which directories are not allowed to crawl. Using different combinations of *User-agent* lines and *Disallow* lines, this protocol can be quite flexible. The "robots.txt" file can express at least the following crawling policies.

- To exclude all web crawlers from the entire server;
- To allow all web crawlers to access the entire server;
- To exclude all web crawlers from parts of the server;
- To exclude certain web crawlers;
- To allow certain web crawlers;
- To allow certain files (which cannot be fulfilled directly because there is no *Allow line*.)

A sample "robots.txt" file to allow only one web crawler, the CINDI Robot, to crawl parts of the server is given below. It is worth mentioning that the Robot Exclusion Protocol is respected in the CINDI Robot.

```
# CINDI Robot knows which parts are prohibited
User-agent: CINDI Robot
Disallow:   /images/
Disallow: /aboutus/contact/
Disallow: /aboutus/stuff/

#for other web crawlers, crawling on this site is not allowed
User-agent: *
Disallow: /
```

2.3 Related Works

In this section, we briefly introduce some related concepts and techniques for focused web crawlers. In addition, we also present three related focused web crawling systems and point out the main differences among the CINDI Robot and these exemplary systems.

2.3.1 Related concepts and techniques

2.3.1.1 Focused web crawler performance metrics

The output of a crawler is a sequence of web pages crawled; thus any evaluation of a crawler's performance is based on this output [27]. The precision and recall are two widely accepted measures of performance for focused web crawlers. In this thesis, these two metrics will be used to evaluate our crawler and comparison web crawlers. Here we give the definitions of these two concepts. Precision is defined as the proportion of retrieved and relevant web pages to all the web pages retrieved, which is formulated as follow:

$$precision = \frac{|\{relevant\_documents\} \cap \{retrieved\_documents\}|}{|\{retrieved\_documents\}|}$$

Recall is defined as the proportion of retrieved and relevant web pages to all the relevant pages in the Web and it can be formulated as given below:

$$recall = \frac{|\{relevant\_documents\} \cap \{retrieved\_documents\}|}{|\{relevant\_documents\}|}$$

A desirable focused web crawler should possess both high precision and high recall but in most cases precision and recall increase at the cost of each other.

2.3.1.2 Context Graph algorithm

The Context Graph algorithm was first proposed by Diligenti and his colleagues in 2000 [28]. In the past eight years, it has been being an active research area: many variations have been presented [29, 30, 31] and many researchers have chosen the context graph algorithm as the basis of their work [32, 33, 34, 35]. By constructing a context graph, users learn web page representations that occur within a certain link distance of the target documents [28]. The link distance is defined as the minimum number of link steps needed to move from one page to another. These representations are used to train a set of classifiers to assign web pages into different layers. In the context graph, each layer is constructed using a strict link distance requirement. Layer 1 web pages have exactly one step from desirable documents and Layer 2 web pages are exactly two steps away from desirable documents and so on. As a result, the context graph can learn topics that are indirectly connected a pre-defined topic and thus increase recall. If there are n layers in the context graph, n classifiers are trained to assign all web pages into different layers. "During the crawling stage the classifiers are used to predict how many steps away from a target document the current retrieved document is likely to be [28]." The idea of using the context of a given topic to guide the crawling process could significantly increase both precision and recall. But we also realize that the context graph approach is based on the assumption that common hierarchies exist on documents within the same topic [29]. However, the Web does not have a homogeneous and well-organized structure. Thus the strict layer strategy is largely constrained. In practical applications, this classic context graph algorithm has several drawbacks. First, the strict link requirement may lower the classification accuracy. Both Layer 1 and Layer 2 web pages can be closely related to the topic of interest while Layer 4 or higher layer web pages can be of totally different topics. In this situation, the classifiers of Layer 1 and Layer 2 can not be delicate enough to make a correct classification while the classifiers for higher layers may not be easily trained. Secondly, the strict link requirement will bring too many classification processes. For example, in our task, we need as many as 5 layers to obtain a reasonable recall and thus in the worst case we need 5 times classifications to assign a web page into a proper layer. This is unacceptable in a real-time application.

Though there are many improvements that have been made on the classic context graph algorithm, no one has tried to remove the strict link distance requirement to make this algorithm more adaptable. Hence we propose a revised context graph as described in Section 4.3.4 to enhance the performance of the CINDI Robot.

2.3.1.3 Tunneling technique

Tunneling is the phenomenon that a crawler reaches some relevant pages on a path which does not only consist of relevant pages [36]. Relevant regions spread over the whole Web and can be visualized as illustrated in Figure 1. Seed URLs locate within relevant regions. But not all relevant regions over the Web can be covered by seed URLs. Relevant regions may not directly connect to each other. The major task of focused web crawlers is to unveil as many bridges among relevant regions as possible. Recall that both *precision* and *recall* are essential performance metrics of a focused web crawler. The tunneling can be prevented by a *pruning strategy* not allowing the crawler to visit any low relevance web page. This strategy can maintain a high precision, but it will result in a low recall. In this case, the discoverability of a focused web crawler only relies on how many seed URLs can be found. An alternative approach is to allow a focused web crawler to temporarily visit low relevance web pages with an expectation of reaching relevant regions that are not covered by seed URLs while trying to keep precision as high as possible.

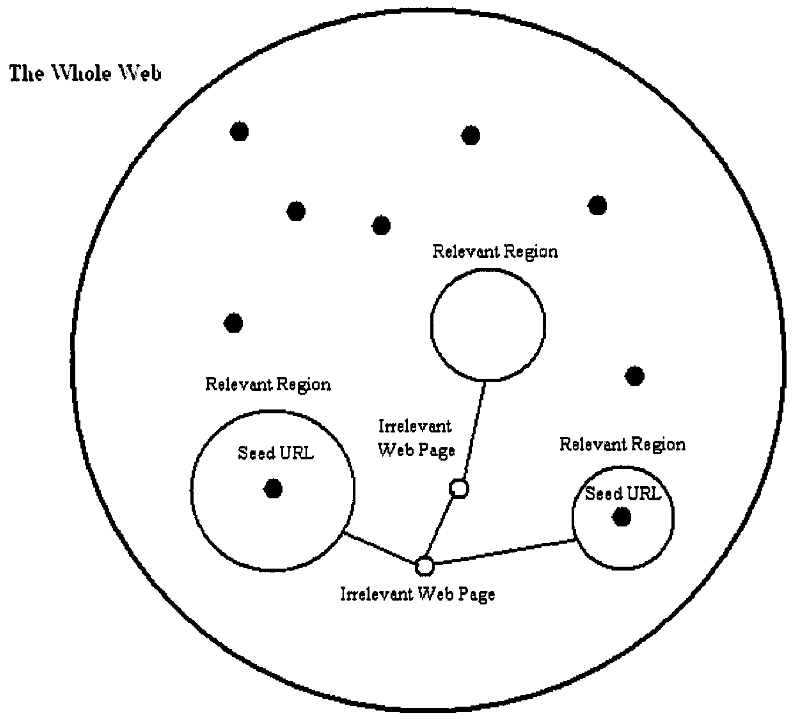


Figure 1 Illustration of tunneling

In [36], Martin Ester et al. implemented their tunneling technique based on taxonomies. During the crawling process, the crawler keeps an eye on the precision. If the precision goes down much faster than expected, it is necessary to broaden the focus of the crawl. For example, if a focused web crawler currently working on topic "basketball" can not find more relevant pages, it can generalize the topic to "sport" and expect to find more bridges leading to other relevant regions. And if the precision gets better than a pre-defined value after the generalization, the topic is specialized again.

Taxonomy based tunneling seems an ideal and intuitive approach to increase recall. However in real operations, it may face some fatal problems and may not work out for all topics. The first problem is the construction of a reasonable taxonomy tree. We have to collect enough training data for each topic and sub-topic. For a huge field like computer science and software engineering, it is not easy to achieve a consensus of how to reasonably divide topics and sub-topics. Even if we can get a reasonable consensus, the taxonomy structure is more complicated than a tree. It can be as complicated as a graph and topics may overlap. These facts make taxonomy based tunneling technique even harder to be used in real applications, especially in CINDI Robot.

2.3.1.4 Support Vector Machine classifier

The Support Vector Machine (SVM) method was developed by V. N. Vapnik [37] based on structural risk minimization principle and was introduced to text categorization by Joachims [38]. "Support Vector Machines are learning systems that use a hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory [39]".

The primary idea of SVM is using a high dimension space to find a hyper plane to do a binary separation and to keep the classification error rate minimum. The SVM exhibits a desirable performance in situations where only limited, nonlinear and high dimensional sample data are available. Even in the worst case where samples are inseparable, the SVM will classify these samples with the lowest error rate.

Consider a given linearly separable training sample

$$S = ((x_1, y_1), \ldots, (x_l, y_l))$$

Where vector $x_i \in R^m, i = 1,\ldots,l$; m is the input dimension; y is labeled as category of +1 (positive) or -1 (negative). In its basic form, the SVM tries to find a hyper plane in forms of $wx + b = 0$ to separate R^m into two half-spaces such that one half-space contains all positive samples and the other half-space contains all negative samples. Meanwhile this hyper plane should be with maximal margin. The idea is illustrated in Figure 2. The corresponding optimal classification is:

$$f(x) = \operatorname{sgn}\{w^* \bullet x + b^*\} = \operatorname{sgn}\{\sum_{i=1}^{l} \alpha_i^* y_i (x_i \bullet x) + b^*\} \qquad (1)$$

where parameters with superscript * mean the optimal solutions and *sgn* denotes the sign function. The parameter α^* can be obtained by solving the following quadratic optimization problem:

$$\textit{maximize}\ \ W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle,$$

$$\textit{subject to}\ \ \sum_{i=1}^{l} y_i \alpha_i = 0,\ \ \alpha_i \geq 0,\ \ i = 1,\dots,l.$$

and b^* can be found by using the primal constraints:

$$b^* = -\frac{\max_{y_i=-1}(\langle w^* \bullet x_i \rangle) + \min_{y_i=1}(\langle w^* \bullet x_i \rangle)}{2}$$

where the weight vector $w^* = \sum_{i=1}^{l} y_i \alpha_i^* x_i$.

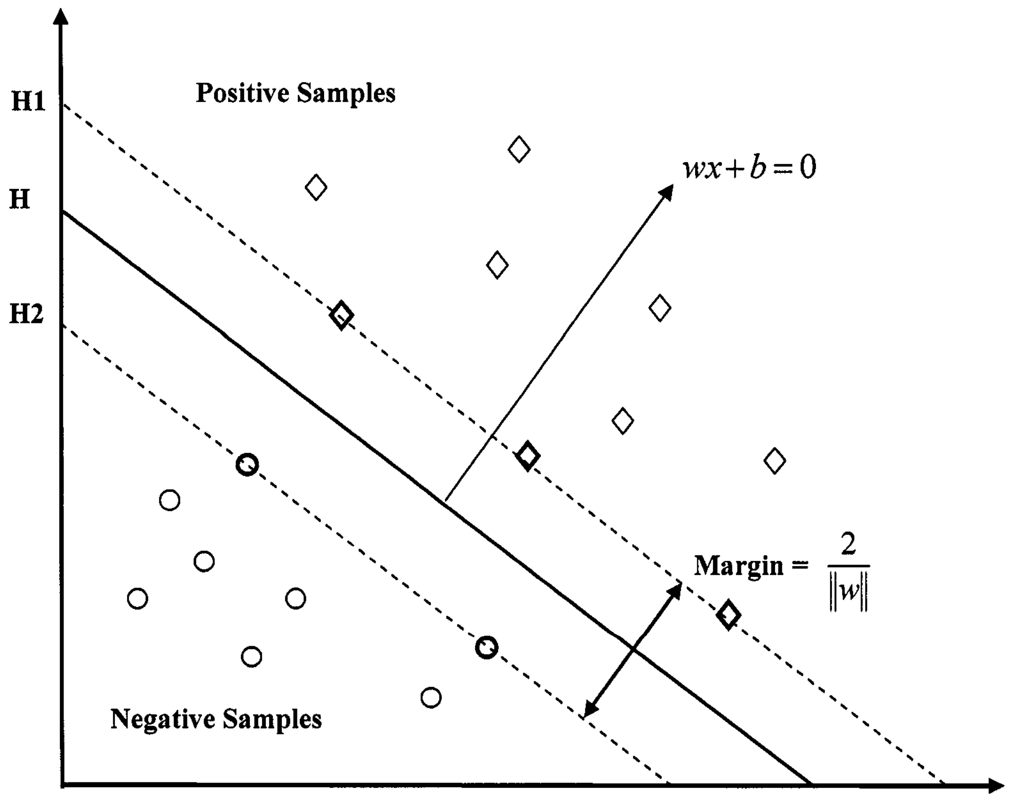


Figure 2 The hyper plane of SVM

Using the *Karush-Kuhn-Tucker complementary conditions* [39], we know that the optimal solutions α^*, (w^*, b^*) should satisfy

$$\alpha_i^*[y(\langle w^* \bullet x_i \rangle + b^*) - 1] = 0, i = 1,\ldots,l.$$

Since only points on hyper planes H1 and H2 can satisfy $y(\langle w^* \bullet x_i \rangle + b^*) = 1$, we can deduce other points' α_i=0. Using this conclusion and Equation 1, we know actually only those points on hyper planes H1 and H2 contribute to the classification. Hence they are called *support vectors*. In Figure 2, *support vectors* are highlighted in bold. This conclusion also explains why SVM is suitable for the situations where only limited training data are available.

For non-linear classification problems, slack variables have to be imported to get the optimal hyper plane. Now we can introduce kernel functions to transform non-linear classifications into linear classifications. A suitable kernel function can solve the non-linear problem without increasing the complexity of the calculation [40]. Different kernel functions are suitable for different types of problems. Four common used kernel functions are listed in Table 1.

Table 1 Kernel functions

Kernel	**Kernel function**
Linear	$K(x_i, x) = x_i \bullet x$
Polynomial	$K(x_i, x) = (s(x_i \bullet x) + c)^d$
Radical Basis Function, RBF	$K(x_i, x) = \exp(-\gamma \lvert x - x_i \rvert^2)$
Sigmoid tanh	$K(x_i, x) = \tanh(s(x_i \bullet x) + c)$

The corresponding SVM classification function can be rewritten by kernel function as follows:

$$f(x) = \mathrm{sgn}(\sum_{i=1}^{l} \alpha_i^* y_i K(x_i \bullet x) + b^*)$$

Using this function and user designed value representations, one can build his own Support Vector Machine classifier.

2.3.1.5 Naïve Bayes classifier

A Naïve Bayes classifier is an intuitionistic and simple probabilistic classifier based on Bayes' theorem with strong independence assumptions. It has been used in many applications, especially in text categorization. Though the assumption of independence for Bayes' theorem in text categorization often doesn't hold, Naïve Bayes classifier still works quite well in many real applications than one might expect. Compared to SVM classifier, Naïve Bayes classifier usually works well under a large training set.

"The probability model for a classifier is a conditional model $p(C \mid F_1, F_2, \dots, F_n)$ over a dependent class variable C with a small number of outcomes or classes, conditional on several feature variables F_1 through F_n [41]". Using Bayes' theorem, we can get

$$p(C \mid F_1, \dots, F_n) = \frac{1}{Z} p(C) \prod_{i=1}^{n} p(F_i \mid C) \qquad (2)$$

Where Z is a scaling factor dependent only on $F_1, \dots, F_n$. In text categorization, C denotes a given category and F denotes distinct words in the text.

2.3.2 Related systems

Since its creation, many focused web crawler systems have been proposed using various techniques [28, 29, 34, 36, 40, 42] and devoting to different goals [18, 24, 43]. In this section, three focused web crawlers are used to present the related works. The LSCrawler system proposed by M. Yuvarani [44] and the crawling system presented by Mohsen Jamali [45] are two relatively new, well-documented focused web crawlers while the meta-search crawler [43] is most similar to our work in terms of its motivation.

The LSCrawler system considers better recall as its goal and uses link semantics to guide the crawling process. The LSCrawler system is made up of seven components: Seed Detector, Crawler Manager, Crawler, HTTP Protocol Module, Link Extractor, Relevant Ontology Extractor and HyperText Analyzer. The Seed Detector is responsible to retrieve seed URLs from general purpose search engines; the Crawler keeps the URLs used for crawling and dynamically generates instances for the crawler based on URL repository size; the Crawler is capable of downloading web pages and storing documents; the HTTP Protocol Module opens HTTP connections upon request; the Link Extractor extracts hyperlinks and anchor texts from downloaded web pages and passes the information to the HyperText Analyzer; the Relevant Ontology Extractor fetches the relevant Ontology from the Ontology Repository; the HyperText Analyzer uses the relevant Ontology from the Relevant Ontology Extractor to determine the relevancy of the terms received from the Link Extractor. The URLs are hence prioritized by their relevancy. The system is compared with the full-text indexed search and exhibits a better recall. In the whole system, the crawling process is actually guided by the semantic of anchor texts. However, anchor texts are often too short to provide adequate information to arrange a reasonable crawling sequence. Content texts and URL patterns are good complements.

The hybrid focused web crawler proposed in [45] uses both link structure of documents and content similarity of pages to the topic to crawl the Web. The authors set the topic of *Sports* for their crawler and launched the crawling using a single seed URL. All web pages are mapped into one of three groups: Seed Pages, Candidate Crawled Pages and Uncrawled Pages. Seed Pages are repeatedly selected from Candidate Crawled Pages using a rank score. Only hyperlinks derived from Seed Pages are crawled. All crawled pages are first stored in Candidate Crawled Pages table and their rank scores are calculated after each session. The one with highest mark will be moved to Seed Pages table. The rank score is calculated using a combination of link structure and content similarity as below:

$$Rank(p) = (links\_to\_seed(p) + links\_from\_seed(p)) \times (0.1 + content\_similarity(p))$$

where p is a page from Candidate Crawled Pages; *links_to_seed* and *links_from_seeds* are number of links from the page p to Seed Pages and number of links from Seed Pages to it respectively. *content_similarity* is a number between 0 and 1 denoting the degree of similarity of a page to a domain. From the above formula, we can observe that this algorithm emphasizes the link structure over content similarity and it is essentially a compromise of the PageRank algorithm [46]. Using Seed Pages to mimic important pages in the PageRank algorithm is a promising way to remove the requirement of a large on-hand Web link structure database but it is also a potential limitation where the accuracy of this algorithm largely depends on the precision of Seed Pages. In addition, the *content_similarity* is calculated using vocabulary vectors and simple keyword matching. We deem that its classification accuracy can not be as good as SVM classifiers or Naïve Bayes classifiers. In addition, using only one seed URL to start the crawling process is an obvious flaw.

The Meta-Search focused web crawler in [43] is designed to build domain-specific web collections for scientific digital libraries. It shares the same goal of the CINDI Robot. The author addressed the drawbacks of traditional focused web crawlers caused by local search algorithms in the digital library construction. Since the Web pages are naturally organized into different regions by special hyperlink structures, most focused web crawlers using local search algorithms including Breadth-First Search, Best-First Search and Spreading Activation Algorithm will be confined by these regions. So the collections built by these traditional techniques often result in low recall, which means that it can not provide diverse enough documents to facilitate the scientific society. As a result, the authors of [43] borrowed the meta-search idea outside the focused crawling domain. They deemed that meta-searching multiple search engines and combining returned results can alleviate the problem of local search. In their implementation, a meta-searching component keeps generating queries from a domain-specific lexicon, retrieving diverse and relevant URLs by querying multiple search engines, and combining their top results. In their experiments, they showed that the meta-search can improve the quality of the collection. Resorting to multiple search engines and combining the results to enlarge the opportunity of finding new relevant Web regions are straightforward and the resulting improvement of recall is also obvious. In CINDI Robot, we also notice the importance of meta-search and hence employ our own meta-search strategy. Our meta-search strategy is used in the seed URL collection phase instead of in the real crawling phase, which can decrease the difficulties of controlling crawling processes. In the CINDI Robot, both Google and AltaVista are used to enrich our seed URL collections. In addition, Open Directory Project is also used, which further relieve the problem of local search. However, we also realize that only meta-search is not enough for an efficient focused web crawler. The meta-search idea unveils more relevant regions, but to efficiently crawl discovered regions, Web analysis algorithms and Web search algorithms are still vital to the success of a focused web crawler, which are especially emphasized in the CINDI Robot. So we are confident that our CINDI Robot can outperform this meta-search focused web crawler.

Content texts, URL patterns and anchor texts are useful clues to guide a focused web crawler. A multi-level inspection infrastructure can avoid many shortcomings of using either one of them alone. However, most current papers ignore the power of such a comprehensive system and the usefulness of URL patterns is overlooked. Therefore, in this thesis we design a novel multi-level inspection infrastructure which maximally makes use of all characteristics of web pages. This is also the main conceptual contribution of this thesis.

