USING EXTRA-TOPICAL USER PREFERENCES TO IMPROVE WEB-BASED METASEARCH

by

Eric J. Glover

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2001

Doctoral Committee:

Associate Professor William P. Birmingham, Co-Chair
Professor Michael D. Gordon, Co-Chair
Professor Edmund H. Durfee
Professor Elliot Soloway

ABSTRACT

USING EXTRA-TOPICAL USER PREFERENCES TO IMPROVE WEB-BASED METASEARCH

by

Eric J. Glover

Co-Chairs: William P. Birmingham and Michael D. Gordon

When searching the web, a user strives to find *useful* documents. Web search engines have been shown to have relatively low coverage, as well as other problems, that limit their ability to return useful documents. One solution is use of a metasearch engine: a tool that sends user queries to multiple search engines and combines the results to increase coverage. Unfortunately, a metasearch engine also has problems, mostly due to the lack of direct control of the underlying search engines, that limit its ability to locate and identify useful results.

To improve the ability of users to find useful results when searching the web, we present the architecture of a preference-based metasearch engine Inquirus 2, which utilizes explicit user preferences in the form of a category, such as “personal homepages” or “research papers”. Inquirus 2 demonstrates five architectural improvements that enhance the ability to locate and identify useful documents, and to increase performance. The architectural improvements are: an incremental user interface, need-based source selection, source and category-specific query modification, selective downloading of results, and need-based scoring.

A user study was performed to measure the effectiveness of the architectural components and to improve our understanding of user judgments of usefulness as related to topical relevance and document category. This user study demonstrated a bounding relation between the levels of user judgments of topical relevance and usefulness, as well as confirming the effectiveness of our architecture.

We also present the Query Modification Learning Procedure (QMLP), a procedure that automatically learns category-specific and source-specific query modifications, along with experimental results confirming the effectiveness of the procedure.

In memory of my grandparents.

ACKNOWLEDGMENTS

If not for the support of so many people, this dissertation would not have been possible. I would like to thank the wonderful people from NEC Research Institute, Steve Lawrence, C. Lee Giles, Gary Flake, and David Pennock, all of whom provided technical expertise, and provided me an excellent research opportunity. In addition, my committee co-chairs Bill Birmingham and Michael Gordon offered not only writing and academic guidance, but also general support and encouragement.

My survival as a student for so many years was made possible by my friends and family. My good friend Dan DeMaggio not only offered his friendship, but also often provided technical and programming advice, helping me to stay on the correct path towards my dissertation. My good friend Jon Blatt would always listen to my personal problems, giving me hope that some day I might actually finish. My parents and my sister, although hundreds of miles away, always made me aware they supported me, and ensured that I would never give up.

In addition to everyone mentioned above, I would like to thank the following people who assisted me with my writing of my dissertation (in alphabetical order): Jason Benjamin, Marion Bush, Noah Cowan, Eloise Hintersteiner, William Walsh, and Ira Zaslow.

I can’t forget the dozens of volunteers of my user study, who spent their valuable time to provide me with data.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

LIST OF APPENDICES

CHAPTER 1

Introduction

When searching the web, a user strives to find documents that satisfy an information need. An information need is a psychological concept that defines the information a user is seeking [Cooper, 1971, Van Rijsbergen, 1979]. Current web search engines allow a user to enter a keyword query, then return a list of pointers to web pages believed to be *topically relevant*. Not all topically relevant documents are useful.

Unlike most collections, such as an online library catalog or journal, the World Wide Web (WWW) has minimal barriers to content creation; anyone can be a publisher. Publishers vary in expertise, thus, some pages can be of little or no value to readers. However, unlike the WWW, a typical online database will be homogeneous, with a set of items that do not change over time, such as a specific version of a book. Unlike a book or a journal article, a single web page may change, move to a different URL, or be removed from the web. The dynamic and heterogeneous nature of the web make it difficult to build effective search systems.

The low barriers to content creation have encouraged the rapid growth of the web from hundreds of thousands of pages in the mid-1990s to billions of pages in 2001 [Cyveillance Corporation, 2000]. Individual general-purpose search engines have been unable to keep up with this growth; the largest search engine in 1999 demonstrated only an estimated 16% coverage of the indexable web [Lawrence and Giles, 1999a]. Several studies have shown that no single search engine has complete coverage and it is unlikely any single web search engine ever will [Lawrence and Giles, 1998c, Lawrence and Giles, 1999a, Selberg, 1999].

Relatively low coverage of the web by individual search engines spurred research into metasearch engines, or tools that send user queries to multiple search engines and combine the results. Research has demonstrated that combining results, in the form of a metasearch engine, produces a significant improvement in coverage and search effectiveness [Lawrence and Giles, 1999a, Gordon and Pathak, 1999].

Although a metasearch engine improves coverage, it is still limited by the results returned from the search engines it queries. A search engine may impose limits on both the number of results that can be seen, and the information about each result. These limits reduce the ability for a metasearch engine to judge usefulness and to find useful results.

To understand the problem of finding useful documents on the web, we present a simple example.

1.1 A simple example

Two users are looking for useful documents about "wireless networking." Each user's need is different, but the subject query is the same. One wants research papers about how wireless networking works, the other wants product reviews about wireless-networking hardware.

Each user first tries her favorite search engines and fails. Each then tries a metasearch engine, again being unable to find any useful documents, despite the fact that there are numerous research papers and product reviews about wireless networking on the indexable web.

First they try Google, a general-purpose search engine, with over one billion pages in its index[1]. For the query of `"wireless networking"` over 70,000 results are reported. Of those, about 500 of them are research papers, and several thousand are product reviews[2]. Despite the large number of useful results contained in Google's index, none of the top ten results is a research paper or product review. The query interface does not permit either user to describe her need in sufficient detail.

The same query, when sent to the FAST (`alltheweb.com`) search engine, also with over 500 million pages, returns no product reviews or research papers in the top 10. FAST reported over 30,000 results that contained the phrase `"wireless networking,"` many of which are research papers and product reviews. All of the top-ten results are the top-level page of organizations or companies related to wireless networking. As before, the users were unable to specify more than their subject query.

A metasearch engine provides a second option for searching the web. Metasearch engines, such as Metacrawler, have demonstrated significantly higher coverage than any single search engine. On Metacrawler, as with other search engines, there are no research papers returned on the first result page (20 documents, instead of the ten default from the regular search engines). However, the eleventh document is a product review. The title, summary and URL of the eleventh document

[1] As of March 2, 2001, Google reported about 1.3 billion pages, although this number includes many pages that are not fully indexed.

[2] These estimates are based on reported results returned from effective query modifications.

provide no obvious indication that it is a product review, making it unlikely the user would click on that result.

In this example, neither user was able to find any useful documents, in the top ten, for her category-specific information need. When searching a general-purpose search engine, the systems did not correctly identify useful documents, ranking documents that were not useful higher than any useful ones. When searching a metasearch engine, only one useful document was located, but was incorrectly ranked, causing the user to miss it.

This dissertation explores several techniques that improve the ability for a metasearch engine to locate and identify useful documents.

1.2 The Web

The indexable WWW is a very large set of heterogeneous documents estimated to be more than four billion pages in early 2001 [Cyveillance Corporation, 2000]. A web search engine is the primary tool used to locate web pages based on content. In February, 2001 nine of the top ten ranked websites had a general-purpose web search capability on their main page, and four of the top ten are major web search engines [Media Metrix Corporation, 2001]. A typical general-purpose search engine can allow users to search about a billion pages in under one second, returning pages that are expected to be *topically relevant* to the provided query. Most search engines are one-size-fits-all, returning the same set of results for all users with the same keyword query.

Most search engines, like classical IR systems, perform document retrieval based on the query terms provided, attempting to find documents that are "similar to the query." The typical method for doing this, based on query-term statistical information, is called TF-IDF (Term Frequency - Inverse Document Frequency) [Salton, 1989, Van Rijsbergen, 1979]. This has been shown to find documents that are deemed *relevant* to the query. To facilitate research on TF-IDF based methods, every year there are large conferences, such as the TREC conference (TREC is not exclusive to TF-IDF based methods), designed to test various algorithms [Harman, 1994]. TF-IDF based algorithms rely on the *similarity assumption*, which states that documents that are "similar to the query" are likely to be useful for the user's need [Voorhees, 1985]. Unfortunately, this has not yet been shown to be generally true for the web, especially when users have a specific search context. Recent research on the web has shown a large difference between documents that are similar to the query and those that are useful [Budzik et al., 2000]. In addition, many new web-page ranking algorithms consider factors other than the keyword contents of the page, suggesting there is more to usefulness than keyword similarity [Brin and Page, 1998].

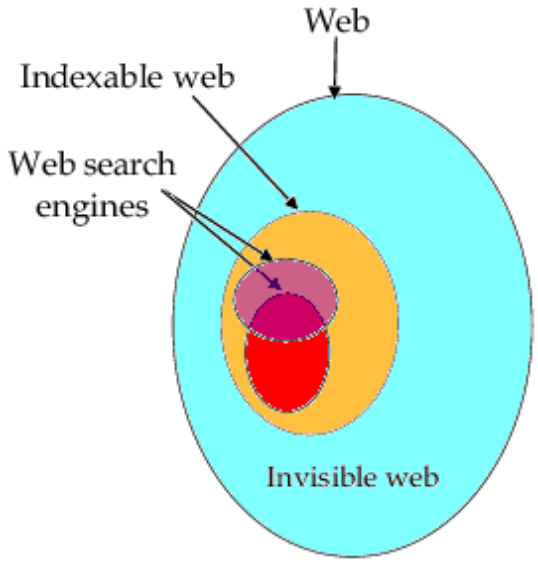

Figure 1.1: The whole web is both the invisible web and the indexable web. Individual search engines only cover some of the indexable web.

The ability for a search engine to return useful documents is limited not only by its ability to identify a document as useful, but also by the requirement that it contain the useful document in its index. The size of the non-indexable web, sometimes referred to as the "hidden web," "deep web" or the "invisible web," is estimated to be as much as 500 times as large as the current search engine's databases, or roughly 500 billion pages, and the large size and rapidly changing contents are at least in part responsible for many search failures [Guernsey, 2001]. The "hidden web" refers to the set of pages that are either not permitted to be indexed[3] or to the set of pages that are not yet indexed, probably due to no inbound links or new pages that did not exist when the previous crawl was made. Figure 1.1 shows the indexable web as distinct from the invisible web. Figure 1.2 shows how a useful document might not be in the index of a search engine, or may fall outside the indexable web.

Even though no single search engine has more than 16% coverage of the web, combining several search engines through the use of a metasearch engine can result in a significant increase in coverage (from 16% to 42%) [Selberg and Etzioni, 1997, Lawrence and Giles, 1999a, Lawrence and Giles, 1998a]. Unfortunately, higher coverage is not a guarantee that users will find useful results. A typical metasearch engine makes ranking judgments based only on the limited information returned by the search engines they query, limiting their ability to accurately predict usefulness.

[3]Although there are few technical means to prevent indexing of any user viewable page, most web crawlers voluntarily follow the robots exclusion protocol, choosing not to index pages that system administrators specify should not be indexed. [Koster, 1994]

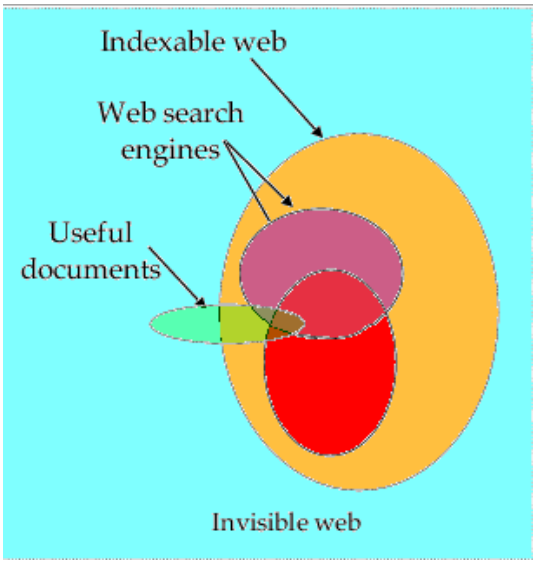

Figure 1.2: Useful documents may be split among multiple search engines, and may even come from the invisible web.

1.3 Contributions

A user searching the web with a category-specific need may be unable to find useful documents using a web search engine. This failure may result from insufficient coverage, or an inability to identify accurately a document as useful. A metasearch engine has been shown to provide higher coverage, but is limited in the information available about results, reducing the ability to judge usefulness. A document that is 'similar to the query' (*topically relevant*) is not necessarily useful.

We present four contributions:

- We performed an empirical user study comparing user judgments of usefulness with user judgments of topical relevance. The study data produced a bounding relationship where the level of the topical relevance judgment provides an upper bound to the level of the usefulness judgment. Our study also quantified the significance of user category in differentiating between highly topically relevant but useful documents from highly topically relevant, but not useful documents.

- To take advantage of the influences of category on the usefulness of web documents, we developed and evaluated an automated algorithm called Query Modification Learning Procedure (QMLP). Query modifications learned using QMLP demonstrated an on-category result rate of 75%, significantly higher than for unmodified queries. QMLP produces a ranked set of search engine, query-modification pairs, without requiring special knowledge of the inner

workings of the search engines, starting with only a set of classified documents and a few test queries.

- To improve web-based metasearch, we developed, implemented, and tested a preference-based metasearch engine that utilizes category throughout the search process. Our preference-based metasearch engine, Inquirus 2, incorporates need-based source selection, source and category-specific query modifications, and need-based scoring. Each of these improvements directly considers user categories, despite the fact that the underlying search engines queried may not.

- We augmented our preference-based metasearch engine to improve search efficiency by combining a selective download module with need-based source selection, source and category-specific query modifications, and an incremental interface. Selective download of results improves the ability to predict usefulness, while not wasting resources if sufficient information to judge usefulness is available. Combining selective download with an incremental interface allows immediate display of results, while the search is still progressing. In addition, we have preliminary data suggesting that selective download can improve the efficiency of web-based metasearch.

1.4 Related Work

There are three areas of related work. First, we describe methods, other than a search engine, for finding web pages. Second, we discuss several studies which have been performed to understand search and metasearch engines better. These studies provide insight into how search engines are used, such as discovering the average user query is about two terms. Third, we discuss the concept of *relevance* in information retrieval, focusing on works related to distinguishing between *topical relevance* and *usefulness.*

1.5 Finding Materials on the Web

One of the first web search engines was The World Wide Web Worm (WWWW), with an index of 110,000 web pages and web accessible documents [McBryan, 1994]. Since 1994, the size of the (indexable) web has grown significantly, estimated to cross 4 billion pages in early 2001, and the variety of materials has also grown with the advent of FLASH, JAVA and JavaScript, as well as with improvements in streaming media, more powerful web servers, and increased integration of dynamic contents.

Chapter 2 describes a model of a web search engine, and the important aspects as related to finding useful documents. In addition to web search engines, there are other ways to locate web pages based on content. One of the components of a web search engine is the crawler, described in detail in Section 2.1.5. A focused crawler is a tool that automatically explores web pages seeking a specific type of content [Diligenti et al., 2000, Chakrabarti et al., 1999]. Unlike a web search engine, a focused crawler cannot return results instantly for a query, but rather finds them over time, possibly hours or days.

Another approach to finding web pages, based on content, is through browsing. Some sites, such as Yahoo, have created a browsable index of content, organized in a hierarchy. A user can follow links, organized by subject, until she homes in on an area that is interesting. Unlike a web search engine, a browsable index can be used without an explicit query. The Yahoo index is human generated and human maintained; however, there has been research on automated hierarchy creation for web content [Chakrabarti et al., 1998b]. There has also been research on browsable interfaces or methods for using browsing technology to improve web search [Lieberman, 1995].

A third method for locating web pages is through a metasearch engine. A metasearch engine is a tool that allows user queries to be sent to multiple search engines as a means of improving consistency and coverage [Selberg, 1999, Lawrence and Giles, 1998c, Gordon and Pathak, 1999]. Several metasearch engines are in use today, including several research metasearch engines. Some commercial metasearch engines include Sherlock by Apple, CNET's main search page, and DogPile. Popular research search engines include SavvySearch [Howe and Dreilinger, 1997], ProFusion [Gauch et al., 1996], Metacrawler [Selberg and Etzioni, 1997, Selberg, 1999], and Inquirus [Lawrence and Giles, 1998a]. It is estimated that there are more than 1200 metasearch engines available on the web today.

