A Web Meta-Search Engine

Using a Ranking Algorithm Based on a Markov Model

Submitted to Committee Members

Dr. Wen-Chen Hu (Chair)
Dr. Gerry V. Dozier
Dr. Dean Hendrix

by

Fangyan Du

In Partial Fulfillment of the
Requirements for the
Degree of
Master of Computer Science and Software Engineering

Auburn University

May 12, 2001
TABLE OF CONTENTS

LIST OF FIGURES ................................................................................................................................. 4

LIST OF TABLES ..................................................................................................................................... 5

CHAPTER 1 INTRODUCTION .................................................................................................................. 6

CHAPTER 2 LITERATURE REVIEW ......................................................................................................... 9

2.1 Current Web search services .............................................................................................................. 9

2.2 Limitations of current search services .............................................................................................. 12

2.3 Meta-search engines .......................................................................................................................... 15

2.3.1 The structure of the meta-search engine ....................................................................................... 15

2.3.2 The pros and cons of meta-search engines ................................................................................... 17

2.4 Ranking algorithms ........................................................................................................................... 19

2.4.1 Traditional ranking algorithms ................................................................................................... 20

2.4.2 Mining the linkage structure ....................................................................................................... 22

2.4.3 User feedback ............................................................................................................................... 23

2.4.4 A ranking algorithm based on Markov model .............................................................................. 23

2.5 The search services used in this prototype ...................................................................................... 24

CHAPTER 3 SYSTEM DESIGN AND IMPLEMENTATION ......................................................................... 28

3.1 System overview ............................................................................................................................... 28

3.1.1 System structure .......................................................................................................................... 28

3.1.2 Environment settings .................................................................................................................. 29

3.1.3 An overview of the programs ...................................................................................................... 31

3.2 User interface .................................................................................................................................... 31

3.3 The dispatcher ................................................................................................................................... 35

3.4 Result integration .............................................................................................................................. 37

3.5 The storage system ........................................................................................................................... 39

3.5.1 The database ............................................................................................................................... 40

3.5.2 The spider ...................................................................................................................................... 40

3.5.3 The result page retrieval .............................................................................................................. 43

CHAPTER 4 A WEB PAGE RANKING ALGORITHM BASED ON A MARKOV MODEL ....................... 45
List of Figures

Figure 2.1 The claimed coverage of major search engines........................................13

Figure 2.2 The architecture of a typical meta-search engine.....................................17

Figure 3.1 The system structure of the proposed meta-search engine........................30

Figure 3.2 The user interface of the proposed meta-search engine..............................32

Figure 3.3 The flowchart of the doGet () method of MetaSearchEngine class.............34

Figure 3.4 The flowchart for the combination of two result vectors, v1 and v2.............38

Figure 3.5 The interface to initiate the Web page crawling and downloading process......41

Figure 5.1 A sample result page....................................................................................50

Figure 5.2 The overlap of our results with those of the five underlying search services..51
List of Tables

Table 2.1 The percentage of dead links and type-400 errors occurring in the query results of selected search engines ................................................................. 14

Table 2.2 A comparison of the features of the selected search engines or directories ....... 25

Table 3.1 Programs written for the prototype meta-search engine .............................. 32

Table 3.2 Possible hyperlinks in HTML files .......................................................... 44

Table 4.1 A sample $u_{ij}$ matrix, $p_{ij}$ matrix, the ultimate distribution vector $P_{\text{limit}}$, and the ranked list of the query “auburn university” ................................................................. 48

Table 5.1 The top five result URLs with their relevance scores ................................ 53

Table 5.2 The ranked places of URL http://www.auburn.edu .................................. 55
Chapter 1 Introduction

The explosive growth of Internet resources has led to an overwhelming amount of available information. According to some statistics, this information will nearly double every year. As the amount of information continues to grow, so does the complexity of finding and retrieving it. Meanwhile, the very nature of Web pages, particularly their diversity, the constant and continuous changing, the lack of a standard structure, and the lack of organization, make searching the Web a challenging task.

Current search services designed to retrieve the Internet information include search engines and directories. Tremendous efforts have been put into developing sophisticated searching and ranking technologies. Unfortunately, search engines possess a number of deficiencies including poor precision, low coverage of the Web, out-of-date databases, a large number of dead links, inconsistent user interfaces, and difficulties with spamming. Meta-search engines have thus been introduced to overcome some of these difficulties.

Meta-search engines collect results by querying a selection of search engines or directories in parallel, and display an integrated result list to users in a uniform format. It thus solves users’ problem of “which search service to use when,” and improves the coverage of the entire Web. However, the success of a meta-search engine depends heavily on the indexing and searching capabilities of the underlying search services. To deal with such problems, the actual HTML pages are downloaded and analyzed at the time of searching to improve the result precision and eliminate dead links. This idea was originally introduced in the Inquirus from NEC Corporation.
The core of a search engine is the quality of the results it generates. Because of their giant indexes, most of the search engines will return thousands or millions of relevant documents for a particular query, only a few of which are valuable to a particular user. Efficient ranking algorithms are then urgently needed. Traditional relevance ranking is usually based on the number times the query terms occur in the documents, their locations, the importance of each term, the relative proximity, and the size of the documents. Some search engines rank the results according to feedback from users such as user popularity or relevance feedback. Recently, some ranking algorithms, like Google’s PageRank and IBM’s Clever, concentrate on mining the lineage structure of the Web. Such algorithms place their emphasis on the anchor text and page authority.

In this project, we developed a prototype meta-search engine which dispatches user queries to multiple search services and than displays an integrated result list to the user. This meta-search engine is composed of a user interface, a dispatcher, result integration, and a storage system. It is implemented with Java technologies, including Java application, servlet, and Java server.

In this prototype, we retrieve the most recent information by downloading and analyzing a portion of the collected results at the time of searching. In this way, the result ranking and display is based on the information from the actual HTML pages; And some of the dead links can be detected and removed. In addition, a storage system is incorporated to improve the performance since some of the results that are already included in the database can be reused. The storage system is initially constructed by a spider program. It is
expanded every time a search is conducted. It is also updated frequently by the spider to index the most recent information available on the Web.

To improve the quality of the results, an efficient ranking algorithm is implemented to rearrange the order of the results before they are presented to users. This algorithm, suggested by Zhang and Dong, abstracts the action of users’ surfing the Web as a Markov model. It is multidimensional because it combines the traditional relevance ranking, the linkage structure of the Web, and the degree of result page differences into one relevance-score using four metrics (relevancy, authority, integrativity, and novelty). It is dynamic because the four parameters used to represent the metrics can be customized and adjusted according to users’ specific needs.

This paper is organized as follows. Related works about Internet searching and result ranking will be introduced in Chapter 2. In Chapter 3, the prototype meta-search engine built on several search engines is presented in detail. The ranking algorithm based on the Markov model is introduced in Chapter 4. Then, some queries are tested and the performance of the proposed meta-search engine is evaluated in Chapter 5. Finally, the advantages and drawbacks of this prototype meta-search engine are discussed and suggestions for future work are proposed in Chapter 6.
Chapter 2 Literature Review

In this chapter, we will present a brief review of current Web search services and ranking technologies.

2.1 Current Web search services

The Web has created new challenges for information retrieval. It has been grown rapidly ever since its emergence. Early 2000, Inktomi and the NEC Research Institute, Inc. completed a new study that verifies the Web has grown to more than one billion unique pages. This is up significantly from the estimated 320 million pages found in December 1997. However, even this may underestimate the actual size of the Web. In June 2000, BrightPlanet has uncovered the so-called “deep Web” – a vast reservoir of public information that is 400 to 550 times larger than the commonly defined surface World Wide Web. Furthermore, Cyveillance Web Study showed that the Internet is growing at an astounding rate with roughly 7.3 million unique pages being added every day, and predicted that the Internet will double in size by early 2001. With the explosive growth of the Web, it is becoming increasingly difficult for users to collect and analyze Web pages that are relevant to a particular topic.

The diverse nature of the Web also makes searching it frustrating. Compared to the traditional text-document collections, the whole set of Web pages lacks a unifying structure and shows far more variations in authoring style and content. Meanwhile, the Web is constantly changing with new URLs added and old pages discarded or modified every day. This level of complexity makes it hard to establish any type of bibliographic control.
Since the growth of the Web is exponential and bibliographic control does not exist, two basic approaches have been developed to help users locate Web resources which will be useful to them -- search engines and directories. Each search service is a general class of programs that search the documents on the Internet for specified keywords and return a list of relevant Web pages where the keywords were found.

