Undergraduate Research Opportunity Program

(UROP) Project Report

Augmenting Focused Crawling using Search Engine Queries

By

Wang Xuan

Department of Computer Science

School of Computing

National University of Singapore

2006/2007

Undergraduate Research Opportunity Program

(UROP) Project Report

Augmenting Focused Crawling using Search Engine Queries

Wang Xuan

Department of Computer Science

School of Computing

National University of Singapore

2006/2007

Project No: U079120

Advisor: Dr. Kan Min Yen

Deliverables:

Report: 1 Volume

Program: 1 Diskette

Abstract

The pervasiveness of the Internet makes it an ideal medium for sharing scholarly information. Nowadays, many authors post their publications online so that others may easily access to them, increasing the author's impact in his/her research area. In this project, we develop a focused crawling to find *publication pages*, web pages that link to online, freely available scholarly publications. In contrast to previous works which only traverse hyperlinks within web pages, our algorithm leverages search engine queries to locate suitable pages for crawling. This strategy allows our crawler to locate more relevant pages and lessens the reliance of the crawler on the quality of seed pages used to start the crawling processing. Our crawler is also able to locate relevant pages that are not accessible by standard crawling method that work only by following hyperlinks. The results show our system is able to avoid slow start, and find publication pages faster, outperform local crawling methods.

Subject Descriptors:

I.2.7 Natural Language Processing

I.2.8 Problem Solving, Control Methods, and Search

H.3.1 Content Analysis and Indexing

H.3.3 Information Search and Retrieval

Keywords:

Focused Crawling, URL-based classification, spidering, search engine

Implementation software and hardware:

Software: Google API, Java, Apache Ant, openNLP MaxEnt

Hardware: Intel-based Linux, Tembusu, Sun Unix

Acknowledgement

The author would like to express her sincerest gratitude to her supervisor, Dr. Kan Min-Yen, for his continuous and invaluable guidance. Without his help, it is impossible for the author to complete the whole process of reviewing literature, planning, coding, experimenting, testing and evaluating. All the comments and advice I truly appreciated.

Special thank to Miss Nguyen Thi Hoang Oanh, for her kind explanation of her project at the very beginning of my work, which forms the baseline system for the work presented here.

I also extend my gratitude to all the members of the WING group, who shared their knowledge with me. This precious learning experience helped me establish a basic understanding of natural language processing, furthering my study beyond the scope of this project.

I also would like to thank the WING administration group for maintaining and updating the workstation on which my research was done with.

Finally, I wish to give credit to those who have in one way or another given support and assisted in the completion of this academic exercise.

Table of Contents

Abstract III

Acknowledgement IV

Table of Contents V

List of Figures and Tables VII

Chapter 1: Introduction 1

1.1 Augmenting Focused Crawling using Search Engine Queries 1

1.2 Crawling 1

1.3 Problem Statement 2

1.4 Summary of achievements 3

1.5 Roadmap 3

Chapter 2: Focused Crawling 5

2.1 Overview of focused crawling 5

2.2 Search and analysis algorithms 5

2.3 Related works 6

2.3.1 Crawling using Naïve Bayes Classifier 7

2.3.2 IBM focused crawler 7

2.3.3 CORA crawler 8

2.3.4 Context Focused Crawler 9

2.3.5 Automatic Publication Data Gatherer 10

2.3.6 PaSE 11

Chapter 3: Baseline 13

3.1 Drawbacks of the Baseline System 14

3.2 Crawling stages in baseline system 15

Chapter 4: Augmenting Focused Crawling using Search Engine Queries 17

4.1 Design 17

4.2 Framework 19

4.3 Discussion on improvements to the Baseline System 21

Chapter 5: Evaluation 23
5.1 Experiment on precision 23
5.2 Estimated Recall 25
Chapter 6: Conclusion and Future Work 29
Bibliography 30
Appendix A - Top Forty Frequently Used Words ..... A

List of Figures and Tables

Figure 1: Baseline System.....13
Figure 2: Web Pages Unconnected to the Seed Page 14
Figure 3: Three Crawling Stages.....15
Figure 4: APDG with Publication Spider modules 19
Figure 5: List Item in Publication Page.....20
Figure 6 Comparison between baseline system and the improved system 24
Table 1: improvement in precision.....25
Table 2: Improvement in recall 27
Table 3: comparison of baseline system and upgraded system.....28

Chapter 1: Introduction

1.1 Augmenting Focused Crawling using Search Engine Queries

Nowadays, many authors post their publications online so that others may easily access to them, increasing the authors' impact in their research areas. A study (Lawrence, 2001) also showed that people prefer to cite the online academic references than the offline ones. Instead of using physical catalogs in the library, search engines and digital libraries are used increasing often by researchers to conduct their research. To assist in building such scholarly literature portals, in this project, I seek to build a focused crawler to locate and harvest such online, freely available publications.

Building such a system requires two stages: crawling and analyzing. In the crawling stage, the system searches the Internet looking for publications. During the analyzing stage, correctly identified publication pages retrieved in the first stage are analyzed for various purposes. In this project, I only focus on the crawling stage.

As publications form only a small part of the web, it is very important to improve the efficiency by eliminating the irrelevant pages. My purpose is to find a more efficient and accurate way to crawl publications. Specifically, this thesis investigates in a method to identify *publication pages*, web pages that link to publications rather than the publications themselves, and to perform crawling within a restricted computer domain (i.e., within a domain name). I will give more details on the problem specifics and its rationale later in this thesis.

1.2 Crawling

A *web crawler* (or simply, *crawler)* is a software program that traverses the hypertext structure tree of the World Wide Web, starting from a set of predefined web pages which are called *seed pages*. The crawler retrieves these pages, both to index them for later searching and to extract the hyperlinks inside these pages to identify

addresses for additional pages to be crawled. In some crawling architectures, this leads to a recursive process.

There are two main crawling methods: exhaustive crawling and focused crawling. *Exhaustive crawling* aims to collect all web pages on the web using traversal strategies such as breadth-first search. As exhaustive crawling traverses all pages regardless of their content, it is most often applied by the crawling component of general search engines like Google™ and Yahoo!™, as they need to acquire the most complete and exhaustive index of pages as possible. Such exhaustive crawlers save an offline cache for each website, to be used in answering user queries at runtime.

Exhaustive crawling is useful in general purpose crawling, requiring a very large storage space. However, it is inefficient when only pages about a specific subject are of interest, as in the case here where we are interested only in academic publications.

In contrast, *focused crawling* aims to collect a subset of documents relevant to a specific topic, avoids links that lead to off-topic documents. Starting from a predefined set of URLs as seed pages, it extracts all the hyperlinks and assigns relevance scores to pages with respect to a predefined subject. Pages with high relevancy are expanded first. Focused Crawler is often applied by search engines for specific purposes, like CiteSeer which harvest only scientific and academic papers.