There are several types of metasearch tools. Tools such as Metacrawler [Selberg and Etzioni, 1997, Selberg, 1999] are centralized metasearch engines, in that all users send their queries to one location. Tools such as Apple's Sherlock are considered personal metasearch agents, since they are installed and run on a user's machine. Another class of metasearch tool includes just-in-time information agents, such as Watson [Leake et al., 1999, Budzik and Hammond, 1999]. A just-in-time information agent will consider user context when making search decisions. There are also non-realtime metasearch tools such as Karnak `http://www.karnak.com/`, a system that sends e-mail if new sites are found for a standing query. Although there are many different classes of metasearch tools, each performs the same set of logical tasks. Chapter 3 describes these tasks and goes into more detail on the issues and challenges for metasearch engines as related to finding useful documents.

A web metasearch engine is a specific type of search engine that utilizes *web* search engines as the data source. One factor that separates a web metasearch engine from other types of distributed information retrieval (IR) is the issue of control. A web metasearch engine does not have control of how its queries are processed by other search engines, nor does it control the information available in search engine responses. Research areas such as distributed databases or query routing may be applicable to the design of a metasearch engine, but are distinct because they often make the assumption of explicit control of the data. Distributed information retrieval refers to the general class of research related to retrieving materials from different (either physically or logically) locations.

Another related area is digital libraries, concerned with the ability to retrieve information goods, often in a distributed fashion [Birmingham et al., 1994]. In the 1980s there were library systems such as WAIS that enabled the querying of many different library catalogs [Davis et al., 1990]. Before the web, protocols such as Z39.50 were developed to enable the querying of different, remote library systems, although typical interfaces allowed selection of only a single database at a time [National Information Standards Organization, 1995].

1.5.1 Digital Libraries

In the mid 1990's there was significant US Government interest in digital libraries. Several of the government funded University projects focused on the concept of distributed IR. Two of the more notable projects include the University of Michigan's digital library [Atkins et al., 1996, Birmingham et al., 1994], and the Stanford Digital Library project [Paepcke et al., 1997]. Both focused on the use of *agents* as tools for performing several distributed search tasks. Several web related search technologies were spawned from these digital library projects, such as FAB, which combined social filtering and word-vectors to make recommendations of web documents [Balabanovic and Shoham, 1997]; and early work on PageRank [Brin and Page, 1998], which later became the basis of the Google search engine. The specific issues for agent-based IR and the relationship between an agent-based digital library and a web metasearch engine architecture are discussed in detail in Section 3.1.6.

1.5.2 Distributed Databases and Query Routing

Two research areas related to metasearch engines are distributed databases and query routing. The first concerns how to distribute content among several databases and the second addresses how to select which databases to query.

A metasearch engine is concerned with finding content, and not necessarily with how to distribute it.

Given a set of already distributed databases, query routing or source selection is an approach to the problem of choosing which databases to query. One of the most widely researched algorithms for database selection is GLoSS [Gravano et al., 1998]. In simple terms GLoSS uses database statistics to predict the likelihood that each database will contain results that contain all the query terms (assuming an AND query). This approach has been shown to be effective when dealing with homogeneous or small distributed text collections, but the effectiveness can be reduced with heterogeneous collections, such as a web search engine's database.

One of the problems with approaches such as GLoSS is the need for database statistics. Although the statistics of the database are only a fraction of the size of the full-text database, it might not always be possible to have access to such information. When dealing with web search engines through a search interface, it is impractical to obtain full statistical information. However, some research has experimented with using test queries to generate summary statistics of web search engines. More recently, Craswell experimented with using proxy statistics to permit a metasearch engine to compute an approximation for TF-IDF, demonstrating that it was more effective than using only local document term statistics [Craswell et al., 1999].

Systems that perform multi-database querying, like systems that may distribute databases, have different problems and different architectural issues than web metasearch engines. When querying databases directly, the search system has implicit control over the form of the query, and has access to all the results in the database. If a query returns 1000 results, it is reasonable to access or view every result; however, if a web search query returns 100 results, it might be unreasonable to view them all through a search engine interface.

1.6 Studying Web Search Engines

To understand how web search engines are used, and their effectiveness, several studies have been conducted. Jansen presents a review of web search studies and makes some suggestions for future research [Jansen and Pooch, 2001]). The three primary studies discussed by Jansen all fall into the category of *transaction log analysis.* In addition to transaction log analysis, there have been some studies that obtain explicit user judgments.

The three primary studies described by Jansen include: *The Fireball Study* [Hoelscher, 1998] , *The Excite Study* [Spink et al., 1999a, Spink et al., 1999b], and *The Alta Vista Study* [Silverstein et al., 1999]. Each of these involved analysis of logs of high-traffic search engines and provides insight into how users use search engines. The important observations from these studies are summarized below:

- Users enter short queries. The average query length is between 1 and 3 terms, with very few users using five or more terms
- Most users did not take advantage of advanced features such as Boolean or quoted phrases
- Limited use of query reformulation and relevance feedback
- Few advanced searches

The primary contribution of such studies is a better understanding of the nature and types of queries entered and of the features utilized. Transaction log studies are unable to make explicit statements about effectiveness or user satisfaction for a particular search. As Jansen did for web search studies, Tonta provided a survey of studies related to search failures of document retrieval systems [Tonta, 1992].

In addition to transaction log studies, several studies have been performed to analyze the effectiveness or properties of specific search engines. Lawrence and Giles studied several search engines to make statements about their overlap to predict coverage and web size [Lawrence and Giles, 1998c]. Gordon and Pathak studied how well each search engine performed for trained experts searching on behalf of actual users [Gordon and Pathak, 1999]. Their data also supported the basic assumptions described by Lawrence and Giles and others regarding the low overlap between search engines. A related study includes the work by Nick Craswell analyzing the effectiveness of several ranking algorithms for metasearch engines [Craswell et al., 1999].

The TREC (Text REtrievaal Conference) conference, held every year, creates test collections to enable comparison of various ranking algorithms [Harman, 1994]. Recently a web track was added to enable studying of ranking algorithms for web pages [Hawking et al., 1999].

Unlike transaction log analysis, we designed our user study to collect actual user judgments of usefulness. We also instrumented our system to collect system performance data, permitting analysis of each component of our architecture. Our user study data confirms the observations of several studies on search and metasearch engines.

1.7 Relevance, Usefulness and IR

There is philosophical disagreement regarding an exact definition of relevance in information retrieval, and it is thought to be multi-faceted [Mizzaro, 1997, Saracevic, 1975, Frants et al., 1996, Schamber et al., 1990]. “Relevance” is often used to describe the agreement of a document with a user’s information need. In other contexts, relevance may mean “answering the question” or “satisfying the query.” The basis for computing relevance is often the similarity between a document’s

description and a query [Van Rijsbergen, 1979, Grossman and Frieder, 1998, Salton, 1989]. Such psychological definitions differ from the computation of relevance, which often is based on the similarity between a user's query and a document description.

Several papers make a clear distinction between the subjective notion of *usefulness* (or utility) and the less subjective notion of *topical relevance* [Mizzaro, 1997, Cooper, 1971, Cooper, 1997, Kochen, 1974, Frants et al., 1996, Saracevic, 1975, Schamber et al., 1990, Budzik et al., 2000, Budzik and Hammond, 1999, Lawrence, 2000, Cole et al., 1996]. Cooper in 1971 described a logical definition of *relevance*, and went on to say that what really mattered was *utility*, or usefulness [Cooper, 1971]. Manfred Kochen described a formal model for computation of utility of documents, and stated that *relevance* is different from utility, and utility subsumes relevance [Kochen, 1974]. Saracevic describes the concept of *relevance* and a survey of the important works related to relevance [Saracevic, 1975]. More recently Mizzaro provides a detailed history of relevance in IR [Mizzaro, 1997]. Cole describes how undergraduate students' notions of what is useful may change throughout the search process, even though they are still looking for materials on the same subject [Cole et al., 1996, Bates, 1986]. Jay Budzik with the Watson project discusses the goal of just-in-time information agents as being one of locating materials that are useful with respect to the current search context [Budzik et al., 2000]. Budzik also describes a small study confirming that user judgments of what is useful are distinct from *similarity to the query*, a proxy for topical relevance. Lawrence describes the importance of context in web search [Lawrence, 2000]. Schamber describes the notion of old relevance as being topical in nature, and new relevance as user-centric [Schamber et al., 1990].

Cox describes the notion of indexing documents based on their expected usefulness for a given query, putting the burden of determining what is likely to be useful on the indexers [Cox, 1994]. To improve IR, several researchers have proposed consideration of non-topical attributes when scoring documents. Buckland, as part of the OASIS project, describes the use of date, language, and physical location of a book as factors used to reduce information overload [Buckland et al., 1992]. Boyce states that to return useful materials, the informativeness or uniqueness of each item with respect to all other items to be returned must be considered [Boyce, 1982]. Croft describes a system that considers many factors in addition to topic for retrieving documents. A user could specify that she was interested in books about a specific subject, but was unsure of the exact year [Croft et al., 1990]. The Decision-Theoretic Interactive Video Advisor (DIVA) project used an explicit utility function over many factors when recommending movies of interest [Nguyen and Haddawy, 1998]. Recently Zhu and Gauch quantified the improvement in effectiveness of a web search for each of six non-topical quality metrics [Zhu and Gauch, 2000].

Our work focuses on the distinction between topical relevance and usefulness, and through our user study attempts to explicitly quantify such differences. When scoring documents, several factors in addition to the topic are considered. Likewise, extra-topical factors are considered when choosing search engines and when choosing query modifications.

1.8 Organization of this Document

This dissertation is divided into seven chapters and four appendices. Chapter 2 describes the architecture of a web search engine, the formal definition of the *web search problem*, and several fundamental problems that limit the ability for a web search engine to return useful documents. Chapter 3 describes the architecture of a web metasearch engine, the formal definition of the *web metasearch problem*, and several fundamental problems limiting the ability for a metasearch engine to return useful documents. Chapter 4 describes the architecture of Inquirus 2, our preference-based metasearch engine. Chapter 5 describes the design and results of our user study. Chapter 6 describes Query Modification Learning Procedure (QMLP), an algorithm for learning category and search engine specific query modifications. Chapter 7 presents a summary and conclusion of our work.

CHAPTER 2

Search Engines

A web search engine is the primary tool used to map between a searching user's desired contents and a URL of a page that is likely to contain those contents, or satisfy a given need. First, we present a general architectural model of a web search engine. Then we formally define the *web search problem*. Next, we present several problems faced by a web search engine that limit the ability to return useful results.

2.1 Web Search Engine Architecture

A web search engine must present a user with a set of results, given her input. There are five logical tasks a search engine performs for each search. Each task corresponds with a specific component of the architecture presented in Figure 2.1.

Five tasks every search engine performs for each search:

- Accept user input
- Process user input
- Apply database query
- Process results
- Display results

Figure 2.1 describes the four required components of a web search engine and the crawler for populating the database. The first component is the *user interface,* which is responsible for accepting user input and presenting the output. Second is the *query processor,* which generates a reasonable database query from the user input. Third is the *database,* which is the component that stores the

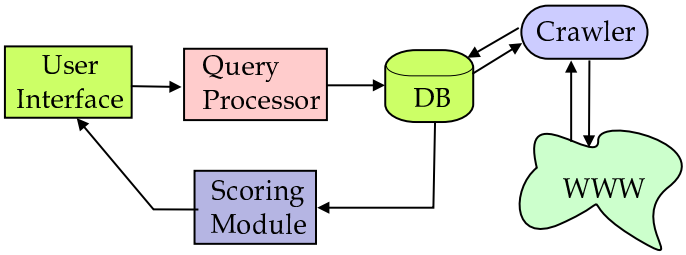

Figure 2.1: Architecture of a web search engine.

knowledge about each result. Fourth is the *scoring module*, which processes each result before sending the result to the user interface for display. In addition to these four components, most web search engines have a *crawler*, which is used to populate and maintain their database.

The main difference between a web search engine and other IR systems is its scale and assumptions about the data. A typical IR system such as an online catalog of journal articles is made up of contents which do not change over time and are often homogeneous. Web pages can be located on any machine on the Internet, and can be written by anyone about anything. A web search engine must be designed to handle many documents and arbitrary contents. A web search engine must also be designed to handle the highly distributed and dynamic nature of the web, not typically present in other online collections.

2.1.1 User interface

The user interface of a web search engine has two tasks:

- Accept user input for use by the system
- Present results to the user

Each of these functions plays an important role in the operation and usability of every search engine. We define the first task as the *input user interface* and the second as the *output user interface.* The input user interface defines what types of information a user can provide. The simplest types of interfaces allow only a keyword query, while more complex input interfaces may allow users to choose options from a list, provide extra context information, or even track user actions. The goal of the input interface is to get as clear a description as possible of the user's information need.

From our original example on "wireless networking", two users with different needs may enter the same query. Without extra information, it is impossible to distinguish between the needs of the users.

There are many research projects on improving web interfaces. Recent work by Marti Hearst describes the importance of a well-designed interface, and provides recommendations on how to design specialized search interfaces for individual sites [Hearst, 2000].

In addition to normal query interfaces, there are research projects on utilizing user feedback or monitoring user actions. The company Direct Hit keeps track of which results users click on for each query, and utilizes this information to provide an improved ranking. Belkin describes a query reformulation system wherein users refine their query based on features extracted from results found so far [Belkin, 2000]. His work concludes that users who are trained on what the system does are more likely to utilize the advanced features. He also noted that sometimes the features that were most effective might seem strange or confusing to users.

User input is more than just the keywords presented. Some search engines such as Northern Light allow users to specify a search category before the search, or choose a cluster after the search. The Watson project considers the current user's context in a word processor when performing searches [Budzik and Hammond, 1999, Leake et al., 1999].

The second task of the user interface is to present results to the user. The output user interface affects how easily a user can identify *useful* documents. Since the web is heterogeneous, and there is no widely adopted metadata standard (some have been proposed and are in limited use, such as XML, or Dublin Core), it is difficult to describe output. A database of books has standard fields such as title, author, etc. A standard web page does not necessarily have an author, or the title may be absent or unrelated to a web page's contents. Barry identified several criteria users considered when making relevance judgments [Barry, 1993]. These factors may be dependent on the needs of the user. A user searching for research papers may desire an abstract, or equations, or figures; a person searching for recent news may want to know the date of the contents and the credentials of the author.

Some works on output interfaces attempt to provide various ways of visualizing the results. Veersasamy describes work on providing extra visual clues to users to help users determine which documents meet their particular needs for a given multi-term query [Veerasamy and Belkin, 1996]. Such output visualization tools may also improve a user's ability to formulate or reformulate their queries through a feedback or an interactive interface.

There are many research projects and organizations dedicated to understanding and creating good user interfaces for web and other information retrieval tools. In our work, we will focus on specific aspects of the user input, such as their subject query and the desired category. For the output interface, we focus on specific properties, such as incremental display of intermediate results.

2.1.2 Query Processor

User input is a representation of an information need. The query processor must convert this input into a database query (or set of database queries) for use by the search engine. Unfortunately, users typically don't enter explicit database queries.

There are several consequences to improperly (not well formed) formulated database queries. If the query formulated is too general, the system may be forced to process an excessive number of results. Even though a typical search engine may report hundreds of thousands of web pages containing a given query, the system does not actually process each of them. Google, for example, will first try to retrieve documents containing all query terms in titles only before searching in the full text [Brin and Page, 1998]. Too general a query might return many documents that, although they contain the user's query terms, may be of little or no value. For example, a user searching for `''fish''` would be likely to find a very long page, mentioning "fish" once near the end of the document.

A database query that is too specific may result in failure to return desirable documents. A user entering a query `lake michigan wildlife fish birds plants` might find a document that is about the plants and animals of Lake Michigan useful, even if it does not mention the word *wildlife*.

The query processor of some search engines has the ability to generate database queries that are different from the query terms entered by the user. For example, some search engines will perform stemming (treating plural words and singular words the same). Some search engines, such as Excite, interpret a user's query conceptually, identifying words of similar concepts as potentially useful, such as *car* and *automobile.* Altavista and Excite also automatically identify phrases. If a user enters the query `lake michigan`, the system treats the query as a phrase, even though no quotes were used.