2.4 CINDI System

Concordia INdexing and DIscovering (CINDI) system was first conceived in 1994 by Dr. B. C. Desai [47]. The purpose of developing such a system is to allow users easy search for and access to resources available on the Internet. It provides fast, efficient and easy access to Web documents by using a standard indexing structure and building an expert system-based bibliographic system using standardized control definitions and terms [48].

The CINDI system is composed of seven subsystems, namely the CINDI Robot, the Conference subsystem [49], the Gleaning subsystem [50, 51], the Automatic Semantic Header Generator (ASHG) subsystem [48], CINDI Registration and Upload subsystem [28], the Search subsystem [51] and the Annotation subsystem [51]. The overall architecture of CINDI system is given in Figure 3.

CINDI Robot uses “pull” mode to discover and download research papers, technical notes, FAQs and so on from the Web, which will be elaborated in following chapters. The remainder of this section will briefly introduce other CINDI subsystems.

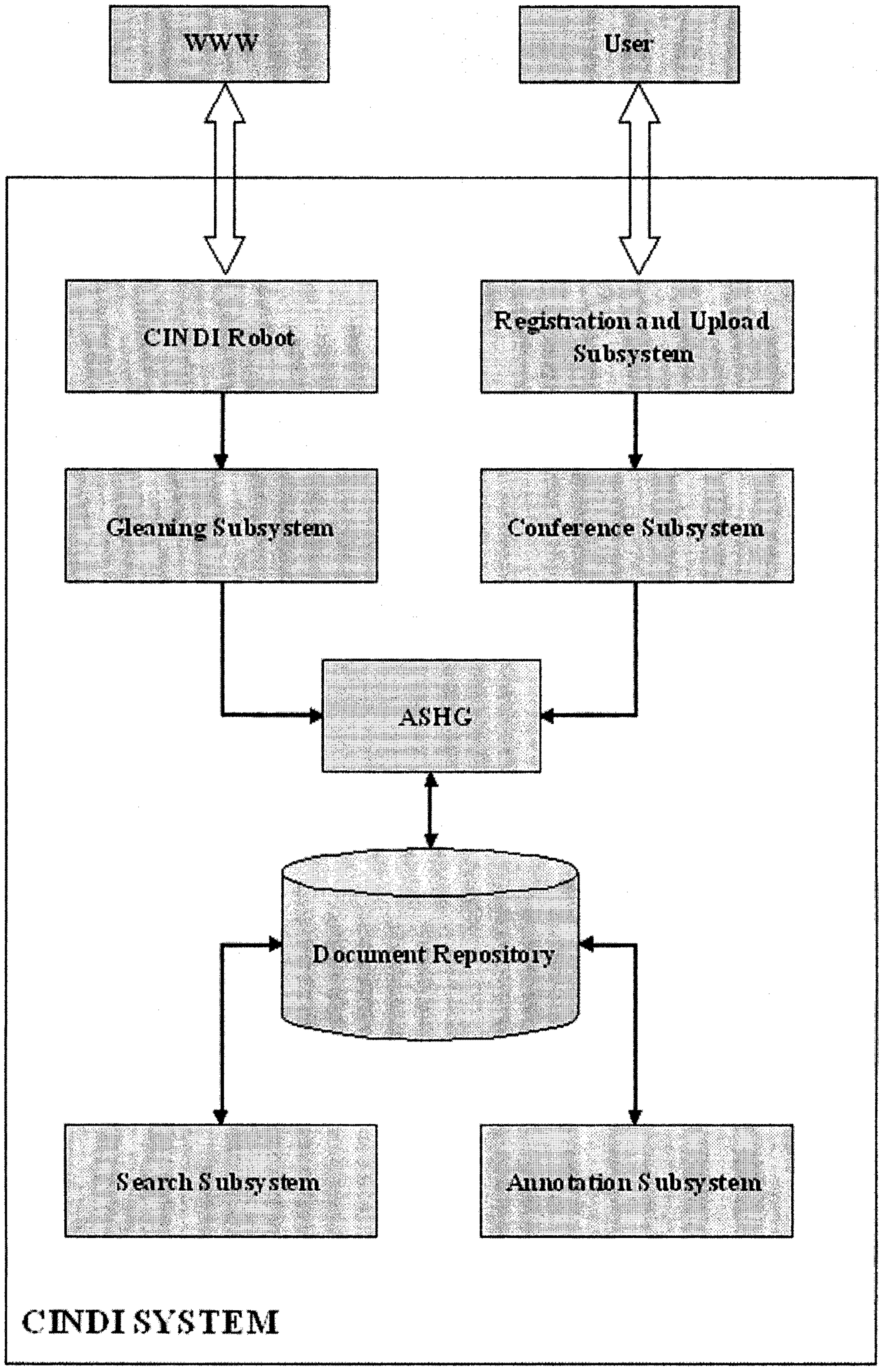


Figure 3 CINDI system architecture

In contrast to CINDI Robot, the CINDI Conference subsystem uses "push" mode to collect academic papers submitted to the CINDI system. The Conference subsystem defines seven types of users: administrators, authors, reviewers, program committee members, program chair, general chair and conference participants. These roles are set up according to a real conference and thus facilitate the management of a real conference. Authors can submit their papers to the CINDI system and this is the second source of documents of the CINDI system.

The Gleaning subsystem further consists of two modules: File Conversion Subsystem (FCS) and Document Filtering Subsystem (DFS). The Gleaning subsystem has direct interactions with CINDI Robot. The CINDI system accepts several file formats, including HTML, TXT, LaTex, RTF, PS, DOC, XML, TEX and PDF. However, all types of files have to be converted into a unified PDF format by the FCS and then be filtered by the DFS. The FCS is to provide a single document format to facilitate document processing in the subsequent CINDI subsystems [51]. The PDF format is chosen because PDF is the most common format of electronic documents. In FCS, a background daemon is established to automatically and regularly convert non-PDF files into PDF format. The DFS performs a sophisticated inspection algorithm to filter out irrelevant documents. The DFS divides all documents into two categories: *accepted* and *rejected* by utilizing Document Type Definition (DTD). The DFS tries to match a document to one of pre-defined DTDs. In CINDI DFS, there are four kinds of DTDs, including thesis DTD, technical paper DTD, academic paper DTD and FAQs DTD. If a document matches one of these DTDs, it is classified as accepted, otherwise it is rejected. The filtering result is sent back to CINDI Robot to guide the crawling process, which will be elaborated in following chapters.

The ASHG subsystem constitutes the core of the entire CINDI system. The ASHG subsystem parses and extracts a number of attributes of a research paper, including author, title, keywords, abstract and subject headings. If any field is absent, the corresponding section in the semantic header will be left blank. ASHG subsystem collaborates closely with the Search subsystem.

Due to the huge amount of documents collected in the CINDI digital library, an efficient Search subsystem is indispensable for users to find specific and appropriate information. The CINDI Search subsystem is composed of two main components: a document locating module and an interface supporting search by keywords and browsing by similarity links. The documents in the CINDI system are parsed by the ASHG subsystem and this information is inserted into the database, which forms the final database that is used for searching and locating a desirable document in the CINDI repository. A full-text indexing is then implemented over ASHG database field to provide a much faster response as compared to the one with the normal LIKE predicate. The CINDI Search module provides two kinds of search approaches, a Google-style basic search module and an advanced search module which supports sophisticated Boolean operators. The basic search performs a keyword based search, in which the conjunctive keyword queries are ranked higher than disjunctive keyword queries. The basic search provides a simple way for novice users to find a desirable document without any prior knowledge about the structure of the underlying data. The advanced search satisfies veteran users to construct Boolean expressions to identify a document from full texts, authors, abstracts or annotations.

The CINDI Registration and Upload subsystem divides all CINDI users into three groups, namely guests, authorized users and contributors. A guest is a user who can only search and download documents from the CINDI system. An authorized user is a user who can make annotations in addition to the privileges possessed by a guest. A contributor is a user who submits academic papers to the CINDI system and also possesses the same privilege as an authorized user.

The CINDI Annotation subsystem permits registered users to comment on CINDI documents. The visibility of annotations is open for all users of the CINDI system. All annotations in CINDI system are made independently, without the need to edit the associated documents, and are stored separately in an annotation database, which is linked with the ASHG subsystem database. For each annotation, the annotation text, the annotation author and the target document are recorded. An annotation search module is provided in the annotation subsystem.

Chapter 3

CINDI Robot System Architecture

The CINDI Robot consists of five indispensable components: Seed Finder, Web Crawler, Link Analyzer, Statistics Analyzer and File Fetcher. As mentioned before, DFS plays an essential role in the execution of CINDI Robot. It provides statistical information to CINDI Robot for an adaptive strategy. Figure 4 illustrates the system architecture of CINDI Robot.

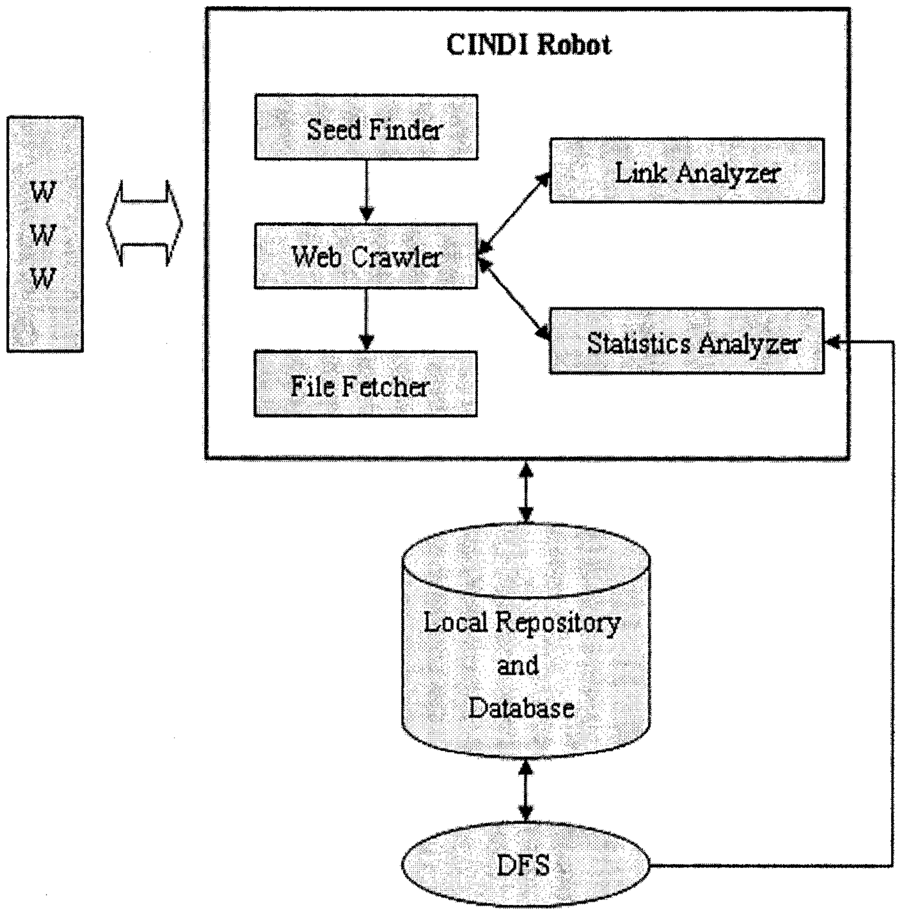


Figure 4 CINDI Robot system architecture diagram

The current CINDI Robot is implemented in JAVA as a multi-threaded application, which uses typically 75 threads in crawling processes. It begins a crawling process by adding seed URLs found by Seed Finder into the URL frontier. The Web Crawler threads get URLs from the URL frontier and use preliminary filters to filter out irrelevant URLs. The page fetcher of Web Crawler downloads web pages using URLs which pass the preliminary filter. The HTML parser embedded in Web Crawler parses the text content and extracts anchor texts and outer links. The text contents of web pages are sent to the Link Analyzer, which classifies a web page into one of three pre-defined categories: relevant pages, irrelevant pages and potentially relevant pages. This classification result will be sent back to Web Crawler to guide the current crawling process and be sent to the Statistics Analyzer. The Statistics Analyzer provides real-time statistical information to help Web crawler threads get out of "trap" sites and also collaborates with DFS to guide further crawling process. Once a potentially relevant document is identified, its URL will be passed to the File Fetcher. The File Fetcher downloads the corresponding document and informs DFS to examine the document.

3.1 Seed Finder

For a focused web crawler, it is difficult to make the crawler remain in the topic while trying to obtain a reasonable Web coverage. So it is vital to collect a large set of relevant seed URLs to initialize crawling processes. According to Brian D. Davison's research result [52], the topical locality is the basis of most web applications, especially for focused web crawlers. The topical locality is defined as the phenomenon that most web pages are linked to others with related content [52]. Thus he concludes the chance of getting relevant pages by following a topic relevant web page is significantly higher than by following a randomly selected page. This forms the basis of the success of a focused web crawler.

For focused web crawlers, a higher precision means a better capability of staying within relevant regions, thus better downloaded document quality and less workload for DFS; a higher recall means a better capability of penetrating the Web, thus better downloaded document quantity. A set of well-selected seed URLs can bring significant crawling performance improvements in terms of both precision and recall.

In CINDI Robot, the Seed Finder aims to find as many web sites as possible, which would most likely contain topic related documents or most likely lead to those web sites containing such documents. These sites are good starting points for the whole crawling process. In CINDI Robot, we propose two methods to detect these initial seed URLs: *ODP* [11] *based approach* and *general purpose search engine based approach.* As mentioned before, ODP is the largest and most comprehensive web directory in the Web. It provides publicly accessible data files of web site links within each category. We deem it as the best source of seed URLs considering the fact that human editing should bring higher quality.

In addition, resorting to general purpose search engine for seed URLs is a popular idea for the construction of a focused web crawler. But how to get best seed URLs from general purpose search engine still poses a huge challenge on focused web crawlers. The previous version of CINDI Robot used this general purpose search engine based approach to get seed URLs [4]. In [4], the author did a survey on three popular general purpose search engines: MSN, AltaVista and Google and chose the latter two as the source of the seed URLs according to a considerably better accessible web page number. He manually submitted the query phrases "computer science department" and "computer science publications" to AltaVista and Google Web API [26] and combined the search results from these two search engines. Duplicate entries obtained from these two search engines were excluded. The combination of results from both AltaVista and Google provided nearly 50% more seed URLs than from a single search engine. Especially, to retrieve seed URLs from AltaVista, two parsing processes were needed in order to get rid of commercial sponsor web sites. However, using only two phrases "computer science department" and "computer science publications" to retrieve relevant seed URLs will largely constrain the number of seed URLs. The obtained seed URLs quality is not as good as we expect. Thus we make some modifications of the previous method. The detailed implementation will be elaborated in Chapter 4.

3.2 Web Crawler

CINDI Robot Web Crawler infrastructure consists of five parts: URL frontier, preliminary filter, page fetcher, HTML parser and anchor text analyzer.

3.2.1 URL frontier

All Web Crawler threads share a single synchronized URL frontier. This is important in order to avoid the abnormality where all threads may converge to the same region. Two priority queues are set up in the URL frontier to express the extent of relevance of incoming URLs. All incoming URLs are classified into two categories: *high priority URL* and *low priority URL* according to how relevant they are related to our topic. All seed URLs found by Seed Finder are marked as high priority. At the beginning of each crawling process, the URL frontier retrieves 200 seed URLs from NEW_SITES table and put them into the high priority queue. During crawling processes, new web sites will be found and put into URL frontier after they are assigned a priority level. Each web crawler thread first retrieves URLs from the high priority queue as long as there is still any high priority seed URL left. Otherwise a low priority seed URL is retrieved.

This is different from the previous version of CINDI Robot. In [4], all new web sites are inserted into a FOREIGN_LINK table and may not be crawled for a relatively long period. For each crawling cycle, only a fixed number of web sites will be traversed. In the current design, the number of web sites that will be crawled in a cycle is decided by how many web sites we can find in this cycle. The biggest benefit of this design is to approach the nature of the focused web crawling: most relevant web pages are crawled first. We have no reason to separate two strongly related web sites into two crawling cycles, especially considering that there is no guarantee when the latter cycle will begin.

One feature of CINDI Robot is that it always crawls within a domain site. That means CINDI Robot won't retrieve a new web site from the URL frontier until it finishes the current one. A crawling cycle ends when the URL frontier is empty.

3.2.2 Preliminary filter

The preliminary filter performs a URL pattern inspection in order to remove useless web pages before actually crawling them. It is an efficient way to increase both precision and crawling speed. Due to the domain hierarchy of URLs, URL patterns can give additional information on detection of a web page's relevance. We also found other heuristics to prune irrelevant web pages. Detailed implementation is elaborated in Chapter 4.

3.2.3 Page fetcher

The page fetcher in the Web Crawler component is different from the File Fetcher component. In CINDI Robot, all existing file formats are grouped into three categories. The first category is formed by file formats that are acceptable to the page fetcher, including: HTML, HTM, ASP, PHP, JSP, SHTML and CFM. In addition, there is a special case on the Web where a URL is like "http://www.cs.concordia.ca/index/". In this case, we deem this URL is of one of the file formats acceptable to the page fetcher. The second category includes file formats that are downloadable by File Fetcher: PDF, PS, DOC, TXT, LaTex, TEX, XML, HTML and RTF. All other file formats are of no interest in CINDI Robot and are excluded during the crawling process. The page fetcher downloads web pages in acceptable formats and temporarily keeps them in a string buffer. Then the page fetcher informs the HTML parser to parse these web pages.

3.2.4 HTML parser

The HTML parser performs three main tasks. The first one is to extract useful parts of a web page's HTML code, including hyperlinks, anchor texts, title if any, "meta keywords" information if any and "meta http-equiv ="refresh"" if any.

Generally, there are four kinds of hyperlinks in HTML documents: anchor tags (<A>), image tags (<IMG>), map and area tags and frame (iframe) tags [53]. Because the main goal of CINDI Robot is to collect academic and scientific documents, only anchor tags are used in CINDI Robot to dig out new web pages. A commonly used anchor tag structure is given below:

<A href="url" name="name" title ="title" target="target">Anchor Text</A>

The href attribute is used to address the URL of the web page to link to and the anchor text is used to predict the relevance of the linked web page. CINDI Robot can use the anchor text to decide whether to follow a URL or not. The HTML parser extracts the href value and the anchor text from an anchor tag for further analyses.

There are several "meta" tags defined in the HTML language, including: generator, keywords, description, author, robots and http-equiv. CINDI Robot is interested in two of them, namely keywords and http-equiv. “meta keywords” information presents web pages’ keywords to a web crawler. “meta http-equiv =“refresh”” prompts a web crawler to reload the current page or redirect to a new web page. The “meta keywords” is used to perform a fast web page classification.

All URLs used by the page fetcher to retrieve a web page should be in the form of absolute URL. However the URLs extracted from anchor tags can be either in an absolute URL form or in a relative URL form. The HTML parser performs the task of converting a relative URL to an absolute URL. Sometimes due to some human mistakes, a URL may be written in a way that can not be recognized by a web crawler. Some of these human mistakes include misuse of back slash “\”, unconventional relative URL and using single “/” after “http:” [4]. The HTML parser is responsible to rectify these mistakes.

Sometimes, a research paper can also be in HTML format. The third responsibility of the HTML parser is to identify this kind of documents. Usually, a research paper falls into a specific pattern. The words, “abstract”, “keywords”, “introduction”, “chapter”, “reference”, “bibliography”, “future work”, “conclusion”, “appendix” and “acknowledgement”, are good indications of a research paper. By identifying these words, the HTML parser can determine if a web page is a research paper.

3.2.5 Anchor text analyzer

One important assumption of focused web crawlers is that anchor text is descriptive of the content of the web page being pointed to [54]. “Focused web crawlers need to predict the benefit of downloading unvisited pages based on the information derived from pages that have been downloaded [55]". Thus using anchor text to guide the crawling process is extremely important for a focused web crawler. The anchor text analyzer uses anchor text obtained from the HTML parser to predict the relevance of a web page. The anchor text analyzer sets up two inspection patterns. If a web page is classified as relevant by Link Analyzer, the anchor text analyzer uses keyword matching to prune irrelevant outer links; if a web page is classified as irrelevant but may lead to relevant web pages, the anchor text analyzer uses keywords matching to quickly identify a potential relevant outer links.

3.3 Link Analyzer

Link Analyzer is the intelligence core of CINDI Robot. In contrast to the previous version of Link Analyzer whose responsibilities are taken over by the preliminary filter in Web Crawler, the current Link Analyzer employs two Naïve Bayes text classifier [55] and one Support Vector Machine text classifier [39] to implement a revised context graph algorithm. The CINDI Robot categorizes all web pages on the Web into three categories. The first category is formed by relevant web pages. The second category is constituted of web pages that are not directly related to computer science and software engineering but may lead to relevant web pages. An example web page from the second category is "www.concordia.ca". This web page is irrelevant per se. However it can lead to a relevant page, "www.cs.concordia.ca". So it is also important for the crawling process. The third category contains totally "useless" web pages.