1.3 Problem Statement

Most crawling methods start the crawling with user defined seed pages, and recursively visit the *neighbors* of the visited. (Neighbors of page M, refer to web pages that are linked from page M through the hyperlinks inside page M.) Start from one page and recursively visit the neighbors of visited is called *local search strategy*.

The local search strategy makes the crawling method's performances highly dependent on the seed pages. This can be a large issue. For instance, if we start from seed pages with no hyperlinks inside, or pages that are unrelated to the subject of interest, even after the whole crawling process, there is no way we can find any pages

of our interest, which we called target pages in this thesis. In addition, when most of the pages in the *Frontier,* the place where we store web pages to be visited, are not target pages, the crawler will randomly visit pages that may not lead to target page. This will make the crawler waste time on traverse irrelevant pages.

In this project, we want to lessen the dependence of the seed pages in the crawling methods. We also want to identify situations when the crawler is traversing on irrelevant pages, and avoid such situations by guiding the crawler to target pages.

1.4 Summary of achievements

In the first stage of this project, we investigated the performance of a baseline previously implemented by a former UROP student. We identified where this baseline is deficient and how to improve it. We decide to improve the performance by introducing search engine queries when the performance of the baseline system is not satisfactory.

In the second stage, we investigate on preparing the search engine queries, and get the result from the search engine. We developed a system to find and extract key words using target pages. Basically, the system identifies candidate publication titles in page, and splitting them using the top frequent words as delimiters. The result is a set of keywords for each publication title that can be used as queries to a search engine. These keywords are used to find more seed pages by querying a search engine. The system adds the resulting pages returned to the Frontier.

In the third stage, we have evaluated our system on a large set of web pages crawled from the `comp.nus.edu.sg` domain. Our results show that our system is competitive with current state-of-the-art technology crawlers, and outperforms the baseline system by a significant margin at certain stages of the crawl.

1.5 Roadmap

In the rest part of this report, we present an overview of focused crawling and related work in Chapter 2. In Chapter 3, we discuss the baseline system: a focused crawler that uses a URL classifier to assign a relevance score to each potential page to be crawled. The implementation detail of our program is presented in Chapter 4, followed by experimental results in Chapter 5. Chapter 6 concludes this thesis and provides some discussion on possible future works.

Chapter 2: Focused Crawling

2.1 Overview of focused crawling

When we view the World Wide Web as a graph, each web page can be treated as a node in the graph, and embedded hyperlinks are edges linking one page to other pages. Exhaustive crawling is similar to a graph search expanding a node to outgoing edges while focused crawling only expands nodes that may lead to relevant pages.

Exhaustive crawling can be viewed as a set of recursive actions that picks one unvisited URL from Frontier, fetches the page, extracts any embedded hyperlinks in the page, and adds any new URLs into Frontier. It usually does not examine the content of the crawling pages.

In contrast, focused crawling assigns higher priority to the pages that are more relevant to the topic in focus by categorizing pages into categories (e.g., a binary categorization such as target pages or not). The topic in which the user is interested in is normally specified using examples instead of keywords. The users need to select and refine specific instances for each categories and also need to select additional URLs as seeds to start with.

2.2 Search and analysis algorithms

Crawlers can be differentiated by the method used to select the URL to be crawled. Two methods are normally used: breadth-first or best-first. *Breadth-first* search is normally used in exhaustive crawling. The search order is based on when the page is initially encountered. However, *best-first* search is commonly used in focused crawling. Unvisited pages are first ranked according to some methods and the crawler then crawls the pages with in ranked order. Both breadth-first and best-first are local search algorithms. They start with a set of seed URLs, and then crawl the neighbors of these seed URLs. Therefore, the crawling result is highly dependent on the quality of

the seed URLs. If the system starts with a poor set of seeds that does not connect directly to any target pages, there is no way it can find any target pages.

Focused crawlers typically have an analysis stage in which the crawler considers which URL to pick from the unvisited pool. These algorithms can also be categorized into two basic types: content-based and link-based analysis. *Content-based* analysis computes a page's importance based only on the page content. The content used for analysis includes the page text (Chakrabarti, van den Berg and Dom, 1999), the page title text (Sun, Lim and Ng, 2002), the page URI (Kan, 2004), the page layout (Shih and Karger, 2004). The page text is commonly used because its high precision, other features are also used when page text is not available.

In contrast, *link-based* analysis does not consider the page content alone. It also considers the relationship of this page with other pages. The most widely used link-base analysis algorithms are Page Rank and HITS. These two are usually not helpful in focused crawling, because it is had to analysis the page which the knowledge about the whole graph is not yet completely known. However, with partial information of the search graph, these algorithms could also be used in crawler. An example is (Diligenti, Coetzee, Lawrence, Giles and Gori, 2000) tries to predict the distance between the current page and the target page, and assign relevance score according to the distance.

Focused crawlers also can be differentiated in what link information they use to classify pages: as hard or soft focus mode crawling. In *hard focus mode*, the priority of the fetching document is determined by its ancestors. If any ancestor is marked as good, the document will be put into frontier, which is a queue of documents waiting to be fetched later. Otherwise this document will be discarded. In *soft focus mode*, the relevance of a document is the maximum of all its neighbors' relevance, before this document is actually visited.

2.3 Related works

2.3.1 Crawling using Naïve Bayes Classifier

(Langely and Sage, 1994) introduce a probably simplest classifier for focused crawling, the Naïve Bayes classifier. The analysis module performs content-based analysis which examines the text extracted from a web page. The text is then assigned a score and the score is used to determine the priority in best-first crawling. The text is represented as a vector of words, weighted by word frequency. The relevance score is the cosine similarity between the page and the focused topic, represented as a query:

$$sim(p,q) = \frac{v_p \cdot v_q}{\|v_p\| \cdot \|v_q\|}$$

v_p and v_q are the vector representations of page p and query q. As with standard best-first crawling, at each crawling iteration, the page with highest score is fetched first. In this design, the classifier will only assign high score for pages that are relevant to topic (target pages). Those pages that are not target pages, but in turn lead to target page will be assigned with low priority. This is a critical weakness. Assigning proper relevance scores to those pages, we will be able to avoid more irrelevant pages.

2.3.2 IBM focused crawler

(Chakrabarti et al, 1999) solved this drawback of the Naïve Bayes crawler by introducing a distiller module, which is a link–based analysis algorithm. The *distiller* finds authorities (bridges) which have high derived centrality, and *topic hubs* (which are not target pages, but link to target pages). Hubs are normally link to good authorities and thereby have high reflected centrality. According to (Kleinberg, 1998), important hubs point to important authorities and vice versa.

In IBM focused crawler, the authors assume that the pages belonging to the same topic are mutually reachable. Central to this focused crawler is the representation of the predefined focused topic as canonical topic taxonomy with examples. The user needs to select and refine specific instances in the taxonomy. User also needs to select seed pages as the starting point of the crawl.