Even more advanced systems, such as AskJeeves, allow *natural language* queries, and must formulate reasonable database queries to ensure retrieval of meaningful documents. The database used by AskJeeves is likely to be different from those of other search engines, optimized for specific types of queries, as opposed to normal full-text retrieval.

Unfortunately, for a variety of reasons, search engines do not generally reveal the methods they use for their query processors. The failure to publish specifications harms research, and can make formulation of effective queries more difficult.

2.1.3 Database

A web search engine can only return documents it knows about. The database is the collective local knowledge about the documents on the web. Currently some search engines boast over one billion pages in their index, suggesting a very large database.

The query processor formulates queries to the database to produce a reasonable number of documents for scoring. Brin describes some of the issues for design of a large-scale search engine, including some specific database design requirements [Brin and Page, 1998].

There are several properties of web search engine databases that differentiate them from most databases for non-web online retrieval systems.

Properties of databases of general-purpose search engines include:

- Very large size- currently estimated to be on the order of ten terabytes
- Contents added at extraordinary rate - the web is estimated to double every 18 months
- High performance - typical searches take under one second
- Dynamic contents - web pages change, some on an hourly or more frequent basis

A web search engine database determines what (local) documents can be returned to a searching user. One reason a user might not find a useful document for a given search is the absence of that document in the search engine's database, also called the *coverage problem* [Selberg, 1999, Lawrence and Giles, 1998c, Lawrence and Giles, 1998b, Guernsey, 2001]. A second reason for search failure is the wrong document returned due to a difference between the actual page contents and the version in the database; referred to as the *coherence problem*. The coverage problem is covered in more detail in Section 2.3.2, and the coherence problem is described in Section 2.3.1.

2.1.4 Scoring Module

Documents returned from the database are ranked by an *ordering policy*. The scoring module determines how documents are scored, and ultimately how they are ranked.

Ordering policy refers to the method used by a search engine to produce a ranking of results.

There are many factors that can be considered when ranking a document. Most classical IR systems use some version of TF-IDF [Van Rijsbergen, 1979, Grossman and Frieder, 1998], where the retrieval status value, or score, of a document is determined based on the term frequency of the query terms in each document relative to the number of documents in the collection (database) that contain that term.

Simple TF-IDF ranking considers only individual keywords, or possibly phrases. Web pages, however, have many features not found in plain text collections, e.g. images, anchortext, link structure, headings, tables, etc. In addition, the large size and wide variety of types of pages makes TF-IDF less effective.

Most search engines closely guard their specific scoring functions. However, a few publications have revealed some of the parameters considered. Mauldin described how Lycos ranked pages (it is not clear that these same rules apply today): factors considered included the specific terms and their placement in the document; terms in the title or top of the document were given more weight than terms found elsewhere in the document. Terms deemed to be the most significant for describing a document were also promoted. There have been some attempts to study various word frequency and location-based ranking methods for web pages as part of the TREC-7 conference [Craswell et al., 1999]. Most TF-IDF methods do not directly consider the location of words or their position relative to other query terms.

Unlike plain-text documents, web pages contain many structural features[1], such as font sizes and colors, headings, boldface, image words, anchortext, titles, meta-tags, etc. It is strongly believed (as indicated by the recommendations of the search engines for how to make pages rank higher) that search engines do consider such tags when ranking pages. One example is that query words of the same color as the background, or in small font, are likely to be given less weight than terms in a title or heading. Many search engines also recommend the use of "meta-tags" or metadata that can be used to help describe a web page. There are several meta-tags specified for use in describing a web page's content. Pages that contain query terms in these meta-tags will almost always rank higher, for search engines that consider meta-tags, than pages that do not; however, only about one third of web pages use them [Lawrence and Giles, 1998c].

Recently, the structure of the web has been used as a ranking factor. Since web pages are interlinked, it is likely that pages that link to each other may be related [Davison, 2000]. Similarly, pages that are linked to very frequently are likely to be more popular, or more authoratitive. Kleinberg uses web link structure to produce sets of *hubs* and *authorities* [Kleinberg, 1999]. Graphically, a hub is a page that is linked to by many authorities, and an authority is considered a page that links to many hubs. In our society, there are instances of hubs and authorities, such as: a specific researcher might be an authority on some research area because she has had many published papers. Kleinberg's work is similar to work on citation analysis used to analyze academic publications [Lawrence et al., 1999]. Another use of the link structure is as an approximation of a web

[1]There are options for structure, but there is no standard metadata currently followed by a large percentage of web page designers.

page's popularity. The search engine Yahoo is probably more popular than Metacrawler and, because of that, people are more likely to link to `http://www.yahoo.com/` on their pages than to Metacrawler. The PageRank algorithm, based on a random walk of the web, produces a numerical score for each web page based on the link structure [Brin and Page, 1998]. This algorithm is the heart of the Google Search Engine. A third use of link structure can be as a method to help determine a description of a web page based on the anchor text of inbound links. For example, a person linking to Bob's homepage might specify anchortext "Bob Smith's homepage", even though the page linked to might not mention "Bob Smith" or the word homepage. Similarly, the search engine Yahoo might not say "search engine" anywhere, but a person linking to it might. This technique is also used by Google to help predict *useful* pages.

Other factors for scoring of results that have been proposed include the relative uniqueness of the information on a given result, or *informativeness* [Boyce, 1982]. Informativeness refers to the amount of "new" information in a given result relative to the other results. Some systems cluster results and may consider the relative clusters when scoring results or when ranking them [Zamir et al., 1997]. Some search engines cluster results based on the domain from which they originate, hoping to provide a wider variety of pages to the user. Northern Light organizes results into "custom folders" based on a patented clustering algorithm.

The scoring module produces a score based on the available information about each result and the user's input. A scoring module that can score each result independently of the other results has the property of *independent result scoring.* Algorithms such as those proposed by Kleinberg require the entire set of data to be retrieved before any score can be computed. If a scoring module has the property of independent result scoring, it can utilize an incremental interface, in which the returned scores do not change as new results are "found". Section 3.8.4 describes this property in more detail, as well as the implications for the user interface.

Scoring modules for current search engines are believed to be based on a combination of factors, both topical and non-topical. The primary components, besides raw text, appear to be link structure, page depth (how far down from the main page of a site), user supplied metadata, and page structural information (title, headings, font color, etc.). A poor representation of a user's need can result in an inability to score a given document properly.

2.1.5 Crawler

A web search engine is a tool that permits users to locate web pages based on content. This content mapping is done through a database of web pages. Most general purpose search engines populate their database through the use of a crawler, also called a web robot. A crawler explores

the web by downloading pages, extracting the URLs from each explored page, and adding the new URLs to its crawl list. Since the web is not static – both pages and links change – and growing at a fast rate, a crawler must make decisions about which pages to examine, as well as which pages to index. Indexing is the, often complex, process of adding a page to a search engine's database. Indexing involves parsing the document using many advanced algorithms to improve the efficiency retrieval. Selberg describes the indexing process and provides several references in [Selberg, 1999].

Tasks of the crawler:

- Retrieve web pages for use in the database
- Minimize the resource requirements – this can include CPU, RAM, disk, network, and time
- Properly balance discovery of new contents with updating of existing contents

The simplest crawler can be thought of as a search algorithm. Beginning from a single page, p_0, download the page, extract the URLs $\{p_1, p_2, \ldots, p_n\}$, then download the new URLs and repeat. The specific ordering could be as simple as a breadth-first search, or possibly some form of best-first search.

The basic purpose of a crawler is to retrieve web pages for incorporation into the database. Given a responsive, finite, and static web, one could imagine a simple crawler search resulting in the discovery of all web pages (for which there exists a path from the start node). Unfortunately, the web is large (effectively infinite), individual pages are dynamic and the bandwidth to any given server is limited. It takes a non-trivial amount of time and network resources to download (retrieve) any given URL, and the contents of any given URL could change at any time. To further complicate the task of crawling, there are many *undesirable* pages, even for a general purpose search engine. There are web pages that produce streams of random data, or pages that are exact duplicates of others, as well as wide variations of quality or significance of desirable pages. A general-purpose search engine strives to cover every subject; however, the usefulness of the search engine is based on its ability to cover pages searchers desire. In addition to general-purpose search engines, there are special-purpose search engines. A special-purpose search engine is a search engine that covers only a specific area, such as research papers or news.

There are two basic types of crawlers, focused and general-purpose. Focused crawlers attempt to minimize the resources required to find web pages of a specific category. Recent research has shown that a focused crawler can offer a significant advantage in resource expenditure when compared to a non-focused crawler [Diligenti et al., 2000, Chakrabarti et al., 1999]. Even for general-purpose search engines, an intelligent crawler can significantly reduce wasted resources. Cho describes how

to design an efficient crawler by using URL ordering in [Cho et al., 1998]. The basic approach is to create a performance metric and optimize the URL ordering with respect to that metric. Some metrics include the PageRank algorithm [Brin and Page, 1998], back link count, forward link count, and a location metric.

PageRank is an algorithm, based on a random walk of the web, that attempts to estimate the relative popularity of a web page. Back link count is the number of links pointing to a particular page. Forward link count is the number of links leaving from a specific page. Location metric refers to properties of the specific URL, such as: *.com* may be more important than *.net*, or a URL that contains the string "home" may be of more interest than other strings. Location metric may also consider page depth (the number of slashes in the URL).

The ordering metric for a focused crawler is typically a function of the contents of the pages, as well as of the link structure. A focused crawler, although designed to find appropriate pages, may also wish to find pages that are not appropriate for the given collection but that link to pages that are appropriate.

The second challenge for a crawler is related to resource requirements. Unfortunately, it is not "free" to download a web page; resources may be required on the part of both the crawler and the servers serving the provided web pages. A company might not mind if their web site is crawled, but they might be upset if a crawler tries to download all hundred thousand of their web pages at once. The network resources are also important since a crawler might have a high-speed network link, but a target domain may not. Requesting dozens of pages that all go through a single network link can cause network problems[Koster, 1995]. In addition to minimizing network and server resources, local resources may be limited. If we assume there are 4 billion URLs, it can be difficult to design a system to keep track of the current set of crawled pages, the remaining pages, and the path being explored.

The third challenge of a web crawler is to ensure that pages are appropriately revisited. Since contents of a web page can change, it becomes necessary to revisit web pages. The greater the number of pages in a given database, the more resources must be expended to ensure that they are all kept up to date, further burdening the crawler. Edwards describes the issues related to balancing the rate of crawling of new pages with the re-visiting of pages that are likely to have changed [Edwards et al., 2001].

Let:

W be the set of all Web documents.

$r(Q, x)$ be a Boolean relevance predicate defined by the user

Given:

A set of Web documents $\mathcal{S} \subseteq W$.

A query Q.

Determine:

The set of relevant documents R defined by:

$\{x \in W : r(Q, x) = True\}$

Figure 2.2: The web search problem as defined by Selberg [Selberg, 1999].

2.2 Web Search Problem

The web search problem is the challenge of a web search engine both to find (likely through a crawler) useful pages, and to be able to return them to a searching user based on the provided user input. Selberg defines the web search problem as one of returning a set of *relevant* web documents from a subset of all web pages. Figure 2.2 shows his definition.

His definition addresses the basic goal of finding *relevant* documents. However, recent work, including this dissertation, shows that there is a difference between topical relevance and *usefulness.* A topical two term query may be insufficient to determine if a user will deem a document useful. In addition, the goal of his definition is to determine the entire set of relevant documents from some subset of the web. As described in section 2.1.3 and 2.1.5, a web search engine can influence the set of documents it has in its database. As a result, we feel the web search problem must include both the notion of usefulness, which is subjective and based on more than the topical query, and the fact that a search engine can influence its own contents. Figure 2.3 proposes our extension to the web search problem.

Our definition of the web search problem can be summarized as locating the n most useful documents, as judged by the user, for some information need. To do this, a search engine must create a database of web documents from which to choose results. The selection of D determines the maximum quality of the result set. A small D may miss many of the best documents. The best possible is where the n best documents from D is the same as the n best from W, the whole web.

It might be possible to produce the n best documents without having a database that covers the

Let:

W be the set of all Web documents.

$U(d)$ a utility function of a given document for the user's information need

Given:

A specification of a user's information need, I (typically a query)

A maximum number of documents to be returned, n

A set of documents, the database, $D \subseteq W$. The contents of D is determined by the policy of the search engine

Determine:

From $\mathcal{D}$, using I, determine a set of results $R = \{d_1, d_2, \ldots, d_k\}$, $|R| \leq n$, $R \subseteq D$, where R contains the n best documents from D as defined by $U(d)$

Figure 2.3: Our definition of the web search problem

entire web. Some special-purpose search engines offer very high coverage of a limited area, and a user searching in this area is less likely to desire documents outside of the area.

It is important to distinguish between the goal of finding the n best documents as opposed to the best set of n. Using a utility function over each document will only result in the set of the best n if the utility of any given document is independent of other documents. Without independence, the best set of n may exclude documents that, when compared individually, are preferred to at least one document not included in the set. An example of this is near duplicates. For example, imagine the world contained only three documents, d_1 d_2 and d_3. A user may state that d_1 is preferred to d_2, and both are preferred to d_3 but when asked to pick a set of the two best, she might choose d_1 and d_3, if d_1 and d_2 are very similar. Given d_1, the marginal utility of d_2 is less than that of d_3.

Boyce suggests considering *informativeness* when scoring documents [Boyce, 1982]. Informativeness is a function of the contents of all retrieved documents, and if significant may suggest that a single document cannot be scored accurately without considering the entire returned set. A user wants to locate useful documents; we argue that a system that finds many useful documents, as judged individually, has succeeded. Simple rules can be applied to remove exact or near duplicates. To find the best set of documents for a task is a different and more difficult problem.

2.3 Fundamental Problems for Web Search Engines

As described by the web search problem, a web search engine strives to locate the n best documents for a given information need. There are several fundamental difficulties limiting the ability of a search engine to return useful documents. First, users don't speak in information needs or utility functions; they communicate an approximation of their information need to a search engine through an interface. Unfortunately, modern interfaces are not very good at capturing an accurate description of a user's information need. In fact, there is a tradeoff between a simple interface (easy to use) that can capture minimal information, and a more advanced interface, more difficult to use, that can allow for a more accurate description of the user's need; this is called the *interface problem*. A web search engine also strives to have all useful documents for all possible information needs; in other words, it tries to index the whole web. To try to possess all documents is called the *web discovery and coverage problem* and the problems associated with document contents changing is called the *coherence problem*.

2.3.1 The Coherence Problem

Unlike most online retrieval systems, a web search engine does not control the pages it indexes. A web page can change at any time; it is the burden of the search engine to recognize such changes. The coherence problem is that of keeping the database contents consistent with the pages to which they refer. For most web search engines, the crawler is responsible for re-checking pages [Selberg, 1999].

The coherence problem affects the ability of a search engine to return useful results because inconsistencies between the database and actual page contents can cause useful pages to be missed, or pages that are no longer useful to be returned.

In addition to page contents changing, sometimes pages are removed. When a search engine result refers to a non-existent page, it is called a *dead link*. Dead links are annoying for users and are one factor used to describe the effectiveness of a search engine – the fewer the dead links the better.

There are several approaches to reducing the coherence problem, or to reducing the consequences of it. The actual problem is generally attacked by recrawling web pages more frequently. Recrawling every page in order can reduce the resources available to the crawler for discovering new contents, potentially lowering search engine coverage. A more effective method is to recrawl pages intelligently based on their expected change frequency and importance [Seltzer et al., 1997]. Some sites use the coherence problem as a method to produce revenue. For example, Inktomi charges web

sites extra if they want to be recrawled more often.

In addition to the research on recrawling, there has been research on developing protocols or methods for notification of content changes. Such methods could also include notification of the addition of new contents. Unfortunately, most system administrators do not use any type of notification system, and it is unlikely a single standard will be adopted.

There are also several approaches to hiding the problem from the user. The search engine Google allows users to view the cached version of a page as it was crawled, providing users with the option of seeing the original contents of a removed page. Some metasearch engines may download page contents to ensure the current contents are still relevant to the user's query and to remove dead links [Gauch et al., 1996, Lawrence and Giles, 1998a, Lawrence and Giles, 1998b, Glover et al., 1999b, Glover et al., 1999a].