The revised context graph maps all web pages into different layers according to the categories defined in the previous paragraph. Figure 5 illustrates the idea of the revised context graph, where the distance requirement is elaborated in Chapter 4. Layer 1 web pages correspond to the web pages in the first category; Layer 2 web pages correspond to the web pages in the second category. All "useless" web pages are excluded from the revised context graph. And in the center are desirable documents. By constructing such a context graph, the crawler learns about which topics are directly or indirectly related to the target topic.

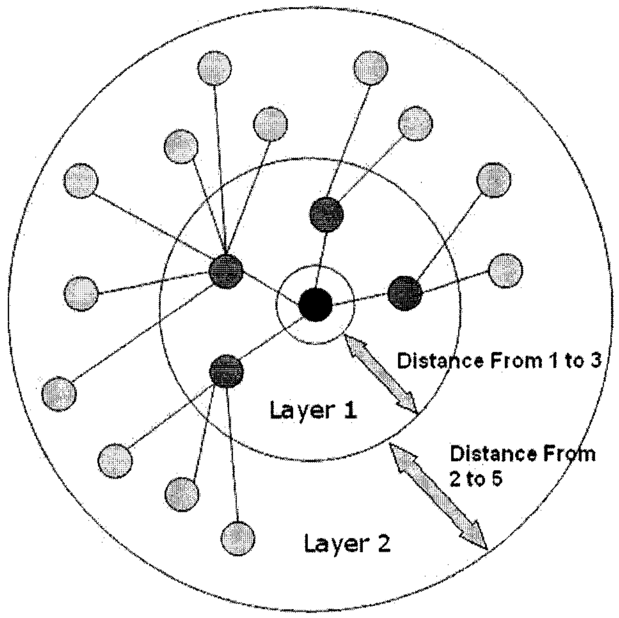


Figure 5 Revised Context Graph

An incoming web page passed to Link Analyzer is first tested by a text classifier to see if it should be put into Layer 1. The text classifier can be a Support Vector Machine text classifier or a Naïve Bayes text classifier. The Support Vector Machine text classifier works in the initial crawling phase while the Naïve Bayes text classifier is activated after the CINDI Robot has established a solid knowledge base. In order to increase recall, another Naïve Bayes text classifier is needed to check if an irrelevant web page is still worthy to be further examined. That is if it is a Layer 2 web page.

On the Web, on-topic regions are usually separated by off-topic regions. Thus it is sometimes necessary for a focused web crawler to go through some off-topic regions to get to the next relevant one [36]. In order to increase recall, a technique is required to allow the CINDI Robot to follow a series of irrelevant pages for a relevant region. This technique is called as *tunneling technique* [54]. A good *tunneling technique* makes the CINDI Robot achieve a high recall while still maintaining a high precision. This is elaborated in Chapter 4.

3.4 Statistics Analyzer

In the previous version of the CINDI Robot, the Statistics Analyzer started after the completion of crawling, file fetching and document filtering [4]. Statistics Analyzer utilized information gained from previous crawling process as well as the feedback provided by DFS to make the robot more selective in subsequent crawling processes. This is considered as a "static" way to guide the crawling process. It is essential for a focused web crawler. However, sometimes on the Web an entrance web page of a web site may be a "trap" page. It is related to the pre-defined topic however the rest of a web site is totally irrelevant. In this case, we need some mechanisms to help CINDI Robot get out of this web site as soon as possible instead of after only collecting statistical data from DFS. The current Statistics Analyzer performs a real-time inspection and uses real-time feedbacks to indicate a crawling termination.

Compared to Link Analyzer which makes its decision based on web page textual contents, Statistics Analyzer works on a higher level. It analyzes at the URL level rather than at the textual content or anchor text level. In order to accommodate the preliminary filter, Statistics Analyzer provides information for the URL pattern inspection. Statistics Analyzer not only collects the Page Relevancy Rate (PRR) and the Document Download Rate (DRR) during the crawling process, but also receives the downloaded files' filter results from DFS to calculate the Accepted Document Rate (ADR). All of these three parameters are defined in the unit of a URL directory.

The Page Relevancy Rate is defined as the ratio of the number of relevant pages over total page number under a URL directory. It can be formulated as follows:

$$PRR = \frac{relevant\_page\_number\_in\_the\_directory}{total\_page\_number\_in\_the\_directory}$$

It reflects how closely a web site (a URL directory) is related to computer science or software engineering. The Document Download Rate is defined as the ratio of the number of documents downloaded over the number of web pages visited under a directory and it reflects the density of the links to files under a directory. It can be calculated by the following equation.

$$DDR = \frac{downloaded\_document\_number\_in\_the\_directory}{total\_visited\_web\_page\_number\_in\_the\_directory}$$

The Accepted Document Rate is defined as the ratio of the number of documents accepted over the number of documents downloaded, which is formulated as below.

$$ADR = \frac{accepted\_downloaded\_document\_number\_in\_the\_directory}{total\_downloaded\_document\_number\_in\_the\_directory}$$

PRR and DDR can be learned by Statistics Analyzer itself during the crawling process; while for ADR CINDI Robot has to resort to DFS.

Using these three parameters, Statistics Analyzer can set up a set of policies for the crawling process. First is the construction of two types of directory lists. In the previous version of the CINDI Robot, two concepts are employed, namely *stop-directory list* and *to-be-avoided directory list* [4]. Former experiments proved they can significantly improve CINDI Robot's performance. Thus they are retained and improved in the current CINDI Robot.

The *stop-directory list* applies to all web pages. If a web page URL is under one of the directories in the *stop-directory list*, it won't be crawled. Due to its generality, we set a strict requirement for stop directory selection. Otherwise, we would miss numerous relevant web pages. In [4], the *stop-directory list* records directory names under which no downloaded documents are found to be accepted. In the current version of CINDI Robot, the *stop-directory list* is augmented to include directory names under which no relevant web pages are found. This modification makes the *stop-directory list* work in a partially dynamic way. To perform the stop directory selection proposed in [4], the *stop-directory list* can only be constructed after completion of a crawling process. While the new *stop-directory list* selection policy makes it work during a crawling process.

The *to-be-avoided directory list* is established for each crawled web site in order to perform an efficient re-crawling. For a specific web site, its *to-be-avoided directory list* records all directory names under which there is an expectation of not finding any relevant documents. Similar to the *stop-directory list*, the *to-be-avoided directory list* selection requirements are also augmented, which is elaborated in Chapter 4. We also notice that the structure of a web site is subject to change, thus new to-be-avoid directories are allowed to be added into the list. To perform a re-crawling, the CINDI Robot will check this web site's *to-be-avoided directory list* and skip all directories in the list.

In addition to these two directory lists, for real-time monitoring Statistics Analyzer also keeps an eye on the PRR and discards a web site if its PRR is less than 5% after crawling 200 web pages. Statistics Analyzer is also responsible to exclude less valuable web sites from re-crawling.

3.5 File Fetcher

File Fetcher downloads various documents using document URLs provided by Web Crawler. In [4], the File Fetcher worked in a sequential order. It was launched after the Web Crawler finished a crawling cycle. In the new CINDI Robot, File Fetcher works parallel with other components. This modification can shorten the whole working cycle of the CINDI Robot.

For a large scale system as CINDI Robot, File Fetcher has to perform many tasks. It can be further divided into two components, namely the file parser and the file writer. The file parser checks if a document has been downloaded by examining the URL of this document in CINDI Robot database. Duplicate documents will be directly removed. The file parser also checks a document's name and a document's size to remove undesirable files. For example, files with names such as "cv", "resume", "homework" are not welcome. An "8K" assumption proposed in [4] is respected in the current CINDI Robot. According to an experiment over 52,552 downloaded PDF documents, the author of [4] found all documents with sizes less than 8K are invalid. Therefore, we set the 8K as the cut-off point. We also notice that file sizes changes under different file formats, so this assumption is only applied to PDF documents. However up to 49.42% of documents discovered by CINDI Robot are in PDF format [4], this assumption can bring substantial time saving for both File Fetcher and DFS. Sometimes, an identical document may come from different URLs; hence a simple comparison on URLs is not enough to remove all duplicate files. The file parser employs a MD5 digital signature checker to verify if two documents with the same name are identical [4]. The file writer performs the actual work of downloading a file from a remote server and inserting file information into database. The file writer also performs the renaming mechanism. Different files may share the same file name. Considering CINDI Robot downloads up to 800,000 files, this phenomenon is very common. According to our survey given in Chapter 4, we can confirm this speculation. The file name "content.pdf" is shared by 306 files. Thus a good renaming mechanism is very important to CINDI Robot. In [4], the file writer attaches a number to the file name to rename a file. For example, if a file is named as "abc.pdf", a second file with the same name will be named as "abc1.pdf", a third file with the same name will be named as "abc2.pdf". Since there are so many documents in CINDI database, this information cannot be kept in memory, but stored in hard disk. In this case, if a file name is very popular, a rename operation may involve as many as hundreds of database operations. In the above extreme example, it takes up to 90 seconds to complete a renaming operation. This is unacceptable for CINDI Robot. A simple but efficient renaming mechanism is used in the current CINDI Robot, which is described in Chapter 4.

3.6 Database of CINDI Robot

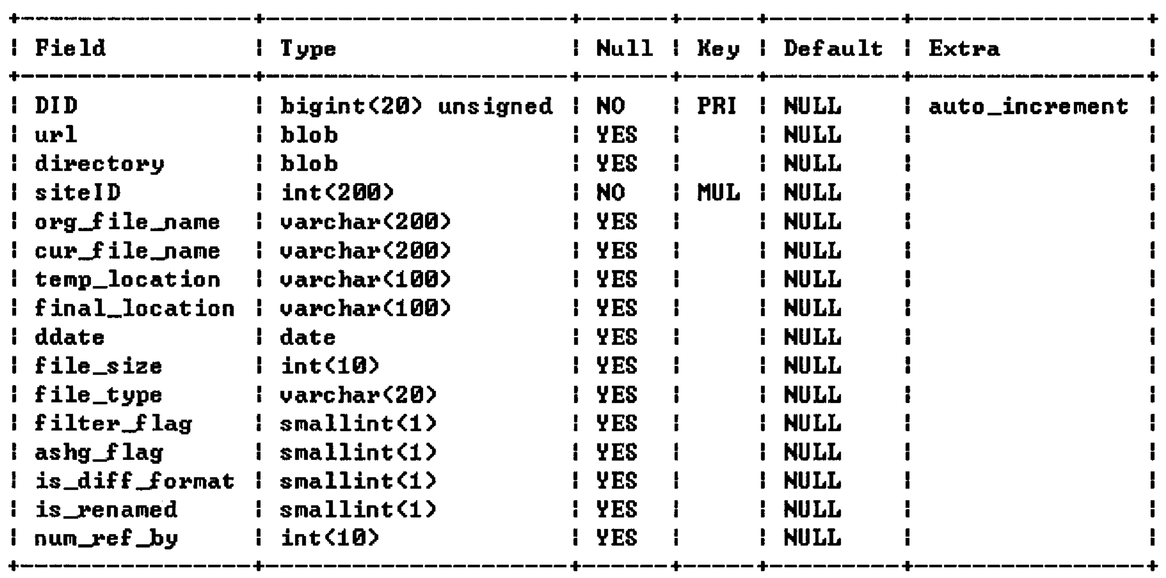
CINDI Robot chooses MySQL as its database server due to its scalability, flexibility and high performance [56]. In the previous version of CINDI Robot, it created and operated on 16 database tables, including NEW_SITES, SEED_URL, VISITED_PAGES, FOREIGN_LINK, PRE_DOWNLOAD_INFO, DOWNLOAD_STATUS, DOMAIN_KEYWORD, DOCUMENT_REF_BY, SITE_REF_BY, LINK_REF_BY, LEVEL_STATS, SITE_STATS, RDVT, STOP_DIR_LIST, DIR_TO_BE_AVOIDED and CRAWLED_SITES. To accommodate modifications made in the current CINDI Robot, we consequently re-design the CINDI Robot database. To accelerate the crawling speed, we choose CSV files to store data instead of a database wherever possible.

NEW_SITES table remains from the previous design. It stores new sites discovered by Seed Finder. It contains only two fields, namely *SID* and *url*. In [4], the SEED_URL table stores URLs used in current crawling. We deem it is not very useful. Instead seed URLs are loaded into Java Vectors at the beginning of each crawling cycle. VISITED_PAGES table also remains in the current design, but the parent ID attribute is abandoned. In [4], the parent ID is designed in order to unveil the web structure of each web site. However, this design does not really contribute to the current CINDI System, but only brings extra cost. Figure 6 gives the schema of the VISITED_PAGES table.

```
+-----------+---------------------+------+-----+---------+----------------+
| Field     | Type                | Null | Key | Default | Extra          |
+-----------+---------------------+------+-----+---------+----------------+
| PID       | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| url       | blob                | YES  |     | NULL    |                |
| title     | varchar(100)        | YES  |     | NULL    |                |
| page_name | varchar(100)        | YES  |     | NULL    |                |
| is_valid  | smallint(1)         | YES  |     | NULL    |                |
| siteID    | int(200)            | NO   | MUL | NULL    |                |
| PDATE     | date                | YES  |     | NULL    |                |
+-----------+---------------------+------+-----+---------+----------------+
```

Figure 6 Schema of the VISITED_PAGES table

Since new discovered foreign links are directly sent back to the URL frontier and get crawled in current crawling cycle, the FOREIGN_LINK table is also removed. PRE_DOWNLOAD_INFO table is also removed from the CINDI Robot database, and corresponding information is kept in a Java Map object. Once a file is retrieved, file related information is inserted in to the DOWNLOAD_STATUS table, which keeps all information of downloaded files. The schema of the DOWNLOAD_STATUS table is showed in Figure 7.



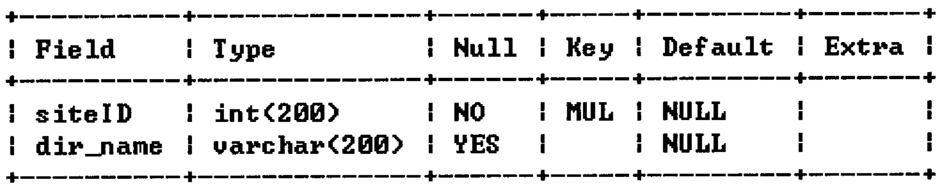
```
+----------------+---------------------+------+-----+---------+----------------+
| Field          | Type                | Null | Key | Default | Extra          |
+----------------+---------------------+------+-----+---------+----------------+
| DID            | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| url            | blob                | YES  |     | NULL    |                |
| directory      | blob                | YES  |     | NULL    |                |
| siteID         | int(200)            | NO   | MUL | NULL    |                |
| org_file_name  | varchar(200)        | YES  |     | NULL    |                |
| cur_file_name  | varchar(200)        | YES  |     | NULL    |                |
| temp_location  | varchar(100)        | YES  |     | NULL    |                |
| final_location | varchar(100)        | YES  |     | NULL    |                |
| ddate          | date                | YES  |     | NULL    |                |
| file_size      | int(10)             | YES  |     | NULL    |                |
| file_type      | varchar(20)         | YES  |     | NULL    |                |
| filter_flag    | smallint(1)         | YES  |     | NULL    |                |
| ashg_flag      | smallint(1)         | YES  |     | NULL    |                |
| is_diff_format | smallint(1)         | YES  |     | NULL    |                |
| is_renamed     | smallint(1)         | YES  |     | NULL    |                |
| num_ref_by     | int(10)             | YES  |     | NULL    |                |
+----------------+---------------------+------+-----+---------+----------------+
```

Figure 7 Schema of the DOWNLOAD_STATUS table

The *directory* field records the directory name of a document link. Suppose a file URL is http://br.endernet.org/~akrowne/elaine/dlib/papers/giles/context_focused.pdf, its directory name is /~akrowne/elaine/dlib/papers/giles/. The *siteID* indicate the web site ID in which it is discovered. The *org_file_name* stores a document's original name and the *cur_file_name* field records its current name. The *filter_flag* field indicates whether a document is accepted or not and it is set by DFS. As aforementioned, CINDI Robot interacts directly with DFS and obtains document relevance feedbacks. The DOWNLOAD_STATUS table is the interface between these two systems. If a document is acceptable, DFS sets its *filter_flag* value to 1, otherwise it sets the *filter_flag* value to 2. The *final_location* field is also set by DFS. The entries of rejected documents are periodically removed. The *ashg_flag* is set by ASHG when a semantic header has been generated for this document. If there are other formats of a file existing in CINDI document repository, the *is_diff_format* field is set to 1. The *num_ref_by* field indicates the number of times a document is referred by other web pages. This information provides an attractive feature for other CINDI subsystems. For example, in future a user may search a paper according to its popularity which is reflected by referred times.

The DOMAIN_KEYWORD table is also removed from the CINDI Robot database. Its content is stored in a CSV file instead and used as prior knowledge of "meta keywords". The previous LEVEL_STATS table maintained the statistic information of document distribution over directory levels. According to the experiment over 106,416 documents [4], the author came to the conclusion that it is unlikely to find a relevant document in a URL with more than 7 directory levels. This conclusion is utilized in the current CINDI Robot for the URL pattern inspection. Since we are following this conclusion, the LEVEL_STATS table is omitted in the current database.

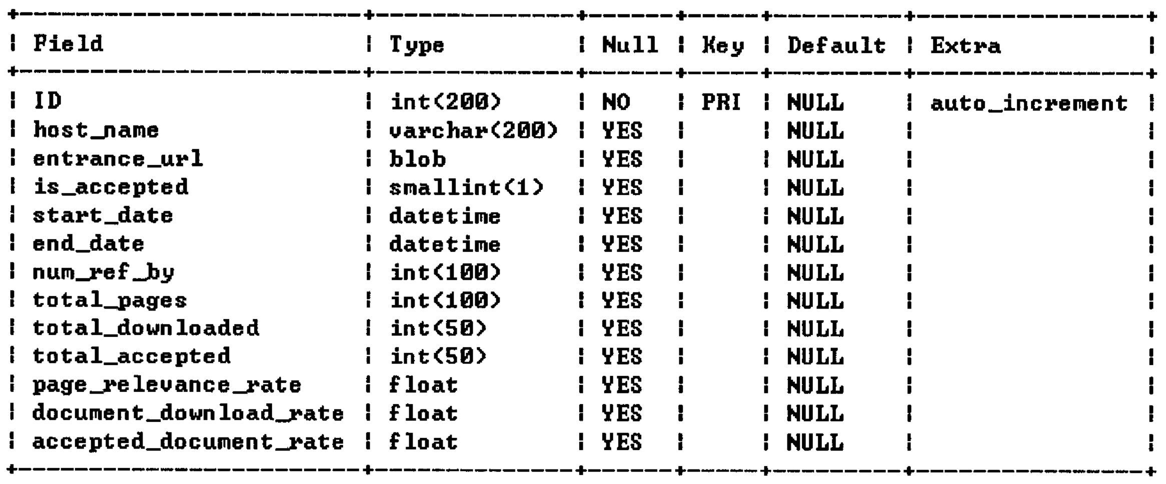
The Representative Document Vector Table (RDVT) was designed to contain anchor texts that appeared most frequently in the home pages of positive sample seeds. In our current design, all anchor text keywords are kept in two CSV files and are used for URL pattern inspection. The STOP_DIR_LIST table and the DIR_TO_BE_AVOIDED table are kept in CINDI Robot database and record *stop-directory list* and *to-be-avoided directory list* respectively. The STOP_DIR_LIST table contains two fields: *did*, *dir_name* which is the name of a stop directory. As we mentioned before, the DIR_TO_BE_AVOIDED table records to-be-avoided directories for every crawled web site. Its schema is given in Figure 8.



```
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| siteID   | int(200)     | NO   | MUL | NULL    |       |
| dir_name | varchar(200) | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

Figure 8 Schema of DIR_TO_BE_AVOIDED Table

The previous LINK_REF_BY table maintained the records of link cross references for downloaded documents [4]. This functionality is condensed to a *num_ref_by* field in the DOWNLOAD_STATUS table. The previous SITE_REF_BY table indicated how often a web site was referred by other sites. This frequency indicated the popularity of a web site and was an indicator for the re-crawling policy. However this information can also be simplified as a *num_ref_by* field in the CRAWLED_SITES table. In addition, we incorporate information of the SITE_STATS table into the CRAWLED_SITES table. Figure 9 presents the schema of the current CRAWLED_SITES table.