**Search engines.** Search engines are automatically generated database systems designed to index Internet addresses and other information of Web page files. Search engines, such as AltaVista, Google, and Northern Light, compile their own searchable databases on the Web. A typical search engine consists of three essential components: the crawler, the index, and the search and ranking software. The crawler is a special program that roams the Internet, scans various Web sites, retrieves a copy of the Web pages it visits, and adds specific information to an index automatically at regular intervals. The index refers to the database of Web pages maintained by the search engine. General-purpose search engines usually have huge databases which hold information in an organized manner, trying to cover larger portions of the Web. When a user submits a search query, the search software goes through the index to find Web pages with keyword matches and ranks the resulted pages in terms of relevance.

**Meta-search engines.** Meta-search engines do not compile databases. Instead, they scan the databases of multiple sets of search engines in parallel and combine the results at a single site and using the same interface. Examples include MetaCrawler, Dogpile, and Mamma. Some search engines, such as All-in-One, Beaucoup, and Proteus, are rather “pseudo-meta-search engines” or “one-stop-shop.” They are actually a collection of different search engines
packed in one site, or a drop-down menu that let users choose among a list of search engines. These search engines do not integrate the search results from each search engine into a uniform format.

**Directories.** Directories are manually administered database systems. They work with descriptions of Web pages submitted either by Webmasters or editors who have reviewed the pages. The editors review and select sites for inclusion in their directories on the basis of previously determined selection criteria. The resources they list are usually annotated. Directory editors typically organize directories hierarchically into browsable subject categories and sub-categories. Directories allow users to click through several subject layers to get to an actual Web page. They can also respond to queries by searching their repositories of descriptions. Subject directories are best for browsing and for searches of a more general nature. Some of the best-known directories are Yahoo, Snap, LookSmart, and Magellan. While both search engines and directories have elements in common, such as the ability to search the database, Boolean expressions, and advanced features, the primary distinction between them lies in how the search services obtain their data. A directory does it manually and a search engine does it automatically.

**Hybrid search engines.** As the Web search services evolve, the line between subject directories and search engines is blurring. Most subject directories have added search engines to query their databases, while search engines are acquiring directories or creating their own. This has led to the creation of hybrid search engine such as Aeiwi and MILK (Multilingual Indexing based on Lexical Knowledge).
Portals. Lots of search services, including AltaVista, Excite, Msn, NBCi.com, and Yahoo, are turning their Web sites into portals. Portals are Web home bases from which users can access a variety of services, including searches, e-commerce, stock quotes, weather forecasts, travel information, e-mail and chat rooms.

2.2 Limitations of current search services

Tremendous efforts have been made to improve the quality and efficiency of search engines over the years. However, relying on any of them is insufficient. The main problems are: poor precision and ranking, low coverage of the Web, out-of-date database, low overlap of the results, inconsistent and inefficient user interfaces, and difficulties with spamming technologies. These are explained in more details in the following text.

Poor precision and relevancy ranking. The diverse nature of Web documents, and the focus of Web search engines on handling relatively simple queries very quickly, leads to search results often suffering from poor precision. Any search may return an abundance of both related and unrelated information. The specialty search engines are likely to generate more relevant results quickly in some areas. However, they are inadequate for most of other topics. Additionally, the practice of “search engine spamming” has become popular, whereby Web builders add possibly unrelated keywords to their pages in order to alter the ranking of their pages. Most of the time, the relevance of a particular page is obvious only after waiting for the page to load and finding the query term(s) in the page.

Limited coverage. Studies indicate that the coverage of any search engine is surprisingly low and the relative coverage to the estimated size of the publicly indexable Web has decreased
substantially. It was estimated that no engine indexed more than about 16% of the estimated size of the publicly indexable Web in 1999\textsuperscript{9}, and this proportion continues to decreases. Figure 2.1 shows the recent claimed numbers (in millions) of Web pages that have been indexed and included in the various search engines' databases\textsuperscript{10}.

![Bar chart showing millions of Web pages indexed by various search engines.]

**Figure 2.1** The claimed coverage of major search engines. Sizes are reported by each search engine as of November 1, 2000. GG=Google, FAST=FAST, WT=WebTop.com, INK=Inktomi, AV=AltaVista, NL=Northern Light, EX=Excite, and Go=Go (Infoseek). The extended bar for Google stands for web pages not really crawled. The extended bar for Inktomi stands for web pages in a second database.

**Limited overlap between engines.** Statistical analysis shows little overlap (less than 60%) between the major search engines at the page level regardless of database growth\textsuperscript{11,14}. Submitting similar queries to several search engines may result in widely different sets of documents, with only a few duplicates, and the majority of pages were found by only one search engine. This has led to the suggestion that if users are unable to obtain satisfactory results from one search engine, they may try a different engine.
Out of date databases. Centralized search engine databases are always out of date. It is important to remember that search engines are actually searching a portion of the Web captured in a fixed index created at an earlier date, rather than the entire Web as it exists at the moment the query is posted. There is often a significant time lag between the time when new information is made available and the time that it is indexed. However, the Web is constantly changing and indexing of new or modified pages by even one of the major search engines can take months. Based on a recent analysis, there are still a substantial number of invalid or dead links occurring in the result lists, though most of the major search engines have made improvements. Table 2.1 presents the percentages of dead links and type-400 error messages in some popular search engines.

Table 2.1 The percentage of dead links and type-400 errors occurring in the query results of selected search engines (data from February, 2000).

<table>
<thead>
<tr>
<th>Search engine</th>
<th>% of dead links</th>
<th>% of type-400 errors only</th>
</tr>
</thead>
<tbody>
<tr>
<td>AltaVista</td>
<td>13.7%</td>
<td>9.3%</td>
</tr>
<tr>
<td>Excite</td>
<td>8.7%</td>
<td>5.7%</td>
</tr>
<tr>
<td>Northern Light</td>
<td>5.7%</td>
<td>2.0%</td>
</tr>
<tr>
<td>Google</td>
<td>4.3%</td>
<td>3.3%</td>
</tr>
<tr>
<td>HotBot</td>
<td>2.3%</td>
<td>2.0%</td>
</tr>
<tr>
<td>Fast</td>
<td>2.3%</td>
<td>1.8%</td>
</tr>
<tr>
<td>MSN Inktomi</td>
<td>1.7%</td>
<td>1.0%</td>
</tr>
<tr>
<td>Anzwers</td>
<td>1.3%</td>
<td>0.7%</td>
</tr>
</tbody>
</table>

Difficulties with Web spamming. The crawler-based search engines are facing a serious problem: the practice of “search engine spamming” has become popular. Examples of spamming include excessive repetition of a keyword in a page, optimizing a page for a
keyword which is unrelated to the contents of the site, using invisible or tiny text, etc. This has forced most search engines to develop sophisticated search software and spamming detection filters and will penalize a page that uses spamming.

*Unequal access.* Search engines are typically more likely to index sites that have more links to them (more “popular” sites). They are also typically more likely to index US sites than non-US sites, and more likely to index commercial sites than educational sites.

*Other problems.* The engines may be limited by network bandwidth, disk storage, computational power, or a combination of three.

### 2.3 Meta-search engines

It is believed that no single search engine is likely to find more than 45% of the relevant pages and users often need to switch from one search engine to another to locate the desired information. The availability of an abundance of search engines often makes users confused about which to select under what conditions. In addition, each search engine has its own unique user interface and features and needs different search strategies. There are many articles on the Web that teach users about the features of search engines and how to take advantage of each one. These facts, together with the limitations of the search services discussed in the previous section, have led to the introduction of meta-search engines.

#### 2.3.1 The structure of the meta-search engine

Meta-search engines like MetaCrawler and SavvySearch are tools that can automatically and simultaneously query several Internet search engines or directories,
interpret and merge all of the results, and display them in a uniform format. Unlike conventional search engines, meta-search engines do not actually crawl the Web pages to maintain their own centralized database. Instead, they pass the queries to a group of pre-selected search services and search their databases for results.

Dreilinger and Howe suggested that a meta-search engine must contain three components: a dispatch mechanism, interface agents, and a display mechanism. The dispatcher is the algorithm or decision-making approach used to determine which search engines or directories a specific query should be sent to. The success of meta-searching depends critically on carefully selecting which resources to use as well as the size, the content, the number of search engines, and how well a meta-search engine selects individual search engines that are most likely return the best results for a particular query. The interface agents are self-contained programs that manage the interaction with a particular search engine. They are responsible for translating the user’s query format into the format of a particular search engine, because the format of the queries and the way search engines process them are far from standardized. The interface agents are also responsible for interpreting the diverse results format of each engine. The display mechanism integrates the raw results returned from each search engine, removes duplicates, and displays them to the users. This is necessary because results should ideally be displayed in a uniform format and be ordered by rank or interleaved.