The IBM crawler thus contains three main components: the classifier, distiller and crawler. The classifier determines which page to expand based on content relevance. The distiller determines the priority based on centrality. Finally, the crawler dynamically reconfigures its crawl based on the priority assigned by the distiller.

The evaluation of the IBM crawler shows that both hard and soft focused crawling methods perform very well. Their work also validates that focused crawling with a distilling module is quantitatively better.

In this paper, (Chakrabarti et al, 1999) makes two assumptions. Firstly, the relevance of a page is based on the relevance of its neighbors. Secondly, multiple citations from a single document are likely to cite semantically related documents. The first assumption is not always true. For example, in our case, most of the publication pages are linked from the author's homepage, which may not be publication pages. Moreover, publication pages will normally not link to other publication pages, but to non-HTML document files such as MS Word, PDF, or MS PowerPoint.

2.3.3 CORA crawler

The CORA search engine (McCallum et al, 1999) is another approach that investigated how to assign proper priority values to hubs. In order to achieve this goal, they introduce the notion of a Q-value, which is assigned to pages according to the number of target pages found among their neighboring pages.

The CORA search engine is developed to automatically crawl and classify computer science research papers from the Web. Web pages are classified according to their neighbors' anchor text (the text that describes the web page when referencing it), its header, page title and URL. Their design applies a reinforcement learning framework in training the crawler. Target pages are immediate rewarded. They also reward pages between seed pages and target pages.

The training phase involves two tasks: assigning a Q-value to the pages in the training data and learning a mapping from neighbor words to Q-values. The focused

crawler has two models to assign Q-value, immediate reward and future reward. Immediate award assigns 1 to the target pages and 0 to other pages. As a result, all the pages will fall into two categories, target pages and others. At this stage, there is little difference from other focused crawling algorithms. However, in future reward, Q-value is assigned according to the number of rewards obtained from its neighbors. In this way, pages linking to more target pages will get higher Q-value. The pages are discretized into *n* bins and one classifier is trained for each of the *n* bins.

In the crawling phase, these *n* classifiers are used to classify a hyperlink to one of the *n* bins. The crawler retrieves the pages with higher Q-value first as with standard best-first crawling.

There are two drawbacks to this method. First, the radius of crawling is limited to 3-4 levels from the starting pages as neighbor text of URLs that are further than 4 levels is too general to learn. Second, it uses a local search strategy. If a target page is not linked from the seed pages, it is not reachable using this method.

2.3.4 Context Focused Crawler

Diligenti et al, (2000) proposed another approach to assign credit to hubs. It is done by capturing the link hierarchies where target pages occur. The paper also describes how to find off-topic content that occurs in web pages whose neighbors are normally target pages. Properly assigning credit to these documents will differentiate them from other off-topic web pages. When all remaining uncrawled pages are not classified as target pages, the crawler chooses a page with a more related topic first. Here, relatedness is defined as closeness in link distance to a target page.

The context focused crawler is built in layers. The crawler tries to find pages linking to pages in layer *n* using search engine such as AltaVista or Google, and puts them into layer *n*+1 recursively. Target pages are put in layer 0. As a result, all the pages in layer *n* are *n* links away from the target pages. This representation is used to train a set of *n* classifiers (one per layer), to assign documents to, based on the expected link distance to any target document.

The first step in training aims to extract the context, within which the target pages are typically found, and encode this information in a *context graph*. A context graph is built by putting the target example pages in layer 0. After that, a number of pages linking to the target pages can be found with the help of search engine, by using the search engine's index of linked pages. These pages that point to target pages are put as nodes in layer 1 with edges between target document nodes and these current layer 1 nodes. This process repeats until a user-specified number of layers is reached. As a result, an induced graph can be created where each document in layer n+1 is linked to at least one document in the layer n.

In the crawling stage, the crawler retrieves one page at a time and the crawler's classifier module predicts how many steps the current document may take to reach a target document. It assigns the page to the queue corresponding to the most likely layer. Each queue is maintained in a sorted state according to the likelihood score associated with its documents. A queue is built for each layer of the context graph. For example, the classifier for layer 0 determines whether a page is a target page while other classifiers are used to predict how many steps must be taken before a target page is found.

This is the first method that we have reviewed that takes a non-local search strategy during training. However, there are still several drawbacks to this method. First, all the hyperlinks from the same page are weighted equally. This can cause a lot of irrelevant pages to be crawled, for example, hyperlinks from one's homepage may link to personal page, timetable and other information besides his publication page. Second, similar to the CORA crawler, this method generally is limited to the immediate layers near to target pages (again 3-4 levels for efficient crawls) for the same reason: pages further than 3-4 links away are too many and too varied in content for classifiers to learn sufficiently distinct models.

2.3.5 Automatic Publication Data Gatherer

The automatic publication data gatherer (hereafter, APDG; Nguyen Thi, 2005)

follows the method used by CORA search engine (McCallum et al, 1999) and builds upon it. Pages in the path from starting pages to target pages are assigned a priority. All the crawling methods stated above require downloading the pages before they are classified. In order to increase the efficiency of the crawler, Nguyen Thi (2005) proposed a method of assigning credit to intermediate pages using the features extracted from the URL itself. This has a distinct advantage of classifying pages before actual crawling, allowing more fine-grained crawling.

Two knowledge resources were investigated to the enhance performance of the web page classifier module: page text and URLs. First, all the words in the page text were extracted, changed to lower case and stemmed. Stop words and words without alphabetic characters were removed. The remaining words are input to classifier. Page titles and anchor texts are operated on separately. Different from all other focused crawling methods that do not distinguish among the outgoing links, the APDG uses URL information to estimate a classification before downloading, so different pages with same parent node can be ranked.

Compared to all the previous methods, the efficiency is increased significantly because with URL classification, we only download those pages with estimated higher priority while others are discarded. Experimental results in their domain (publication pages; identical to ours) shows this method finds target pages faster and avoids irrelevant page better, outperforming other classification methods. However, it is still a local search algorithm and is sensitive to seed URLs and may not be able to recover all target pages when they are not directly linked from the initial seeds. Our own analysis shows that this method has another drawback: a relatively slow start at finding target pages.

2.3.6 PaSE

Finally, On and Lee (2004) proposed a method to locate online copy of publications effectively, with the help of a search engine. For each citation, they submit its title to Google web search engine, and retrieve the top 10 links. Using

breadth-first crawling, it identifies which document is most likely to be the online copy.

Once arriving at the right web page, the system attempts to select the correct <citation, PDF> pair among a list of publications. To make the problem easier, instead of considering all the fields of citation (i.e., title, author, year, etc), On and Lee only consider the "title" field as they reason that the title has much less probability to be written in different formats. They use a citation matching algorithm (CM) to find the most similar token and the citation block, based on word matching using token based and subfield matching algorithms. The CM algorithm breaks a citation into each field (title, author, year, etc) and compares each candidate citation with the query citation separately. After the citation block is identified, the CM module measures the distance from each citation block to each neighboring PDF or PS document, and picks the one with the shortest distance.