```
+-------------------------+--------------+------+-----+---------+----------------+
| Field                   | Type         | Null | Key | Default | Extra          |
+-------------------------+--------------+------+-----+---------+----------------+
| ID                      | int(200)     | NO   | PRI | NULL    | auto_increment |
| host_name               | varchar(200) | YES  |     | NULL    |                |
| entrance_url            | blob         | YES  |     | NULL    |                |
| is_accepted             | smallint(1)  | YES  |     | NULL    |                |
| start_date              | datetime     | YES  |     | NULL    |                |
| end_date                | datetime     | YES  |     | NULL    |                |
| num_ref_by              | int(100)     | YES  |     | NULL    |                |
| total_pages             | int(100)     | YES  |     | NULL    |                |
| total_downloaded        | int(50)      | YES  |     | NULL    |                |
| total_accepted          | int(50)      | YES  |     | NULL    |                |
| page_relevance_rate     | float        | YES  |     | NULL    |                |
| document_download_rate  | float        | YES  |     | NULL    |                |
| accepted_document_rate  | float        | YES  |     | NULL    |                |
+-------------------------+--------------+------+-----+---------+----------------+
```

Figure 9 Schema of the CRAWLED_SITES table

In the CRAWLED_SITES table, the *is_accepted* field records if this web site is a “trap” web site and if it is worth being recrawled. The *num_ref_by* indicates how many times a web site is referred. The *total_pages*, *total_downloaded* and *total_accepted* fields present the total number of web pages crawled, total number of documents downloaded and total number of accepted documents respectively.

As a summary, we recall all tables used in the current CINDI Robot database: NEW_SITES table, VISITED_PAGES table, DOWNLOAD_STATUS table, STOP_DIR_LIST table, DIR_TO_BE_AVOID table, and CRAWLED_SITES table.

Chapter 4

CINDI Robot Implementation and Heuristics

In this chapter, we describe the implementation details of CINDI Robot. Recall that CINDI Robot is a multi-threaded, large-scale focused web crawling program that is implemented in pure JAVA and runs on the Linux platform. To enhance the CINDI Robot's performance, we implement a novel multi-level inspection infrastructure which includes URL pattern inspection, anchor text inspection and content text inspection. We start the elaboration with implementations of CINDI Robot's five main components followed by an illustration of an overall crawling process at the end of this chapter and the multi-level inspection infrastructure is emphasized.

4.1 Seed Finder

The Seed Finder aims at finding as many high quality seed URLs as possible. Seed URLs are the starting point of each crawling process, so their quality and quantity partially decide the following crawling performance. In [4], the author submitted queries using keyword "computer science department" and "computer science publications" to both Google Web API and AltaVista. Since Google places a constraint of a maximum of 10 results received for each query up to 1000 results [25] and there is also a similar limit on AltaVista, the actual quantity of seed URLs returned is not satisfactory. In fact, even the quality of seed URLs is unsatisfactory. Hence, in the current CINDI Robot, we propose using Open Directory Project (ODP) to increase seed URL quantity and quality. In addition, we also revise the method proposed in [4] with an expectation of getting more and better seed URLs from Google and AltaVista.

4.1.1 ODP based approach

Open Directory Project provides a set of publicly accessible data, including category hierarchy information, links within each category, category move history and so on. Among their free data, we are interested in two UTF-8 files, named *structure.rdf.u8.gz* and *content.rdf.u8.gz* [11]. The former file gives the category hierarchy information which will be used to identify categories that are relevant to computer science and software engineering. Figure 10 gives a short example of *structure.rdf.u8.gz* file which defines the first level categories.

```
<Topic r:id="Top">
    <catid>1</catid>
    <d:Title>Top</d:Title>
    <lastUpdate>2004-04-13 23:40:59</lastUpdate>
    <narrow r:resource="Top/Arts" />
    <narrow r:resource="Top/Shopping" />
    <narrow r:resource="Top/Science" />
    <narrow r:resource="Top/Games" />
    <narrow r:resource="Top/Business" />
    <narrow r:resource="Top/Computers" />
    <narrow r:resource="Top/Health" />
    <narrow r:resource="Top/Sports" />
    <narrow r:resource="Top/World" />
    <narrow r:resource="Top/Test" />
    <narrow r:resource="Top/Society" />
    <narrow r:resource="Top/News" />
    <narrow r:resource="Top/Home" />
    <narrow r:resource="Top/Bookmarks" />
    <narrow r:resource="Top/Regional" />
    <narrow r:resource="Top/Recreation" />
    <narrow r:resource="Top/Kids_and_Teens" />
    <narrow r:resource="Top/Adult" />
    <narrow r:resource="Top/Reference" />
    <narrow r:resource="Top/UTF8" />
    <narrow r:resource="Top/Netscape" />
</Topic>
```

Figure 10 *structure.rdf.u8.gz* file example

By analyzing the *structure.rdf.u8.gz* file, we choose categories in Table 2 from which Seed Finder retrieves seed URLs.

Table 2 ODP categories used in Seed Finder

Top/Computers/Computer_Science	Top/Computers/Hardware
Top/Computers/Internet	Top/Computers/Security
Top/Computers/Software	Top/Computers/Systems
Top/Computers/Algorithms	Top/Computers/Artificial_Intelligence
Top/Computers/Artificial_Life	Top/Computers/Bulletin_Board_Systems
Top/Computers/CAD_and_CAM	Top/Computers/Data_Communications
Top/Computers/Data_Formats	Top/Computers/Desktop_Publishing
Top/Computers/E-Books	Top/Computers/Emulators
Top/Computers/Graphics	Top/Computers/Hacking
Top/Computers/Home_Automation	Top/Computers/Human-Computer_Interaction
Top/Computers/Intranet	Top/Computers/MIS
Top/Computers/Mobile_Computing	Top/Computers/Multimedia
Top/Computers/Operating_Systems	Top/Computers/Parallel_Computing
Top/Computers/Performance_and_Capacity	Top/Computers/Programming
Top/Computers/Robotics	Top/Computers/Speech_Technology
Top/Computers/Supercomputing	Top/Computers/Usenet
Top/Computers/Virtual_Reality	

After that, a parser is developed to extract desirable seed URLs from *content.rdf.u8.gz* file. Figure 11 gives a short example of *content.rdf.u8.gz* file. It gives all links under category "Top/Arts/Movies/Titles/1/187".

```
<Topic r:id="Top/Arts/Movies/Titles/1/187">
    <catid>97826</catid>
    <link r:resource="http://www.wbmovies.com/187/" />
    <link r:resource="http://www.movieweb.com/movie/index.html" />
    <link r:resource="http://www.filmscouts.com/scripts/film.cfm" />
    <link r:resource="http://www.mrqe.com/lookup?187" />
    <link r:resource="http://us.imdb.com/title/tt0118531/" />
</Topic>
```

Figure 11 *content.rdf.u8.gz* file example

By parsing the *content.rdf.u8.gz* file, we get 30,655 high quality seed URLs. Since ODP is edited by human editors, we deem it can provide seed URLs with higher quality. However, due to its human constructed nature, it can not scale to the whole Web. Thus the entries retrieved from general purpose search engines are good complements to enrich seed URLs.

4.1.2 General purpose search engine based approach

General purpose search engines contain large volume of link information. Upon well-selected queries, they can also provide numerous good seed URLs. Inspired by the CiteSeer Web crawler [24], we implement a new method to construct query phrases submitted to AltaVista and Google. These queries have to meet two requirements: 1) they can obtain a good coverage of computer science and software engineering related web sites; 2) the obtained seed URLs should be strongly related to our topic.

To cover all computer science subfields, we utilize the computer science directory provided by CiteSeer [57] to construct the query repository. Table 3 presents the top level of the CiteSeer computer science directory. We manually collect 200 research papers from all subdirectories and extract their main titles and keywords. Removal of stop words is manually performed and meaningful phrases are retained [58]. The words and phrases retained are grouped into query entries and are submitted to AltaVista and Google to retrieve seed URLs. These query entries are given in Appendix A. Since these queries are specific terms used in computer science and software engineering field, we consider all returning results are on-topic. Among returned seed URLs, those document URLs, like www.ed.ac.uk/papers/Sta003.pdf, are excluded because usually it is not possible to find any hyperlink in this kind of documents. To further ensure a better relevance of seed URLs, we only keep the first 200 query results from both AltaVista and Google results for each query and combine the results. In this way, we obtained 27,056 high quality seed URLs in contrast to 3,130 seed URLs found in [4]. One potential advantage of general purpose search engine based approach is that we can increase the seed URLs by creating more well-selected queries whenever we feel it is necessary.

Table 3 CiteSeer computer science directory

Topic Name	Subtopics Number
Agents	6
Applications	3
Architecture	3
Artificial Intelligence	6
Compression	3
Databases	8
Hardware	8
Human Computer Interaction	8
Information Retrieval	8
Machine Learning	8
Networking	9
Operating Systems	9
Programming	10
Security	5
Software Engineering	3
Theory	5
World Wide Web	4

All seed URLs found by these two approaches are inserted to NEW_SITES table and all duplicate URLs are removed.

4.2 Web Crawler

The previous Web Crawler was basically designed as a Breadth-First Search (BFS) crawler [4]. However, more and more research [28, 59, 60] indicates that BFS is not suitable for focused web crawling. Another problem of the previous CINDI Robot is lack of intelligent guidance of crawling processes. As a result, the current Web Crawler crawls the Web in a sequence guided by machine learning and heuristics.

4.2.1 URL pattern inspection

Unlike text categorization in Information Retrieval systems in which only content texts are used for classifying a document, for web crawlers the URL pattern can provide additional clues for web page classifications. This observation brings better efficiency for a focused web crawler. URL pattern inspection of CINDI Robot is implemented to include 5 main aspects: computer science department web site speculation, stop-directory and to-be-avoided directory filtering, protocol and file format filtering, seven directory level exclusion and the Robot Exclusion Protocol.

A URL may explicitly tell us if it is from a useful web site. This is most common for web pages of a computer science department at a well recognized university. For example, a URL such as "http://www.cs.concordia.ca" indicates it is worth crawling. By collecting over 500 computer science departments' URLs, we deem URL keywords in Table 3 are good indications of computer science department web pages. Usually a web page with appearance of "edu" or "ac" in its URL is even more likely to be a computer science department web page. However, we also notice that some institutions' domain names are not constructed using "edu" or "ac". Due to the existence of Link Analyzer and Statistics Analyzer, we can perform a less strict classification policy. Any URL containing a section matching URL keywords in Table 4 is classified as relevant and if it appears as a foreign link, it will be sent to the high priority queue of the URL frontier.

Table 4 URL keywords of computer science department web pages

ID	URL keywords	ID	URL keywords	ID	URL keywords
1	cs	20	umcs	39	csis
2	cpcs	21	ccs	40	soi
3	scs	22	secs	41	macs
4	site	23	mathcs	42	comp
5	cis	24	mtcs	43	cctm
6	csm	25	ecsu	44	soc
7	csce	26	macsc	45	computing
8	csc	27	compsci	46	inf
9	ecs	28	csdept	47	scm
10	csci	29	itec	48	eeecs
11	ecst	30	computerscience	49	csd
12	cse	31	insttech	50	cmis
13	soe	32	encs	51	cswww
14	scis	33	cap	52	cms
15	eecs	34	seas	53	scom
16	csee	35	lcsee	54	dcs
17	informatics	36	scit	55	it
18	cscs	37	cems	56	ece
19	cti	38	csse	57	mcs

The text in a URL string can contain even more important information than just deciding if it is a computer science department web page. Since we are looking for research documents in computer science and software engineering field, a URL string containing “medicine” is less likely to be relevant than a URL containing the word “computing”. The stop-directory list is inspired by this idea. When Web Crawler retrieves a URL, it first extracts all directory names of this URL. If one of these names appears in the stop-directory list, this URL won’t be crawled. For example, given a URL "http://www.cs.concordia.ca/programs/ugrad/coop/calendar/index.html", Web Crawler extracts the following directory names: "programs", "ugrad", "coop" and "calendar". Since "calendar" falls into the stop-directory list, this URL won't be crawled. The to-be-avoided directory list is constructed for each web site and the directories in this list are more specific. In the former example, its to-be-avoided directory is "/programs/ugrad/coop/calendar/" in contrast with "calendar".

One big difference between stop-directory list and to-be-avoided directory is that stop-directory list can be used for crawling new websites while to-be-avoided directory list is only used for re-crawling. When CINDI Robot re-crawls a web site, its to-be-avoided directory list is loaded at the beginning of crawling. During the crawling process, Web Crawler compares each URL with the directories in the to-be-avoided directory list and all URLs falling into the to-be-avoided directory list are removed. Both stop-directory list and to-be-avoided directory list are built by Statistics Analyzer, which is elaborated in Section 4.4. The speed-up obtained by using the to-be-avoided directory list is studied in Chapter 5.

In CINDI Robot, all other hyperlinks extracted from web pages need to be further selected because CINDI Robot is designed only accessing HTTP protocol and only traversing certain types of web pages. Web pages ending with "html", "htm", "asp", "php", "cfm", "jsp" and "shtml" are considered as acceptable web pages and will be traversed by Web Crawler; web pages ending with "pdf", "ps", "doc", "txt", "latex", "tex", "xml" and "rtf" are considered as downloadable web pages. For acceptable web pages, a special URL format, like "http://www.cs.concordia.ca/index/" in which no file format is specified, is also considered as acceptable. When URL pattern inspection filters outer links, acceptable web pages are sent to either *toSearch* vector which keeps all unvisited web pages of the current web site or the URL frontier which keeps all foreign links while downloadable web pages are passed to File Fetcher. In addition, "html" web pages can also be downloaded if the HTML parser identifies them as useful documents (see Section 3.2.4).

In [4], the author found that there is no possibility to get a document in a URL with more than 7 levels. This finding is utilized in the current version of CINDI Robot. If Web Crawler finds a URL has more than 7 levels, it won't be crawled. The Robot Exclusion Protocol mentioned in Section 2.2 is also incorporated into URL pattern inspection because the Robot Exclusion Protocol also operates on the URL level. Thus the full URL pattern inspection algorithm is exemplified as follows:

```
Begin
    get next URL u

    if u has more than 7 levels
        discard u
    else
         if u has a directory in stop directory list
            discard u
        else
            if for recrawling & u contains a directory in to-be-avoided directory list
                discard u
            else
                if u is not allowed by robots.txt
                    discard u
                else
                    download page u
                    extract outer link v in u

                    if v can not pass protocol and file format filter
                        discard v

                    if v is speculated as a computer science department web page
                        send v to the high priority queue of URL frontier
End
```

Figure 12 Pseudocode of URL pattern inspection

4.2.2 Anchor text inspection

Anchor texts are of special importance to a focused web crawler. In addition to URL pattern inspection, it is the only way to provide clues to classify a web page before actually crawling it.

Both layers of web pages in the revised context graph are worth anchor text inspection. Since Layer 1 web pages are strongly related to the pre-defined topic, it is reasonable to hypothesize that most of the outer links of Layer 1 web pages are relevant. In this case the main responsibility of the anchor text inspection is going to prune very few irrelevant outer links if any. To discover the most common irrelevant anchor texts in relevant web pages, we manually collect anchor texts leading to totally irrelevant regions (neither Layer 1 nor Layer 2 web pages) from 1000 Layer 1 web pages to construct a *pruning anchor text list*. The collected anchor texts are also preprocessed (including stop word removal and stemming, elaborated in Section 4.3.1) to be satisfactorily representative. When the anchor text analyzer gets a Layer 1 web page, it first extracts and preprocesses all anchor texts from this page. Then it compares each anchor text with the *pruning anchor text list*. If an anchor text falls into the *pruning anchor text list*, its corresponding hyperlink is eliminated. For example, for web page http://en.wikipedia.org/wiki/Focused_web_crawler, “About Wikipedia”, “Help”, “Donate to Wikipedia”, “Upload file”, “Contact Wikipedia”, “Printable version”, “Privacy policy”, “Disclaimers”, “GNU Free Documentation License”, “Copyright” and “Login” are excluded. This can significantly save crawling time.

For Layer 2 web pages, we assume that there are only a limited number of hyper links that are related to our topic. Thus we constructed an *extracting anchor text list* to identify those few relevant hyperlinks. We manually collect anchor texts leading to Layer 1 or

Layer 2 web pages from 5000 Layer 2 web pages, which we think is large enough. Preprocessing is also needed. When the anchor text analyzer gets a Layer 2 web page, it extracts all anchor texts and preprocesses these anchor texts. After that, it also performs a simple anchor text match to extract relevant outer links. Figure 13 gives a simplified algorithm of anchor text inspection.

```
Begin
    if web page w is not from either Layer 1 or Layer 2
        discard w
    else
        get anchor text and hyperlink pair <u, v> from w
        if w is a layer 1 web page
            if anchor text u is in pruning anchor text list
                discard v
            else
                retain hyperlink v
        else if w is a layer 2 web page
            if anchor text u is in extracting anchor text list
                retain v
            else
                discard v

        if v is a domain web link
            send v to toSearch vector
        else if v is a foreign link
            send v to URL frontier
End
```

Figure 13 Pseudocode of anchor text inspection

4.2.3 Error handling

Since the World Wide Web is a dynamic, heterogeneous information repository and the HTML language is semi-structured, it is common for a web crawler to run into some unexpected problems, for example, 400 Bad Request, 403 Forbidden, 404 Not Found and so on. It is vital for a web crawler to get out of an unexpected problem as soon as possible. According to our experience, error handling can be categorized into active error handling and passive error handling.

Active error handling means to fix a URL problem before actually retrieving its content while passive error handling means to get out of a connection error elegantly during the retrieving process. In [4], the author proposed a series of active error handling methods, including absolute URL conversion, avert spider trap detection, back slash mistake and so on. These methods can rectify a large number of mistaken URLs. However in [4], passive error handling is ignored. According to our experiments, passive error handling is essential to CINDI Robot's performance. For previous version of CINDI Robot, in one crawling cycle (involves only 20 web sites) it can get stuck up to 20 times and each time can take up to 1 hour for various unknown connection problems. Since the previous CINDI Robot was a single-thread program, this drawback can be fatal. In the current CINDI Robot, multi-threading can relieve this problem but it is still necessary to implement passive error handling.

In current CINDI Robot, we impose a time limit on each web page retrieval task. It is not allowed to spend too much time on retrieving an abnormal web page. We implement the web page retrieval task in a separate thread and use *Timer* and *TimerTask* classes to enforce the time constraint. If a web page retrieval task takes more than 10 seconds, it is abandoned and its corresponding thread is stopped and released.

4.2.4 Additional heuristics

One feature of a successful focused web crawler is that it can make use of every clue to supplement its crawling strategy. As mentioned in Section 3.2.4, "meta keywords" and web pages' titles can provide additional information to unveil the nature of web pages. The current CINDI Robot inherits, revises and augments the domain keyword list provided in [4]. And this new domain keyword list is used to perform simple text matching over an incoming web page's "meta keywords" and title. If there is a keyword in the domain keyword list, this web page is directly classified as relevant and thus no further content text classification is needed.

4.3 Link Analyzer

Link Analyzer is the intelligence core of the current CINDI Robot. In [4], the main responsibility of Link Analyzer is to construct the stop-directory list and the to-be-avoided directory list. In the current design, this responsibility is taken over by Statistics Analyzer. The current Link Analyzer is responsible to identify the relevance of web pages. The content text classification is the most important intelligent mechanism of CINDI Robot. In the current design, CINDI Robot possesses two kinds of content text classifiers in order to perform the content text inspection, namely SVM classifier and Naïve Bayes classifier. They are used in different crawling phases and different scenarios based on their characteristics. A revised context graph is proposed to classify all web pages into three categories, which augments the opportunities of finding more relevant web sites. A novel tunneling technique is designed to increase the recall. URL ordering policies are set up to realize the essence of a focused web crawler, "Most relevant web pages get crawled first [23]".

4.3.1 Preprocessing

Before extracting the web page features used for SVM classifiers and Naïve Bayes classifier, four steps of preprocessing are done. These four steps are removal of HTML tags, alphabetic character extraction, removal of stop words and stemming [61]. These techniques are typical steps used in Information Retrieval Systems and Indexing Systems. However, our experiments demonstrate they can be applied to web crawlers and are essential for the performance of focused web crawlers.

Removal of HTML tags

The web content directly retrieved from URLs can be quite annoying since it usually contains lots of HTML tags. For content text classifiers, HTML markup tags are meaningless because they are not representative enough to be considered as features. So when CINDI Robot gets the source code of a web page, it first removes all HTML tags and passes a clean string for alphabetic character extraction in order to reduce the cost of storage and computing. In particular, to speed up further processing, anchor texts and outer links of a web page are extracted and stored in a temporary repository in this step. Given a section of a web page source code as follow:

```
<b>Abstract:</b> We consider the problem of using a large unlabeled sample to boost performance of a learning algorithm when only a small set of labeled examples is available... <a href="http://citeseer.ist.psu.edu/update/73282">(Update)</a><br><br><font size=-1 color= "#907050"><b>Cited by: &nbsp; <a href="http://citeseer.ist.psu.edu/context/31351/73282"> More</a></b></font><br><a class=tl href="http://citeseer.ist.psu.edu/5benchmark.html">A Benchmark Dataset for Audio Classification and.. - Homburg, Mierswa.. (2005)</a>
```

Link Analyzer strips all HTML markup tags and generates the following string. This string is also used to illustrate latter steps.

Abstract: We consider the problem of using a large unlabeled sample to boost performance of a *learning algorithm when only a small set of labeled examples is available... (Update) Cited by: More A Benchmark Dataset for Audio Classification and.. - Homburg, Mierswa.. (2005)*

Especially "*&nbsp*" is not surrounded by "<" and ">", so it is temporarily kept here and is removed in following steps.

Alphabetic character extraction

The extraction result from HTML tag removal may still contain punctuations, numbers and other non-alphabetic characters. In addition to not being able to contribute to the classification, they lower the processing speed. Thus we have to remove them from the string obtained after HTML tags are removed and convert all characters into lower case. The static method *isLetter* in *Character Class* is used to exclude non-alphabetic characters. After the processing, the output string from step 1 becomes:

abstract we consider the problem of using a large unlabeled sample to boost performance of a learning algorithm when only a small set of labeled examples is available update cited by nbsp more a benchmark dataset for audio classification and homburg mierswa

Removal of stop words

A stop word is a commonly used word (such as "the") that are filtered out prior to, or after processing of natural language data [62]. Articles, prepositions and conjunctions are common candidates of stop words. The stop word list is also language and task dependent [58].

In CINDI Robot, stop words are filtered out before the classification of web pages for three main reasons. The first one is to reduce the vocabulary size and accelerate the computing speed of subsequent steps. The second reason is to lower the classification error rate caused by stop words, which is demonstrated in [58]. The third reason is that stop word removal can bring significant improvements on recall and this is also proved in [58].

In addition to common stop words with certain semantic meaning, other special strings of the HTML language are also included in our stop word list, for example, "nbsp", "reg". By incorporating several popular stop word lists and considering our topic, we construct an augmented stop word list in order to exclude as many stop words as possible. Appendix B gives the stop word list used in CINDI Robot which is two times larger than the one used in [51].

The string used for illustration changes to the following after stop word removal.

abstract consider problem large unlabeled sample boost performance learning algorithm small set labeled examples available update cited benchmark dataset audio classification homburg mierswa

Stemming

Stemming is the process for conflating inflected or derived words to their stem, base or root form. "The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root [61]". For example, "compute", "computer", "computers", "computing" and "computes" are all mapped to the same stem "comput". Stemming is an efficient way to avoid feature explosion in text categorization. According to [58], stemming does not make significant contribution to recall value, but it plays an important role in precision. The classic Porter Stemming Algorithm [63] is used in CINDI Robot. The Porter Stemming Algorithm was proposed in 1979 and was targeted for English words. It is the most widely accepted algorithm; however it leads to loss of precision and introduces some anomalies. Considering the requirement of CINDI Robot, the Porter Stemming Algorithm is a good trade-off between efficiency and accuracy since it is simple and fast. The string used for illustration looks as follows after stemming.

abstract consid problem larg unlabel sampl boost perform learn algorithm small set label example avail updat cite benchmark dataset audio classif homburg mierswa

Comparing the number of words after Step 4 and the word number after Step 1, we can observe there is a significant decrease in the distinct number of words. Table 5 further demonstrates the importance of preprocessing by showing the distinct number of words before preprocessing and after preprocessing.

Table 5 Comparison of distinct number of words

	Distinct Number of Words
Without stop word removal and stemming	26,876
With stop word removal and stemming	17,533

4.3.2 Vocabulary construction

Vocabulary construction is essential to facilitate both the Naïve Bayes classifier and the SVM classifier. Distinct words obtained after Step 4 are used as elements in the vocabulary. In addition to distinct words themselves, the vocabulary also records other parameters to measure the degree of representation of keywords in order to support further classifications. There are two categories defined in CINDI Robot: computer science and software engineering (CSE) category and non computer science and software engineering (Non-CSE) category. The CINDI Robot vocabulary records information for both categories. Usually, two well-known functions are utilized to measure the degree of representation of a term for a specific category, namely Keyword Frequency (KF) and Document Frequency (DF) [64].

Keyword Frequency function is defined as the average occurrence frequency of a keyword per document over all documents in each category and it is formulated as follows:

$$lk_i^k = \frac{\sum_{i=1}^{n_k} kf_i^j}{n_k}$$

Where lk_i^k is the KF score of keyword i in category k; kf_i^j is the occurrence frequency of keyword i in document j; n_k is the total number of documents in category k. The idea behind KF function is that if a keyword appears frequently in a category's documents, then it can be used to represent this category.

Document Frequency function is defined as the frequency of documents containing a specific keyword in a given category. It can be written as below.

$$ld_i^k = \frac{\sum_{kf_i^j \neq 0}^{j=1,\dots,n_k} 1}{n_k}$$

where ld_i^k is the DF score of a keyword i in category k. Document Frequency function provides another view of a keyword's degree of representation.

Though Keyword Frequency function and Document Frequency function are widely used, they are not suitable in our design. They will bring substantial extra computing cost to further text classifications. Thus we import two new concepts as Revised Keyword Frequency (RKF) function and Revised Document Frequency (RDF) function. RKF is defined as the total occurrence frequency of a keyword in all documents in a given category and it is formulated as follows:

$$lk_i^k = \sum_{j=1}^{n_k} kf_i^j$$

where kf_i^j is the occurrence frequency of keyword i in document j; n_k is the total number of documents in category k. RDF is to find the document occurrence of a keyword in a given category. It can be written as follows:

$$ld_i^k = \sum_{kf_i^j \neq 0}^{j=1,\ldots,n_k} 1$$

Using these two concepts, we can construct our vocabulary by recording distinct keywords along with their RKFs and RDFs for both categories. Table 6 shows a subset of the vocabulary.

Table 6 A subset of the vocabulary

Keywords	CSE		Non-CSE	
	RKF	RDF	RKF	RDF
comput	638	90	26	18
gnu	6	4	0	0
decod	7	3	0	0
queue	26	4	0	0
hash	32	3	0	0
acm	71	25	0	0
queri	22	8	6	4
oracl	20	6	0	0
latex	3	2	0	0
perl	4	2	0	0
linux	41	11	0	0
grid	24	12	0	0
protocol	4	4	8	5
model	85	31	31	17
advertis	0	0	42	20
agricultur	0	0	26	14
film	0	0	105	17
concurr	11	6	4	3
program	32	27	14	13
multicast	5	3	0	0
databas	257	52	77	32
morphism	7	5	1	1
cryptographi	3	2	0	0
system	43	20	96	38

* based on 100 positive samples and 200 negative samples.

4.3.3 Content Text Inspection

In addition to URL pattern inspection and anchor text inspection, content text inspection provides the most straightforward information on web pages' relevancy. It is also a reasonable way to speculate the relevancy of their outer links. In the CINDI Robot, the content text inspection is realized by both the Support Vector Machine classifier and the Naïve Bayes classifiers.

4.3.3.1 Support Vector Machine classifier

As introduced in Section 2.3.2.3, the Support Vector Machine classifier is suitable for our task especially when only limited training samples are available at the initial phase. The RBF kernel function is used in the CINDI Robot. The RBF kernel nonlinearly maps samples into a higher dimensional space, so it can handle the case when the relation between class labels and attributes is nonlinear [65]. According to [38], it shows that using RBF one can improve text categorization performance up to 1-2% on Reuters-21578 dataset.

In CINDI Robot SVM classifier, distinct words appearing in training documents constitute the feature space. As shown in Table 5, even after preprocessing steps there are still 17,533 keywords left. Without further keyword selections, the feature space can be quite large. So we have to perform feature space reduction. According to our experiments, we only select words with total RDF (the sum of the RDF in CSE category and in Non-CSE category) bigger than 3 under which we can get a best tradeoff between processing speed and accurate classification rate to form the feature space. After feature space reduction, we get only 9,773 distinct words. Using this reduced feature space, we can map incoming web pages to feature vectors. The next step is to choose an appropriate feature value representation. A well-selected feature value representation can bring substantial improvements of classification accuracy. In order to gain the best performance, we design four kinds of value representations for CINDI Robot SVM classifier and in Chapter 5 we will compare the performance of these four representations.

a) *Using keyword occurrence times in a document as feature values;*

b) *Using keyword's document occurrence (if the keyword appears in the document, it is 1, otherwise it is 0) as a feature value;*

c) *Using Term Frequency and Inverse Document Frequency ($TF \times IDF$) score* [66] *as feature values.* The $TF \times IDF$ value is widely used for term weights. It says that the importance of a word increases proportionally to the number of times a word appears in the document but is in inverse proportion to the frequency of the word in the corpus. The $TF \times IDF$ value can be calculated as below.

$$w_{ij} = t_{ij} * \log_2 \frac{N}{n}$$

Where w_{ij} is the score of Term T_j in Document D_i; t_{ij} is the frequency of Term T_j in Document D_i; N is the number of documents in collection; n is the number of Documents where Term T_j occurs at least once.

d) *Using a combination of RDF and RKF as feature values*, which is formulated as:

$$RDF + \gamma * RKF$$

Where γ is a scaling factor to balance the degree of representations of RKF and RDF, which is calculated by the following formula:

$$\gamma = \frac{\sum_{k=1}^{m} \sum_{i=1}^{n_k} ld_i^k}{\sum_{k=1}^{m} \sum_{i=1}^{n_k} lk_i^k}$$

Where m is the number of categories and in CINDI Robot m equals to 2; n_k is the total number of documents in category k. γ is used as a constant once getting calculated after the training process.

According to our experiments, the fourth representation is finally used in CINDI Robot because it gains nearly the same classification accuracy as the $TF \times IDF$ value but with less computing cost. Practically, we use SVM^{light} [67] for CINDI Robot. Though SVM^{light} has reasonable speed for real-time classification, the training time would still be relatively long and thus it is not suitable for real-time application. So we only retrain our SVM classifier after we crawl every 50,000 web pages.

4.3.3.2 Naïve Bayes classifier

In the CINDI Robot, the Naïve Bayes classifier only works after the CINDI Robot collects enough training data because the Naïve Bayes classifier generally works better under a large training set and it can avoid the very large quadratic programming optimization problem in the SVM classifier.

In CINDI Robot, two classes are defined and distinct keywords are used as features for a web page. In addition, smoothing is usually needed, so the conditional probability of any word w_i for a given category can be written as:

$$P(w_i \mid category) = \frac{n_{w_i}^k + 0.5}{n^k + 0.5 \times |V|}$$

Where $n_{w_i}^k$ is the number of times word w_i appears in the category k; n^k is the total number of words in the category k; V is the number of entries in the vocabulary. To avoid arithmetic underflow, we use the log of the probability instead of the probability itself. Applying Equation (2) in Section 2.3.1.5, we can calculate the probability of a web page falling into a given category as below.

$$\log P(category \mid newWebPage) = \log P(category) + \log P(w_1 \mid category) + \ldots + \log P(w_n \mid category)$$

However for CINDI Robot, we find a weighted probability formula is more powerful to make an accurate classification. We assign a weight to every distinct word in a web page. Here we use $TF \times IDF$ scores as the weighting scheme. Thus we can get:

$$\log P(relevant \mid newWebPage) = \log P(relevant) + W_{w_1} \log P(w_1 \mid relevant) + \ldots$$
$$+ W_{w_n} \log P(w_n \mid relevant)$$

and

$$\log P(irrelevant \mid newWebPage) = \log P(irrelevant) + W_{W_1} \log P(w_1 \mid irrelevant) + \ldots$$
$$+ W_{W_N} \log P(w_n \mid irrelevant)$$

Where W_{w_i} is the $TF \times IDF$ score of keyword w_i. If $\log P(relevant \mid newWebPage)$ is greater than $\log P(irrelevant \mid newWebPage)$, then the new page is relevant, otherwise it is irrelevant. The idea behind the weighting scheme is to emphasize the most "useful" words for the classification. There is another Naïve Bayes classifier used in CINDI Robot to classify Layer 2 web pages and it is elaborated in the next section.

4.3.4 Revised context graph

As analyzed in Section 2.3.1.2, there is a necessity of constructing a revised context graph which can remove the strict link distance requirement. To construct the revised context graph for our CINDI Robot, there are two essential steps. The first step is to learn the context within which relevant documents are typically found and encode this information in our revised context graph. Here the context in the revised context graph is different from the one used in the classic context graph. First we manually collect a set of relevant document URLs to form the core of our revised context graph. Here we set the document URL number to be 100. Subsequently, a separate revised context graph is built for each relevant document URL. To get web page contents used for constructing the revised context graph, we have to first find their corresponding links. Using the back link tracking function provided by Google, we can get all links which can lead to a document URLs within up to 5 link steps. To use the back link tracking function, we can simply submit a query beginning with "link:" and appending a URL we want to track. For example, to track the document URL,

http://www2003.org/cdrom/papers/poster/p181/p181-tsoi/p181-tsoi.html,

we simply provide following query to Google:

"*link: http://www2003.org/cdrom/papers/poster/p181/p181-tsoi/p181-tsoi.html*".

We notice that a too specific URL may not be able to get enough back links even using the back link tracking function of Google. Inspired by the path-ascending algorithm [68], a similar path-ascending strategy may be needed. In this case, we will try to ascend to every path in each URL to track its back links. For example, for the previous URL, we will try to track back

http://www2003.org/cdrom/papers/poster/p181/p181-tsoi/,

http://www2003.org/cdrom/papers/poster/p181/,

http://www2003.org/cdrom/papers/poster/

and so on. In this way, we can guarantee getting enough web pages. Thereafter, web pages retrieved by all resulting links are classified by the content text classifiers. Web pages that can pass the classifications are labeled as Layer 1 web pages. All other web pages are kept as Layer 2 web pages. Once we construct independent revised context graphs for all document URLs, the corresponding layers from all these revised context graphs are merged to provide the training set for Layer 2 web page classifier.

The second step is using the merged Layer 2 web pages mentioned above to train a Naïve Bayes classifier. This Naïve Bayes classifier is used in the actual crawling process to indicate if a web page can lead to relevant web regions. So in CINDI Robot, an incoming web page is first classified by Layer 1 content text classifiers to see if it is relevant. If yes, it is put into Layer 1. Otherwise it is classified by Layer 2 classifier. If it passes the Lay 2 web page classification, it is put in Layer 2. Otherwise it is discarded. According to our statistics, Layer 1 web pages usually have link steps from 1 to 3 while the link distances of Layer 2 web pages usually span from 2 to 5. Figure 14 illustrates classification procedures of CINDI Robot.

```
Begin
    get next web page w

   if CINDI Robot has established enough training data
       classify w using Layer 1 Naïve Bayes classifier
   else
       classify w using Layer 1 SVM classifier

   if classification result is positive
       w is a Layer 1 web page
   else
       classify w using Layer 2 Naïve Bayes classifier
       if classification result is positive
               w is a Layer 2 web page
       else
               w is discarded

End
```

Figure 14 Classification procedures of CINDI Robot

4.3.5 Tunneling

To relieve the difficulties of the taxonomy tree construction in the classic tunneling technique, CINDI Robot employs a simple tunneling strategy while it still achieves desirable performance. In each CINDI Robot crawling cycle, low priority web pages in URL frontier won't be crawled until CINDI Robot exhausts all high priority web pages. This policy guarantees that most relevant web pages are crawled first. When CINDI

Robot has traversed all web pages in a relevant region, it is a good time to launch the tunneling strategy in order to dig out new relevant regions. In CINDI tunneling strategy, Layer 2 Naïve Bayes classifier is used for tunneling. In tunneling, its classification policy changes to that as long as $\log P(inLayer2 \mid newPage)$ is greater than $\gamma \times \log P(outLayer2 \mid newPage)$ it is classified as relevant; otherwise it is irrelevant, where $0 \le \gamma \le 1$. We call γ a bias factor. A smaller bias factor enlarges the opportunity to discover new relevant regions at the cost of precision; a larger bias factor corresponds to a more strict selection policy but a lower recall. When γ equals 0, CINDI Robot degenerates to a general purpose web crawler. In CINDI Robot, we set γ to 0.75. CINDI Robot switches the bias factor back to 1 when CINDI Robot discovers 30 new Layer 1 web sites. If this can not be achieved, the crawling cycle ends in a normal way. And the bias factor switches back to 1 for the next crawling cycle. In this way, we can discover all relevant web regions from a seed URL within acceptable link distances.

4.3.6 URL ordering

In an ideal case, one would like to retrieve pages according to the order of their relevance to the specific topic [35]. A URL ordering strategy is usually established based on other techniques used in focused web crawlers. Thus we developed our URL ordering strategy by considering revised context graph, tunneling and real-time relevance feedback. Since there are two priority queues in the URL frontier, URL ordering in CINDI Robot is to assign each incoming web page into an accurate queue or discard it. An efficient URL ordering policy plays a vital role in the success of a focused web crawler. According to techniques implemented in CINDI Robot, we set following rules in order to crawl most relevant web pages as early as possible.

1. *All URLs found by Seed Finder are assigned to the high priority queue.* Since we perform a strict seed URL selection procedure and trust all seed URLs as good start points of crawling processes, we put them in the high priority queue.

2. *All outer links extracted from web pages which are classified as Layer 1 web page are assigned to the high priority queue.* Since all Layer 1 web pages are directly related to computer science and software engineering, it is reasonable to deduce all foreign links derived from Layer 1 web pages are also on-topic. Especially, considering the anchor text inspection implemented in CINDI Robot can largely remove the few existing off-topic outer links from Layer 1 web pages, this guarantees this rule works more efficient.

3. *All outer links extracted from web pages which are classified as Layer 2 web page are assigned to the low priority queue.* Usually a Layer 2 web page is not directly on-topic, but they are promising bridges to on-topic web pages. We realize that even we perform an anchor text inspection, it is still too optimistic to directly put foreign links mined from Layer 2 web pages into the high priority queue. So these outer links are put into the low priority queue. There is one exception. If a foreign link is identified as a computer science department web page by the URL pattern inspection, it is directly put into the high priority queue regardless of where it is found.

4. *All outer links extracted from web pages which are classified out of Layer 1 and Layer 2 are discarded.* This rule efficiently prevents CINDI Robot from crawling irrelevant web regions and thus increases the precision and crawling speed of CINDI Robot.

4.4 Statistics Analyzer

The previous CINDI Robot collected various statistical data from SEED_URL, VISITED_PAGES, FOREIGN_LINKS, DOMAIN_KEYWORD and DOWNLOAD_STATUS tables to produce a serious of statistical results and stored resulting data into tables SITE_STATS, SITE_REF_BY, LINK_REF_BY, RDVT, and LEVEL_STATS [4]. The modifications of CINDI Robot database and the change of working mode require redesign of Statistics Analyzer.

Statistics Analyzer is responsible to update the CRAWLED_SITES table. After crawling a web site, Statistics Analyzer inserts corresponding fields into the database. The *total_pages* field shows how many web pages of this web site are crawled. This value may not reflect the real number of web pages in a web site. Due to multi-level inspections implemented in CINDI Robot, some parts of a web site may be skipped. The *total_downloaded* field records how many document URLs are founded in this web site. The *total_accepted* field refers to the total number of Layer 1 web pages and Layer 2 web pages in this web site. The *page_relevant_rate* field records the proportion of Layer 1 web pages to total crawled web pages. And the *document_download_rate* field records the proportion of document URL number to total crawled web page number, which reflects the document URL density of a web site. The *accepted_document_rate* field can not be filled until the DFS completes the filtering. It records the proportion of accepted document number to document URL number, which reflects the quality of a web site's documents. The *number_ref_by* field of CRAWLED_SITES table indicates the popularity of a web site. At the first time when a web site is inserted, the value of *number_ref_by* field is 0. During crawling processes, when CINDI Robot finds a foreign link, Statistics Analyzer first extracts the host name from this foreign link. Then Statistics Analyzer checks the CRAWLED_SITES table to see if this host name has already been discovered. If yes, the *number_ref_by* field of this web site increases 1. The *is_accepted* field indicates if a web site is worth re-crawling. It can be set by the real-time monitoring module of Statistics Analyzer during a crawling process or by inspecting the feedback from DFS. If a web site meets one of following conditions, its *is_accepted* field is set to 0 and will not be re-crawled, which leads to significant time saving.

- Average Document Download Rate < 1%
- Average Accepted Document Rate < 5%
- Average Page Relevant Rate < 5%

Statistics Analyzer is also responsible to build the STOP_DIR_LIST table and the DIR_TO_BE_AVOIDED table. The stop-directory list is usually constructed in a long period of time. As mentioned before, the stop-directory list records the directories under which either no accepted download document is found or no relevant web page is found.

For stop-directory list under which no accepted download document is found, we perform the construction after each crawling cycle. After we get the filtering results from DFS for each crawling cycle, we begin to identify all distinct directory names appearing in this cycle. We divide all directory names into two sets, directory names under which accepted documents are found (in following this set is referred as A) and directory names under which no accepted document is found (in following this set is referred as B). For a directory from *directory* field of the DOWNLOAD_STATUS table like /cdrom/papers/poster/p181/p181-tsoi/, "cdrom", "papers", "poster", "p181" and "p181-tsoi" are identified as distinct directory names. Two temporary tables, ACCEPTED_DIR and NO_ACCEPTED_DIR, are used to record this information.

For set A, all distinct directory names since the first crawling cycle are recorded in ACCEPTED_DIR table while for set B only distinct directory names from the last crawling cycle are recorded in NO_ACCEPTED_DIR table. Now we can get a potential stop directory set by performing a set-difference operation over these two sets. All elements in this potential stop directory set satisfy the requirements that there is no accepted document under them. The next step is to check if they meet the "5 distinct web sites" requirement. For each element in the potential stop directory set, we simply check how many times this directory name has occurred in different web sites. If the times exceed 5, it is added to the STOP_DIR_LIST table. Otherwise, the directory names and their occurrence numbers are recorded in another temporary table, POTENTIAL_STOP_DIR. An example of the POTENTIAL_STOP_DIR table is given as follow.