Some web sites, such as stock or news sites, have a continually updated database of local content. Local control ensures that all pages returned are valid and have not had a content change. A metasearch engine that queries such a site will also be guaranteed of the currency of the result, allowing it to return pages that might not be contained in the database of a general-purpose search engine.

2.3.2 The Web Discovery and Web Coverage Problem

The rapid growth of the web presents two fundamental problem for search engines. First is the problem of containing and obtaining all possible pages. Second is the problem of locating all possible pages. The measure used to describe the percentage of the indexable web covered is called coverage. Several studies have been conducted to estimate the coverage of web search engines [Lawrence and Giles, 1999a, Lawrence and Giles, 1998c, Selberg, 1999, Gordon and Pathak, 1999, Bharat and Broder, 1998]. Although they do not all agree on the exact sizes and percentages, there is consensus that no single search engine can cover the entire web. The estimates by Lawrence and Giles from a sampling of all IP addresses in February of 1999 concluded that the largest coverage by a single search engine is about 16%.

Despite a rapid decrease in the costs for storage over the past decade, and rapid increases of network bandwidth, it is not expected that any search engine will individually crawl the entire web. Even if a crawler could crawl the entire web, it is unlikely that it would have the most recent contents for every page. Selberg describes several reasons why it is unlikely for a single search engine ever to be comprehensive [Selberg, 1999].

The first problem in discovering every web page is the ability to obtain and store every web page (or even just every URL listed on the seen web pages). Even if there were a master list of

all known URLs and disk space were free, the sheer size of the web and limited performance of web servers and network links would make it difficult to download every page in a reasonable time. The *web coherence problem* requires that web pages be re-visited, or otherwise updated, to ensure contents are kept synchronized. A slow crawler may know of every page (assuming the web was not growing), but if it takes ten years to get them, the value is questionable. This problem is made impossible when the growth rate of the web is larger than the growth rate of available network resources for a crawler (assuming a single centralized crawler with no cooperation between search engines).

The second problem is determining every URL. Unfortunately, there is no master list of all URLs. As a result, URLs must be discovered. The *web discovery problem* deals with the task of determining the set of all (indexable) web pages, given an initial set. Erik Selberg formally defines the web discovery problem in his dissertation [Selberg, 1999]. His definition begins with an initial set of web pages. By following hyperlinks (URLs), beginning with the initial seed, determine all web pages.

Although this might seem easy, there are several stumbling blocks. First, web pages are dynamic, and links from a given page may change or may point to non-existent pages. Second, new pages are added, and they might not be linked to. The problem is made worse by the robot exclusion standard that states that a web crawler or robot should not index or crawl pages that are marked as excluded [Koster, 1995]; any web server can indicate any directory or set of pages as excluded. As a result, even a page that has a path to it from an initial page may never be crawled because the path goes through an excluded site. The large size of the web also creates challenges due to the need to keep track of what has been seen and what has not.

Another problem faced by crawlers is that of "spider traps." Since web pages can be dynamic, some individuals will generate page contents from programs, or CGI scripts. These contents may include dozens of randomly generated URLs, many of which point to the same program, resulting in an apparently infinite path, even though in actuality it is only one program that responds to many different URLs. *Spider traps* are sometimes used as a mechanism for contaminating the e-mail lists generated by e-mail searching crawlers.

There are several approaches taken to reduce the effects of the web discovery and web coverage problems. First, a metasearch engine has been shown to provide higher coverage, by combining results of several search engines, than the largest single search engine: 42% vs 16% [Lawrence and Giles, 1999a]. In addition, new technologies may develop to allow for alternate methods of distributed search, allowing for the entire web to be searched without having direct access to a single centralized database of all pages. Distributed search can be accomplished when

each web server maintains its own index; and either the indices are combined, such as with Harvest [Bowman et al., 1994], or searches are applied directly to the smaller indices in a metasearch fashion. When combining indices, it is necessary for there to be a single standard, and universal cooperation, neither of which is likely to exist. In addition, if the change rate of the individual web indices exceeds the bandwidth to the central index, it is impossible to achieve complete coverage.

The current approach by search engines is to focus their crawler on sites believed to be the most important [Cho et al., 1998]. Excluding pages that are unlikely to be desired by searching users can also reduce resource costs and improve performance. Some search engines are special-purpose, covering only one topic or genre, and as a result can ignore large portions of the web.

2.4 Resource Costs and Scalability of Web Search Engines

There are resource costs associated with maintenance of a web search engine, and separate costs for responding to queries. Most general-purpose web search engines are designed to handle high query loads on the order of tens of millions of queries per day. A well designed search engine can in theory be scaled to handle arbitrarily large numbers of user queries given enough computational resources. However, network bandwidth may limit the maximum performance of a crawler.

2.4.1 Search Engine Maintenance Costs

A search engine as described in Figure 2.1 must maintain its database. Database maintenance requires ensuring current contents are synchronized with corresponding web pages and crawling new pages to ensure sufficient coverage is provided. The web crawler is responsible for both tasks, and the limiting resource is the available network bandwidth both at the crawler and at the target web sites being crawled. As described in Section 2.3.2, it is unlikely that a search engine will ever crawl the entire web. The more resources provided to the crawler, in general, the more pages that can be crawled or recrawled.

2.4.2 Query Costs

A search engine expends resources to respond to user queries. It is estimated that the size of the textual content on the indexable web is several terabytes. Several advanced algorithms have been developed to permit search engines to respond to most queries in fractions of a second. Even with sublinear, but not constant time, algorithms, as the size of the index grows, the response time per query will increase. This increase is separate from the total number of queries a search engine can process in any given amount of time. Duplication of a search engine's database can permit

multiple instances of a given search engine to respond to simultaneous user queries, providing a nearly unlimited scalability for user queries.

CHAPTER 3

Web Metasearch

A metasearch engine is a “meta” search engine; it is a search engine that searches other search engines. A metasearch engine takes user queries and submits them to multiple underlying search engines, and combines the results into a single interface. Metasearch engines are primarily used to improve coverage over a single search engine.

In this chapter, we present a description of the architecture of a web metasearch engine. Then, we define the web metasearch problem. Finally, we present several problems faced by a metasearch engine that limit its ability to find useful results.

This chapter describes the architecture of a metasearch engine. To understand some of the concepts, we define a *search engine request* as the query sent to a search engine requesting results. All requests are assumed to be syntactically correct for the search engine being queried.

3.1 Architecture of a Metasearch Engine

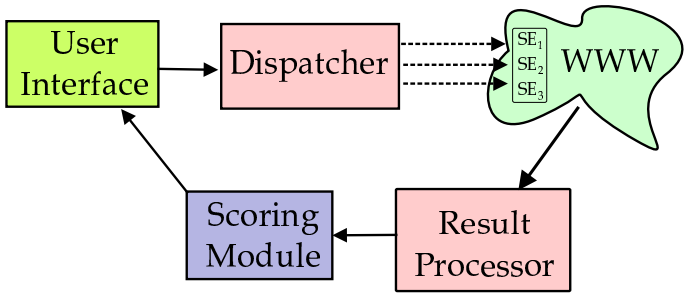

Figure 3.1: Architecture of a web metasearch engine.

The architecture of a web metasearch engine is similar to the architecture of a regular web search engine. The primary difference is that the database of a web search engine is replaced by a virtual

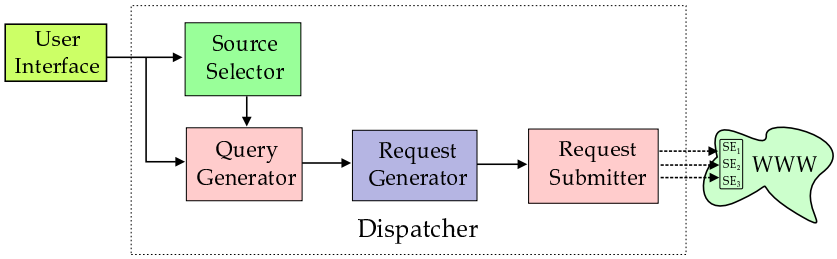

Figure 3.2: A possible design of a dispatcher.

database, consisting of the dispatcher, other web search engines, and a result processor. A virtual database functions like a local database in that queries are applied and results returned, but does not require local storage. In addition to the web, there are four components of a web metasearch engine: user interface, dispatcher, result processor, and the scoring module.

3.1.1 User Interface

The user interface of any search engine (metasearch engine or regular search engine) is the interface between the "user"[1] and the system. It is responsible for accepting user input and displaying output. Both a metasearch engine and a regular search engine must capture a description of the searching user's information need. A metasearch engine may have a few extra options related to the decisions about where to search, but otherwise is similar to that of a regular search engine.

One of the differences between a metasearch engine and a regular search engine is the expected search time. A regular (centralized) search engine can be designed for a specific performance need. A metasearch engine is limited by the performance of the search engines it queries, and as a result may take significantly longer than most of the search engines it queries. This extra time presents a problem for the user interface, since a normal metasearch engine interface must wait for all search engines to respond before acting.

3.1.2 Dispatcher

The dispatcher of a metasearch engine is similar to the query processor of a regular search engine. A query processor generates database queries based on the input from the user interface; a dispatcher generates *search engine requests* from the user's input. A dispatcher must determine which search engines to query, and how to query them. Figure 3.2 shows one possible design of a dispatcher with an explicit source selector to choose search engines to query, and a query generator to modify queries appropriately for each source.

[1] A user could be a metasearch engine.

The dispatcher makes the primary search decisions for a metasearch engine. The decisions of which search engines to query, and how to query each source, directly affect the ability for a metasearch engine to locate useful results. Poorly chosen sources or ineffective queries will result in few useful results. A dispatcher also influences the resource requirements of a metasearch engine. A dispatcher that sends too many search engine requests requires significantly more network resources, and will probably take longer to complete a search, than a metasearch engine that queries only a few resources.

The dispatcher first decides what and who to query, then must formulate an appropriate request. The request generator combines the user's query, the selected source and chosen query modification to produce a valid web request (typically a URL). The request submitter takes that request and executes it. The request submitter must interact with the low level protocols and ensure that errors are appropriately recorded.

3.1.3 Result Processor

The result processor of a metasearch engine produces the output of the virtual database, returning results with associated metadata. Results sent from the result processor to the scoring module are similar to results returned from a database in a regular web search engine. The result processor accepts search-engine responses and extracts from them the individual results.

Figure 3.3 shows a simple two-part view of the tasks of the result processor. First, a result processor retrieves pages from the web (search engine responses). Second, it extracts the results contained with them.

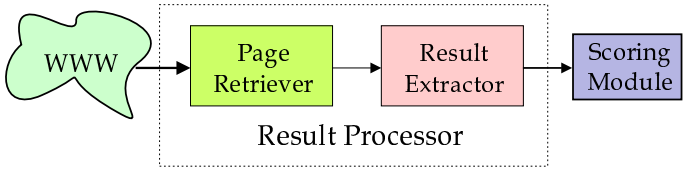

Figure 3.3: Two subcomponents of a generic result processor.

A result processor has several things that limit the ability for a metasearch engine to locate useful results. The response page from a search engine contains only minimal information about each result; a large search engine database contains all the necessary information to score each result, but only a small subset of this information is provided in the search-engine output. The information provided in the outputs of different search engines may vary. Each search engine may have a different output format, and the information provided by each search engine may be different. One search engine may provide a title, URL, and a document summary, while another might provide

a title, date, URL, and query-term context. The same web page returned by two different search engines may appear differently.

An advanced result processor may perform *information-gathering actions* to supplement the data about each result to improve the ability to score results. Typical information gathering actions include the downloading of web pages to provide full HTML.

3.1.4 Scoring Module

The scoring module of a metasearch engine, like the scoring module of a regular search engine, produces a score for each result from the associated data about each result. If a metasearch engine cannot directly compare results, a *fusion policy* is used to combine the ranked lists of results into a single ordered list. Unlike a regular search engine, a metasearch engine may have limited information about each result. The missing information may make it difficult to identify a result as useful for a given information need.

3.1.5 Fusion Policies

A metasearch engine that combines the results from different search engines, based only on the limited information returned from each search engine, uses a *fusion policy* since it is fusing the results without being able to compare them directly.

Craswell discusses the differences between integrated merging methods and isolated merging methods (metasearch engine ranking) [Craswell et al., 1999]. He examines four sources of merging information: ordinal rank assigned to a document, score assigned to a document, server-promise metric, and the contents of the document itself.

In addition to these four sources, some metasearch engines consider query terms in the document summaries or URLs returned from multiple search engines. A document that contains the query terms in the titles, summaries, or URL is believed to be more likely to be relevant to the query. A document that is ranked highly by many search engines for the same query is also assumed to be more likely to be useful since several different ranking functions scored it highly [Selberg, 1999]. It should be noted that data from our user study, as well as data from the work of Gordon suggest that overlap between search engines occurs infrequently [Gordon and Pathak, 1999].

Selberg describes the ranking functions used by MetaCrawler [Selberg, 1999]. When MetaCrawler is instructed to download all the results, and the user is searching for “All of these words” or “Any of these words”, an algorithm similar to TF-IDF is used to score each document. If the documents are downloaded and the user specified “phrase” search, MetaCrawler scores documents based on the term proximity.

When MetaCrawler does not download any documents, the *Normalize-Distribute-Sum* (NDS) algorithm is used to fuse the documents. NDS considers the original scores from each search engine,[2] and normalizes to prevent a single search engine from dominating the ranked list. NDS is designed to re-weight the individual search engines, as well as to provide a score bonus to documents ranked highly by multiple search engines.

Other metasearch engines, such as ProFusion, use similar fusion policies [Gauch et al., 1996].

3.1.6 Wrappers and Agents

Figure 3.1 describes a logical architecture of a metasearch engine. The dispatcher and the result processor are explicit. Some metasearch engines are built using special software called *wrappers* or *agents*. Typically, an agent or wrapper is associated with a single search engine, and performs some of the sub-tasks of the dispatcher and result processor. Figure 3.4 shows an architectural representation of a metasearch engine based on wrappers. In Figure 3.4 there is no explicit dispatcher or result retriever.

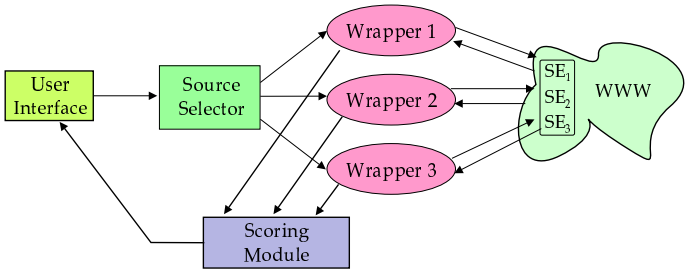

Figure 3.4: A metasearch engine based on wrappers.

A wrapper is a name given to a piece of code that can accept queries (inputs) and generate web requests, typically for a specific search engine, and process the results. These tasks address some of the sub-tasks of the dispatcher and the result processor. Wrappers are typically used to build modular systems that can easily incorporate new search engines. Wrappers can be used with a standardized interface to allow third parties to create wrappers and to allow wrapper use for other research systems.

Like wrappers, agents can form part of a metasearch engine. Agent based information retrieval has been the focus of many recent research projects [Atkins et al., 1996, Paepcke et al., 1997]. An agent refers to an independent entity (could be a human) that has goals and can make decisions. One area of agent research is digital libraries. Agents are useful for such a system because, like wrappers, they allow modular design; each agent can be designed independently and allow for a

[2]Currently few search engines provide numerical scores for results, so a modified version of NDS would be required.

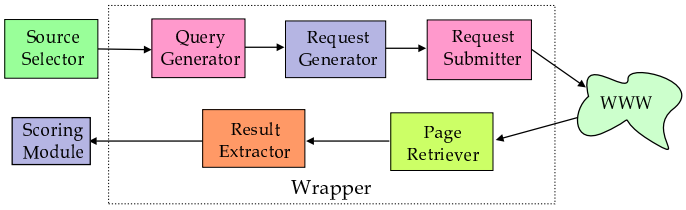

Figure 3.5: The internal subcomponents of a wrapper.

highly distributed system.