Figure 2.2 illustrates a simple architecture of a typical meta-search engine. When a query is submitted to a meta-search engine, the dispatcher determines to which search engines the query should be sent. The interface agent converts the query to the proper format.
and submits it to selected search engines in parallel, and transforms the results into a uniform format when they are returned. The display mechanism then merges the results, eliminates identical hits, and displays the results in order.

![Diagram of a typical meta-search engine architecture](image)

**Figure 2.2** The architecture of a typical meta-search engine.

### 2.3.2 The pros and cons of meta-search engines

The primary advantages of current meta-search approaches are the ability to access the databases of more than a single search engine or directory, the ability to combine the results of multiple search engines or directories, and the ability to provide a consistent user interface for searching these engines. By searching multiple engines simultaneously, meta-
search can significantly improve the coverage that is available on the Web. In Lawrence and Giles’s study, the total coverage of six search engines was found to be approximately 3.5 times as much of the Web as for a single engine on average, and about twice the coverage of the largest engine\(^1\). Thus, they will often return answers to relatively obscure queries that a single engine may miss. Moreover, users can avoid the necessity to keep switching among many engines, and no longer need to decide “which to use when.”

However, meta-search engines suffer from their own drawbacks. The quality of the results ultimately relies on the indexing and searching capabilities of the search services used. If even one of them returns a large number of low relevance documents, it will degrade the overall quality of the meta-searcher’s results.

Also, most of the meta-search engines rely on the documents and summaries returned by search engines and inherit their limited precision. NEC Research Institute has developed a meta-search engine, Inquirus, which avoids this problem by downloading and analyzing each document and then displaying the local context around the query term in the original Web page\(^2\). Downloading the actual Web page at the time of searching also leads to the elimination of out-of-date links, thus improving the quality of the results. Another meta-search engine, MetaCrawler, also incorporates a mechanism to verify if a hit in the result list is actually accessible and relevant to the query before it is displayed.

Slow response time is another drawback of the meta-search engine. One slow search engine can impose delays on the display of all the results obtained. Many meta-search engines, therefore, have a timeout period, so that attempts to work with a particular search engine can be abandoned if no response comes from it within a set period of time. However,
to improve speed and efficiency, the number of results that can be obtained is often compromised. Most of the search engines only spend a short time in each database and retrieve only 10% of the available results. This is generally considered to be acceptable because the most relevant results are always given earlier in the result list, while it is not worth the time to retrieve less relevant documents.

Inquirus is claimed to be very efficient. It downloads search engine responses and Web pages in parallel, and can typically returns the first result faster than the average response time of a search engine. However, if a final ranked results list is needed, it still has to wait till the individual search engines finish their retrieval work. Moreover, since it downloads the actual Web pages of all the results, it takes more time to obtain the final results list.

A query submitted to a meta-search engine, with its uniform search interface and syntax, is to be applied against the diversity of search engines. It is therefore impossible for meta-search engines to take advantage of all the features of the individual search engines. Boolean searches and advanced search services, for example, are lost.

### 2.4 Ranking algorithms

With the rapid growth of the entire Web and consequently large size of most popular search engines, there will always be a huge amount of relevant pages from the database for most of the queries, with few of them valuable to a particular user. For example, if you query “search engine” using AltaVista, over 7 million pages will be found. However, the majority of users are reluctant to check the hits beyond the first few result pages. The best
possible ranking algorithms are thus necessary for high-quality search engines to emphasize the more relevant results and eliminate the rest. The most relevant pages are those which are of most value to a user. Relevancy is usually measured by a score that intends to represent the quality and importance of the hit to users.

2.4.1 Traditional ranking algorithms

The precise methods that each search engine uses for determining the relevance score, and thus the ranking, are top secrets and closely guarded. However, they follow a set of general rules, with the main rules involving the location and frequency of keywords on a Web page.

Relevance ranking is usually based on the number of time query terms occur in the document, the importance of each one (tf.idf – term frequency times inverse document frequency), their locations in the text, the relative proximity, and size of the document.

**Term frequency.** Term frequency plays a major role in how search engines determine relevancy. A search engine will analyze how often keywords appear in relation to other words in a Web page. Those with a higher frequency are often deemed more relevant than other Web pages with fewer or no occurrences of the same term. Using this factor alone may artificially raise the ranking of very long pages that contain many words. This is sometimes evident on Web search engines when a very long page, such as a log file, is ranked high. A more helpful approach is where the frequency of the term is compared to the total number of words on the page.
**Inverse document frequency.** A classic information retrieval approach often combines the term frequency with one other important quality of natural-language text – inverse document frequency. The inverse document frequency is one divided by the number of times the word appears in the entire collection of documents, which in this case would be the engine’s database. When considering this factor, the keywords that occur rarely in the database contribute more weight to the relevancy score in a multi-term query. For example, in the query of “Markov model”, the word “Markov” is likely to occur much less frequently in the database than the word “model”, and so contributes a greater weight.

**Term positioning.** Term positioning also certainly plays a role. When a search term is found in certain sections of a Web page, it is considered more important and consequently receives more weight. For example, pages with keywords appearing in the title are assumed to be more relevant than others to the topic. Search engines will also check to see if the keywords appear near the top of a Web page, such as in the headline or in the first few paragraphs of text. This is based on the assumption that any page relevant to the topic will mention those words right from the beginning.

**Term proximity.** When a query contains several terms, the proximity of the search terms to each other will affect the relevance scores. Basically, the closer the search terms are to each other, the more relevant the Web page is considered to be.

**Meta-tags.** Meta-tags are also a common factor that the search engines consider for ranking. Meta-tags are hidden keywords and descriptions that are inserted in the HTML pages by Website builders. They are supposed to accurately represent the topic of the pages, allowing search engines to give the words in author-supplied meta-tags a higher relevance weight.
Above ranking techniques have certain limitations. They evaluate only the contents of Web pages rather than their quality. Little human intelligence work has been integrated. Also, these ranking algorithms are easily manipulated. Many Website builders try to enhance the rank of their sites by inserting popular and irrelevant words into the title, metadata keywords or the body of the page.

2.4.2 Mining the linkage structure

New directions have been introduced to support relevancy ranking for search engines. These algorithms concentrate on mining the linkage structure of the Web. The typical search engines that apply such algorithms are IBM’s Clever and Google \(^3,^4\). Two factors weigh heavily in these methods: anchor text reference and source authority.

The anchor text refers to the words that have been hyperlinked to a new URL. When several or even many other Web sites all point to the same Web page from the same anchor text, the page to which they point is quite likely to be highly relevant to anyone searching on the term or terms within the anchor. Google associates the anchor text with the page the link points to as well as that the link is on. This makes it possible to return Web pages which have not been actually crawled.

Unfortunately, using just the anchor link technique could rapidly fall prey to a new spamming technique. Web index spammers might just create loads of new pages that consist of unrelated anchors that point to their Web site. To avoid this, Google adds a layer of weighting links from authoritative or well-known sites higher than anchors from unknown
situations. Combining this source authority with the anchor text references can achieve highly relevant results.

2.4.3 User feedback

Some search engines take into account feedback from previous user searches in assigning ranks. For example, Direct Hit piggybacks on traditional keyword search engines, kicking in after a user gets a list of search results. By monitoring the amount of time users spend on the sites yielded by their list of search results, Direct Hit comes up with a rough gauge of the sites' popularity, giving the sites that are visited longer higher rankings in future searches.

2.4.4 A ranking algorithm based on Markov model

Researchers have pointed out that simply ranking the hits by traditional information retrieval technology, mining the linkage structure, or considering user feedback alone is not enough. A good ranking algorithm should be multidimensional, simultaneously considering the metrics such as relevance, authority, integrativity, and novelty at the same time.

The relevance measures the distance between the content of a Web resource, R, and a user’s query, Q. This is usually calculated using the traditional IR methods. Authority indicates how many Web resources refer to the Web resource, R. Ideally, the resources referred to by high quality resources should be assigned with higher authority. Integrativity means how many Web resources are pointed to by the Web resource, R. A page that provide the best resource of information on a given topic, or that provide collections of links to authorities, should be assigned a higher integrativity. Novelty means how much the Web
resource, R, is different from others. Highly ranked pages will be very representative and have few overlapping links within them.

Zhang and Dong proposed a ranking algorithm of Web resources based on a Markov model from a dynamic viewpoint. Four parameters were included to present the four metrics; and they can be easily adjusted according to particular needs.

2.5 The search services used in this prototype

Five search engines or directories are included in the prototype meta-search engine described in this report. All are well known public general-purpose search engines or directories. A comparison of some of their features is listed in Table 2.2. Following are brief descriptions for each one of them.