```
+----------------+-------------+
| directory_name | occur_times |
+----------------+-------------+
| administration |           1 |
| alumni         |           2 |
| help           |           3 |
| jobs           |           4 |
| location       |           2 |
| login          |           3 |
| notices        |           3 |
| phpdig         |           2 |
| visitors       |           3 |
+----------------+-------------+
```

Figure 15 An example of POTENTIAL_STOP_DIR table

The POTENTIAL_STOP_DIR table is used for future crawling cycles. After the completion of next crawling cycle, Statistics Analyzer still finds potential stop directory names for this crawling cycle. The result will be merged with the records in POTENTIAL_STOP_DIR table. Directory names exceeding 5 occurrence times are moved to the STOP_DIR_LIST table. Meanwhile, all old entries in POTENTIAL_STOP_DIR have to be checked to see if new documents are found under them afterward. A simple SQL query is used to perform the checking:

```
SELECT COUNT(*)
FROM DOWNLOAD_STATUS
WHERE directory LIKE '%element_in_POTENTIAL_STOP_DIR%'
AND is_accepted = 1
```

If new documents are found, the corresponding directory name is removed from POTENTIAL_STOP_DIR table.

Since the construction of the stop directory list caused by relevancy is dynamic during the crawling process, we design a simpler strategy. During crawling processes, Statistics Analyzer keeps a list of directory names under which no relevant web pages are found and corresponding occurrence times. When this number exceeds 5, it is added to STOP_DIR_LIST table. After each crawling cycle, potential stop directory names and their occurrence times are kept in a data file and are further used in following crawling processes. Table 7 gives a subset of the stop-directory list.

Table 7 A subset of stop-directory list

Stop-directory Name
audio(s)
image(s)
section(s)
puzzle
calendar
lecturenotesweb
transparencies
contact(s)
login
admission(s)

Compared to the construction of stop-directory list, the construction of the to-be-avoided directory list is more straightforward. Unlike stop directories, the directory hierarchy is important to to-be-avoided directories. So a to-be-avoided directory is recorded with its hierarchy. For a specific web site, its to-be-avoided directory list can be built in two phases. During the crawling process, directories with a Document Download Rate less than 1% or Page Relevant Rate less than 5% are directly inserted into the DIR_TO_BE_AVOIDED table because this kind of directories is not cost efficient. After the crawling process, Statistics Analyzer also checks the filtering feedback from DFS and adds all directories under which Accepted Document Rate is less than 5% to the DIR_TO_BE_AVOIDED table. Figure 16 gives a subset of the DIR_TO_BE_AVOIDED table, where "siteID = 16" corresponds to www.cs.concordia.ca.

```
+--------+---------------------------+
| siteID | dir_name                  |
+--------+---------------------------+
|     16 | /department/admissions/   |
|     16 | /department/announcement/ |
|     16 | /department/floors/       |
|     16 | /department/policies/     |
|     16 | /department/facilities/   |
|     16 | /search/                  |
|     16 | /department/ugrad/coop/   |
+--------+---------------------------+
```

Figure 16 A subset of to-be-avoided directory of a web site

As we mentioned in Section 3.4, Statistics Analyzer also performs a real-time monitoring to get rid of trap web sites. When CINDI Robot begins crawling a web site, Statistics Analyzer updates its Page Relevant Rate after every 100 web pages. If Statistics Analyzer finds the PRR of a web page is less than 5% after crawling 200 web pages, this web site will be skipped and in CRAWLED_SITES table its *is_accepted* field will be set to 0. The 200 web page threshold is selected according to our experiments that 200 web page can on the average span over at least 5 directories to make the topic of a web site representative.

4.5 File Fetcher

File Fetcher is responsible to download all documents discovered by Web Crawler and to remove as many duplicate files as possible. For each web crawler thread, there is a corresponding File Fetcher thread.

The first step of fetching a document is to check if its URL has already been downloaded by CINDI Robot. It is realized by executing a SQL query

```
SELECT   COUNT(*)
FROM DOWNLOAD_STATUS
WHERE url = to_test_URL
```

over the DOWNLOAD_STATUS table. If the returning result is 1, it means this document URL has been downloaded and hence this document URL is discarded. If a document URL has not been downloaded, File Fetcher further checks its file name to see if it is worth downloading before opening a HTTP connection. From our experiments, several file name keywords are not welcome and they are given in Table 8.

Table 8 Not welcome file name keywords

ID	Keywords
1	homework
2	hw
3	assignment
4	syllabus
5	cv
6	resume
7	week
8	lecture
9	exercise

Alphabetic character extraction is performed over file names and thereafter a simple keyword matching is used to filter out undesirable file URLs.

As for the Web Crawler, the File Fetcher may also encounter several unexpected failures in the connection phase, including invalid URLs, password protections and remote host failures. When errors occur, the current document URL gets simply discarded by a similar passive error handling method as mentioned in Section 4.2.3. Once the connection is successfully open, before downloading the actual file content, File Fetcher can first get some parameters of the corresponding file, for example, the file size. The file size of a file is of special importance. First of all, it can be used to perform the "8K" assumption. If the file type is PDF and the file size is less than 8k, this file won't be downloaded. Secondly, the file size can also partially indicate if two files with the same name are identical. A document URL can be found in different web pages and different web sites. So we check each document URL to see if it is discovered and downloaded before. In addition, the same file can be cached in different URLs. So if two files are with the same file name and same file size, it is possible that they are the same file and MD5 tool is needed to make sure if they are the same file and if yes, the duplicate one is removed from CINDI document repository. Otherwise they are not identical and only renaming is needed.

If two files share the same file name or a file is with an extraordinarily long file name, renaming is needed. After downloading a file, File Fetcher executes a SQL query:

```
SELECT COUNT(*)
FROM DOWNLOAD_STATUS
WHERE org_file_name = current_downloaded_file_name.
```

If the returning result is greater than 1, renaming mechanism is launched. In [4], the author proposed a renaming policy by recursively appending digital numbers to the tail of a file name. However, this simple mechanism turns out to be a bottleneck of File Fetcher. Considering the large volume of documents discovered by CINDI Robot, the phenomenon that several files share the same name is quite common. We did a survey on this phenomenon and the top 10 most common file names are given in Table 9.

Table 9 A survey of identical file names

File Names	Occurrence Times
content.pdf	306
thesis.pdf	278
syllabus.pdf	155
paper.pdf	142
intro.pdf	140
project.pdf	95
final.pdf	92
notes.pdf	72
report.pdf	66
assignment.pdf	46

Based on 300,000 downloaded files, the file name, "content.pdf", is shared by 306 files. If we meet the 307th "content.pdf" file, according to the design in [4], File Fetcher has to first check if "content0.pdf" is already used and if yes, File Fetcher will check if "content1.pdf" is already used and so forth. So in this case, File Fetcher has to try 306 times before it can successfully rename the incoming file. As we see in Table 9, the identical name phenomenon is quite common. So an efficient renaming policy is indispensable. It is reasonable to assign files unique file names that don't need to be further checked. Time stamp is one of many possible solutions and it is easy to implement. When File Fetcher finds renaming is needed, it first removes a file's extension name. Then it attaches current time stamp to the stripped file name. A time stamp looks like "2008-02-22-12-23-22". So if there is a new incoming document named "content.pdf", File Fetcher will rename it according to current time stamp and its new name becomes "content-2008-02-22-12-23-22.pdf" (suppose that current time stamp is "2008-02-22-12-23-22"). Current time stamp can be obtained by following codes using *java.util.Calendar* class and *java.text.SimpleDateFormat* class.

```
Calendar cal    = Calendar.getInstance();
SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss");
String TimeStamp=formatter.format(cal.getTime());
```

Sometimes, renaming is also needed if file names from Web are extraordinarily long. In this case, we only retain the first 10 characters of the original file name and attach current time stamp.

File Fetcher is also responsible to update the *num_ref_by* field and *is_diff_format* field in the DOWNLOAD_STATUS table. This former field reflects the popularity of a document and it is an important feature for a digital library. There are two ways to update the *num_ref_by* field. When File Fetcher finds a document URL has been downloaded, this document URL is discarded but its corresponding *num_ref_by* field increases 1. When File Fetcher finds two files are identical after the MD5 verification, duplicate file is dropped and the corresponding *num_ref_by* field increases 1. Sometimes, web page author may post the same document in different formats to facilitate potential readers. If two documents come from the same URL prefix and with the same striped file name but with different file extensions, the *is_diff_format* field of both two files are set to 1.

4.6 General Crawling Process

Since we employ a lot of new techniques in the current CINDI Robot, we re-design the general crawling process. In the previous CINDI Robot, components work in a sequential mode. For example, File Fetcher is launched after completion of Web Crawler; Statistics Analyzer is started after completion of File Fetcher. In current CINDI Robot, all components work in a parallel mode in order to increase the efficiency. In the current crawling algorithm, some drawbacks observed from the previous version are rectified. In the previous version of CINDI Robot, *robots.txt* is retrieved and analyzed every time crawling a web page. This brings 50% waste of network and computing resources. In the new design, this drawback is fixed. Robot Exclusion Protocol is retrieved at the first time of crawling a web site and kept in memory. As aforementioned, the current CINDI Robot is a multi-threaded program and the overall crawling process of each thread crawling a web site is given as below.

```
Begin
    load a seed URL u from URL frontier into vector toSearch
    // if there is a high priority seed URL, load it first
    remove u from URL frontier

    while (toSearch.size()>0)
        get the first URL f from toSearch vector
        remove f from toSearch vector

        perform URL pattern inspection on u
        //for seed URL u, Robot Exclusion Protocol is not checked
        //because robots.txt will be downloaded later on
        //computer department speculation is not performed here

        if u fails to pass the URL pattern inspection
            continue

        if f == u
```

```
            retrieve the robots.txt
            add robots.txt to URL pattern inspection
            initialize Statistics Analyzer

        download f
        preprocess web page f
        classify web page f

        if f is Layer 1 or Layer 2 web page
            extract all hyperlinks and corresponding anchor texts <h, a>

            if f is a Layer 1 web page
                add documents URLs to toBeDownloaded vector
                inform File Fetcher
                perform anchor text inspection on <h, a>
                get useful hyperlinks set <m>

                if a hyperlink n is within domain
                    add n to toSearch vector
                else
                    send n to URL frontier's high priority queue
            else
                perform anchor text inspection on <h, a>
                get useful hyperlinks set <m>

                if a hyperlink n is a domain link
                    add n to toSearch vector
                else
                    send n to URL frontier's low priority queue

            add f to searched vector

            update Statistics Analyzer
            continue

        else
            add f to searched vector
            update Statistics Analyzer
            continue