This method improves the accuracy, but their research only tries to retrieve the actual publication document, not the other pages or hubs that lead to them. Another drawback is that they just assume that all the PDF/PS documents are publications (target pages), but this is not always true as sometimes PDF files contains slides or other information.

Chapter 3: Baseline

Our project is built on top of the prior work by Nguyen Thi (2005) in her work on the APDG. In this chapter, we examine the baseline project and show the motivation of our further study. The framework for APDG is shown in Figure 1.

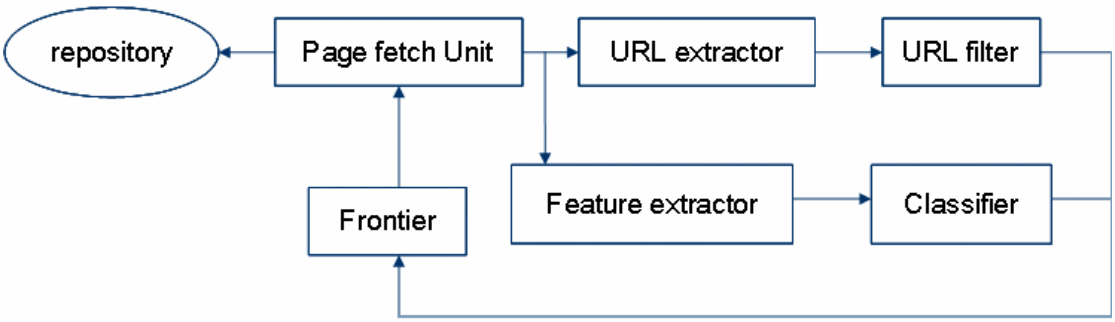

Figure 1: Baseline System

The page fetch unit, URL extractor, URL filter, feature extractor and classifier are the five components of this crawler. Repository and Frontier are built as data storage. The *frontier* contains pages that haven't been visited before. As with standard graph search algorithms, the internal structure of the frontier describes the crawler strategy: whether it is a breadth-first, depth-first structure or best-first search. When content analysis on the web page or URL is performed, the problem usually results in a best-first search, requiring a priority queue data structure for the frontier. The *page fetch unit* removes pages from the frontier one at a time, feeding them to the *URL extractor*, which extracts hyperlinks embedded in the page. A separate *domain filter* is used to remove URLs from other domains outside of the one of interest, identical to our target definition; a *duplication removal filter* is used to remove URLs that refer to the same page.

Furthermore, a *feature extractor* and *classifier* are added to analyze the crawled pages and rank the URLs in the frontier. In the baseline system, only the URL classifier is used: the feature extractor only extracts a set of features based on the URL. In the classification stage, each URL is assigned a priority value. The system assigns a numeric priority value to each of the pages classes as was done in other previous work

of CORA (McCallum et al, 1999). In specific, Nguyen Thi assigns 1.0 to publication pages (target pages). Homepage is assigned with priority 0.9 as they are most likely to be publication pages and most likely to link to publication pages directly. Staff listing pages are most likely to link to staffs' homepages, so they are assigned with priority 0.8. The rule is also applied to department web pages, so they are assigned with priority 0.7. Other pages are assigned to priority of 0. When the page fetch unit gets a page from the frontier, it follow best-first search and always fetches the page with the highest priority among the remaining unvisited URLs. For example, if there is no target page in the frontier, the page fetch unit will look for homepage instead.

3.1 Drawbacks of the Baseline System

Observations of the APDG noted a serious shortcoming in that it cannot exhaust all the publication webpages within a certain domain, as it is a local-search algorithm. It traverses the web by starting with a set of seed URLs, relying on the user to provide high quality seeds and necessitating that from the seed URLs all target pages are reachable. In our case, we start with `www.comp.nus.edu.sg`, and limit the domain to be `comp.nus.edu.sg`. If the target pages have no path linking from the seed URLs, or if the target pages are in the domain but only an indirect link to some other pages in the domain exist, there is no way we can find such target pages using the APDG. This scenario is illustrated in Figure 2.

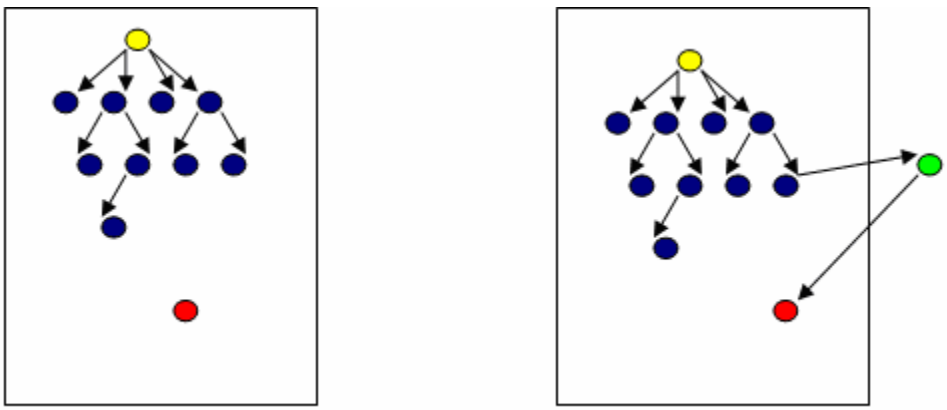

Figure 2: Web Pages Unconnected to the Seed Page

As illustrated in Figure 2, the rectangular box represents the entire domain (e.g., `www.comp.nus.edu.sg`). In the left part, the target page (the red point) has no link path from the seed page (the yellow point). In the right part, the target page has indirect link from the seed page. The only page pointing to the target page is a page (green point) that is out of the domain. As a result, the green page will be caught by the "Domain Filter". Thus the red one will not be visited as well.

3.2 Crawling stages in baseline system

Furthermore, we would like to improve the efficiency of the APDG. From Figure 3, we can see that the crawler starts with a relatively low efficiency in finding publication pages. In period A, the URL links added to the frontier are pages that are several links away from the target pages.