**AltaVista.** AltaVista is one of the largest and most comprehensive general Web search engines. It supports two search modes: Simple Search and Advanced Search. Searching AltaVista gives results from several different sources such as RealNames Internet Keywords, Ask Jeeves question and single answer database, Open Directory subject directory, and the main, very large, AltaVista database. Language capabilities include 25 listed languages. Machine translation is available. The hit display shows title, URL, first two lines, date modified, size in bytes, and language. The results are arranged in the order: matches in the Ask Jeeves database, in the RealNames Internet Keyword database, in the actual Web database. Relevance is determined by location of the terms, proximity of multiple search terms to each other, and the frequency of the terms. All results are clustered by site, so that only one record per site appears on the main results page. In order to provide results quickly,
AltaVista may stop processing a search and only provide partial results. So repeating the exactly the same search several times may give inconsistent results.

**Table 2.2** A comparison of the features of the selected search engines or directories.

<table>
<thead>
<tr>
<th>Database</th>
<th>AltaVista</th>
<th>Direct Hit</th>
<th>Excite</th>
<th>Google</th>
<th>Yahoo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content Size</td>
<td>250M pages</td>
<td>Not known</td>
<td>250M pages &amp; media objects</td>
<td>1.25 billion sites</td>
<td>1.7M pages</td>
</tr>
<tr>
<td>Full-text</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Logic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default word</td>
<td>OR</td>
<td>Not directly</td>
<td>OR</td>
<td>AND</td>
<td>AND</td>
</tr>
<tr>
<td>Boolean connectors</td>
<td>AND, AND NOT, NEAR</td>
<td>Not directly</td>
<td>AND, AND NOT</td>
<td>Limit including and excluding words</td>
<td>AND, OR</td>
</tr>
<tr>
<td>Phrase search</td>
<td>Quotation marks</td>
<td>Not directly</td>
<td>Quotations marks</td>
<td>Quotation marks</td>
<td>Quotation marks</td>
</tr>
<tr>
<td>Truncation</td>
<td>No, use *</td>
<td>Not directly</td>
<td>No</td>
<td>Automatic</td>
<td>No, use *</td>
</tr>
<tr>
<td>Case sensitive</td>
<td>Yes</td>
<td>Not directly</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Words included</td>
<td>Use +</td>
<td>Not directly</td>
<td>Use +</td>
<td>Use +</td>
<td>Use +</td>
</tr>
<tr>
<td>Word elimination</td>
<td>Use -</td>
<td>Not directly</td>
<td>Use -</td>
<td>Use -</td>
<td>Use -</td>
</tr>
<tr>
<td>Duplicate detection</td>
<td>Grouped under one title</td>
<td>Yes</td>
<td>Grouped Under Categories</td>
<td>Grouped Under Categories</td>
<td></td>
</tr>
</tbody>
</table>

25
**Direct Hit.** Direct Hit\(^6\) is a popularity engine which gives results based upon what other searchers have chosen on previous searches for the same query terms. It is ideal for seeing what other searchers choose, especially on popular topics. Direct Hit displays a title linked to the URL, an extract, the URL, and the relevant icons for each hit. Sites are sorted based on how popular they were among previous searchers. Popularity is measured by which hits people click on.

**Excite.** Excite\(^7\) is one of the smaller search engines. However, it is very well known, provides sophisticated personalization, offers excellent relevant results for very popular queries, and its News Search provides important access to Web versions of newspapers, magazines, and news wires. Excite provides simple and advanced interfaces and allows natural language searching as well as complex Boolean searching. The display includes the title, URL, a brief summary, and sometimes the directory match. The results are sorted by a % of confidence - similar to relevancy ranking. Boolean operators must be in all uppercase.

**Google.** Google\(^8\) is one of the largest Web search engines. It is well known for its relevance ranking based on link analysis. Results are sorted by relevance which is determined by Google's PageRank analysis, determined by links from other pages, with a greater weight given to authoritative sites. Pages are also clustered by site. The display includes the title, URL, a brief extract showing text near the search terms, and the file size. Additional features are a unique "cached copy" link, where Google takes a "snapshot" of a page as it crawls the Web, although this may be older than the version currently available on the Web, and a "similar pages" link, which prompts GoogleScout to search the Web for pages related to the resultant search. Recently, Google has begun indexing PDF files,
becoming the first engine to offer searchers an easy way to find these high-quality documents that make up a significant portion of the Deep Web.

**Yahoo.** Yahoo ¹⁹ is one of the best-known Internet subject directories and is particularly good for popular and general information. Yahoo provides high quality results, although the size of its index is relatively small. Entries in Yahoo are gathered from user submissions and the Yahoo editorial team. It can be searched directly or browsed by category. Search statement is automatically sent to the Google database if no matches are found within Yahoo directories. Yahoo displays the category name and hierarchy, the site title, URL, and sometimes a brief description.
Chapter 3 System Design and Implementation

This chapter introduces the system design and the implementation details of a prototype meta-search engine.

3.1 System overview

In this project, a prototype meta-search engine was developed. This meta-search engine collects results by searching the databases of five search services, removes duplicated result pages, and ranks the results based on a Markov model. The ranking and display are based on the Web page information stored in its own database.

3.1.1 System structure

This prototype meta-search engine consists of a user interface, a dispatcher, result integration, and a storage system.

- The user interface accepts users’ query terms and initiates the search process.
- The dispatcher searches a subset of selected search services and collects a certain number of hits from each of them.
- The result integration combines the results returned from the underlying search services, ranks the results based on a Markov model, and displays them to the user.
- The storage system holds information of the Web pages crawled by a spider program in the database and expands the database by retrieving results pages that are not already stored in the database of the storage system.
An overview of this prototype meta-search engine is presented in Figure 3.1. When a query is posted on the user interface by a user, it is sent to the dispatcher where the five search services are searched and raw results are collected. The raw results are combined with the duplicates deleted. The results’ actual HTML pages are downloaded and analyzed if they do not already exist in the database. The results are then placed in order according to the ranking algorithms based on the Markov model. Finally, the results are displayed to the user. A database is maintained in the background by the storage system. Initially, the database was constructed by a spider. It is expanded every time a search is performed by adding new documents that are in the results but not the database.

3.1.2 Environment settings

The whole system is implemented by using Java technologies including Java applications, servlets, and Java server.

**Java.** Java has gained enormous popularity since it first appeared. Java was chosen as the programming language for network computers and has been perceived as a universal front end for the enterprise database. Java™ 2 Platform, Standard Edition (J2SE) version 1.3 was downloaded from Java’s homepage and installed on the PC.

**Oracle 8i.** The database used for this prototype was built using Oracle technology. All Oracle8i databases are compatible, portable and highly scalable, and may be used to host Internet applications and content, in addition to serving as a repository for traditional relational data. Oracle8i Enterprise Edition release 2 (version 8.1.6) was downloaded from Oracle.com and installed with the global database named as “hannah.”
Figure 3.1 The system structure of the proposed meta-search engine.
**JDBC.** The Java application is connected with the database through JDBC. The combination of Java and JDBC has an essential advantage over other database programming environments since it allows one to develop programs that are platform independent and vendor independent. The JDBC is included in the Oracle8i Enterprise Edition v 8.1.6.

**Java Server.** Tomcat was used to set up the server. It is a servlet container and JavaServer Pages implementation. Tomcat version 3.2.1 was obtained and installed under the directory C:Jakarta-tomcat\. Servlet was used to develop the client-server application in this prototype. This high-level view of networking in Java is based on the request-response model of communication. A Servlet extends the functionality of a server by enabling it to act dynamically upon receiving a request from a client. The javax.servlet package and javax.Servlet.http package provide the classes and interfaces used to define Servlet.

### 3.1.3 An overview of the programs

Table 3.1 is a list of the programs developed for the entire system. The HTML files serve as the user interfaces. Two servlet classes were developed for the prototype: MetaSearchEngine.java and MySpider.java. The former captures the user’ query and performs the search for the meta-search engine. The latter is used to initiate the Web crawling process and update the database.

### 3.2 User interface

The screen of the user interface of the meta-search engine is captured and displayed as Figure 3.2. Users can customize the search services by selecting a subset of listed search engines or directories and deciding how many hits should be collected from each engine.
Table 3.1 Programs written for the prototype meta-search engine.