End
```

Chapter 5

Experiments and Results

In this chapter, we present various experimental results and analyses of these results. As mentioned before, recall, precision and crawling speed are main concerns of the current CINDI Robot, thus they are emphasized in our experiments. In real applications, true recall is usually hard to be parameterized since it is impossible to identify the true relevant set for any topic over the Web [27]. As a result, target recall is used as a reasonable estimate of the true recall [27]. Target recall is defined as the proportion of relevant web sites that are retrieved and in the target set to the whole target set. It is formulated as below.

$$Target\_recall = \frac{|retrieved\_relevant\_websites \cap websites\_in\_the\_target\_set|}{|websites\_in\_the\_target\_set|}$$

In following, we compare the precision, target recall among different web crawlers and also inspect the performance improvements of the current CINDI Robot due to individual heuristics. The previous CINDI Robot as a Breadth-First Search (BFS) crawler and a context graph crawler [28] are employed as comparisons. Here one modification of the previous CINDI Robot is needed. The previous CINDI Robot was designed to crawl exactly 30 web sites in each crawling cycle. The 30 web sites limitation in each crawling cycle is removed and it is allowed to start from a given number of seed URLs and to exhaust all foreign web sites found during a crawling cycle. The crawling speed is compared between previous CINDI Robot and current one. We also notice that the network speed may vary from time to time during each day. So we arranged the experiments to start at the same time of a day.

5.1 Experiments on Four Representations of SVM

In Section 4.3.3, we proposed four representations for SVM feature values and conclude that using a combination of RDF and RKF as feature values is the most efficient choice. Here we design an experiment to unveil the SVM accurate classification rates under different representations. We manually collect 200 positive web page examples and 400 negative web page examples to train the SVM classifier. Web page examples are converted into feature vectors using different representations and are normalized to unit length. We set the number of negatives to be twice the number of positives on purpose because this ratio appears to be the optimum value [69]. To test the accurate classification rate, we randomly collect 600 web pages and divide them into three sets. Then a SVM classifier is used to classify these three sets using different representations. We manually check all testing web pages and use manual classification results as the benchmark to calculate the accurate classification rates for different sets under different representations. The classification results are given in Table 10.

Table 10 SVM classifier accurate classification rate comparison

Representation	**Accurate Classification Rate**		
	Set 1	**Set 2**	**Set 3**
Keyword occurrence	76%	81.5%	79.5%
Document occurrence	81%	80.5%	79.5%
$TD \times IDF$	89.5%	93%	90%
Combination of RDF and RKF	88.5%	93.5%	91.5%

According to the classification results from Table 10, we can observe that $TD \times IDF$ representation and combination of RDF and RKF representation obtain significantly better accurate classification rates than keyword occurrence representation and document occurrence representation. Between keyword occurrence representation and document occurrence representation, the latter one brings a slightly better result. Both combination of RDF and RKF representation and $TD \times IDF$ representation get about 10% improvement of the accurate classification rates. Between these two representations, combination of RDF and RKF representation performs even slightly better than $TD \times IDF$ representation. This unveils one reason of choosing combination of RDF and RKF for CINDI Robot. The other reason is that using $TD \times IDF$ representation involves logarithm operations very often, which is not cost-efficient.

5.2 Experiments on Weighted Naïve Bayes Classifier

To enhance the performance of the classic Naïve Bayes classifier, we propose a weighted classification scheme. In the weighted classification scheme, most representative words for a topic are highlighted and therefore it may bring a better accurate classification rate. This speculation is validated in the following experiments. The vocabulary used for both classic and weighted Naïve Bayes classifiers is first constructed by 500 manually collected positive web pages and 1000 manually collected negative web pages and then is augmented after CINDI Robot crawls 5,000 relevant web pages and 10,000 irrelevant web pages. To test the accurate classification rate, we also randomly collect 600 web pages and form three testing sets. The classification results are given in Table 11.

Table 11 Naïve Bayes classifier accurate classification rate comparison

Scheme	Accurate Classification Rate		
	Set 1	Set 2	Set 3
Classic Scheme	87%	90%	89%
Weighted Scheme	89.5%	91.5%	91%

From the experiments, we can find that weighted Naïve Bayes classifier gains about 2% improvement of the classification results than classic Naïve Bayes classifiers and its accurate classification rate is quite close to the rate of the Support Vector Machine classifier. However, the classification process of a Support Vector Machine classifier involves a very large quadratic programming optimization problem. Considering the limited computing resources and strict timing requirements of a large-scale, real-time application, like CINDI Robot, Naïve Bayes classifier is a good choice once we have a relatively large training data.

5.3 Experiments on General Crawling Processes

To compare the performance of the current CINDI Robot, previous CINDI Robot and the classic context graph crawler, we randomly select 5 seed URLs as the start point of the experiment and set the crawling termination limit to 400,000 web pages. We choose these parameters for two reasons; the first is that a larger seed URL number may bring too many foreign links and make the crawling process extraordinarily lengthy, especially for the previous CINDI Robot; the second is if we start with too many high quality seed URLs, we have to wait for a long time before web crawlers complete the seed URLs to precisely judge the performances of these web crawlers. However, in real application, 200 seed URLs are used for each crawling cycle. In the experiments, 75 threads are used in the current CINDI Robot. In this experiment, we compare the performance of these three focused web crawlers in terms of precision, target recall, crawling speed and downloaded document acceptance rate.

5.3.1 Comparison of precision

We use Naïve Bayes classifiers for both previous CINDI Robot and context graph crawler to classify incoming web pages in order to calculate the precision of 400,000 web pages, but not to guide the crawling process. This strategy does not change the precision and target recall of the previous CINDI Robot and the context graph crawler. But it may influence the crawling speed of the previous CINDI Robot. Its effect is analyzed in Section 5.3.3. Figure 17 gives the real-time precision during the crawling processes for different web crawlers, where "CINDI Robot" denotes the current CINDI Robot, "BFS" denotes the Breadth-First Search crawler (the previous CINDI Robot) and "Context Graph" represents the classic context graph crawler.

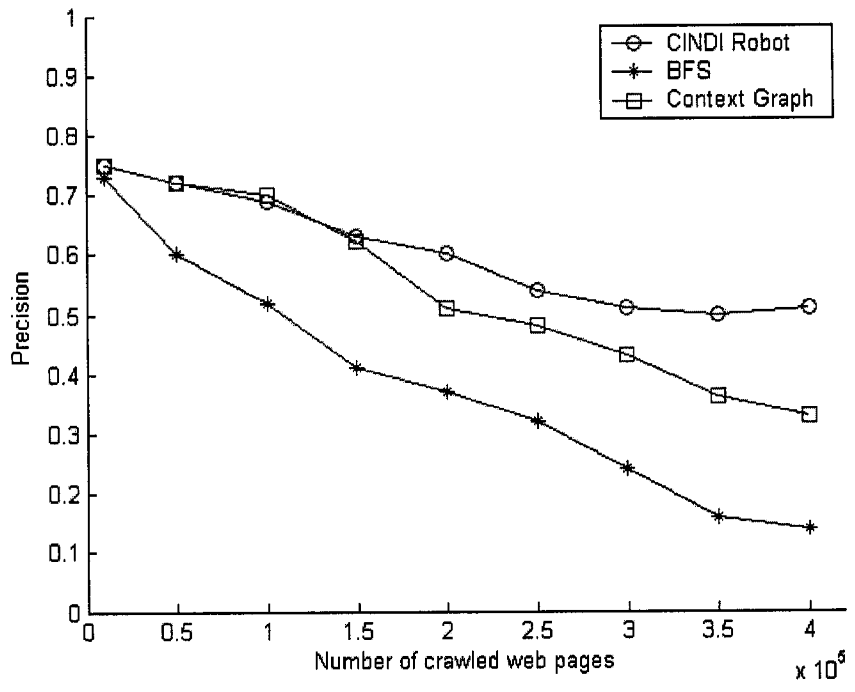


Figure 17 The precision of web crawlers

Generally, the precision of any focused web crawler declines during a crawling process. Thus it is important for a focused web crawler to slow down the precision decreasing process. From Figure 17, we can observe that the current CINDI Robot outperforms other web crawlers in terms of precision. Even after crawling 400,000 web pages, the precision still stays about 50%, which means the current CINDI Robot has the capability of remaining in relevant regions in long-term crawling. Its desirable precision attributes to URL pattern inspection, anchor text inspection, content text classification, revised context graph, real-time relevance feedback and URL ordering.

The context graph crawler exhibits a better precision than the previous version of CINDI Robot because it benefits from its layer classification mechanism and URL ordering policies. The previous CINDI Robot, as a Breadth-First Search crawler, gets worst precision. The previous CINDI Robot can only get a good precision when crawling well-selected seed web sites. This explains its good precision of the first 50,000 web pages. After that, its precision declines dramatically, which reflects the fact that the previous CINDI Robot lacks enough mechanisms to guide a focused web crawling.

5.3.2. Comparison of target recall

As indicated in [27], true recall is hard to measure since we can not know the actual relevant web site set of the Web. Thus target recall is used to substitute the true recall. Here we randomly select 10,000 web sites from the results obtained by Seed Finder as the target set. If a page in a certain web site is crawled, we consider this web site as retrieved. The target recall comparison is given in Figure 18.

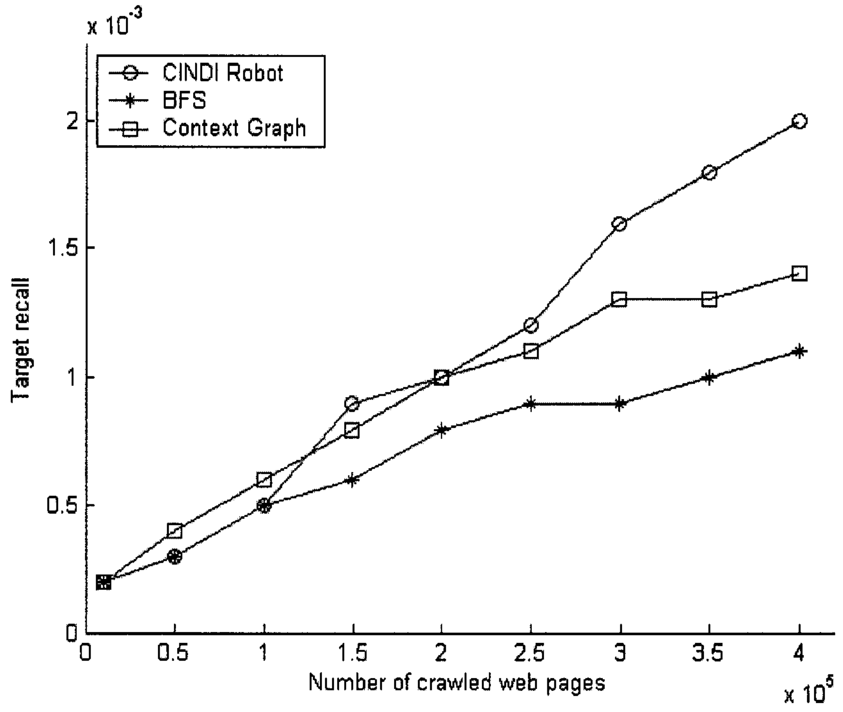


Figure 18 The target recall of web crawlers

The experimental results demonstrate that CINDI Robot exhibits a more desirable target recall than BFS and context graph crawlers. The better performance of the current CINDI Robot mainly ascribes to tunneling and the revised context graph strategies. The context graph crawler also has the ability to discover indirectly related topics. However its ability is largely confined by its strict link distance requirements. The previous CINDI Robot has the poorest target recall due to its poor precision which makes it hard to find on-topic web sites.

5.3.3 Comparison of crawling speed

This section we compare the crawling speed improvement of the current CINDI Robot with the previous CINDI Robot. Table 12 gives the hours needed to complete 400,000 web pages for both versions of CINDI Robot.

Table 12 Comparison of crawling speed

	Time
Previous CINDI Robot	376.4 hours
Current CINDI Robot	4.1 hours

The processing time of a web page is made up of two components: downloading time and classifying time. The classifying time of a web page is relatively invariable. However, the downloading time of web pages may vary drastically from site to site. Since CINDI Robot traverses the Web in the unit of web sites, its crawling speed may be largely affected by slow web sites. However, normally it should not take more than 10 seconds to retrieve a web page. According to our experience, the chance of failure of a web page which takes more than 10 seconds to download is as high as 99%. This is the reason why we force CINDI Robot to drop a web page downloading task after 10 seconds. Under our fixed network bandwidth and classification speed, in the optimum case it takes on average only 0.8 second to process a web page in a single-thread application. Thus we can calculate that in an ideal case a 75-threads application takes only about 1.2 hours to crawl 400,000 web pages. However in real crawling, it is not the case.

From Table 12, we can observe that the current CINDI Robot has $376.4/4.1 = 91.8$ times speed-up. Beyond this number, we should realize three points. First of all, we introduced a Naïve Bayes classifier into the previous CINDI Robot, which should cause some crawling speed degradation of about 5%. Thus the current CINDI Robot would still have at least a speed-up of $376.4 \times 95\%/4.1 = 87.2$. Secondly, using 75 threads cannot result in 75 times speed-up even though it provides significant improvement of crawling speed. The last thing is that the actual time of crawling not only depends on the crawling speed, but also relies on if the crawler can efficiently prune irrelevant web regions.

The speed-up shown in Table 12 is reached by two factors: multi-threading and rectifications of previous CINDI Robot drawbacks. Considering that *in an ideal case* it only takes on average 0.5 second to process a web page in previous CINDI Robot [4], the previous CINDI Robot should take only about 89 hours to complete 400,000 web pages *in an ideal case*. The difference between 376.4 hours and 89 hours not only indicates the challenges a focused web crawler has to face, but also indicates the drawbacks embedded in the previous CINDI Robot. Passive error handling and Robot Exclusion Protocol re-implementation are two main modifications from the previous CINDI Robot.

5.3.4 Comparison of downloaded document relevant rate

In this section, we examine the downloaded document relevant rates among these three web crawlers. Table 13 gives three web crawlers' numbers of discovered documents, numbers of downloaded documents, numbers of accepted documents and document acceptance rates. The number of discovered documents shows how many document URLs are found during the crawling process. Not all discovered documents are downloaded. Some of them may encounter connection failures and some of them are directly filtered out by File Fetcher. The number of downloaded documents reflects the number of actually downloaded documents. All downloaded documents are filtered by DFS and thus document acceptance rates can be calculated.

Table 13 Comparison of downloaded document relevant rates

	Num of Discovered Documents	Num of Downloaded Documents	Num of Accepted Documents	Document Acceptance Rate
Previous CINDI Robot	129,298	128,973	11,946	9.26%
Context graph crawler	118,823	118,485	34,761	29.34%
Current CINDI Robot	127,275	124,963	45,121	36.11%

As mentioned before, the margins between numbers of discovered documents and numbers of downloaded documents are caused by two reasons: 1) the pre-filtering functions introduced in File Fetchers of both versions of CINDI Robots; 2) unexpected connection failures. From Table 13, we can observe that the enhanced File Fetcher in current CINDI Robot can more effectively remove undesirable document URLs. As to document acceptance rates, context graph crawler and the current CINDI Robot obtain much better results. Starting from a small set of relevant seed URLs, the previous CINDI Robot is easy to lose its focus. Thus its document acceptance rate is extremely low. URL ordering and content text classifiers are implemented in both context graph crawler and the current CINDI Robot, which help them obtain a much better document acceptance rate. In addition, the URL pattern inspection, anchor text inspection and real-time relevance feedback bring additional improvements to current CINDI Robot's document acceptance rate.

5.4 Experiments on Individual Heuristics

In this section, we inspect the performance improvements due to individual heuristics implemented in current CINDI Robot. URL pattern inspection, anchor text inspection, tunneling and revised context graph are examined. However, content text classifiers, URL ordering and real-time relevancy feedback are considered as essential parts of the current CINDI Robot, so we do not examine the improvements attributed to them. These experiments are performed using 10 randomly selected seed URLs and 75 threads. Here we examine the experimental results using the first 500000 web pages of the crawling process.

5.4.1 URL pattern inspection

URL pattern inspection mainly contributes to the precision. One main responsibility of URL pattern inspection is to exclude those to-be-avoided directories during re-crawling processes, which will be studied in Section 5.5. URL pattern inspection brings much more salient precision improvement for re-crawling than for initial crawling.

Figure 19 shows the precision curves of CINDI Robot with and without URL pattern inspection, where "full-version" denotes the CINDI Robot using complete mechanisms, "without-upi" denotes CINDI Robot without URL pattern inspection.

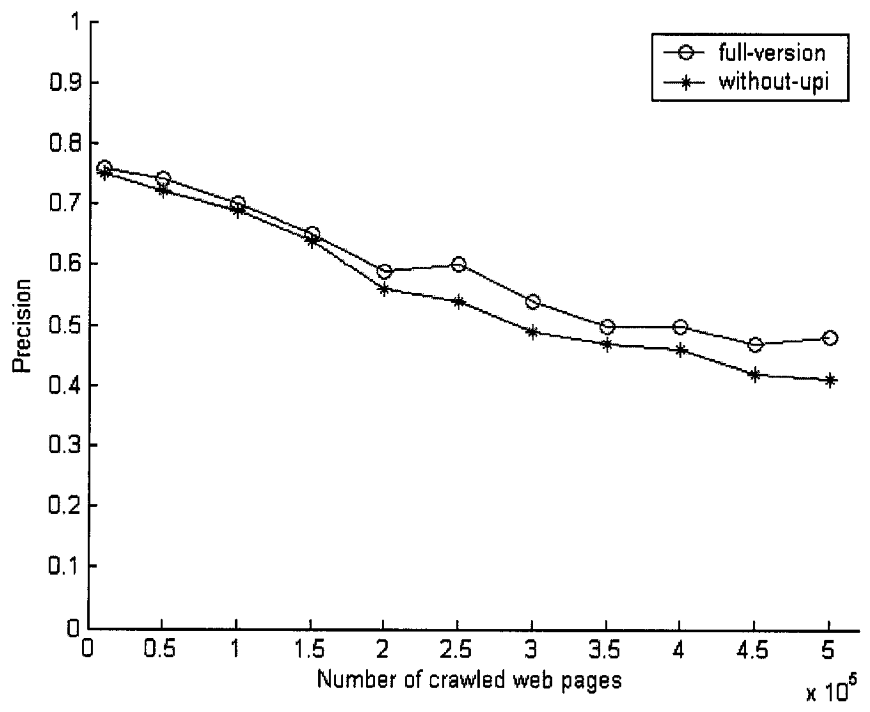


Figure 19 Precision improvement due to URL pattern inspection

From Figure 19, we can find that URL pattern can bring about 3% improvement. Stop-directory filtering, seven directory level exclusion and the Robot Exclusion Protocol are main reasons of precision improvement. Other aspects of URL pattern inspection, such as computer science department web site speculation and protocol and file format filtering, can accelerate the crawling process. Computer science department web site speculation can save time on content text classifications while protocol and file format filtering can save time used to crawl useless web resources.

5.4.2 Anchor text inspection

Compared to URL pattern inspection, anchor text inspection enhances more precision increase. On the one hand, anchor text inspection prunes irrelevant hyperlinks from relevant web pages; on the other hand, it extracts only potentially useful hyperlinks from Layer 2 web pages. This strategy keeps CINDI Robot from useless web pages, which is demonstrated in Figure 20, where "without-ati" denotes the CINDI Robot without anchor text inspection.

Figure 20 shows that using anchor text inspection the precision of CINDI Robot can improve about 6%. This improvement gets more obvious as the crawling process goes on. This feature makes anchor text inspection even more desirable for long time crawling. Recall that in Figure 17 the precision of the previous CINDI Robot drops dramatically during the crawling process. Here anchor text inspection can to a great extent offset the precision degradation.

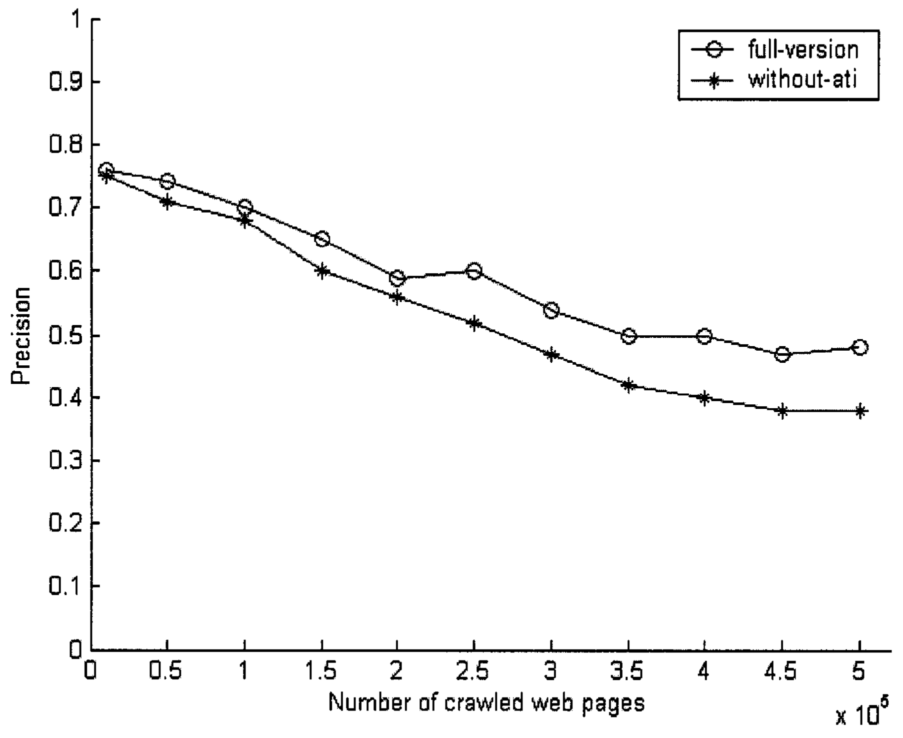


Figure 20 Precision improvement due to anchor pattern inspection

5.4.3 Tunneling and revised context graph

Tunneling and revised context graph are two main strategies used in current CINDI

Robot to increase recall. The revised context graph allows CINDI Robot to include Layer 2 web pages into URL frontier and the tunneling strategy looses the Layer 2 web page selection requirement to increase the chance of finding more relevant web sites. Figure 21 gives the target recall comparison between the complete CINDI Robot and the one without the revised context graph and tunneling. The target set is also formed by 10,000 randomly selected web sites from Seed Finder.

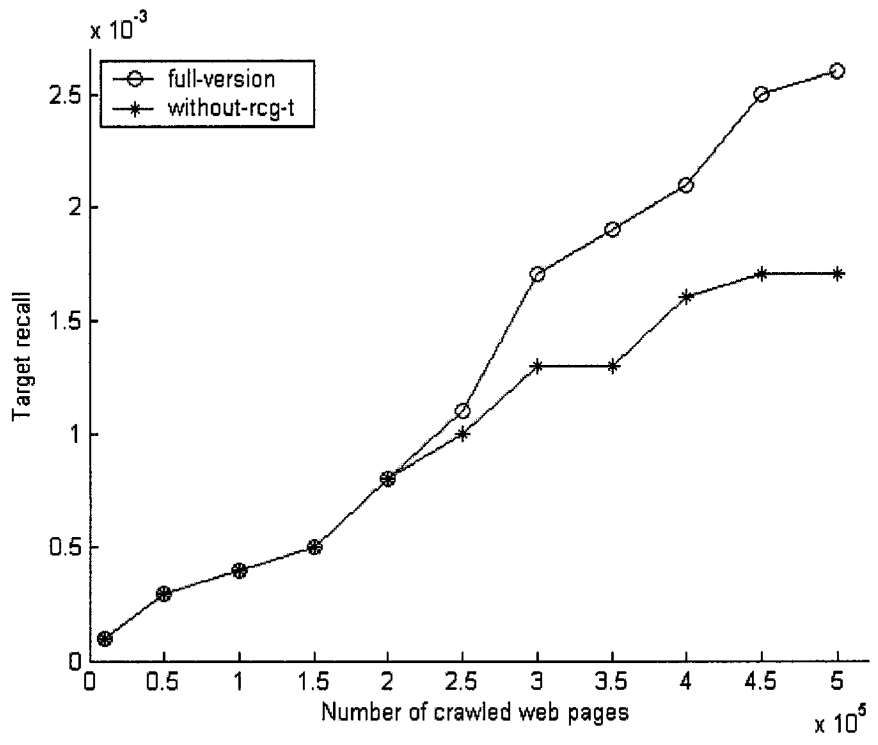


Figure 21 Target recall improvement due to tunneling and revised context graph

Figure 21 shows that using the revised context graph and our tunneling strategy, CINDI Robot can discover more relevant regions by traversing some less relevant intermediates. At the initial crawling phase, two web crawlers crawl the same set of seed URLs, thus they have the same target recall. After the completion of seed URLs, the revised context graph and tunneling strategy begin to show their contributions. As the crawling process goes on, the precision goes down. At this time, the revised context graph and tunneling strategy can identify more web pages leading to relevant regions and thus increase the recall.

5.4.4 Summary

Based on the above experiments, we can summarize the contributions of all heuristics in terms of precision, recall and crawling speed, which are shown in Table 14. Content text inspection, URL ordering and real-time relevance feedback are implemented as essential parts of CINDI Robot. It is easy to see how content text inspection and real-time relevance feedback contribute to precision. However it is not easy to observe that URL ordering contributes to all performance metrics by ordering incoming web sites, excluding useless web sites and including Layer 2 web sites.

Table 14 Contributions of individual heuristics

	Recall	**Precision**	**Crawling Speed**
Multi-threading			√
URL pattern inspection		√	√
Anchor text inspection	√	√	√
Content text inspection		√	
Revised context graph	√	√	
Tunneling	√		
Real-time relevance feedback		√	√
URL ordering	√	√	√

5.5 Experiments on Re-crawling

In [4], the author used www.cs.concordia.ca as a sample seed to demonstrate the power of keeping CINDI Robot crawling within domain. He mentioned that after optimization it still took 3 hours and 10 minutes for the total of 9,578 visited web pages. Here we use these data as the benchmark to show the performance improvement of the current CINDI Robot, especially for the re-crawling process.

Since there is only one seed URL in the whole crawling cycle, we manually tune the current CINDI Robot to allow all threads to collaborate on crawling web links from the same web site. Here only 30 threads are used in order to lower the possibility of congesting the web server and thus lowering the CINDI Robot performance. For initial crawling, it takes only 5.8 minutes to crawl the whole web sites and 9,164 web pages are visited. Some web pages are skipped by the URL pattern inspection and the anchor text inspection. This time saving is brought by multi-threading and also code optimization as mentioned in Section 4.6. After the first crawling, to-be-avoided directory list for www.cs.concordia.ca is built and we perform the re-crawling. In the re-crawling, only 8,027 web pages are visited and it only takes about 4.9 minutes. That means by importing to-be-avoided directory list, we can bring approximately (5.8 – 4.9)/5.8 = 15.5% speed-up.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Online documents are good sources for CINDI digital library. CINDI Robot is consequently developed to traverse the whole Web to collect on-topic documents. Due to the huge volume and dynamic nature of the World Wide Web, general purpose web crawlers are infeasible to keep all on-topic documents up-to-date. Thus CINDI Robot is designed as a focused web crawler, which crawls the Web based on web pages' relevancy in order to crawl most relevant web pages as early as possible and to skip irrelevant regions. CINDI Robot is implemented in JAVA as a multi-threaded program in order to increase crawling speed. CINDI Robot consists of five components, namely Seed Finder, Web Crawler, Link Analyzer, Statistics Analyzer and File Fetcher. Seed Finder identifies seed URLs which are used as starting points for focused crawling processes. Both ODP based approach and general purpose search engine based approach are utilized to augment the number of seed URLs. Web Crawler performs both URL pattern inspection and anchor text inspection, retrieves web pages, extracts hyperlinks and identifies document URLs. Link Analyzer utilizes content text classifiers, revised context graph and tunneling to rank web pages based on their relevancy to the topic. Statistics Analyzer constructs a stop-directory list and to-be-avoided lists to exclude useless URL directories. It also performs a real-time relevancy monitoring. File Fetcher downloads documents and performs file related operations. In Chapter 5, various experiments demonstrate the current CINDI Robot has exhibited desirable improvements on the precision, recall and crawling speed and thus is able to scale up to the whole Web.

6.2 Contribution of This Thesis