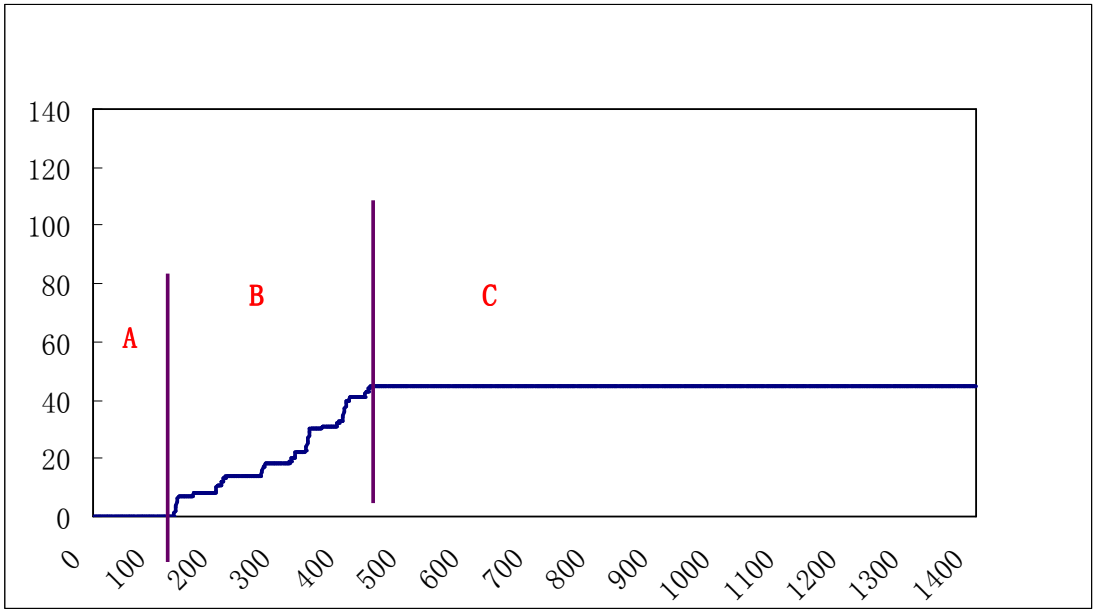

Figure 3: Three Crawling Stages

For example, if we want to find `http://www.comp.nus.edu.sg/~kanmy/pubs.html`, which is a publication page, the shortest path will be as following:

- `http://www.comp.nus.edu.sg/` (seed page)

- `http://www.comp.nus.edu.sg/cs/` (department webpage)
- `http://www.comp.nus.edu.sg/cs/acad_staff.html` (staff_list webpage)
- `http://www.comp.nus.edu.sg/~kanmy/` (staff's homepage)
- `http://www.comp.nus.edu.sg/~kanmy/pubs.html` (publication page)

As more other links may be added within this period of time, it may take a while before crawling proceeds to find the department webpage, staff_list webpage, finally reaching the staff's home page and his publication page. This explains the low efficiency in period A.

During period B, crawler has already reached each staff's homepage, so the publication pages are found at a faster rate. This is due to that some staffs' homepages are publication pages themselves. Others will have a direct link to their publication pages.

During period C, a very large amount of URLs are present in the frontier. Thus the number of URLs with high priority increases due to the total number increases. Since high priority may not direct indicate publication page, page fetch unit may pick a page that we predict to be target page, but actually not target page. This will make crawler waste time on many irrelevant pages. Therefore the rate will decrease again in this phase.

Improving the accuracy and efficiency are two major motivations for us to upgrade the system. On one hand, we want to find a way to get target pages that are inside the search domain but not directly linking from seed pages. On the other hand, we want to improve the system efficiency by helping the crawler find more target pages or good hubs during periods A and C, as mentioned above.

Chapter 4: Augmenting Focused Crawling using Search Engine Queries

In our project, we develop a new focused crawler called the Publication Spider, or PS. This program is built on top of the previous APDG work by Nguyen Thi (2005), and specifically integrates new work to address the shortcomings of previous methods reviewed in Chapters 2 and 3.

4.1 Design

When we do experiment analysis the Frontier, we found that at the starting stages, the number of pages classified as publication pages periodically become zero. This reflects the situation as shown in Figure 2 in period A. Moreover, after iteration in which the page fetch unit gets a page from the frontier, the APDG extracts the embedded hyperlinks, classifies them and add them back to the frontier. We analysis the frontier again, it shows that there are significantly more non-target pages than target ones added in.

To solve these problems, we need to find another way to add more relevant pages to the frontier at this stage. The following is what we do. If the percentage of target pages in the frontier keeps decreasing, or there is no target page in frontier, local search is no longer productive. In the PS system, we augment the APDG by introducing a global method. It finds additional publication pages by using a search engine and adds these paegs to the frontier. This effectively introduces a mechanism to achieve the spidering configuration as shown in Figure 2 period B, in which pages non-connected from the initial seeds can be found via external links. We invoke this global PS crawling mechanism to add more publication pages directly to the frontier by extracting the keywords from a visited page which the URL-classifier predict it as publication page, and getting result using search engine query. This global PS system is only invoked with the percentage of estimated target pages harvested drops over a

period of α crawling iterations.

We have done several experiments varying α and have set α empirically to 10. In other words, we invoke the Publication Spider system after the percentage of estimated harvested publication pages decreases for 10 consecutive iterations. To explain our setting of 10, we offer two main justifications. First, we believe that some publication pages do not mutually link to / are not reachable from each other. Thus, when we fetch a publication page from the frontier and extract its hyperlinks, we may not find URLs for other publication pages. Second, in the PS system, we also use top 10 results returned from the search engine results (in our case we use Google's Query API via SOAP). If adding these 10 hits does result in the URL classifier deciding that at least one is a publication page, we need to make an additional query.

We also do not use the PS immediately when there is no estimated target page on the frontier. When there is no target page in the Frontier, we will treat this situation as one decrease in percentage for target pages. This situation occurs at the starting stage, when we have not found any estimated target page yet. In such case, there is no way we can generate search queries. However, once one page is estimated to be a target page by the classifier it can be used to generate queries in the PS system.

Our experimental results also indicate that it is better to use percentage instead of the total number of estimated target pages found in the Frontier to measure the system performance. We find that after several crawling steps, the number of pages with high priority increases with the increase of the size of the frontier. Therefore, an increase in the number of estimated target page does not indicate high performance of crawling, especially if the classifier's judgment is not accurate. Efficiency may also decrease in this case as a high estimated priority does not guarantee a true publication page. Instead, the percentage of estimated publication pages in the frontier is a better measurement of system efficiency, as the total number of pages in the frontier is taken into consideration. Furthermore, measuring percentage of estimated publication pages can also reflect among all the URLs extracted from the page we fetched, how many are estimated as publication page. This will reflect the quality of pages inside the frontier.

Furthermore, we retrain the classifier to obtain better performance. In this project, we remove the domain name and only classify the rest part of the URL. Because here we are only interested in pages inside one certain domain, the domain name is no longer a feature to distinguish pages. Remove this feature will make the classifier focuses more on other features when classifying one page.