<table>
<thead>
<tr>
<th>Components</th>
<th>Programs</th>
<th>Java applications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTML files</td>
<td>Servlets</td>
</tr>
<tr>
<td>User interface</td>
<td>Search.html</td>
<td>MetaSearchEngine.java</td>
</tr>
<tr>
<td>Dispatcher</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Storage System</td>
<td>Spider</td>
<td>MySpider.java</td>
</tr>
<tr>
<td>Database</td>
<td>Spider.html</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2 The user interface of the proposed meta-search engine.
The user interface triggers the servlet class MetaSearchEngine.class through the code:

```html
<FORM NAME=searchform ACTION=/examples/servlet/MetaSearchEngine
    METHOD=GET>
```

The validation of the input fields is checked with JavaScript language. If an invalid input is detected, a small window with an alert message will pop up to remind the user. For example, following code is included in the HTML page to check if the search box is blank when a user clicks the search button:

```javascript
var s1 = document.searchform.query.value.toString();
if (s1 == "") {
    alert('Please input the query.');
    searchform.query.focus( );
    return false;
}
```

A help button is added using JavaScript language. It explains everything shown on the user interface and teaches a naïve user how to perform a search.

The MetaSearchEngine class provides all the functionality of the prototype meta-search engine. It extends class HttpServlet and overrides its doGet ( ) method. Figure 3.3 is the flowchart of this method. It first gets the user input by:

```java
String input = request.getParameter( "query" );
```

It then sends the user query directly to the selected search services and searches their databases. The returned results are merged together by the method Combine (v1, v2) with
duplicates deleted. If a result is not already recorded in the database of the storage system, its source page is downloaded by Retrieve class and desired information from the page is stored into the database by InsertInfo class. The results are then ranked based on a Markov model that is implemented as Rank class. Finally the ranked results are displayed to users through the object response that implements interface HttpServletResponse.

**Figure 3.3** The flowchart of the doGet ( ) method of MetaSearchEngine class.
3.3 The dispatcher

The dispatcher searches for the user query in the databases of selected search services and collects the results. It is implemented as MetaSearch class. The MetaSearch class contains a series of public methods; each searches an engine and stores the retrieved URLs of the hits in a vector object. These methods are searchAv(), searchDh(), searchExcite(), searchGoogle(), and searchYahoo(). The setSize(int i) method of this class enables the user to decide how many hits to obtain from each engine.

The query. The query is passed directly to each search engine without any modification except for Direct Hit. If double quotes occur in the query, it will be eliminated before it is searched in Direct Hit, since it will cause Direct Hit to produce no results. Queries are sent to remote search engines via HTTP protocol in a similar manner to Web browsers. For each search engine, the results are retrieved by passing on the query term as well as the required parameters, including the starting number of the results on the page, the number of results on the page, and the type of result pages (Web sites or Web pages if applicable). The query URL sent to each search engine will be in a form that can be handled acceptably, and will not be incorrectly encoded. These characters in the query sent to search engines must be encoded: "&", "+", and " " . The following code applies the static method String encode(String s) of java.net.URLDecoder class to return the URL-encoded form of the original query term.

    input = URLDecoder.encode(input);
An example query URL is:

```
http://www.directhit.com/fcgi-bin/DirectHitWeb.fcg?
qury=auburn+university&base=10&pgsz=10&alias=websrch
```

In this URL, “http://www.directhit.com/fcgi-bin/DirectHitWeb.fcg?” is the search program that dynamically generates the search results for the engine. “qry=auburn+university” is the encoded query term. “&base=10” indicates the starting number of the results on the result page is 11. “&pgsz=10” indicates that there are 10 hits listed on the page. “&alias=websrch” means searching for Web sites or Web pages. So the above query URL searches the Direct Hit database for “auburn university” and returns an HTML page of search results ranked from 11-20. Only the simple search form of each search engine or directory can be queried in this prototype and advanced search features are lost.

**Download.** The URL and URLConnection classes of Java encapsulate much of the complexity of retrieving information from a remote site. Therefore we can easily download HTML pages using the advanced services that the standard edition of the Java platform provides. The following segment of code opens a connection to the URL represent by String currentSite. The input stream “in” is then used to read the data from the resource line by line using the readLine( ) method of BufferedReader class.

```java
URL urlName = new URL(currentSite);
URLConnection connection = urlName.openConnection();
connection.connect();
BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
```
**Result retrieval.** Once an HTML results page from a search service is downloaded, the page is parsed to retrieve the URLs of the hits on the page. This is done according to our knowledge of the results page format of each search service included in the prototype. This leads to a major drawback of our prototype: if one of the underlying search services changes their results page format, we have to change our code accordingly or the system will crash. From each search service, if too many results are available for a query only a certain number of results are retrieved. The number is set by the method setSize (int i). Otherwise, the search method stops when there are no more results available. The URLs of hits returned from each search engine are stored as Java vector objects.

### 3.4 Result integration

**Combination of the raw results.** All of the results are then merged together and the duplicated pages are deleted to form a single result vector. Combine(v1, v2) is the method in MetaSearchEngine class that performs the combination of the result vectors of two single engines. The logic is presented in Figure 3.4. Only duplicates of two URLs that are exactly same are eliminated. Some URLs (ie, http://www.searchenginewatch.com and http://searchenginewatch.com), although they point to the same page, are different in terms of character strings. They are thus considered to be two different links in this prototype. Such page duplication still exists. Also, Web pages are not clustered into sites.
**Figure 3.4** The flowchart for the combination of two result vectors, $v_1$ and $v_2$. 

```plaintext

Vector $v_1, v_2$

Yes

$v_2.size() = 0$

No

$v_1.size() = 0$

Yes

$v_1 = v_2$

No

$v_1.size() = 0$

Yes

j ++

$v_1 = v_2$

No

Return $v_1$

Yes

j >= $v_2.size()$

No

i >= $v_1.size()$

Yes

v1.add($v_2.elementAt(j)$)

No

j ++

i ++

j >= $v_2.size()$

No

i >= $v_1.size()$

Yes

v1.add($v_2.elementAt(j)$)

No

j = 0

i = 0

v1.addElementAt(i) = $v_2.elementAt(j)$

Yes

i ++

No

j = 0

i = 0

v1.addElementAt(i) = $v_2.elementAt(j)$

Yes

i ++

No

j = 0

i = 0

```
**Results ranking and display.** The merged result list is then ranked based on a Markov model. The details of ranking will be discussed in the next chapter. Finally, the ranked results are displayed to users in the browser. The display of each result includes the hypertext of its title, the metadata of its description followed by the first sentence of the page, and its URL. The segment of code that realizes the display is as follows.

```java
response.setContentType( "text/html" );
PrintWriter responseOutput = response.getWriter( );
StringBuffer buf = new StringBuffer( );
buf.append( "<html>
" );
... 
buf.append( "</html>" );
responseOutput.println( buf.toString( ) );
responseOutput.close( );
```

### 3.5 The storage system

A storage system is incorporated in this prototype to improve the efficiency. If a result page already exists in the database, its source page need not be downloaded and analyzed at the time of searching and the stored information is used instead. In this way, the search speed can be improved. To keep the stored information as current as possible and avoid dead links, the database can be updated frequently. Thus, the prototype achieves a balance between up-to-date information and search speed. The storage system consists of a database, a spider, and the results page retrieval.
3.5.1 The database

The database used in this prototype search engine is quite simple. It contains two tables: DOWNLOAD and LINKTABLE. DOWNLOAD stores the information of all the Web pages downloaded by both the spider and the results page retrieval process. It contains five fields for HTML pages: URL, title, meta-keyword, meta-description, and the first sentence. LINKTABLE is used to store the link structure of the pages used for the purpose of ranking. It has two attributes, both of which are URLs. The Web page of the first URL has a hyperlink to the second URL.

The database is initially created by class CreateTable. JDBC is used to connect the Java application with the Oracle database. Access to the database is realized by the following code segment:

```java
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection con = DriverManager.getConnection(
    "jdbc:oracle:oci8:@hannah", "scott", "tiger");
```

The retrieved information of Web pages is inserted into the tables by class InsertInfo. InsertInfo has two constructors, one of which accepts a Spider object as its parameter, and the other a Retrieve object.

3.5.2 The spider

Updating the database is not an automatic process in this prototype. The user interface shown in Figure 3.5 is used to initiate the indexing process. It makes updating of the index easier. It is required to enter a URL as the seed URL, specifying where the spider
starts to crawl the Web, and the maximum number of Web pages to be downloaded by the spider. Clicking on the Start_Crawling button triggers the spider to start to crawl the Web and updates the database.