Although this dissertation focuses primarily on centralized, web-based, metasearch engines, a digital library could have the same functionality as a metasearch engine. The University of Michigan Digital Library Project, for example, would send a user's query to several different sources, including some web search engines. Figures 3.2 and 3.3 show a possible design of the dispatcher and result processor. Similar to the metasearch engine architecture shown in Figure 3.1, the UMDL has an interface that accepts user inputs and displays output. In the UMDL, a task-planner agent generates a plan, from which "requests" are sent to multiple Collection Interface Agents, some of which are proxies to web search engines. The results are received and processed by the User Interface Agent (UIA). The tasks performed by the agents of the UMDL project are identical to the sub-tasks performed by the dispatcher, result processor, scoring module, and user interface.

3.2 Web Metasearch Problem

The goal of a web search engine is to find the n most useful documents for a given search. A web search engine influences the documents in its database and the ranking of the documents returned for a given search. A metasearch engine has the same goal, to return the best n documents, as judged by the user. However, a metasearch engine does not necessarily have a database, but rather relies on results from other search engines. A metasearch engine controls the set of results through the dispatcher; the set of results that can be returned is determined from the responses to the search engine requests generated by the dispatcher. Like a regular web search engine, a metasearch engine can choose the ranking of the documents it returns; however, it often must do so with less information about each result.

Figure 3.6 describes the web metasearch problem. The important aspects are the selection of search engine requests and the ranking of results. Metasearch engines were originally designed to improve coverage. Research has shown that combining results from several search engines significantly improves coverage over any single search engine alone. A metasearch engine may be able to have indirect access to more results than any one single search engine alone, but there are many

Let:

$U(d)$ a utility function of a given document for the user's information need

$SD^* = (D_1, D_2, \ldots, D_r)$ is the set of search engine databases corresponding to S

Given:

A specification of a user's information need, I (typically a query)

A maximum number of documents to be returned, n

A set of search engines $S = \{se_1, se_2, \ldots, se_r\}$

Determine:

Using S and I, generate a set of search engine requests: $Q = \{< se_k, q_k >\}$, where q_k is query to search engine se_k

$\mathcal{D}'$ is the union of the results returned from all of the search engine requests in Q

Using I, determine a set of results $R = \{d_1, d_2, \ldots, d_k\}$, $|R| \leq n$, $R \subseteq \mathcal{D}$, where R contains the n most useful documents from SD as defined by $U(d)$

Figure 3.6: Definition of the web metasearch problem

restrictions on this access. The limited access directly affects the ability for a metasearch engine to return *useful* results. We present five specific problems faced by a metasearch engine.

- First, a metasearch engine has a limit to the number of results retrievable for any given search. This limit, called the *web search horizon*, is either artificially imposed by a search engine, or is bounded by the resources or results available.
- Second, metasearch engines may have access to the URLs of many results, but the information used to score each result is not always provided in the output interface of the queried search engine – *the limited information problem.*
- Third, a metasearch engine is dependent on the resources of outside search engines whose performance they cannot control. As a result, a metasearch engine has search time issues – *the temporal search problem.*
- Fourth, the attitude of a particular search engine directly affect the ability of a metasearch engine to find useful results. A cooperative search engine will make special affordances to improve metasearching, while a hostile search engine will block or interfere with the metasearch process – *the limited cooperation problem.*

- Fifth, individual search engines are independent entities, each with their own interfaces. Each unique interface presents problems for a metasearch engine wishing to locate useful results – *multiple interfaces problem.*

3.3 Web Search Horizon

A web search engine has access to all results in its own database when making ranking decisions. However, a metasearch engine is limited by the returned set from the underlying search engine. This is called the *web search horizon* problem.

Definition 3.3.1 *The web-search horizon (SH) for a specific search request is the minimum of the following three things:*

- L_{se}*: The artificial limit on the number of results returned by a search engine*
- L_a*: The point at which it is no longer practical or desirable to retrieve more results: the stopping point*
- $|D'|$*: The number of results retrieved for a particular query (input)*

The web search horizon emphasizes the difficulties a metasearch engine faces when trying to find useful results from a specific search engine. A single search engine may contain a million results for a particular query, but because of the web search horizon, only a tiny fraction may be seen. The problem is further complicated by the fact that the ordering policy of the search engine may vary from that of the metasearch engine making the request, causing useful results to be ranked beyond the web search horizon.

A metasearch engine can influence the web search horizon by expending more resources in the form of extra search requests. Most search engines allow a "user" (in this case a metasearch engine) to request more than the default ten results, but at a resource cost for both the requester and the search engine. The extra requests not only require extra network bandwidth, but also may increase the total search time.

To reduce the effects of the web search horizon, it is desirable that the search engine's ordering policy match that of the metasearch engine, causing the best results to be ranked near the top. If it is not possible to alter the search engine's ordering policy, other steps may be taken to improve the precision of the results (the precision is the density of useful results). Query modification may be used both to alter the search engine's ordering policy and to improve precision of the results.

Several search engines offer options, such as “sort by date” or “restrict by category.” A good dispatcher will utilize all available options to align the ordering policy of the search engine with its own. Query modification, through addition of extra words or constraints, may also have an effect on the ordering policy, as well as on the set of results returned. Adding extra terms may restrict the result set to those more likely to be useful, or may cause a search engine to rank more useful documents near the top. For example, when searching for research papers on AllTheWeb, adding `introduction ‘‘shown in figure’’` to the query will cause nearly all of the top 20 results to be research papers.[3]

3.4 Limited-Information Problem

A web search engine typically has access to the full-HTML and other information about each result in its database. This information is used to determine the ranking and the summary displayed in the output user interface. A metasearch engine queries other search engines and uses the presented result page to generate its own set of results. Unfortunately, a metasearch engine is not presented with all the information that was used by the queried search engine to make the decision to return the result. The lack of available information reduces the ability for a metasearch engine to judge the usefulness of a result.

Definition 3.4.1 *The limited-information problem*

The information provided by a search engine, in the output user interface, may be less than the information is used to make the ranking and display decisions.

There are many factors a search engine considers when ranking documents. Three common features include: link structure; query-term frequency in the document or collection; and special HTML structural information, such as META-TAGS or font color. In general, none of these three factors are presented in the output user interface.

For example, most search engines will score documents based at least in part on query term frequency in the document and the collection (TF-IDF). A search engine has access to term frequency in both the documents and the collection; however, they do not display such information in the results presented in the output interface. A protocol called STARTs was developed to allow search engines to provide this information in a standard way [Gravano et al., 1997]. Unfortunately, this protocol is not in wide use.

[3]This query modification’s effectiveness was demonstrated at the time our learning algorithms were run, but due to changes in the ordering policy or database contents, it may become less effective.

As a result, a typical search engine returns only the title, URL, and a short summary or query term context information. It is possible that a useful result would contain none of the query terms in the title, URL or summary. Query context, although helpful, does not provide much information about where the terms occur relative to each other or their frequency in the document or collection. A very large but not useful document that contains one relevant paragraph may have the same displayed query term context as a document that is entirely about the user's query.

The primary approach to reducing the effects of this problem is through the use of an information-gathering action. This action, such as downloading the result, may provide access to some of the missing information, such as query term frequency in the document. The second approach is to use proxy statistics to estimate TF-IDF [Craswell et al., 1999]. Proxy statistics may allow computation of inverse document frequency (IDF), but do not provide term frequency in a document where only a summary is available.

3.5 Temporal-Search Problem

Another consequence of relying on other sources for results is the lack of control over the latency. An individual search engine can be designed for specific performance requirements as needed; in general, the more resources, the faster. No matter how many resources are available to a metasearch engine, it cannot return results faster than the search engines it queries. The latency is further increased when a metasearch engine requests more than the default number of results from a search engine, or when expensive (both in terms of time and network resources) information gathering actions are performed.

The increased latency may not directly affect the quality of the final returned results, but rather affects the ability for a metasearch engine to operate in real-time.

On average, a typical search engine may respond in under one second; the worst time can be ten or more seconds. In some cases, a search engine request may be unanswered, timing out in as much as 30 or more seconds. The temporal-search problem simply states that: A metasearch engine is limited by the performance of the underlying search engines it queries. Responses are not instantaneous, and may come inconsistently. Selberg describes as one of the advantages of a metasearch engine the ability to return results even when a single underlying search engine fails [Selberg, 1999]. Unfortunately, this advantage comes at a cost: time.

Currently, there are two user-interface approaches to deal with the increased total search time. First is the use of feedback to the user that the system is working. Second is the use of an incremental interface that provides both feedback and usable results. Most metasearch engines choose the first

approach by offering colored icons that indicate the status of each search engine. As a search engine responds, the associated icon turns from yellow to green; if there is an error, it turns red. A second type of feedback is a two screen output interface. The first screen lists the names of the search engines as they respond. After the search has completed, the second screen lists the scored results.

This research uses an incremental interface as a means of reducing the perceived latency. Although it is impossible to return results faster than the search engines queried, it is possible to return scored results as soon as they are found, even before the search completes. The incremental interface and the property of independent result scoring is described in more detail below in Section 3.8.3.

In addition to reducing the consequences of the latency, several actions can be taken to reduce the latency itself. Many metasearch engines will impose explicit time limits on each search engine queried. If the search engine does not respond within 30 seconds, for example, the search engine is ignored and the search terminates. A time limit will place an upper bound on the total search time, although 30 seconds may be considered by many users too long to wait.

Another approach is the use of source selection based on expected performance. Each search engine performs differently: if time is important, then choosing to only query those likely to respond quickly can reduce total latency. The metasearch engine ProFusion allows selection of sources based on expected search latency [Gauch et al., 1996].

Three other methods used in our research include the use of alternate stopping conditions, web caches, and the intelligent, selective downloading of results. A metasearch engine may choose to stop based on conditions other than all queried search engines responding. Inquirus 2, our research, terminates the search when a specified number of results have been processed. It may also be desirable to stop based on the scores of results or other factors. By stopping based on results, a single, slow search engine will not necessarily adversely affect the total search time.

The second approach we use is web caches. Inquirus 2 downloads many results. The downloading of results can be very expensive and can increase search latency. Web caches can be used to reduce the costs of downloading results. A third technique, described in detail in Section 4.2, is intelligent selective download. Choosing to download only some results can significantly reduce the search costs and total search time. Scoring a result without download can take fractions of a second, while downloading may take several seconds.

3.6 Limited Cooperation Problem

A web metasearch engine utilizes the resources of other search engines. Each search engine has an attitude towards a metasearch engine. There are three possible attitudes: cooperative, non-cooperative but not hostile, and hostile. Each of these attitudes has implications for the ability of a metasearch engine to locate useful results. In addition, a metasearch engine, through its actions, can affect the attitude of other search engines towards it.

- cooperative – A search engine works with a metasearch engine, either through special interfaces, early notification of interface changes, or possibly private access to their resources.
- non-cooperative, but not hostile – A search engine is mostly indifferent to a metasearch engine. In general, blocking is not performed, and no special affordances are provided.
- hostile – A search engine forbids access by a metasearch engine. Often carried out by technical means, i.e., blocking or difficult to parse interfaces, required authentication, user tracking, etc., This attitude implies a strong desire not to be remotely queried. In the extreme case, a hostile search engine may use legal remedies to stop remote accesses by a specific party.

The best relationship, from the perspective of a metasearch engine, is a cooperative one. A search engine that is cooperative will take actions to assist a metasearch engine, such as providing a private interface or making design alterations to facilitate searching. In general, this relationship will not occur without some form of incentive, such as payment for each query.

The second, and probably most common, type of relationship is non-cooperative, but not hostile. A search engine of this type will permit a metasearch engine to query it, but will not make special affordances. A search engine may allow metasearch engines, without helping them, as long as the perceived resource burden is kept small. For example, a search engine might not block a metasearch engine if fewer than 10,000 queries are sent per day. Our research assumes all search engines that are queried have this type of attitude.

The third type of relationship is hostile. A search engine that does not want a metasearch engine querying it, and takes actions accordingly is said to be hostile. Typically, hostile search engines will use technical means such as IP address or USER-AGENT based blocking mechanisms. A persistent metasearch engine could be viewed as participating in a limited denial of service attack and might be stopped through legal action.

The attitudes affect the ability for a metasearch engine to provide useful results in several ways. First, a search engine that is hostile will make it difficult or impossible to be metasearched. Second, a search engine may change its interface or ordering policy at any time, and if the relationship is

not cooperative, the metasearch engine may require higher maintenance costs. Third, a cooperative search engine may make special affordances, such as providing extra information along with each result, improving the ability for a metasearch engine to score results, possibly with lower resource costs. A cooperative search engine may provide, through special interfaces, lower latencies than would be found through the normal interface, reducing the effects of the temporal search problem.

A metasearch engine can affect the attitudes towards it. A metasearch engine that submits many requests per day to a search engine might cause that search engine to become hostile; asking for more results is almost identical to making a distinct request. The web search horizon should be kept as low as possible to minimize the total number of requests to any search engine. A metasearch engine that performs source selection may be able to spread the total query load among multiple search engines. A metasearch engine that performs caching of search engine responses may also reduce the total number of queries. There are also technical methods that can be used to reduce accountability of queries. For example, a personal metasearch engine, such as Sherlock, makes all queries from a user's machine instead of a central server. Distributing the load makes it harder to attribute all the requests to a single source, reducing the chance of being blocked.

3.7 Multiple Interfaces Problem

A metasearch engine submits queries to several search engines and extracts the results from their responses. Each search engine has a unique input interface and a unique output interface. There are not currently any widely used standards for search engine interface design. The lack of standardization of both the form of the interfaces and their functionality can make metasearch more difficult. To formulate a valid web search engine request or to extract results from a search engine response page, a metasearch engine must have knowledge of the specific input and output interface. Changing interfaces causes higher maintenance costs for a metasearch engine.

In addition to the maintenance costs associated with a changing interface, the interface differences present a problem for the result processor and scoring module. Output interfaces may provide different and often irreconcilable fields. For example, one search engine may present a title, URL, and a document summary, while another provides a title, URL, query term context and date last crawled. The result processor must be able to extract each field, and when possible reconcile them. Two search engines, each presenting a last changed date, may use different formats, or two search engines may both present a "relevance score" that was computed differently. The result processor may have many different fields for each search engine. The scoring module must be able to score documents appropriately given the differences in the document metadata.

A fusion policy may allow combination of results from different search engines. Section 3.1.5 describes fusion policies in more detail. If individual results are to be scored, it is necessary that each document to be scored has appropriate metadata.

Research on wrappers can allow for each search engine to be represented by a stand-alone piece of code that can be maintained by a third party, allowing for more rapid response to interface changes. Apple's Sherlock will automatically download the most recent parsing code from Apple each time Sherlock is started. It may also be possible to generate wrappers for a search engine automatically, further reducing the maintenance costs.

To deal with differences in the semantics of the output interfaces, research has been done on integrating ontologies into searching. Work by Weinstein on differentiated ontologies studies the ability to learn how to communicate when there is a subtle conceptual language difference, such as one agent not knowing the color pink but understanding shades of red [Weinstein and Birmingham, 1999]. The University of Michigan Digital Library developed and utilized ontologies for use with describing collection contents, as well as for the input queries to them [Atkins et al., 1996].

Our work reduces this problem through improvements in the result processor and scoring modules. Downloading a document produces a uniform set of attributes to enable scoring, independent of what fields were returned by a search engine. The result processor, using a selective download module, will download a result if there is not sufficient information available to score it. The result processor will also utilize knowledge about the search engine to aid in the scoring. For example, results from a search engine that specializes in research papers are all assumed to be research papers. The scoring module allows scoring of results that were downloaded and have the full document information, or results that were not downloaded, but have sufficient metadata to score them.

The approach of allowing smart downloading of results allows all results to be scored, regardless of the search engine, and when a search engine has a field the user desires, the results might not be downloaded for a performance boost. For example, a result returned by a search engine that provides a date might be scored without download if the user was searching for "current events." However, a general-purpose search engine that did not provide a date could still be used, but all results would likely have to be downloaded.