4.2 Framework

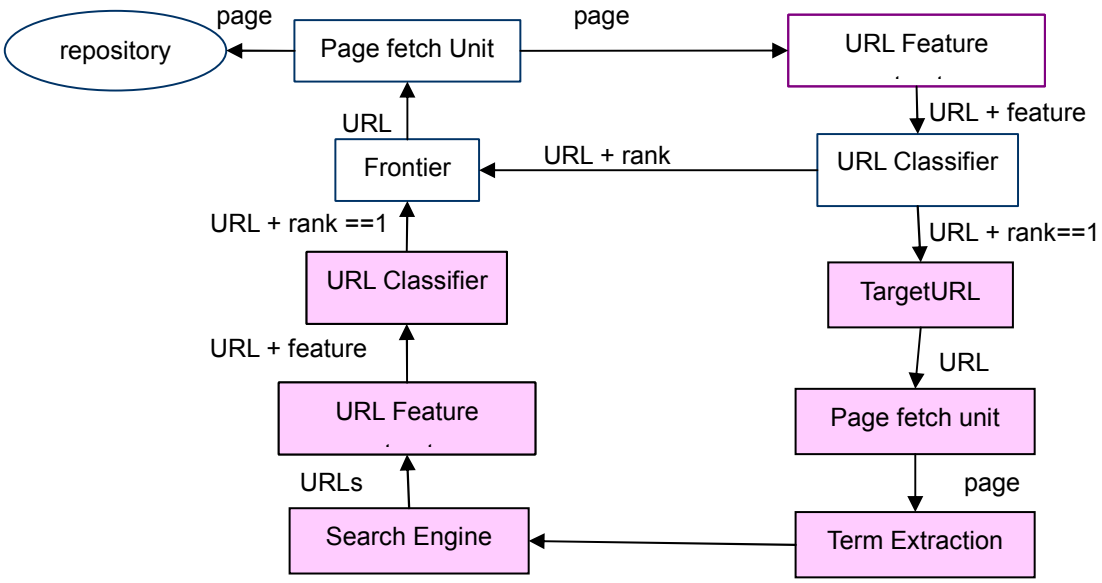

Figure 4: APDG with Publication Spider modules

The general framework of our improved work is shown in Figure 4. We add two components highlighted in purple to the APDG. We add a targetURL to store all the URLs of the estimated target pages, and the Term Extraction module is used send appropriate queries to the search engine. Our Publication Spider module consists of the following steps:

In the crawling stage, all the URLs classified as URLs of publication pages will be stored into the targetURL immediately.

When the percentage for publication pages in the frontier has decreased for 10 consecutive iterations, the PS system will fetch a page using the last URL we put in the targetURL.

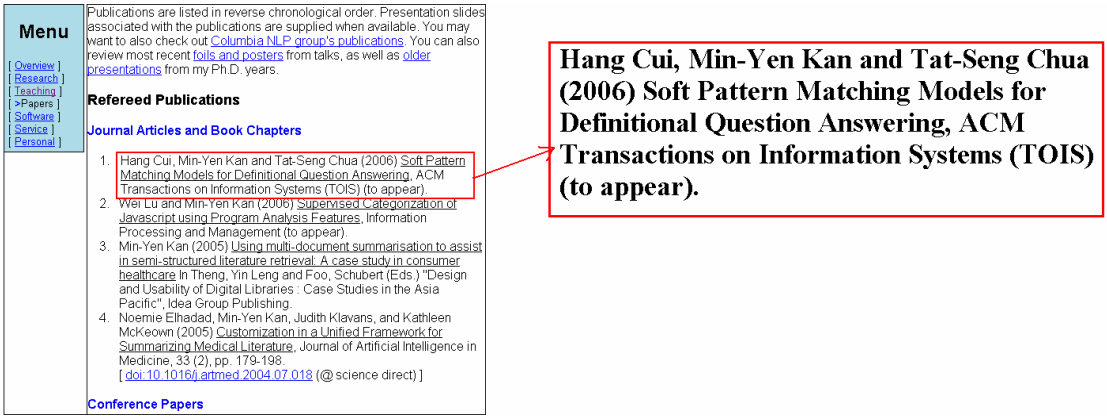

Figure 5: List Item in Publication Page

We assume almost all the publications on a publication page are listed in bulleted form or numbered form. This assumption is made based on our manual observation of many publication pages. Using the pages we get in last step, we extract all text in bulleted and numbered environments. Each item is deemed as containing candidate publication information, and is saved in a string as shown in Figure 5. We attempt to extract the title of the publication using a set of different regular expressions. It is important to note here that while regular expressions have problems in identifying infrequent patterns (i.e., their recall is low), the goal of these expressions is to discover terms for querying and the system does not require an exhaustive list of terms.

If we just query the search engine for the web pages with the title string itself, we will likely get only same page which was used to obtain this title string, as it is quite rare for a title to be reproduced in two different publications in a single domain. Thus we need to break the titles into shorter, more usable chunks for the querying process. We use the top 40 most frequently used words (listed in complete form in Appendix A) to separate the title into series of tokens. Each piece contains partial information about this publication. For example, as shown in Figure 5, we get the title of the publication "Soft Pattern Matching Models for Definitional Question Answering". After splitting, we get two strings: "Soft Pattern Matching Models" and "Definitional Question Answering". Using the string "Soft Pattern Matching Models",

the system can obtain all the applications done using this model and similar models. Using the string "Definitional Question Answering", publications solving this problem using different techniques can be found.

These queries are sent to Google via its Google SOAP Search[1]. The Google SOAP Search API service enables software developers to query billions of web pages directly from their own computer programs. We query the search engine for relevant search results for each resulting strings, observing the following two restrictions. First, we restrict the search domain to be `comp.nus.edu.sg`. Second, we restrict maximum number of queries generated from one page to 10. The Publication Spider program is built to help and guide the baseline APDG system in order to improve its performance. This setting has been set empirically through experiment trials. Adding too many search results tends to overwhelm and entirely replace the functionality of the baseline system, leading to poorer results.

The URLs for the top 10 results are returned by the Google SOAP search and are passed into the URL filter and classifier before finally placing them into the frontier.

4.3 Discussion on improvements to the Baseline System

First, during the starting stage (period A), Automatic Publication Data Gatherer suffers from linking steps before finding publication pages and staff's homepages. It can be avoid by calling our Publication Spider. The PS system can help the crawler proceed directly to retrieve target pages if at least one crawled page is estimated to be a target page. The system put these pages into the frontier, so that it can directly jump

[1] http://www.google.com/apis/index.html. On a technical note, the Google SOAP Search sets the daily access limit to 1000 times, so we add a hash table to cache the results for each query together with the results. We will check the cache each time before we send the query to conserve bandwidth.

over the linking steps.

Second, some of the pages may not link from the seed pages, therefore they cannot be found using the baseline Automatic Publication Data Gatherer. After we add the Publication Spider program, the system has a greater chance of finding them through the search engine query process.

Chapter 5: Evaluation

We now evaluate whether the APDG with our added PS modules are able to correct for the shortcomings of the original APDG baseline system, and assess the difference in performance among the two systems. Our hypotheses are that the addition of the PS modules are able a) to improve the specific weaknesses in the target harvesting rate graph in Figure 3 of the baseline system, and that they lead to b) an overall improvement in harvesting rate and c) better final recall. To validate these hypotheses, our evaluation consists of consists of two halves: the first part to determine precision and harvesting rate and the second to address recall.