The main goal of this thesis is to enhance CINDI Robot's performance in terms of precision, recall and crawling speed. To achieve this goal, we propose a novel multi-level inspection infrastructure which includes content text inspection, anchor text inspection and URL pattern inspection for the current CINDI Robot. This infrastructure maximally takes advantage of the characteristics of web pages rather than only focuses on content texts. Content texts, anchor texts are usually considered independently in other papers and URL information is ignored in most papers. In this thesis, we demonstrate that combining content texts, anchor texts and URL patterns can result in an obvious improvement of performance.

To support this infrastructure, we investigate some state-of-the-art techniques of focused web crawlers and propose our own strategies to enhance the CINDI Robot. For content text inspection, we propose a weighted Naïve Bayes classifier which increases the accurate classification rate by using a $TF \times IDF$ scheme; we design a simple tunneling strategy which gets rid of the error-prone taxonomy tree construction in the classic tunneling technique; we introduce a revised context graph algorithm which removes the strict link distance requirement of the classic context graph algorithm. For URL pattern inspection, we utilize the URL information to speed-up the crawling process and to increase the precision. The usefulness of URL patterns for a focused web crawler is elaborated and emphasized in this thesis. For anchor text inspection, we developed a unique double-mode identification strategy which increases both precision and recall. The experimental results demonstrate that this multi-level inspection infrastructure helps the current CINDI Robot outperform other traditional focused web crawlers.

6.3 Future Work

Since focused web crawlers are still at their incipient stage, there is still much room to improve a focused web crawler. For CINDI Robot, we consider following six directions as future work.

In content classification, phrases may provide more precise semantic meaning than single words. For example, "mobile" and "agent" as single words may appear in other topics while "mobile agent" is a good indicator of a computer science and software engineering related web page. Thus we can add phrase extraction in the preprocessing step and in content text classification. The second direction is developing a distributed crawling system. Distributed web crawlers are the general developing tendency and have been used for many mature, large-scale search engines, for example, Google and AltaVista. Although a focused web crawler can be more efficient to scale up the whole Web in contrast with a general purpose web crawler, a distributed system may also be necessary due to the explosion of the Web. A distributed focused web crawler can bring more speed-ups, which can make crawled web pages up-to-date. The third direction is to further enlarge high quality seed URLs number. In the current CINDI Robot, we use manually collected query to cover all computer science and software engineering fields. This process can be automated. Many query construction techniques have been proposed from which we can get a list of well-selected queries by simply inputting a topic name. Since CINDI system is going to expand to a comprehensive digital library to cover all academic topics, this automated query construction technique is essential. The fourth direction is employment of a more sophisticated URL ordering mechanism. In the current CINDI Robot, we classify all web pages into three categories and order web pages based on their categories. In future work, we can use a more sophisticated scoring scheme to assign each web page a real number score and sort web pages based on their scores. The fifth direction is to take advantage of other web page quality metrics in addition to relevancy-based page quality metric in order to improve crawling performance, for example connectivity-based page quality metric. The deep Web [57] is also a valuable working direction. In spite of huge volume of web pages on the surface Web, even more web pages exist as the form of deep web pages. In 2000, it was estimated that the deep Web contained approximately 7,500 terabytes of data and 550 billion individual documents [58] while even in 2007, there are only 29.7 billion surface web pages [8].

References

[1] CINDI digital library project, available at: http://cindi.encs.concordia.ca

[2] ISC Domain Survey: Number of Internet Hosts, available at: http://www.isc.org/index.pl?/ops/ds/host-count-history.php

[3] Nihar Bihani, "Search Engine Optimization Blog", available at: http://www.ecnext.com/ecnext/blogs/organic/

[4] Cong Zhou, "CNDROBOT – A Robot for the CINDI Digital Library", Master Thesis, Department of Computer Science, Concordia University, December 2005.

[5] Shkapenyuk V, Suel T, "Design and implementation of a high-performance distributed Web crawler", the Proceeding of the 18th International Conference on Data Engineering, 2002.

[6] Lyman, P. and Varian, H. R. "How much information", available at: http://www.sims.berkeley.edu/how-much-info-2003/

[7] World Wide Web, available at: http://en.wikipedia.org/wiki/World_Wide_Web

[8] Andrei Broder, Ravi Kumar, et al, "Graph structure in the Web", available at: http://net.pku.edu.cn/~wbia/2005/public_html/papers/webGraph/Graph%20structure%20in%20the%20web.pdf

[9] How many websites are there, available at:
http://www.boutell.com/newfaq/misc/sizeofweb.html

[10] Internet Usage Statistics, available at: http://www.internetworldstats.com/stats.htm

[11] Open Directory Project, available at: http://www.dmoz.org/

[12] Open Directory Project instruction, available at:
http://www.dmoz.org/help/geninfo.html

[13] Ted Goldsmith, "Case Study – Search Engine Censoring of Open Directory Data", available at: http://www.searchenginehonesty.com/searchvsodp.pdf

[14] Web directory, available at: http://en.wikipedia.org/wiki/Web_directory

[15] Google search engine, available at: http://www.google.com

[16] AltaVista search engine, available at: http://www.altavista.com

[17] CiteSeer digital library, available at: http://citeseer.ist.psu.edu

[18] Soumen Chakrabarti, Focused Crawling: The Quest for Topic-specific Portals, available at: http://www.cse.iitb.ac.in/~soumen/focus/

[19] Monica Peshave, "How Search Engines Work and A Web Crawler Application", available at: http://www.micsymposium.org/mics_2005/papers/paper89.pdf

[20] Google Search, available at: http://en.wikipedia.org/wiki/Google_search

[21] A standard for robot exclusion, available at: http://www.robotstxt.org/orig.html

[22] Menczer, F., "ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery", Proceedings of the 14th International Conference (ICML97).

[23] Chakrabarti, S., et al, "Focused crawling: a new approach to topic-specific web resource discovery", Computer Networks, 31(11-16), pp. 1623-1640.

[24] Huajing Li, Isaac Councill, et al. "CiteSeer$^{\chi}$: an Architecture and Web Service Design for an Academic Document Search Engine", 15th International World Wide Web Conference, 2006.

[25] Gautum Pant, Kostas Tsioutsiouliklis, et al. "Panorama: Extending Digital Libraries with Topical Crawlers", Proceedings of the 2004 Joint ACM/IEEE Conference on Digital Libraries.

[26] Google Web APIs, available at: http://www.google.com/apis/

[27] Gautam Pant and Padmini Srinivasan, "Link Contexts in Classifier-Guided Topical Crawlers", IEEE Transactions on Knowledge and Data Mining, Vol.18, No.1, 2006.

[28] M. Diligenti et al., "Focused Crawling using Context Graphs", 26th International Conference on Very Large Databases, VLDB 2000, pp.527-534.

[29] Ching-Chi Hsu, Fan Wu, "Topic-specific crawling on the Web with the measurements of the relevancy context graph", Information Systems 31, 2006, pp. 232-246.

[30] R. Babaria, J. Saketha Nath, Krishnan S, SivaramakrishnanK R, C. Bhattacharyya,M. N.Murty, "Focused Crawling with Scalable Ordinal Regression Solvers", Proceedings of the ICML-2007 conference.

[31] Marc Ehrig and Alexander Maedche, "Ontology-Focused Crawling of Web Documents", Proceedings of ACM Symposium on Applied Computing 2003, Melbourne, Florida, USA.

[32] Knut Eivind Brennhaug, "EventSeer: Testing Different Approaches to Topical Crawling for Call for Paper Announcements", June 2005, available at: www.diva-portal.org/diva/getDocument?urn_nbn_no_ntnu_diva-601-1__fulltext.pdf

[33] Jingru Dong et al., "Focused crawling guided by link context", Proceedings of the 24th IASTED International Conference on Artificial Intelligence and Applications, 2006.

[34] Nicola Baldini and Federico Neri, "A Multilingual Text Mining based Content Gathering System for Open Source Intelligence", Proceedings of IAEA International Atomic Energy Agency, October 2006.

[35] Jyh-Jong Tsay et al., "AutoCrawler: An Integrated System for Automatic Topical Crawler", Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science, 2005.

[36] Martin Ester, Matthias Grob, Hans-Peter Kriegel, "Focused Web Crawling: A Generic Framework for Specifying the User Interest and for Adaptive Crawling Strategies", 27th International Conference on Very Large Database, Rom, Italien, 2001.

[37] Vladimir N. Vapnik, "The Nature of Statistical Learning Theory", Springer, 1995.

[38] Joachims T. "Text Categorization with Support Vector Machines: Learning with Many Relevant Features", Proceedings of ECML-98, 10th European Conference on Machine Learning, 1998.

[39] Nello Cristianini, John Shawe-Taylor, An introduction to Support Vector Machines and other kernel-based learning methods, Cambridge University Press, England, 2000.

[40] Rung-Ching Chen and Chung-Hsun Hsieh, "Web page classification based on a support vector machine using a weighted vote schema", Expert Systems with Applications 31, 2006, pp.427-435.

[41] Ioan Pop, "An approach of the Naïve Bayes classifier for the document classification", General Mathematics Vol. 14, No. 4, 2006, pp. 135-138.

[42] Jia-he, Zhao, et al., "Design and implementation of focused web crawler based on semantic analysis", Journal of Computer Applications, v 27, n 2, Feb.2007, pp.406-408.

[43] Jialun Qin, Yilu Zhou and Michael Chau, "Building Domain-Specific Web Collections for Scientific Digital Libraries: A Meta-Search Enhanced Focused Crawling Method", Proceedings of the 2004 Joint ACM/IEEE Conference on Digital Libraries, Tucson, Arizona, USA.

[44] M. Yuvarani, et al., “LSCrawler: A Framework for an Enhanced Focused Web Crawler based on Link Semantics”, Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence.

[45] Mohsen Jamali, et al., “A Method for Focused Crawling Using Combination of Link Structure and Content Similarity”, Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence.

[46] Sergey Brin and Lawrence Page, "The anatomy of a large-scale hypertextual Web search engine". Proceedings of the Seventh International Conference on World Wide Web pp.107-117, 1998.

[47] B. C. Desai, May 1994, “A system for Seamless Search of Distributed Information Sources”, available at: http://users.encs.concordia.ca/~bcdesai/web-publ/w3-paper.html.

[48] Sami Samir Haddad, “Automatic Semantic Header Generator”, Master Thesis, Department of Computer Science, Concordia University, 1998.

[49] Yuwei Feng, “CONFSYS: Enhancement and Integration”, Master Thesis, Department of Computer Science, Concordia University, August 2003.

[50] Tong, Z, “A Gleaning Subsystem for CINDI”, Master Thesis, Department of Computer Science, Concordia University, 2004.

[51] Krishma Dutta, “Enhancement and Integration for CINDI System”, Master Thesis, Department of Computer Science, Concordia University, August 2007.

[52] Brian D. Davison, "Topical Locality in the Web", Proceedings of the 23rd Annual International Conference on Research and Development in Information Retrieval (SIGIR 2000), Athens, Greece, 2000.

[53] Neel, S., Jeonghee, L., Anital, H. "Using MetaData to Enhance a Web Information Gathering System", available at:
http://www.research.att.com/conf/webdb2000/PAPERS/1b.ps

[54] Donna Bergmark, Carl Lagoze, and Alex Sbityakov, "Focused Crawls, Tunneling, and Digital Libraries", Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries, pp. 91-106, 2002.

[55] Domingos, Pedro and Michael Pazzani, "On the optimality of the simple Bayesian classifier under zero-one loss", Machine Learning, pp.103-137, 1997.

[56] MySQL Home page, available at http://www.mysql.com

[57] CiteSeer Computer Science Directory, available at:
http://citeseer.ist.psu.edu/directory.html

[58] Silva, Catarina and Ribeiro, Bernardete, "The importance of stop word removal on recall values in text categorization", Proceedings of the International Joint Conference on Neural Networks, Vol. 3, pp. 1661-1666, 2003.

[59] G. Almpanidis, et al., "Combining text and link analysis for focused crawling- An application for vertical search engines", Information Systems, 2006.

[60] Baeza-Yates, R., Castillo, C., Marin, M. and Rodriguez, A., "Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering", Proceedings of the Industrial and Practical Experience track of the 14th conference on World Wide Web, pp. 864–872.

[61] Stemming, available at: http://en.wikipedia.org/wiki/Stemming

[62] Stop words definition, available at: http://en.wikipedia.org/wiki/Stop_words

[63] C. J. Van Rijsbergen, S. E. Robertson and M. F. Porter, "New models in probabilistic information retrieval", British Library Research and Development Report, no. 5587, London: British Library.

[64] Chih-Ming Chen, et al., "An intelligent web-page classifier with fair feature-subset selection", Engineering Application of Artificial Intelligence 19, 2006, pp.967-978.

[65] Chih-Wei Hsu, et al., "A Practical Guide to Support Vector Classification", available at: www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[66] Salton, G. and McGill, M. J. Introduction to modern information retrieval, McGraw-Hill, 1983.

[67] T. Joachims, "Making Large-scale SVM Learning Practical", Advances in Kernel Methods – Support Vector Learning, MIT Press, 1999.

[68] Cothey, V., "Web-crawling reliability". Journal of the American Society for Information Science and Technology 55 (14), 2004.

[69] Ayache, S. and Quenot,G., “Evaluation of Active Learning Strategies for Video Indexing”, Content-Based Multimedia Indexing, 2007. CBMI '07. International Workshop, 2007, pp.259 - 266”.

[70] Deep Web, available at : http://en.wikipedia.org/wiki/Deep_web

[71] Bergman, Michael K., "The Deep Web: Surfacing Hidden Value". The Journal of Electronic Publishing 7 (1), Aug 2001.

Appendix A

Query List

This list presents queries that we use to obtain seed URLs in the general search engine based approach as described in Section 4.1.2. It is organized by computer science subfields provided by CiteSeer computer science directory.

Agent

mobile ambient
Oz programming model
real-time dynamic programming
BDI agent
mobile agent
concept language
markov game
multi agent
reinforcement learning
complex group action

Applications

modular eigenspace
face recognition
FERET
recurrent net
speech recognition
hidden markov model
motion regularization
HMM-based
linear transformation
global optimization
neural network
context decode
interactive translation

Architecture

integrated communication computation
intelligence reason
software architecture
distribute shared memory
virtual memory
network interface
weak ordering
vectorization multiprocessor
sequential consistency
compiler transformation
optimization parallelism

Artificial Intelligence

bagging predicate
graph analysis algorithm
general purpose planning
STRIPS planning
propositional logic
stochastic search
subset selection
model render architecture
constraint satisfaction schedule
least commitment planning

Compression

packet video
CMOS design
framework simulate prototype
heterogeneity mix mode
variable bit rate VBR
schedule operating system
time scale traffic

multiple time scale
protocol network communication
multicast transcode
predictability data value
delta encode
data compression algorithm

Database

data query language
semistructure unstructured data
semantic logic program
unfounded set logic
object-oriented database OODBS
data model DBMS
persistent scalable database
query formulation optimization
knowledge representation
relational schema integrity expression
incremental recomputation
data integration approach
cache index data
XML query language

Hardware

parallel distributed compute
resource management operating system
symbolic Boolean binary manipulation
hardware architecture program
value locality prediction
cluster scalable network

Human Computer Interaction

seamless interface user
coupling query filter
starfield display information seeking
affective compute
image motion model
adaptive hypermedia navigation
uncertain dynamic agent
DOF structure visual track
autonomous interface agent
ubiquitous computing

Information Retrieval

agent communication language
integration heterogeneous information
text categorization feature selection
Support Vector Machines learning
asymmetric communication environment
linear algebra information retrieval
vector space database
TF IDF
text classifier
incremental index full text
transductive inference classification

Machine Learning

machine learning algorithm
fuzzy systems
neural networks
mining set rules database
pattern recognition
combinatorial geometric computing
approximation algorithm
nearest neighbor search
cluster high dimensional data
additive logistic regression
hash based algorithm
rocchio probabilistic algorithm
support vector machine
genetic algorithm GA

Networking

random detection congestion avoidance
multicast framework framing
Ethernet traffic LAN
poisson modeling network
real-time packet network
self-similarity web traffic
network route protocol

IP mobile internetworking
synchronization periodic networking
network throughput layer

Operating Systems

cache file system memory
kernel binary run-time
operating system design
fault tolerant distributed system
time sharing system interface
remote client memory file
page uniprocessor software
schedule CPU computer
microkernel OS thread
network file system
multiprocessing architecture context switch

Programming

software fault isolation
uniprocessor garbage collection
logic programming language
imperative functional programming
object concurrent programming
aspect oriented programming
compiler design optimization
data locality transformation
loop fusion distribution
linear algebra software
system verification validation

Security

digital signature public key
cryptography protocol
access control groupware
encryption algorithm
intellectual property protection
intrusion detection
information warfare

Software Engineering

software architecture
formal methods
program slicing technique
software process modeling
object oriented programming
software visualization
legacy code
agent oriented analysis design
software design pattern
software reliability debugging
software specification morphism
random sampling computational geometry

Theory

computational complexity
quantum computation
logic disjunctive database
deadlock free routing
knowledge compilation horn approximation
higher order logic programming
wavelet closed subset
adversarial queueing theory
intersection type bounded polymorphism
calculus symmetric concatenation

World Wide Web

hypertextual web search engine
metacrawler resource aggregation
focused web crawler discovery
semi-automatic wrapper generation
link topology hypertext
context graph crawl
automated negotiation system electronic
formal ontology information

Appendix B

Stop Word List

This list enumerates the stop words used in the preprocessing as described in Section 4.3.1.

a	anyways	believe	contains
able	anywhere	below	corresponding
about	apart	beside	could
above	apparently	besides	couldn't
according	appear	best	course
accordingly	appreciate	better	currently
across	appropriate	between	d
actually	are	beyond	definitely
after	aren't	both	described
afterwards	around	brief	despite
again	as	but	did
against	aside	by	didn't
all	ask	c	different
allow	asks	came	do
allows	asking	can	does
almost	associated	cannot	doesn't
alone	at	can't	doing
along	available	cant	don't
already	away	cause	done
also	awfully	causes	down
although	b	certain	downwards
always	be	certainly	during
am	became	change	e
among	because	clearly	each
amongst	become	co	edu
an	becomes	com	eg
and	becoming	come	eight
another	been	comes	either
any	before	concerning	else
anybody	beforehand	consequently	elsewhere
anyhow	begin	consider	end
anyone	beginning	considering	ending
anything	behind	contain	enough
anyway	being	containing	entirely

especially	going	i'd	latterly
et	gone	i'll	least
etc	got	i'm	less
even	gotten	i've	lest
ever	greetings	ie	let
every	gt	if	let's
everybody	h	ignored	like
everyone	had	immediate	liked
everything	happens	in	likely
everywhere	hardly	inasmuch	little
ex	has	inc	look
exactly	hasn't	indeed	looking
example	have	indicate	looks
except	haven't	indicated	lt
f	having	indicates	ltd
far	he	inner	m
few	he'd	insofar	made
fifth	he'll	instead	make
first	he's	into	mainly
five	hello	inward	man
followed	help	is	many
following	hence	isn't	may
follows	her	it	maybe
for	here	it's	me
former	here's	its	mean
formerly	hereafter	itself	meantime
forth	hereby	j	meanwhile
four	herein	just	merely
from	hereupon	k	might
further	hers	keep	more
furthermore	herself	keeps	moreover
fully	hi	kept	most
g	him	know	mostly
gave	himself	knows	much
get	his	known	must
gets	hither	l	my
getting	hopefully	largely	myself
given	how	last	n
gives	howbeit	lately	namely
go	however	later	nd
goes	i	latter	near

nearly	ought	recent	shouldn't
necessary	our	recently	shown
need	ours	refs	shows
needs	ourselves	reg	significantly
neither	out	regarding	similar
never	outside	regardless	similarly
nevertheless	over	regards	since
new	overall	relatively	slightly
next	own	respectively	six
nine	p	resulting	so
no	particular	right	some
nobody	particularly	s	somebody
non	per	said	somehow
none	perhaps	same	someone
nonetheless	placed	saw	something
noone	please	say	sometime
nor	plus	saying	sometimes
normally	possible	says	somewhat
not	possibly	second	somewhere
nothing	potentially	secondly	soon
now	predominantly	see	sorry
nowhere	present	seeing	specifically
o	presumably	seem	specified
obviously	previously	seemed	specify
of	primarily	seeming	specifying
off	probably	seems	still
often	promptly	seen	strongly
oh	provides	self	sub
ok	q	selves	substantially
okay	que	sensible	successfully
old	quickly	sent	such
on	quite	serious	sufficiently
once	quot	seriously	sup
one	qv	seven	sure
ones	r	several	t
only	rather	shall	take
onto	rd	she	taken
or	re	she'd	taking
other	readily	she'll	tell
others	really	she's	tends
otherwise	reasonably	should	th

than
thank
thanks
thanx
that
that'll
that's
that've
thats
the
their
theirs
them
themselves
then
thence
there
there'd
there'll
there're
there've
thereafter
thereby
therefore
therein
theres
thereupon
these
they
they'd
they'll
they're
they've
think
third
this
thorough
thoroughly
those
though
three
through
throughout
thru
thus
to
together
too
took
toward
towards
tried
tries
truly
try
trying
twice
two
u
un
under
unfortunately
unless
unlike
unlikely
until
unto
up
upon
us
use
used
useful
usefully
uses
using
usually
uucp
v
value
various
very
via
viz
vs
w
want
wants
was
wasn't
way
we
we'd
we'll
we're
we've
welcome
well
went
were
what
what'll
what's
what've
whatever
when
whence
whenever
where
where's
whereafter
whereas
whereby
wherein
whereupon
wherever
whether
which
while
whither
who
who'd
who'll
who's
whoever
whole
whom
whomever
whose
why
widely
will
willing
wish
with
within
without
wonder
won't
would
wouldn't
x
y
yes
yet
you
you'd
you'll
you're
you've
your
yours
yourself
yourselves
z
zero