3.8 Metasearch Engine Properties

The simplest metasearch engine will take a user query and submit it to multiple search engines, combining the results. There are many features a metasearch engine can utilize to improve effec-

tiveness and performance. The first property we describe is *source selection.* Source selection is the ability for a metasearch engine to determine at query time which sources to query to balance resource costs with the expected search benefits. A simple metasearch engine, such as DogPile, may always query the same three sources, while a more advanced tool such as ProFusion may decide to query a different set of three based on the expected performance or expected quality of results for the provided query.

A second property of a metasearch engine is *query modification.* Query modification is used to enhance the quality of the results. The Watson project submits user queries both to special-purpose and to general-purpose search engines as appropriate. When querying a general purpose search engine, the queries are modified to reflect the user's search context. A user searching for web pages about firearms in the context of the early US history, using a query of `firearm`, would probably prefer results related to the creation of the Bill of Rights or the use of firearms in the Revolutionary War of the US to a recent article about the NRA. The query `firearm` will contain both useful and not useful results, but the query `firearm US history` may narrow the focus. Our research focuses on three properties a query modification can have: Source-dependent, category-specific and non-obvious.

A third property of a metasearch engine is called *independent result scoring*, or the ability to score a result independent of the set of already found results. Sherlock presents results before a search is completed. When a new result is found, the scores of the already returned results do not change. Independent result scoring enables the use of an incremental interface that will not change the relative ranking of already returned results.

3.8.1 Source Selection

The primary advantage of a metasearch engine is a substantial improvement in coverage over any single search engine. The total coverage of a metasearch engine is related to the coverages of the individual search engines queried. One could, therefore, conclude that to maximize coverage, a metasearch engine should query all possible search engines all of the time. Although coverage may be increased by querying more search engines, querying is not free, and in some cases may harm the result quality. Because of the *temporal search problem* and *limited cooperation problem*, there is incentive to query only search engines that are likely to benefit the user. Querying too many search engines can slow the search down, and cause the search engines being queried to change their attitude, as well as increase the local network costs for the metasearch engine. In addition, querying a search engine that does not return any useful results wastes local processing resources and angers the user, who may be forced to examine large numbers of undesirable results.

Figure 3.6 states that a search engine must determine a set of requests, from which the set of results are derived. If a metasearch engine does not query the same set of search engines for each user query, it is said to have the property of source selection. Source selection can be used to improve the precision of the search and lower the search costs by querying only "good" sources.

Several metasearch engines perform source selection. Most metasearch engines that perform source selection explicitly acknowledge the resource costs in their published papers.

The metasearch engine SavvySearch determines the number of search engines to query based on the current system load. As the load increases, fewer search engines are queried per user search [Howe and Dreilinger, 1997]. The Watson project automatically chooses the most appropriate special-purpose search engines based on the user's search context [Leake et al., 1999, Budzik and Hammond, 1999]. SavvySearch also allows users to specify a search category, such as auctions or news, and only searches "appropriate" special-purpose search engines [Howe and Dreilinger, 1997]. ProFusion can perform source selection based on the predicted subject of the query [Gauch et al., 1996]. ProFusion also allows users to specify that only the fastest search engines be queried, choosing search engines based on expected (time) performance [Gauch et al., 1996].

Source selection may also be used as a means of varying sources queried to improve scalability and reduce the chance of a negative attitude change by a queried search engine. A metasearch engine that queries the same three general purpose search engines for every query can process fewer simultaneous queries than a metasearch engine that chooses three random search engines from a set of one hundred.

Our work focuses on *need-based source selection.* Users provide a category in addition to their subject query. The specified category is used to select a set of search engines (and associated query modifications) that are expected to provide the best balance of precision and recall. Choosing sources based on user's specified need can enable the appropriate selection of special-purpose sources, even when the user's query may be ambiguous. Combining the selection of the sources and the selection of query modifications can allow use of general-purpose search engines for special-purpose queries. Sending a vague query to a general-purpose search engine may have very low result precision when the user was looking for results of a specific category. Chapter 6 describes the procedure used by Inquirus 2 to learn the search engine query modification pairs. The data from our learning procedure demonstrates that for some cases, a simple, unmodified query may have less than 2% precision for a specific category when sent to a general-purpose search engine.

3.8.2 Query Modification

Query modification is the process of altering a user's query with the intention of improving the result quality, usually precision. Most metasearch engines perform *syntactic query modification*, or modifying the "symbols" without altering the "meaning" of the query, to ensure the target search engine interprets it correctly. In addition to syntactic modification, there are other types of query modifications. First, we discuss how query modifications, other than syntactic modifications, are currently used in metasearch engines or related systems. Second we define three dimensions that can be used to compare query modifications: Source dependent, category dependent, and non-obvious.

When formulating a query to a search engine, it is essential that the query be consistent with the published rules for the search engine, for example rules that specify to put a "+" (plus) in front of every "required" term, or to use the term "AND" to indicate that several terms are required. These simple rules refer to the syntax of the query; to modify the syntax without altering the meaning (semantics) the modification is called a *syntactic modification.* Typically semantic or syntactic modification is performed by the dispatcher, in the request generator sub-module, as shown in Figure 3.2.

In addition to syntactic modification, queries may be modified in a way that alters their semantics (meaning). Several basic methods are currently used to produce modified queries. Some systems will make use of an ontology or thesaurus to modify the query terms to alter the expected number and specificity of the results (the precision recall balance). Some systems may use WordNet to find related concept terms to broaden or narrow the query, or to generalize a concept. Question-answering systems, such as the work of Agichtein use query modifications to improve the chance that a search engine result answers the question [Agichtein et al., 2001]. For example the original query `When was George Washington born?` may be modified to `''George Washington'' was born` or `''George Washington'' lived`.

Some work on query reformulation, such as the work by Belkin, allows users to choose extra words or features to help clarify their information need [Belkin, 2000]. Query reformulation is typically done in a multi-step process, and the final query is a modification, typically formed by adding extra words or phrases, of the original presented topical query. Belkin described how modifications chosen by a user were not always as effective as less obvious ones recommended by the computer.

The metasearch engine Watson considers a user's context when formulating queries to general-purpose search engines. The user's working document helps to clarify the user's subject area, which is used to modify the query [Budzik et al., 2000].

We define three properties of query modifications that relate to the ability to locate *useful* documents:

Source specific: Source specific describes how much a particular query modification takes advantage of specific features or knowledge of the source to which it is applied. For example, knowing a search engine allows stemming can be used to improve query modifications to that search engine. Likewise, some search engines allow queries on specific features such as title or URL. Source specific query modifications can be more effective at finding on-category documents from a general-purpose search engine. Chapter 6 describes our procedure for learning source-specific query modifications, which demonstrates a wide variation of the effectiveness of query modifications across different search engines. The level of specificity is a function of the knowledge and features utilized.

Category specific: Category specific refers to the ability for a query modification to find results that are of a specific category. The level of category specificity can be determined as a function of the precision of the returned results. A category-specific query modification can take advantage of an underlying user need by focusing on the "category" rather than the subject of the query. For a given category there is a reasonable domain of input queries. For example for a category of "research paper authors" the query input domain would be author names. A good category-specific query modification should perform equally well for any query within this domain. The effectiveness of category-specific query modifications can be measured based on the precision of returned results. It is also desirable to consider recall and consistency across queries.

Non-obvious: Query modifications may not always make sense to a human. Non-obvious query modifications are those that do not make sense to the uninformed, and or are unlikely to be guessed as being effective. An example of a non-obvious query modification could be requiring the title to contain the phrase "s home" when trying to find personal homepages. Although this query modification might make sense to some people, it is unlikely to be guessed, and in general demonstrates very high precision (in the area of personal homepages). Chapter 6 describes our procedure for learning query modifications. For the category of product reviews, the top ranked query modification for the search engine of AllTheWeb, was adding `reviews t`. The addition of "reviews" makes sense, but adding of "t" is clearly non-obvious, and would be unlikely to be guessed by a human.

Query modifications used by Inquirus 2 were learned using QMLP as described in Chapter 6. QMLP is designed to take advantage of knowledge and some functions specific to each search engine. The learned query modifications are also optimized for the given category, and since they are learned can be non-obvious. Chapter 5 describes the effectiveness of the query modifications used by Inquirus 2.

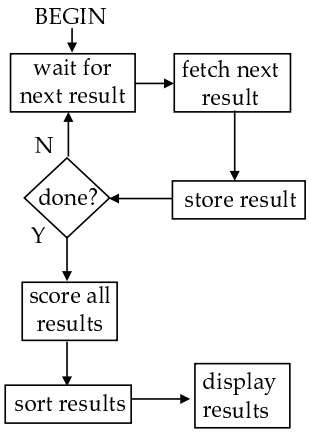

Figure 3.7: A flow chart diagram of an interface that processes all the results before displaying the sorted, scored list

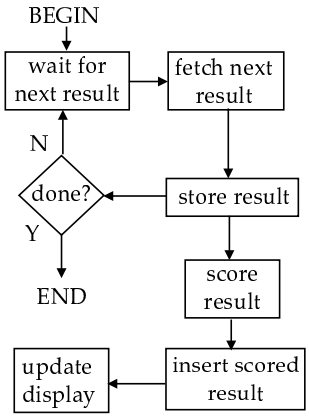

Figure 3.8: A flow chart diagram of an interface that displays scored results as soon as they are processed, assuming the property of independent result scoring

3.8.3 Incremental Interface

One of the problems faced by a metasearch engine is the *temporal search problem*, described in Section 3.5. A metasearch engine may be required to wait seconds, or even minutes, before a search can be completed. The overall utility of a search may be reduced as the time for the search to produce results is increased. One approach to reducing the effects of this is the use of a incremental interface. Although the total search time may not be reduced, the time until a user sees useful results, the perceived search time, may be reduced.

Apple's Sherlock, and Inquirus [Lawrence and Giles, 1998a] both list results as they are found. Sherlock will insert results into the correct position based on score. Inquirus will list results with context, regardless of final rank, until it has processed enough results, after which it produces a full ranking.

3.8.4 Independent result scoring

The property of independent result scoring is that the score of any result can be computed independently of all other results. A metasearch engine that considers link structure or statistical information about the retrieved set cannot score results independently. However, using proxy information that does not change as new results are found may permit independent result scoring.

Independent result scoring enables an incremental interface. Figure 3.8 shows a basic flow chart of an incremental interface which takes advantage of independent result scoring. Figure 3.7 shows a flow chart of a typical interface, where results are scored after they are all processed. A system with independent result scoring does not need to have an incremental interface, it only requires that scores be computed independently of the other results (both already seen and not yet seen).

CHAPTER 4

Preference-Based Metasearch

A preference-based metasearch engine is a metasearch engine that incorporates explicit user preferences. Explicit preferences are used to improve the ability to find useful documents and improve performance.

Three ways to utilize explicit user preferences in a preference-based metasearch engine:

- Improve the ability for a metasearch engine to *locate* useful documents
- Improve the ability for a metasearch engine to *identify* a document as useful
- Improve performance by reducing search latency and lowering resource costs

This Chapter presents Inquirus 2, our preference-based metasearch engine. Inquirus 2 implements architectural improvements to each component of the standard metasearch engine architecture, reducing search costs and perceived latency, improving the ability to locate useful documents and improving the ability to identify a document as useful.

Sub-component	**Improvements**
User interface	Input interface provides an option for a search category and output interface is incremental
Dispatcher	Need-based source selection and source specific, category-specific query modification
Result Processor	Intelligent selective result download
Scoring module	Need-based scoring

Table 4.1: Inquirus 2 architectural improvements

Inquirus 2, an extension of the architecture of Inquirus [Lawrence and Giles, 1998a, Lawrence and Giles, 1999b], is a content-based preference-based metasearch engine. A user of Inquirus 2 describes the topic or subject, via a topical query, and specifies preferences, in the form

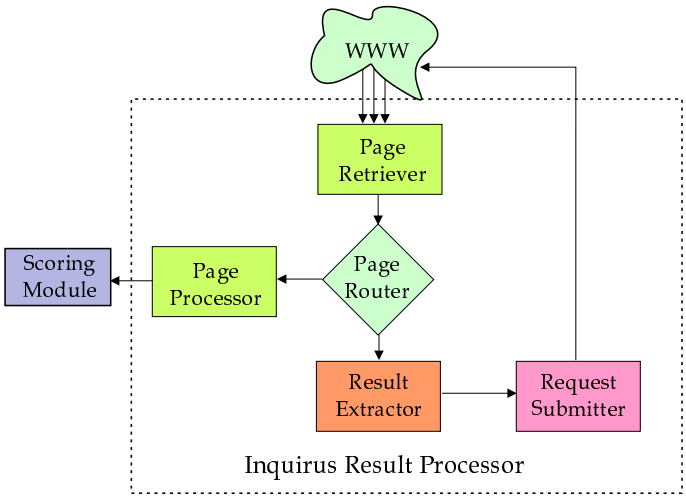

Figure 4.1: The improved result processor of Inquirus.

of a category, to aid in locating and interpreting results. Similar to Inquirus, Inquirus 2 performs information gathering actions in the form of downloading results. Unlike Inquirus, Inquirus 2 chooses sources and query modifications to improve the precision of the results, and utilizes a more advanced incremental interface to reduce perceived latency.

4.1 Inquirus

The metasearch engine Inquirus presents several improvements over a standard metasearch engine. Inquirus is a content-based metasearch engine since it utilizes full document contents when making both scoring and user interface decisions. Full document contents are obtained through the relatively expensive information gathering action of downloading all results.

The main architectural improvements of Inquirus over a regular metasearch engine are to the result processor. Figure 3.3 shows a sample result processor for a typical metasearch engine. In a typical result processor, responses from the search engines are processed, then each result is extracted and sent to the scoring module. Figure 4.1 shows the result processor used by Inquirus. In the result processor of Inquirus, pages associated with the URLs extracted from the results returned by each search engine are downloaded and analyzed. Individual summary information provided by the search engines is not directly considered by Inquirus when scoring results.

4.1.1 Inquirus Result Processor

The Inquirus result processor (Figure 4.1), as in the result processor of a regular metasearch engine, Figure 3.3, takes as input web pages, and outputs results with their associated attributes. The

differences lie in both the detail of the attributes for each result and the specific actions taken. Details of the implementation of the architecture of Inquirus can be found in [Lawrence and Giles, 1998a, Lawrence and Giles, 1999b].

There are five sub-components of the Inquirus result processor:

Page retriever: All responses from the web are processed by the page retriever. The page retriever must associate any information with the requested URL, as well as ensure that the pages are properly downloaded; i.e. no errors. Each requested page may have some associated metadata, such as the time or nature of the request, the source search engine that returned the result, etc. Also, received pages may have errors, such as *page has moved*. Such errors require specific actions by the page retriever.

Page Router: For a typical metasearch engine, all web pages returned are responses from search engines. However, for Inquirus, a web page returned could be either a search engine response or a web page that was requested for download. Based on the URL and information from the page retriever, the router determines the module that processes a particular page. If a page is a search engine response, and not an error, it is sent to the result extractor for processing. If it is a response of a download request, it is sent to the page processor so the necessary data for scoring can be extracted.

Result Extractor: Similar to the result extractor of a typical metasearch engine, as shown in Figure 3.3, the result extractor extracts the individual results from a search engine response. The result extractor for Inquirus does not need to extract search engine result summaries, only the URLs. The result extractor of Inquirus also extracts the URL for the "next" search results, to be used to request more results from the search engine. For each result extracted, and the "next" URL (as needed), a request is sent to the request submitter to be queued for download. The result extractor keeps track of already requested URLs to prevent duplicate requests and stores basic information about each result URL, such as the source search engine, for display and analysis purposes.

Although not explicitly shown, Inquirus utilizes a specialized cache. The cache performs two purposes: first, web caching is used so that URLs in the cache (and considered sufficiently recent) do not need to be downloaded. Second, Inquirus caches specific results for specific query terms. A duplicate query may jump directly to already known results containing the provided query.

Request submitter: Each result returned from a search engine may be queued for download. The request submitter manages the process of requesting URLs, either pages for download, or requests for more results from a search engine, generated by the result processor[1]. Since this could potentially mean hundreds of URLs requested, it is necessary to perform basic prioritization and

[1]The dispatcher also has a result submitter which manages original web search engine requests.

resource management. In the same way a crawler must not overwhelm individual web servers, the request submitter must balance the resources utilized and the expected burden for any given web server.