5.1 Experiment on precision

We compare the baseline and the improved system using same input data. Figure 6 below shows the target page harvesting rate of our PS system superimposed on the same graph as in Figure 3. For efficiency reasons we have set the maximum page fetch limit to 1,400. The graph indicates how many true target pages each system finds. For both systems, a single seed URL (http://www.comp.nus.edu.sg/) is used and the domain restriction is set as `comp.nus.edu.sg`. Both systems use the URL classifier for classification.

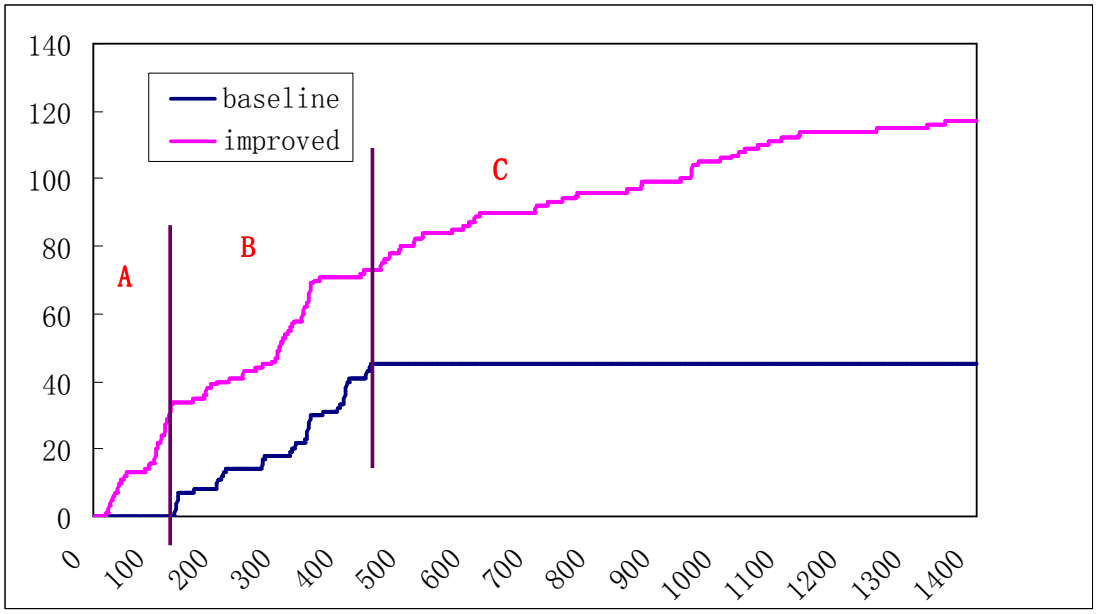

Figure 6 Comparison between baseline system and the improved system

Figure 6 shows the upgraded system has found more actual publication pages than the baseline system.

The upgraded system is able to detect and avoid situation when there is no publication pages in the frontier. Similar as what we shown in Figure 3, we divide the crawling stage to three periods. During period A, Publication Spider is invoked as we discuss in chapter 4, and add estimated target pages directly into the Frontier. In this way, we avoid the slow start stage. During period B, both baseline system and the upgraded system perform quite well. During period C, in the improved system, if the percentage of pages classified as publication pages in the Frontier keeps decreasing, Publication Spider program is able to find publication pages at a constant speed.

The following table lists the improvement in the precision in the upgraded system. In this table we list the number of publication pages baseline system and upgraded system is able to find. The first column is the total number of pages crawled, with an increment of 100 pages. The second and third column shows the number of publication pages each system is able to find. After that, we calculate the precision for each system at every crawling stage. And calculate the improvement we made.

URL crawled	Baseline	Improved	PrecBase	PrecImp	Improvement
0	0	0	-	-	-
100	0	20	0.00%	20.00%	Infinity
129	1	34	0.78%	26.36%	3400%
200	11	40	5.50%	20.00%	364%
300	18	53	6.00%	17.67%	294%
400	38	71	9.50%	17.75%	187%
500	45	80	9.00%	16.00%	178%
600	45	87	7.50%	14.50%	193%
700	45	90	6.43%	12.86%	200%
800	45	96	5.63%	12.00%	213%
900	45	99	5.00%	11.00%	220%
1000	45	106	4.50%	10.60%	236%
1100	45	112	4.09%	10.18%	249%
1200	45	114	3.75%	9.50%	253%
1300	45	115	3.46%	8.85%	256%
1400	45	117	3.21%	8.36%	260%

Table 1: improvement in precision

From the table we can see after crawling 100 pages, the baseline system cannot find even one publication page, instead, the upgraded system is able to find 20 publication pages. When the baseline system finds its first publication page, the upgraded system has already find 34 publication pages. Using these numbers we can show that the upgraded system is able to avoid the slow start. Moreover through the whole process, the improved system is discovering publication pages at almost constant speed. After crawling 500 pages, the baseline system is not able to find more publication pages, but the upgraded system keep finding publication pages with almost constant speed. Through all the crawling process, the improved system finds publication page at double speed of the baseline.

5.2 Estimated Recall

As extensive coverage of scientific papers is especially critical in scholarly

publication gathering systems, recall of target pages is very important. In this second half of the evaluation we would like to ascertain whether the upgraded system is able to retrieve all the publication pages within a restricted computer domain. However, unless we manually examine each page within an entire domain we will not be able to calculate the exact recall. Even for a fairly small domain such as in the School of Computing, this is infeasible as there are over 2 million web pages that are accessible through the School's web server[2]. Instead of true recall figures, we try to calculate a reasonable estimate of the total number of publication pages by another means.

One way is to note that publication pages are almost always created and maintained by individuals. Also note that an estimate based on this ignores publication pages that represent an entire research group. If we have census data on how many individuals are in a domain and the average number of publication pages per individual on average, we can extrapolate a reasonable estimate of the total number of publication pages. We also note that academic staffs normally have publication pages linking from their homepage, but graduate students do not always have a publication page, and very few administrative staffs may have them. So we decide to ignore the administrative staffs, estimate the number of publication pages by academic staffs and graduate students separately, and add up the number to get the total number of publication pages. We will give a brief description of our calculation here. The detailed data will be provided in the CD.

For our specific experiment on the School of Computing (SoC), we obtained a list of all graduate students and staff from enrollment data[3]. The total number of full-time graduate students in SoC is 114. We randomly selected 28 students (about one quarter of the full time students), and browsed their home pages. Only 10 have publication pages. From this, we estimate that full-time graduates will contribute $114 * \frac{10}{28} \approx 41$ publication pages in the `comp` domain. Similarly, the total number of part-time graduate students in SoC is 83. Of a randomly selected 21 students, 7 have a

[2] As estimated by Dr Min-Yen Kan during his crawl of the www.comp.nus.edu.sg domain.