![My Spider - Microsoft Internet Explorer](image)

**Figure 3.5** The interface to initiate the Web page crawling and downloading process.

The Spider class is responsible for downloading the Web pages and retrieving the desired information from each page. Six private vectors (sites, titles, keywords, descriptions, fstSentences, and linkTable) are created to hold the downloaded information. Accordingly, six public get methods are included to obtain the information stored in these vectors. The variable linkTable is used for the purpose of ranking discussed in Chapter 4. It contains pairs of links: the URL to be crawled, and one URL contained in the page.
In this class, Boolean method download () is used to download Web pages. Method crawl (Boolean more) and isLink (String input, Boolean more) are used to retrieve the links within the downloaded page. Method processWork (int i) is used to retrieve the desired information from the Web pages, including the title, metadata, and the first sentence.

The seed URL is the root where the spider starts to crawl. The crawling uses a breadth first search technology. All the links on the first page are retrieved at a time and put into a queue. Then these links are downloaded in order and the links in them are retrieved and added to the end of the queue. If a page cannot be downloaded for any reason, it is assumed to be a dead link and will be removed from the queue. The process continues until the required number of links have been obtained.

Web page download is implemented by using the URL and URLConnection classes as in Section 3.3. In this prototype, only Web pages using the HTTP protocol are downloaded. All others are discarded. In addition, pages with HTTP status code greater than 400 (error pages) are discarded to avoid dead links. This is done using the following code:

```java
if (connection instanceof HttpURLConnection) {
    HttpURLConnection check = (HttpURLConnection)connection;
    int code = check.getResponseCode();
    if (code >= 400)
        return false;
} else {
    System.out.println("Other than http protocol");
    return false;
}
```
Once an HTML page has been downloaded, the content of the page is converted to lower cases. The comments enclosed in <!-- and --> and the parts of script language are deleted. Method crawl (Boolean more) retrieves all the links following string “href” in the page. Since links can be made in many different ways, method isLink (String input, Boolean more) is used to check if a link is valid or not before it is added to the sites vector and linkTable vector. The Boolean parameter more is included to control whether a new link obtained need to be added to the sites vector or not. Once the size of the sites vector equals the maximum number of Web pages required, variable more is assigned to the value of “false”; and no new links are added to the sites vector.

Connecting to valid links from an HTML page is quite complex and tedious. A link in HTML text can be either an absolute or relative pathname. All the possible cases are listed in Table 3.2. Only HTML pages are retrieved; others such as images, mailto tools, PDF files, and CSS files are discarded. While a link is a relative pathname, the type scheme, the hostname, and the path to the current directory can be added to form an absolute pathname. If a URL contains the “#” sign to jump within a file, the anchor name following the “#” sign is discarded to avoid duplication of the page. Finally, a URL is added to the site vector only when it is not already included and the size of the sites vector is less than the required maximum number.

3.5.3 The result page retrieval

The class Retrieve was developed to download and analyze the result pages. Its functionalities are very similar to those of class Spider, except that it does not crawl the Web pages. Instead, it accepts a vector parameter that contains the result URLs. It first checks if
a URL already exists in the database. If not, it then starts to download the page by method download (), retrieve the links by method crawl (), and extract the desired information by method processWork (int i).

**Table 3.2 Possible hyperlinks in HTML files.**

<table>
<thead>
<tr>
<th>HTML code</th>
<th>Link Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;a href=&quot;file.html&quot;&gt;hypertext&lt;/a&gt;</code></td>
<td>Local Hypertext Link</td>
<td>Link to another document in the same folder/directory.</td>
</tr>
<tr>
<td><code>&lt;a href=&quot;data/file.html&quot;&gt;hypertext&lt;/a&gt;</code></td>
<td>Local Hypertext Link</td>
<td>Link to another document in a directory named &quot;data&quot; that is inside the directory containing the calling HTML document.</td>
</tr>
<tr>
<td><code>&lt;a href=&quot;../file.html&quot;&gt;hypertext&lt;/a&gt;</code></td>
<td>Local Hypertext Link</td>
<td>Link to another document in a directory that is one level higher relative to the calling HTML document.</td>
</tr>
<tr>
<td><code>&lt;a href=&quot;URL&quot;&gt;hypertext&lt;/a&gt;</code></td>
<td>Internet HyperText Link</td>
<td>Link to another Internet Site, specified by URL.</td>
</tr>
<tr>
<td><code>&lt;a href=&quot;file.html#xy&quot;&gt;hypertext&lt;/a&gt;</code></td>
<td>Link to Named Anchor</td>
<td>Jump to a named anchor within the same or another document.</td>
</tr>
<tr>
<td><code>&lt;a href=&quot;mailto:doe@xyz.edu&quot;&gt;... &lt;/a&gt;</code></td>
<td>Internet Mail Link</td>
<td>Sets up email message to specified address.</td>
</tr>
<tr>
<td><code>&lt;base target=&quot;window_name&quot;</code></td>
<td>Base tag</td>
<td>The <code>&lt;BASE...&gt;</code> tag makes the target effective for all links that follow it in the HTML code.</td>
</tr>
</tbody>
</table>
Chapter 4 A Web Page Ranking Algorithm Based on a Markov Model

Results of the prototype meta-search engine are ranked according to the model suggested by Zhang and Dong\(^5\). We can view the search results for a query Q as a set of related Web resources R \((r_1, r_2, \ldots, r_n)\). Assume a user surfing on the Web is browsing the Web resource \(r_i\) in probability \(P_i(t)\), and will jump to the Web resource \(r_j\) in probability \(P_{ij}\).

When users check the search results, their behavior can be described as follows. Suppose the user is browsing page \(r_i\) at time \(t\), then at time \(t+1\), he may:

- Keep browsing the same page \(r_i\);
- Jump to a new page using a hyperlink in page \(r_i\);
- Return to a previous page through the “BACK” button; or
- Select another page from the result list (resources R).

The tendency of a user to choose among these options can be measured using four parameters, \(a*corr[i], b, c, d\). The four parameters represents the four metrics (relevance, authority, integrativity, and novelty) discussed earlier in Section 2.4. \(a, b, c,\) and \(d\), are four constants for a specific user which satisfy the conditions: \(0 < a, b, c, d < 1\) and \(a + b + c + d = 1\). These four parameters can be adjusted according to users’ particular needs. \(corr[i]\) is the relevance function obtained from conventional information retrieval methods. Particularly, if a record in the database matches a word or phrase of the query in the meta-tag keyword field, it is assigned a weight of 2; if a match appears in any of the fields of title, description, or first sentence, each is assigned a weight of 1. The total weight of a query is the relevancy score of a hit.
The tendency matrix is defined as:

\[
U_{ij} = \begin{cases} 
  a \ast \text{corr}[i] & \text{if } i = j \\
  b, & \text{if } (v_i, v_j) \in E \\
  c, & \text{if } (v_j, v_i) \in E \\
  d, & \text{otherwise} 
\end{cases}
\]  

(Equation 4.1)

where \((v_i, v_j) \in E\) means page \(r_i\) points to \(r_j\) through a hyperlink. The tendency matrix of Web resource \(R\), \(U = (u_{ij})_{n \times n}\), synthesizes the four metrics defined in the previous paragraph using the determined parameters. Therefore, this result ranking is multidimensional.

A user jumps to a Web page \(r_j\) from \(r_i\) with a probability \(p_{ij}\). The transition probability matrix, \(P = (p_{ij})_{n \times n}\), for Web resource \(R\) can be constructed by normalizing the tendency matrix as

\[
p_{ij} = \frac{u_{ij}}{\sum_{j=1}^{n} u_{ij}}
\]  

(Equation 4.2)

\(P\) is a square matrix with nonnegative entries and row sums all equal to 1. It is called a stochastic matrix. A stochastic process for which the probability of entering a certain state depends only on the last state occupied is called a Markov process. Therefore, a user’s searching action can be abstracted as a Markov chain. If at time \(t\) a distribution vector \(p(t) = (p_1(t), p_2(t), \ldots, p_n(t))\) represents the probability of each hit to be surfed, then \(p(t+1) = p(t) \ast P\). When time reaches the limit, the ultimate probability vector should reflect the relevancy of
each Web resource in the result set. The ultimate stable vector, \( P_{\text{limit}} = (p_1, p_2, \ldots, p_n) \), is the unique solution of the equation

\[
P_{\text{limit}} = P_{\text{limit}} \cdot P
\]

(Equation 4.3)

with the sum of all \( p_i \) equals 1. This leads to a system of linear equations and can be solved using the Gaussian’s elimination method.

The result ranking is implemented as Rank class and the class Gaussian solves the equations. The field seq [] (an integer array) of Rank class is used to hold the ranked sequence of the results list. It can be accessed through the public method getSequence (). Table 4.1 lists a sample \( u_{ij} \) matrix, \( p_{ij} \) matrix, the ultimate distribution vector \( P_{\text{limit}} \), and the ranked list.
**Table 4.1** A sample $u_{ij}$ matrix, $p_{ij}$ matrix, the ultimate distribution vector $P_{\text{limit}}$, and the ranked list. The query term is “auburn university.” All underlying search services are selected. The number of search results returned from each service is 3. The four parameters are set to be $a = 0.6$, $b = 0.2$, $c = 0.19$, and $d = 0.01$. All underlying search services are selected. The final results list contains 9 hits.