Page Processor: Inquirus downloads each result. These results must be processed before they can be scored. The page processor performs several tasks:

- Examines the location of user query terms for relevance prediction
- Generates query context summaries for later display by the output user interface
- Extracts URLs on page for analysis of possible hubs and authorities using a modified Kleinberg algorithm
- Considers common phrases or other features to be used for alternate term recommendation
- Performs appropriate logging and statistical analysis

Since Inquirus computes topical relevance locally, it must analyze the location and frequency of each query term in each result. The actual function used for result scoring is described in [Lawrence and Giles, 1998a] and is similar to the topical relevance function used by Inquirus 2. Section 4.1.2 describes the scoring module of Inquirus in more detail. To preserve the property of independent result scoring, the page processor of Inquirus 2 did not implement all of the functions of the Inquirus page processor.

Downloading each document provides the ability to generate the summary information provided by the output user interface. Inquirus displays query context information to allow users to better predict if a presented result will be useful or not. The page processor's second task is to extract the query context for display by the output user interface.

The page processor also extracts all URLs. Many search engines consider web link structure when ranking pages [Brin and Page, 1998]. In addition, considering local link structure may be useful when trying to identify hubs or authorities [Kleinberg, 1999]. Inquirus uses a slightly modified hub authority algorithm that utilizes only the URLs found from the results returned by the search engines [Lawrence and Giles, 1999b]. This algorithm may discover authorities that were never returned by the search engines, but were linked to by many results.

4.1.2 Inquirus Scoring Module

The scoring module of Inquirus is responsible for computing a *relevance* score for each result. Equation 4.1 shows the scoring equation used. To predict topical relevance, Inquirus considers

three factors: The percentage of query terms that occur, the relative proximity of the query terms to each other, and the total number of occurrences. A document missing a query term is likely not as topically relevant as one that contains them all. A document where the query terms occur as a phrase, in close proximity is likely more topically relevant where the query terms occur far apart. A document that mentions the query terms many times is likely more topically relevant than one that mentions the query terms only once. Equation 4.1 balances these three factors with the weights c_1, c_2, and c_3. N_p is the number of query terms that occur anywhere in the document, N_t is the number of times the query terms occur. The proximity is computed using a distance metric between each query term. Inquirus used values of 5000, 100 and 1 for c_1, c_2, and c_3 respectively. These weights produce an ordering where the the word proximity and query term occurrences act as tie breakers for documents with the same percentages of the query terms.

$$R = c_1 N_p + \frac{\left[c_2 - \frac{\sum_{i=1}^{N_p - 1} \sum_{j=i+1}^{N_p} min(d(i,j), c_2)}{\sum_{k=1}^{N_p - 1} (N_p - k)} \right]}{\frac{c_2}{c_1}} + \frac{N_t}{c_3} \tag{4.1}$$

Although the relevance scores can be computed independently, Inquirus does not utilize an incremental interface that preserves relative result rank. However, Inquirus does return results as found, with context information, but without a score. The incremental interface of Inquirus both provides feedback that the system is working and allow users to click on results prior to search completion. Query term context helps a user to determine if a result is likely to be useful for their needs [Lawrence, 2000]. In addition to computing a relevance score, Inquirus computes an approximation of the hub and authority score by utilizing the link structure from the set of all retrieved results. It is possible to return a page that is an authority but whose URL was not returned by any of the queried search engines, because many pages contain links to the URL. The computation of hubs and authorities cannot be computed independently, since each document found can change the scores of other URLs.

4.1.3 Content-Based Metasearch

Inquirus is a content-based metasearch engine. Content-based metasearch has many advantages and disadvantages when compared to a metasearch engine that is not content-based.

Advantages of content-based metasearch as compared to a metasearch engine using a fusion policy:

- Reduces limited-information problem; can theoretically improve ability to predict usefulness by utilizing the full HTML of each result;

- Has access to a limited view of the web structure; can be utilized to locate other results, as well as provide partial hub or authority information;
- Guarantees most recent version of each result is utilized, significantly reducing several aspects of the coherence problem; and,
- Can utilize full page text for improved display of results.

Disadvantages of content-based metasearch as compared to metasearch engines that utilize a fusion policy:

- Significantly worsens temporal search problem, as search time is increased dramatically
- Significantly increases resource requirements, reducing scalability.

Content-based metasearch engines perform information-gathering actions to permit them to utilize an ordering policy as opposed to fusing results. Unfortunately, the typical information-gathering action, downloading a result, can be very expensive, severely limiting the scalability.

The scalability of a metasearch engine is fundamentally limited by the available resources of the search engines it queries. Downloading of every result emphasizes the local network throughput limits. A web search engine typically has a wide inbound network pipe allowing thousands of simultaneous requests; a low-capacity web server may only be able to handle a few requests at a time, limiting the ability for download. Parallel download might not be reasonable if several results reside on the same web server.

One approach to reducing temporal and other resource costs for downloading results is the use of a web cache. Inquirus and Inquirus 2 both utilize web caches for this purpose. The performance gain of a web cache is dependent on the specific requests made. A system that makes many requests, often for the same pages, will benefit most. A system that is used infrequently, or that rarely has a duplicate request, will benefit very little. Web caches must handle pages whose contents may change. Many sites instruct web clients (or caches) to not cache their pages since their contents are frequently updated. As a result, such pages may not offer any resource savings when a web cache is used. In the limit, a web cache approaches the contents of a search engine database.

4.2 Inquirus 2 Architecture

Inquirus 2 is an extension to the architecture of Inquirus. Each sub-component of the architecture of a metasearch engine is improved as described in Table 4.1. Each of these improvements directly relates to one or more of the problems described in Chapter 3.

Inquirus 2 provides several improvements over a generic metasearch engine and a content-based metasearch engine. Inquirus 2 explicitly considers extra-topical user preferences throughout the search process to improve the ability to locate and identify useful results. Inquirus 2 also uses selective download to reduce significantly the number of documents downloaded, reducing time and network costs. To reduce perceived latency, scored results are displayed when they are found using an incremental interface.

In the next several sections, we describe the details of the architecture of Inquirus 2. The specific implementation details are described in 4.3. User preferences affect the selection of search engines and the way they are queried, as well as how results are scored, breaking the "one-size-fits-all" mold. Two users with the same query, but different (extra-topical) preferences, might search different sources, with different (modified) search engine requests, and results would be scored differently.

4.2.1 User Interface

The user interface of Inquirus 2 has two major improvements over the interface of a typical metasearch engine. First, the input interface makes preferences explicit in the form of a category. Second, the output interface is incremental; i.e., as results are found they are scored and displayed immediately.

Input user interface

The input user interface of Inquirus 2 requires users to provide a category for their query. Our interface was designed to make extra-topical category information explicit, although it is likely that a better input interface could be developed. Figure 4.2 shows the simple version of the input interface. A user enters a query (topical) and then chooses her desired category from the presented options (there are currently eight default categories). Our interface is "single-click" in that a single click chooses the category and begins the search.

The explicit category selection helps to clarify the user query, providing more information about her information need than a query alone. This information is utilized to choose the sources and query modifications, as well as to select the result scoring function.

For our simple interface, we provided eight default categories from which the user could choose. Any authenticated user can modify the categories listed on her simple interface. Table 4.2 provides a list of the eight default categories and their description. This interface makes it easy to utilize a small list of categories, but it is unlikely that this type of interface would scale to thousands of categories.

In the simple interface, besides entering a query and choosing a category, a user has two other options. A user can specify the number of results the system processes, between 10 and 500, and

Figure 4.2: The Simple Input Interface of Inquirus 2

can specify the type of output interface, Java, Javascript, or plain HTML.

The advanced interface adds several categories and options for the user. Table 4.3 describes the available advanced interface options and their description.

For both the simple and the advanced interfaces, the list of possible categories is determined runtime. Prior to the user study, Inquirus 2 provided users with the ability to log in and customize their personal interface. Any user could easily choose which categories to display, and what text was on each button. Inquirus 2 supports the ability for individual users to have different personal categories. At the time of this writing, it did not support the ability for external users to create custom categories, even though the process has been automated, as described in Chapter 6.

Output user interface

To reduce the perceived search latency, and to improve the ability for users to identify a result as useful, Inquirus 2 utilizes an incremental output interface and query-term context. As results are scored by the system, they are immediately sent to the interface for display. Each result provides a score, title, URL, and query-term context. The score is based on both the user query and the chosen preference category. Query-term context provides users extra information about how particular query terms are used in each document, allowing a user to predict more accurately if a document is likely to be useful for her needs. To improve compatibility, Inquirus 2 supports three different incremental interfaces. The Java-based incremental interface is the most powerful, but may not operate on all browsers. The Javascript-based incremental interface only allows the top four results to be displayed. The HTML 1.0-compliant interface will work on all browsers, but does not re-order listed results before the search has completed.

Java interface

Figure 4.3 shows a screen shot of the Java incremental interface for a search in progress. The

Name	Description
Topical relevance	This category is a default category when none of the others work; only the query is considered
Personal homepages	A user is searching for either someone's homepage, or a person's homepage about a subject
Research Papers	Query is treated as either a research paper subject or an authors name
Product Reviews	Pages that are reviews, intended for hardware or software products, but also designed to work for movies, music or any other review page
Guide FAQ how to	Pages that are guides about a topic, a FAQ about a topic, or a general "how to" document about a topic
Calls for Papers	Pages are the actual CFP for a conference, meeting, or workshop. Pages which simply list many CFPs by name are not counted. Date is not considered in this category
Genealogy	Pages related to genealogy, such as family trees, information on immigration, pages related to locating information about a persons heritage or history
ergonomics	Pages related to ergonomic products, or RSI and related conditions as well as general RSI and related information

Table 4.2: Categories from Inquirus 2 simple interface

listed results are always in sorted order, with new results inserted as appropriate. A user can click on any result and view the resultant web page in a new window to prevent interference with the running search. The interface retains the status of each result and sets the color accordingly: unclicked results are blue, clicked results are black, and the currently selected result is red. A user can scroll up or down in the list of results to see any block of ten results. As new results are found, the set of results in any given range, say 11-20, can change. The Java interface also supports the ability for users to change scoring criteria while the search is running by clicking on the "Sort +" button. Section 4.3.1 describes the design of the applet and interface architecture in more detail.

Option	Description
Hits	The number of results the system will process: 10 - 500
Display	Choice of display query term context, summaries, or just the URLs
Display num	The number of characters of query context: 50 – 200
Tracking	Enable or disable result click tracking
Java	Choose the format of the output interface, either Java, Javascript, Firewall, or standard HTML
Category	This option allows the user to specify the desired search category

Table 4.3: Inquirus 2 advanced interface options

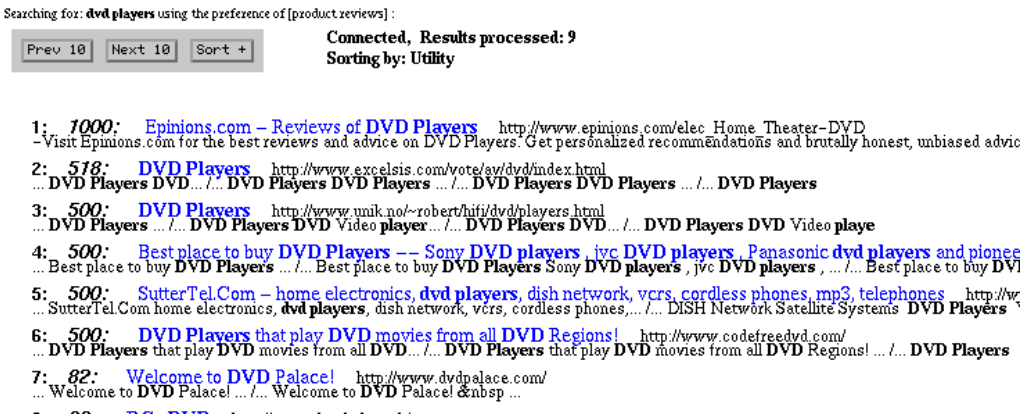

Figure 4.3: Incremental JAVA interface of Inquirus 2

Figure 4.4: Incremental Javascript interface of Inquirus 2

Javascript output interface

Since some users cannot or do not wish to enable Java, we created a Javascript version of the incremental interface. Figure 4.4 shows the Javascript incremental interface. Like the Java interface, the results are always kept in sorted order. Unlike the Java interface, there are no user options, and only the top four results are displayed. The Java interface operates inside the main browser window; the Javascript interface operates as its own window, updated each time there is a change to the current set of top four results.

Section 4.3.1 describes the design and implementation of the Javascript-based incremental interface in more detail.

HTML 1.0 incremental interface

To improve compatibility and to provide users with system feedback, the main search page uses an HTML 1.0 compliant incremental interface. This interface may be used in conjunction with either the Java or Javascript interface. Due to limitations in HTML 1.0, it is not possible to change

content already displayed. The Java and Javascript incremental interfaces will always keep the results in correct sorted order; however, the HTML interface does not resort printed results while the search is running, but only prints those scoring over a given threshold.

When each result is processed, if the score is above the minimum threshold, the summary and score are displayed immediately. If the score is less than the threshold, then either the top scoring result found so far (less than the threshold), or a text message indicating progress, is printed about once every five seconds. This policy ensures that users are provided feedback as the search progresses, and guarantees that a user will never have to wait more than five seconds to see the "best so far" result. If a search finds many high-scoring results, the user will have many results to choose from before the search completes.

At the end of the search, the main search page always returns the top 100 results in sorted order.

4.2.2 Dispatcher

The dispatcher of Inquirus 2 (Figure 4.5) provides two primary improvements over the dispatcher of a regular metasearch engine. The dispatcher of Inquirus 2 utilizes need-based source selection combined with category-specific, source-specific query modifications. The combination of these two options reduces the effects of the web search horizon by increasing the precision of the results, allowing Inquirus 2 to ask for fewer results for each search engine request. Source selection, although not used for this purpose in Inquirus 2, can be utilized to reduce the chance that a search engine will become hostile, as described in the limited cooperation problem. Choosing different sources for each search reduces the query load to individual search engines. This also improves scalability of a metasearch engine.

Inquirus 2 allows selection of both special-purpose and general-purpose search engines as appropriate for the user's need. The combination of the decisions of source selection and query modification into a single decision allows for more than one query to be made to a single search engine. The best two search engine requests may be to the same search engine, but with different query modifications. It may also be the case that for a certain query, a general-purpose search engine with a modification may perform better than a special-purpose search engine.

Section 4.3.2 describes the implementation of the dispatcher in more detail.

4.2.3 Result Processor

The result processor of Inquirus 2 (Figure 4.6) is an extension of the result processor of Inquirus (Figure 4.1). To improve performance, a selective download module is added. Inquirus 2, like Inquirus, has the ability to download results to provide more information, improving its ability to

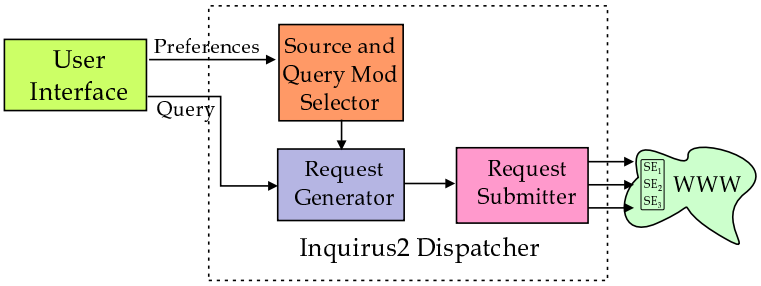

Figure 4.5: The dispatcher of Inquirus 2

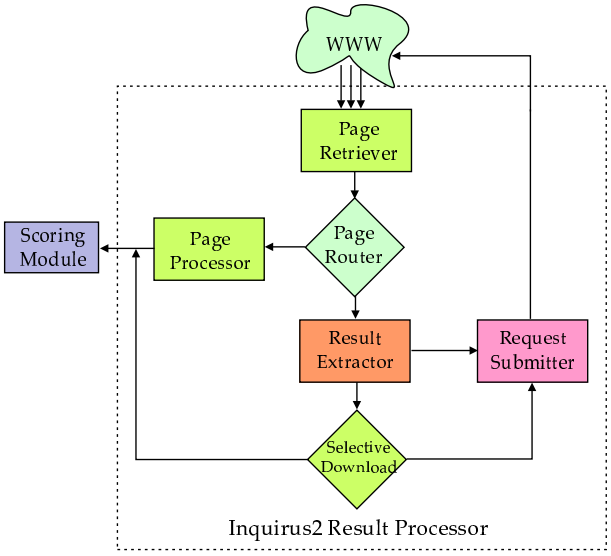

Figure 4.6: The Result Processor of Inquirus 2

predict usefulness. Inquirus 2 improves performance, and reduces search latency, by choosing not to download results that can be scored without download.