[3] Courtesy the Graduate Office

publication page, leading to $83 * \frac{7}{21} \approx 28$ publication pages. Teaching Assistants and Research Assistants are already counted in this part-time graduate student count. Finally, we browsed all the 105 academic staffs' home pages, in which a total of 100 publication pages are contributed. Thus the estimated total is 100 + 41 + 28 = 169 publication pages.

URL crawled	Baseline	Improved	RecallBase	RecallImp	Improvement
0	0	0	-	-	-
100	0	20	0.00%	11.83%	infinity
129	1	34	0.59%	20.12%	3400%
200	11	40	6.51%	23.67%	364%
300	18	53	10.65%	31.36%	294%
400	38	71	22.49%	42.01%	187%
500	45	80	26.63%	47.34%	178%
600	45	87	26.63%	51.48%	193%
700	45	90	26.63%	53.25%	200%
800	45	96	26.63%	56.80%	213%
900	45	99	26.63%	58.58%	220%
1000	45	106	26.63%	62.72%	236%
1100	45	112	26.63%	66.27%	249%
1200	45	114	26.63%	67.46%	253%
1300	45	115	26.63%	68.05%	256%
1400	45	117	26.63%	69.23%	260%

Table 2: Improvement in recall

Table 2 shows the comparison of recall of baseline system and the upgraded system. Similar to Table 1, the first three columns shows the total number of pages we crawled, the number of publication pages found by the baseline system and the upgraded system. The following two columns show the estimated recall of the baseline system and the upgraded system. We calculated the recall using the number of pages each system is able to find, divide by the total number of publication pages as we estimated just now. The recall of the upgraded system is about twice higher than the baseline system in the whole crawling process. After crawling 1400 pages inside SOC domain, the baseline system is able to get 45 publication pages, the recall is

26.63%. The upgraded system is able to get 117 publication pages. The recall is 69.23%.

The comparison of the precision and recall of the baseline system and upgraded system at the end of our experiment is shown in Table 3 below. After crawling 1400 pages, the precision of the baseline system is 3.21% where the upgraded system has a precision of 8.36%. The recall of the baseline system is 26.63%, compare to 69.23% for the improved system. It shows that the upgraded system outperform the baseline system significantly in both precision and recall. We also calculate the F1 Score for each system. The F1 score is a statistical measure of a test's accuracy. It is defined as 2pr/(p+r), where p is the precision and r is the recall of the test. The F1 value shows that the upgraded system performs about three times better than the baseline system.

	Baseline system	Upgrade system
Publication pages found	45	117
precision	3.21%	8.36%
recall	26.63%	69.23%
F1	0.057	0.149

Table 3: comparison of baseline system and upgraded system

Chapter 6: Conclusion and Future Work

In this project, we aim to improve the performance of focused crawling using search engine. We investigate in a baseline system (APDG) which is focused crawler with URL-classifier, and improve its performance in terms of precision and recall. There are two problems with the baseline system. First, APDG suffers from a slow start and low harvesting rate at some stages. Second, with the local search method, APDG is not able to exhaust all the publication pages under a restricted computer domain.

We propose a method to enhance the precision and recall of focused crawling by using Search Engine queries. Refer to Figure 6, in phase A, we will add pages directly into the frontier in order to guide the crawler jumps over the linking steps and avoid the slow start. In phase C, the improved system is able to detect the situation when the baseline system is failing to detect more publication pages, and assist the crawler to avoid such situation. With the help of search engine, the upgraded system is able to get more external pages, lessening the reliance of the seed page. The evaluation results show that the upgraded system improve the baseline system significantly in both precision and recall.

However, the evaluation is only limited to School of Computing domain only. Future work might expand the evaluation to other web spaces. Also, instead of using Google result in this project, other search engine might be tried for obtaining better performance. In this project, we use the title of publication to form search engine queries, better ways (i.e, include the publisher's information) to form queries may be investigated in the future work.

Bibliography

Chakrabarti, S., van den Berg, M. and Dom, B. (1999). Focused crawling: a new approach to topic-specific Web resource discovery. Computer Networks (Amsterdam, Netherlands: 1999), 31(11-16):1623-1640, 1999.

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. and Gori, M. (2000). Focused crawling using context graph. In Proceedings of the 26th VLDB Conference, (Cairo, Egypt, 2000.) pp. 527-534.

Kan, M.-Y. (2004). Web page classification without the web page. In Proceedings of the 13th International World Wide Web Conference (WWW2004), (New York, USA, May 2004.) pp. 262-263.

Kleinberg, J. (1998). Authoritative sources in a hyperlinked environment. In Proceedings of 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 668-677.

Langely, P. and Sage, S. (1994). Induction of selective Bayesian Classifiers. In Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, (Seattle, WA, Morgan Kaufmann, 1994.) pp. 399-406.

Lawrence, S. (2001). Online or invisible? Nature, Vol. 411, No. 6837, Jan 2001, p.521.

McCallum, A., Nigam, K., Rennie, J. and Seymore, K. (1999). Building domain-specific search engines with machine learning techniques. Proc. AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace, 1999.

Nguyen, H.O.T. (2005). Automatic Publication Data Gatherer. UROP report, National University of Singapore, Singapore; 2005.

On, B. and Lee D.(2004). PaSE: Locating Online Copy of Scientific Documents Effectively. In Proceedings of the 7th International Conference of Asian Digital Libraries (ICADL), (Shanghai, China, December 2004.) pp. 408-418.

Sun, A., Lim, E.-P. and Ng, W.-K. (2002). Web classification using support vector machine. In Proceedings of the fourth international workshop on Web information and data management, ACM Press, 2002, pp. 96--99.

Shih, LK. and Karger, D.(2004). Using URLs and table layout for web classification tasks. In Proceedings of the 13th International Conference on the World Wide Web, (New York, NY, 2004.) pp. 193—202

Appendix A - Top Forty Frequently Used Words

The Top Forty Frequently Used Words are provided by a joint effort of the UC Berkeley and Stanford Digital Library projects.

Terms are sequences of one or more of the characters from the following set:

0 to 9, A to Z, a to z

ASCII 150-160, 170, 181, 186, 192-214, 215-246, 248-255

Terms are terminated when the first non-term character is encountered. The non-term character is treated as a term break. The next term starts with the next valid term character. Thus, character strings such as "it's" are tokenized as two terms, "it" and "s".

We count the number of documents in which each term appears to compute that term's **document frequency**.

Terms	Frequency
the	65002930
a	62789720
to	60857930
of	57248022
and	54078359
in	52928506
s	50686940
for	49986064
on	45999001
this	42205245
is	41203451
by	39779377
with	35439894
or	35284151
at	34446866
all	33528897
are	31583607
from	30998255
e	30755410
you	30080013

Terms	Frequency
Be	29669506
that	29417504
not	28542378
An	28162417
As	28110383
home	28076530
It	27650474
I	27572533
have	24548796
if	24420453
new	24376758
t	24171603
your	23951805
page	23875218
about	22292805
com	22265579
information	22107392
will	21647927
can	21368265
more	21367950