**Table 4.1a** A sample tendency matrix, $U = (u_{ij})_{nxn}$. It is calculated according to Equation 4.1.

\[
\begin{array}{cccccccccc}
2.40 & 0.20 & 0.20 & 0.01 & 0.19 & 0.20 & 0.01 & 0.20 & 0.20 \\
0.19 & 2.40 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\
0.19 & 0.01 & 3.60 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\
0.01 & 0.01 & 0.01 & 0.00 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\
0.20 & 0.01 & 0.01 & 0.01 & 6.00 & 0.20 & 0.01 & 0.01 & 0.20 \\
0.19 & 0.01 & 0.01 & 0.01 & 0.19 & 4.80 & 0.01 & 0.01 & 0.19 \\
0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 1.20 & 0.01 & 0.01 & 0.01 \\
0.19 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 4.80 & 0.01 & 0.01 \\
0.19 & 0.01 & 0.01 & 0.01 & 0.19 & 0.20 & 0.01 & 0.01 & 2.40
\end{array}
\]

**Table 4.1b** A sample transition probability matrix, $P = (p_{ij})_{nxn}$. It is calculated according to Equation 4.2.

\[
\begin{array}{cccccccccccc}
0.66482 & 0.05540 & 0.05540 & 0.00277 & 0.05263 & 0.05540 & 0.00277 & 0.05540 & 0.05540 \\
0.07143 & 0.90226 & 0.00376 & 0.00376 & 0.00376 & 0.00376 & 0.00376 & 0.00376 & 0.00376 \\
0.04922 & 0.00259 & 0.93264 & 0.00259 & 0.00259 & 0.00259 & 0.00259 & 0.00259 & 0.00259 \\
0.12500 & 0.12500 & 0.12500 & 0.00000 & 0.12500 & 0.12500 & 0.12500 & 0.12500 & 0.12500 \\
0.03008 & 0.00150 & 0.00150 & 0.00150 & 0.90226 & 0.03008 & 0.00150 & 0.00150 & 0.03008 \\
0.03506 & 0.00185 & 0.00185 & 0.00185 & 0.03506 & 0.88561 & 0.00185 & 0.00185 & 0.03506 \\
0.00781 & 0.00781 & 0.00781 & 0.00781 & 0.00781 & 0.00781 & 0.93750 & 0.00781 & 0.00781 \\
0.03755 & 0.00198 & 0.00198 & 0.00198 & 0.00198 & 0.00198 & 0.00198 & 0.94862 & 0.00198 \\
0.06271 & 0.00330 & 0.00330 & 0.00330 & 0.06271 & 0.06601 & 0.00330 & 0.00330 & 0.79208
\end{array}
\]
Table 4.1c The ultimate stable distribution vector with corresponding URLs. It is obtained by solving equation 4.3 using the Gaussian method with $P$ from Table 4.1b. $P_{\text{limit}} = \{0.11106, 0.08567, 0.12432, 0.00253, 0.20088, 0.17599, 0.04055, 0.16297, 0.09602\}.$

<table>
<thead>
<tr>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.11106</td>
<td><a href="http://www.auburn.edu">http://www.auburn.edu</a></td>
</tr>
<tr>
<td>0.08567</td>
<td><a href="http://search.auburn.edu">http://search.auburn.edu</a></td>
</tr>
<tr>
<td>0.12432</td>
<td><a href="http://www.auburn.edu/athletics">http://www.auburn.edu/athletics</a></td>
</tr>
<tr>
<td>0.00253</td>
<td><a href="http://www.universities.com/elsewhere.asp?univkey=85">http://www.universities.com/elsewhere.asp?univkey=85</a></td>
</tr>
<tr>
<td>0.20088</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>0.17599</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>0.04055</td>
<td><a href="http://www.auburn.edu/business">http://www.auburn.edu/business</a></td>
</tr>
<tr>
<td>0.16297</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissio">http://www.auburn.edu/student_info/student_affairs/admissio</a></td>
</tr>
<tr>
<td>0.09602</td>
<td><a href="http://oasis.auburn.edu">http://oasis.auburn.edu</a></td>
</tr>
</tbody>
</table>

Table 4.1d The ranked URL list with their relevance score according to the seq[ ] array: Seq [i] = \{4, 5, 7, 2, 0, 8, 1, 6, 3\}.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.20088</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>2</td>
<td>0.17599</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>3</td>
<td>0.16297</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissio">http://www.auburn.edu/student_info/student_affairs/admissio</a></td>
</tr>
<tr>
<td>4</td>
<td>0.12432</td>
<td><a href="http://www.auburn.edu/athletics">http://www.auburn.edu/athletics</a></td>
</tr>
<tr>
<td>5</td>
<td>0.11106</td>
<td><a href="http://www.auburn.edu">http://www.auburn.edu</a></td>
</tr>
<tr>
<td>6</td>
<td>0.09602</td>
<td><a href="http://oasis.auburn.edu">http://oasis.auburn.edu</a></td>
</tr>
<tr>
<td>7</td>
<td>0.08567</td>
<td><a href="http://search.auburn.edu">http://search.auburn.edu</a></td>
</tr>
<tr>
<td>8</td>
<td>0.04055</td>
<td><a href="http://www.auburn.edu/business">http://www.auburn.edu/business</a></td>
</tr>
<tr>
<td>9</td>
<td>0.00253</td>
<td><a href="http://www.universities.com/elsewhere.asp?univkey=85">http://www.universities.com/elsewhere.asp?univkey=85</a></td>
</tr>
</tbody>
</table>
Chapter 5 Experimental Results

In this chapter, we will evaluate the performance of our meta-search engine prototype using a group of trial searches. Also, we will study the effect of different weights for the four parameters \((a, b, c, d)\) on the results.

Figure 5.1 shows a sample result page. The display of each hit consists of a hypertext of the page title, the URL, and a meta-description followed by the page’s first sentence.

![Search Results](image)

Figure 5.1 A sample result page.
To evaluate the performance of this prototype, a group of experimental searches were conducted. Three query terms (“auburn university,” “camry, Toyota,” and “Java servlet”) were tested. Three series of parameter weights were used: series 1 (a = 0.60, b = 0.20, c = 0.19, d = 0.01), series 2 (a = 0.25, b = 0.25, c = 0.25, d = 0.25), and series 3 (0.25, b = 0.35, c = 0.30, d = 0.10). The number of results returned from each underlying search service is either 5 or 10.

**The overlap of prototype results with those of the underlying search services**

![The overlap of prototype results with those of the underlying search services](image)

**Figure 5.2** The overlap of our results with those of the five underlying search services when the number of results returned from each engine is 10. The numbers shown are the summation of three queries: “auburn university,” “camry, toyota,” and “Java servlet.” Top ten results of each search are examined with a total of 30 results.
Figure 5.2 uses a bar graph to present the overlap of our results with the five underlying search services. Table 5.1 shows the first five URLs with their relevance scores in our result lists when the four parameters were assigned to different weights for the query term “auburn university.” Table 5.2 shows the ranked place of URL http://www.auburn.edu in the result lists for the query “auburn university” using different parameter weights.

From a scan of the experimental result lists, all top ten results are relevant to the query. The presented results suggest that different weights of the four parameters affect the ranking. For example, Figure 5.1 shows that series 3 provides the best overlap for the top ten results with all five underlying search services selected, while series 2 the worst. In addition, it is common sense that when one queries for “auburn university,” the university’s homepage “http://www.auburn.edu” should have a relatively high rank, if not at the first place. However, this was not the case using series 2 parameter weights (Table 5.2). Series 1, although better, is not a satisfactory result list either. This is understandable since for series 1, parameter $a$ (coefficient of traditional IR score) has a higher weight, but this prototype calculates the keyword relevance quite simply. Only the title, meta-keywords, meta-description, and first sentence are used to match the query. Parameter $d$, used to represent the novelty factor, actually stands for the randomized selection of a page from the result set. When it is high, as in series 2, ranking of the results is certainly worse.
Table 5.1 The top five result URLs with their relevance scores when the four parameters, \( a, b, c, \) and \( d \), are assigned to different weights. The query term is “auburn university.” \( n \) stands for the number of results returned from each underlying search service. \( k \) is the number of results returned in our result lists.

Table 5.1a \( a = 0.60, b = 0.20, c = 0.19, d = 0.01, n = 5, \) and \( k = 17. \)

<table>
<thead>
<tr>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.11568</td>
<td><a href="http://www.eng.auburn.edu">http://www.eng.auburn.edu</a></td>
</tr>
<tr>
<td>0.11377</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>0.10046</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>0.09210</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissio">http://www.auburn.edu/student_info/student_affairs/admissio</a></td>
</tr>
<tr>
<td>0.07544</td>
<td><a href="http://www.theplainsman.com">http://www.theplainsman.com</a></td>
</tr>
</tbody>
</table>

Table 5.1b \( a = 0.60, b = 0.20, c = 0.19, d = 0.01, n = 10, \) and \( k = 32. \)

<table>
<thead>
<tr>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.06414</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>0.06362</td>
<td><a href="http://www.eng.auburn.edu">http://www.eng.auburn.edu</a></td>
</tr>
<tr>
<td>0.05514</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>0.05249</td>
<td><a href="http://www.auhcc.com">http://www.auhcc.com</a></td>
</tr>
<tr>
<td>0.05188</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissio">http://www.auburn.edu/student_info/student_affairs/admissio</a></td>
</tr>
</tbody>
</table>
**Table 5.1c** \( a = 0.25, \ b = 0.25, \ c = 0.25, \ d = 0.25, \ n = 5, \) and \( k = 17. \)

<table>
<thead>
<tr>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.07324</td>
<td><a href="http://www.eng.auburn.edu">http://www.eng.auburn.edu</a></td>
</tr>
<tr>
<td>0.07323</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>0.06760</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>0.06760</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissio">http://www.auburn.edu/student_info/student_affairs/admissio</a></td>
</tr>
<tr>
<td>0.06479</td>
<td><a href="http://www.theplainsman.com">http://www.theplainsman.com</a></td>
</tr>
</tbody>
</table>

**Table 5.1d** \( a = 0.25, \ b = 0.25, \ c = 0.25, \ d = 0.25, \ n = 10, \) and \( k = 32. \)

<table>
<thead>
<tr>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.03587</td>
<td><a href="http://www.eng.auburn.edu">http://www.eng.auburn.edu</a></td>
</tr>
<tr>
<td>0.03587</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>0.03500</td>
<td><a href="http://www.auhcc.com">http://www.auhcc.com</a></td>
</tr>
<tr>
<td>0.03412</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>0.03412</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissio">http://www.auburn.edu/student_info/student_affairs/admissio</a></td>
</tr>
</tbody>
</table>

**Table 5.1e** \( a = 0.25, \ b = 0.35, \ c = 0.30, \ d = 0.10, \ n = 5, \) and \( k = 17. \)

<table>
<thead>
<tr>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.08827</td>
<td><a href="http://www.eng.auburn.edu">http://www.eng.auburn.edu</a></td>
</tr>
<tr>
<td>0.08612</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>0.08556</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>0.08213</td>
<td><a href="http://www.auburn.edu">http://www.auburn.edu</a></td>
</tr>
<tr>
<td>0.07360</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissio">http://www.auburn.edu/student_info/student_affairs/admissio</a></td>
</tr>
</tbody>
</table>
Table 5.1f \( a = 0.25, \ b = 0.35, \ c = 0.30, \ d = 0.10, \ n = 10, \) and \( k = 32. \)

<table>
<thead>
<tr>
<th>Relevance score</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.04439</td>
<td><a href="http://www.auburn.edu">http://www.auburn.edu</a></td>
</tr>
<tr>
<td>0.04400</td>
<td><a href="http://www.vetmed.auburn.edu">http://www.vetmed.auburn.edu</a></td>
</tr>
<tr>
<td>0.04342</td>
<td><a href="http://www.eng.auburn.edu">http://www.eng.auburn.edu</a></td>
</tr>
<tr>
<td>0.04193</td>
<td><a href="http://www.lib.auburn.edu">http://www.lib.auburn.edu</a></td>
</tr>
<tr>
<td>0.03897</td>
<td><a href="http://www.auburn.edu/student_info/student_affairs/admissions">http://www.auburn.edu/student_info/student_affairs/admissions</a></td>
</tr>
</tbody>
</table>

Table 5.2 The ranked places of URL http://www.auburn.edu in the result lists for the query “auburn university” using different parameter weights. \( n \) is the number of results returned from each underlying search service.

<table>
<thead>
<tr>
<th>Parameter weights</th>
<th>( n = 5 )</th>
<th>( n = 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = 0.60, \ b = 0.20, \ c = 0.19, \ d = 0.01 )</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>( a = 0.25, \ b = 0.25, \ c = 0.25, \ d = 0.25 )</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>( a = 0.25, \ b = 0.35, \ c = 0.30, \ d = 0.10 )</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Most of the meta-search engines rely on the documents and summaries returned by search engines and inherit their limited precision. NEC Research Institute has developed a meta-search engine, Inquirus, which changes the situation by downloading and analyzing each document and then displaying the local context around the query term in the original Web page. Downloading the actual Web page at the time of searching also leads to the elimination of out-of-date links, thus improving the quality of the results. We adopted this strategy in our prototype so that our ranking depends solely on the information from the original Web pages. In addition, our result display can include the title, meta-description, and first sentence from the actual pages.
The delay of the response time is usually another drawback of the meta-search engine. One slow search engine can impose delays on the display of all the results. Inquirus is claimed to be very efficient. It downloads search engine response and Web pages in parallel, typically returning the first result faster than the average response time of a search engine. However, if the results are being presented as a single integrated list, it takes more time to obtain the final results list. To solve this problem, we incorporated a storage system in this prototype to improve the efficiency, since Web documents in the database need not be retrieved again. Our experience with this prototype shows that a search takes much less time if the query has been searched before by the system.

It is worth mentioning that the goodness of the ranking and the quality of the search results are based on their values to a particular user. A thorough evaluation of the quality of this proposed system would involve an extensive user study that has not yet been done.
Chapter 6 Conclusions

In this project a prototype meta-search engine was developed. The meta-search engine collects a certain number of search results for a query from a subset of pre-determined search engines and directories, merges the results, and displays them to users in a ranked order.

6.1 Advantages of this prototype

This prototype meta-search engine presents the following advantages.

• As with other meta-search engines, this prototype improves the coverage of the information that is available on the Web by searching multiple search services simultaneously.

• The prototype meta-search engine also provides a uniform search interface for users. Users can avoid switching among many engines, having to learn how to search different engines, and being confused about which engine to select.

• This prototype downloads and analyzes the actual Web pages at the time of searching. Both the ranking and the display are based on the newly retrieved information, so that the quality of the results can be less dependent on the underlying search services.

• A storage system is included to improve the speed and efficiency by avoiding retrieving Web documents that are already stored in the database.

• The results are ranked based on a Markov model. This algorithm synthesizes the relevance, authority, integrativity and novelty of the results so that it ranks the search
results in a multidimensional manner. It is also dynamic, since the four parameters used can be adjusted according to users’ needs.

In this way, the proposed system offers an excellent alternative for users who need to search the Web.

6.2 Drawbacks of this prototype

However, this prototype meta-search engine introduced its own deficiencies.

It relies on the underlying search services in some ways. First, the result URLs are obtained by parsing the returned results pages from underlying engines based on our knowledge of their result page formats. If an engine changes its results display, we have to modify our code correspondingly. Second, documents of low relevance returned from any of the underlying engines will degrade the quality of the final results.

In this prototype, the response time is particularly slow because the engines are searched sequentially and the results are not displayed until all results are ranked. The two bottlenecks of the speed are database connection and Web information retrieval.

The query is passed to selected search services simply as it is without any refinement according to each service’s query syntax. It is therefore impossible for this prototype to take advantage of the advanced features of each of the search engines.

Due to very limited system resources, only limited information for each Web page is retrieved and stored in the database. Thus, the ranking performance is degraded.
6.3 Suggestions for future work

Future work is expected to improve the performance of this meta-search engine.

**Efficiency.** The system performance needs to be improved. This can be done by searching the selected search services in a parallel manner. To improve the speed, result page retrieve can also be done immediately after a result URL is available.

**Ranking.** The relevance factor \( (a^{*corr[i]}) \) needs to be calculated using a more complex algorithm. In addition, more dimensions such as users’ feedback can be added to the model when the tendency matrix is calculated.

**Automatic routing.** This prototype, although allowing a certain degree of query routing, can only do so manually. The ideal meta-search engine should include many subject-specific search engines that are particularly good for specific subjects and possesses the intelligence to identify the type of query, automatically selecting those engines which would perform best for that query.
References


2. S. Lawrence and C. L. Giles, “Context and Page Analysis for improved Web Search,” 