Selective download reduces the effects of the limited information problem, the temporal search problem, and the multiple interfaces problem. Even though a search engine might not provide full document text, in some cases the title, URL, and summary are sufficient for scoring. Downloading results only when necessary allows use of the information provided, when it is sufficient. When the information is not sufficient, downloading provides a method for obtaining the necessary information. To reduce the effects of the temporal search problem, scoring some results without download significantly reduces the resource costs, and when combined with the incremental interface, reduces perceived latency. The selective download module considers the specific scoring function for each search. When a special-purpose search engine is used, the specific fields provided may be useful, such as a news specific site providing a date. The ability to utilize the fields provided when needed reduces the effects of the multiple interfaces problem.

Figure 4.3.3 shows the details of the implementation of the selective download module. The goal of the selective download module is to determine if there is sufficient information to score a given result accurately, based only on the summary information (title, URL, etc...) returned by the search engine. If yes, then the result is sent directly to the scoring module for scoring; if no, then the result is queued for download to obtain values for the needed attributes. Certain attributes, such as the number of links on a page, are never known (as reported by the current output interfaces of the search engines queried), and if required for scoring would always require a download. Other attributes, such as number of words in the title, or if a particular query term is in the URL, are always known from the summaries (assuming the summaries are believed to be accurate). Some attributes, such as the predicted topical relevance or the predicted score for a category classifier, may not be known exactly, but can be guessed with some degree of certainty. The level of certainty can be considered when queuing a result for download. Although not implemented in the current version of Inquirus 2, it would be very easy to reduce the required certainty requirements if system load were high, reducing the number of results downloaded. Likewise, if the system load were low, and the user were very patient, then more certainty could be required, causing more accurate predictions but more result downloads.

Of the 11,279 total documents processed during the user study, 1844 or about 16% of them were scored without download. The design of the selective download is such that documents selected not to be downloaded are always classified as positive and predicted as topically relevant, implying a very high score. Of the documents users judged as “highly useful” in our user study, 29% were found without download. The user study, Chapter 5, describes the experiments and results in more detail.

4.2.4 Scoring Module

Inquirus 2 improves the ability to identify useful results by considering the user’s need when scoring results. Need-based scoring is the ability for a metasearch engine to choose the scoring functions based on the user’s information need, as opposed to considering only the user’s query. The scoring module of Inquirus 2 provides independent result scoring, which is combined with an incremental interface and selective download to reduce the effects of the temporal search problem. The scoring module of Inquirus 2 also allows use of meta-attributes, where the specific attributes used depend on the search engine queried. For example, a different research paper classifier can be applied to results from Google than to results from a research paper-specific search engine such as CiteSeer. The use of meta-attributes reduces the effects of the multiple interfaces problem by allowing a score to be computed from the attributes available for each search engine, even though

Scripts	Description
Main search script	The main script which performs the search
Logging scripts	User clicks are logged through a logging CGI script
User study scripts	Scripts to support the voting for results
User authentication	Scripts to support authentication and account management
User customization	Scripts to allow the user to control and customize their interface

Table 4.4: Types of CGI scripts used for Inquirus 2

the attributes available may be different.

Scoring of a document can be based on any of the attributes Inquirus 2 can compute. In addition, the scoring module allows use of arbitrary page classifiers, providing a nearly infinite array of attributes determined at run-time. Any user can use her own scoring functions as desired. Chapter 6 describes the procedures used to learn new preference categories and the associated page classifiers.

The scoring module of Inquirus 2 provides several options for improved performance. There are dozens of attributes and a virtually infinite array of classifiers that could be used to score a document. The scoring module is designed to load only the code that is needed. A search that scores documents based on a research paper classifier and topical relevance will not load the page classifier for personal homepages. The design is modular and allows for easy addition of new attributes and new scoring functions (preference categories). The implementation of the scoring module for Inquirus 2 is described in detail in 4.3.4.

4.3 Implementation

Inquirus 2 is implemented as approximately 20,000 lines of PERL CGI scripts and associated libraries. The basic libraries to download web pages were based on freely available sources. To improve performance of Inquirus 2, a SQUID web cache was used. Each architectural component was implemented to facilitate easy incorporation of new options and functions, and to allow the specific features to be used by other projects at NEC.

The overall project implemented five classes of CGI scripts. Table 4.4 describes each class of scripts. The main search script implemented the modules that define the Inquirus 2 architecture, and the remaining four classes of scripts facilitated use and study of the system.

4.3.1 User Interface

The input user interface for Inquirus 2 is a simple HTML web page, generated through the main CGI script. The specific categories listed and the associated options are determined at run-time based on the options files associated with the individual user and the entire search system. Authenticated users can easily edit their options on the main interface via an associated customization CGI script. Currently any authenticated user can change the category buttons. Buttons can be removed, or new buttons corresponding to a different category can be added. The text on the buttons can be edited as desired. The heading, title, and a few other options of the input user interface, can be altered by modifying the configuration file. Inquirus 2 can be run in "single-preference" mode by specifying the default preference, and by providing a new title and options choices – all by changing a few lines in the main configuration file. The customization CGI scripts operate by modifying the associated configuration files.

There are three different output user interface options for Inquirus 2. The default interface is Java-based and is supported through addition of special functionalities to the main user interface code. The Javascript-based and HTML-based interfaces were implemented through the regular interface libraries and were written entirely in PERL.

The HTML 1.0 interface is implemented by examining every result processed and determining if the result's score is either above the specified threshold, or is the highest scoring so far (if no results over the threshold have been found). All results scoring over the threshold are immediately printed. If no output has occurred for five seconds, then the system prints the highest scoring so far or a message indicating search progress.

The Javascript-based user interface sends several Javascript functions to the output HTML stream. These functions, when called, set the result in a given rank position, one through four, and push the remaining results down by one, followed by a re-render of the output window. The server keeps track of the current top four results, and if a change occurs to the top four, the server sends the appropriate Javascript function call to the output HTML stream. From the user's perspective, every time a new high-scoring result is found, the Javascript-based incremental interface window gets redrawn, ranking this new result accordingly.

The Java-based incremental interface is more complicated and required creation of a multi-threaded Java applet. The applet has two threads, one for the network and receipt of results from the server, and one for display. Results are sent asynchronously from the server, and must be processed immediately. The applet must also respond to external (and internal) redraw events requiring separate threads for the interface and the network. When a result is received, the data is stored and the new item is inserted into the correct rank position. Each result has fields of: title, display URL,

click URL, summary, and score array. A result can have multiple scores, and the user can change which score is used as the sort key by clicking on the "sort +" button.

Each result has an associated status: seen or unseen. There is also a currently selected result. The currently selected result is colored red, all seen results are colored black, and unseen results are colored blue.

To facilitate quick redraw and easy insertion, a modified binary tree structure was used. The generic balanced binary tree structure was modified to include a "next sort" and a "previous sort" pointer. Given any result, the next sorted result can be found in constant time. The extra fields allowed for constant time to render any block of ten results, regardless of the total number of results. The balanced binary tree structure ensured that new results were inserted in $log(n)$ time. Special precautions were taken to prevent the receiving thread from modifying the pointers for an item at the same time the display thread was scanning for the next result, causing the possibility of a runtime error.

The Java applet supports two communication methods: TCP sockets, and HTTP. If the applet is using HTTP to connect, then the CGI script does not send anything other than the raw applet data. The normal mode, TCP socket-based, is preferred since the server can both send HTML data to the user, providing more information and a better interface, and send the raw result data to the applet. The CGI script allows for the applet to connect back into it on a random socket to receive the raw data. If used in socket mode, every time a result is processed, the raw data is printed out to the socket to which the applet is connected. In socket mode, the user's web browser has two simultaneous connections to the server, one for the HTML data and one for the raw data. These connections must remain open for the entire length of the search, and as a result, hardware costs for scaling the architecture may be increased.

4.3.2 Dispatcher

The dispatcher is implemented as a set of PERL code designed to choose the sources and query modifications, and to output a set of valid search engine requests. The user's selected preference corresponds to a single line in an associated text file. This line, loaded at run-time, defines the preference name, the preference description, the set of sources and associated query modifications and the maximum number of requests for each, and the scoring function. The dispatcher extracts the associated list of search engines and query modifications.

For each search engine there is an associated set of functions for request generation and result extraction. The request generation code accepts as input the user's topical query and an optional query modification. The output of the request generation code is a valid search-engine request.

Currently, Inquirus 2 does not support POST requests.

The associated file listing the search engines and associated query modifications can be edited easily to add or remove search engines, or to change query modifications. The search engine functions are modular in design, allowing for easy maintenance and easy addition of new parsers. These stand-alone functions have been used by other projects at NEC.

Currently, each preference has a fixed set of search engines and associated query modifications. This set is determined based on the results of the QMLP algorithm described in Chapter 6.

4.3.3 Result Processor

The result processor is probably the most complicated module of Inquirus 2. Figure 4.6 shows the sub-modules of the Inquirus 2 result processor. The page retriever and request submitter are implemented using custom data structures combined with the freely available LWP libraries. The custom data structures facilitate keeping track of the source and status of pending requests. Each web page has an associated search id. The search id indicates if the result is a page requested for download or a search-engine response. If it is a search-engine response, the specific search engine is contained in the search id.

Search-engine responses are sent to the appropriate search-engine parsing libraries. Each parsing function is independent, and returns a list of "page items". Each "page item" is an associative array wherein each field corresponds to an attribute extracted from the parser. Typical fields for a "page item" include the title, URL, search engine rank, and summary. From those fields, a prediction of topical relevance and of the category is made, if possible.

If a response is from a search engine, the result processor can request more results. Associated with each search engine and query modification is a "request limit". The request limit specifies the maximum number of requests that can be made to a particular search engine for a given query. If a search engine returns only ten results per request, and the request limit was four, then the top forty results could be retrieved, ten at a time. The request limit is used to allow more than the minimum number of results (typically ten), but to provide an upper bound to prevent overburdening any single search engine. If more than one query modification is used for the same search engine, each request is treated independently with its own limit.

When a document is downloaded, the result processor must evaluate the required attributes. The scoring function specifies which attributes are needed. The result processor saves system resources by loading code only for functions that are needed. A scoring function that requires a category classifier, such as "research papers", will load only that classifier's feature vector.

Topical relevance is a complicated attribute. When a document is downloaded, a topical-

relevance function similar to that used by Inquirus is used. Section 4.1.2 describes the three factors considered when evaluating topical relevance. Inquirus assigns an overwhelming weight to the percentage of query terms, while Inquirus 2 uses a more balanced weighting.

The selective-download module may guess the topical relevance based on the summary information provided from the search-engine response.

Selective Download:

The selective download module compares the required attributes for the scoring function with those that are known with sufficient certainty. If a required attribute is missing, the document is downloaded. If all required attributes are known, the document is not downloaded, and instead is sent directly to the scoring module for scoring.

Many attributes, such as the percent of query terms in the title, are almost always known from the summary information. Many attributes such as the total wordcount are never known from the summary information. The challenge is for attributes that may be guessed based on the summary, such as topical relevance or a category classifier.

For example, a document titled: "Top ten DVD players reviewed" with a URL of "http://www.reviews.com/dvd_reviews.html" and a summary of: "Reviews.com presents comprehensive reviews of the ten best selling DVD players." is probably topically relevant for a query of "DVD players" and would most likely classify as positive for a classifier for product reviews. A document titled "RandomSite.net" with a URL of "http://www.randomsite.net/stuff/today.html" and a summary of "Randomsite.net presents our usual array of stuff today's special is ..." does not seem obviously about DVD players, and is not obviously a product review, but it could be.

The selective download module can utilize attributes whose values are not certain. In the above case we could imagine two attributes: *topical relevance* and *reviews*. Topical relevance refers to the extent to which the document is about (or relates to) the query. Reviews refers to the page classifier for product reviews; a positive classification indicates the document is in the category of product reviews, and a negative classification indicates the document is not. When trying to determine values for these attributes from a summary, there is uncertainty. The second document could be topically relevant and a product review, but the summary might not be representative.

The selective download module allows use of summary classifiers and predicted topical relevance when making download decisions. When available, a summary classifier can be used to predict the category of a given document, based only on the summary information. Summary classifiers are trained on short summaries of known documents, and utilize title, URL and summary-text features. Figure 4.7 shows a sample graph of the numerical output of a regular binary classifier and the meaning of that output. When the classifier returns a positive score, the document is assumed

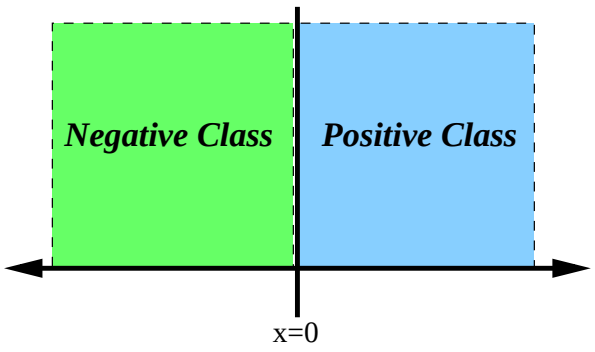

Figure 4.7: A normal classifier has two regions: positive to the right of 0 and negative to the left of zero

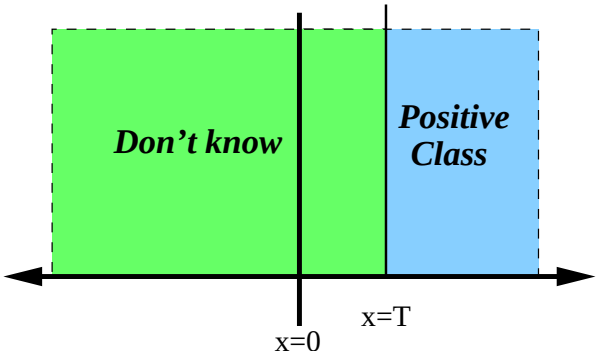

Figure 4.8: The modified summary classifier has two regions: positive to the right of the threshold and don't know to the left of the threshold

classified as positive; negative score implies a negative classification. Zero is equally likely positive or negative.

The problem with using a summary classifier is that the negative classification is not necessarily a true negative. A summary that is not representative of the document may result in a false negative. Downloading a document can provide the full-HTML, allowing a more accurate classification, but at a higher resource cost. To reduce the number of false negatives, the output of the summary classifier is redefined as shown in Figure 4.8. Instead of having a negative document assumed to be negative, the output is "Don't Know." In addition, a slightly positive document may be a false positive. To reduce the number of false positives, a threshold is used. If $x = T$ then the results can be classified either way. The higher the threshold, the fewer the number of documents that are classified as positive, but the lower the false-positive rate. Although not used in Inquirus 2, it is possible to add a third region, as shown in Figure 4.9, with a negative threshold. Documents that are far to the left (highly negative scores), can be assumed to be definitely negative. A document considered as "don't know" can be queued for download. The threshold can be adjusted based on the resources available. In some cases, such as for personal homepages, the summary classifiers

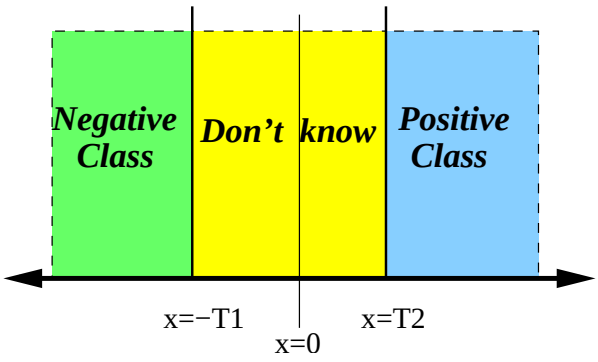

Figure 4.9: A three-output classifier: The right region is positive, the left region is negative, and the middle is don't know. Two different thresholds provide the boundaries

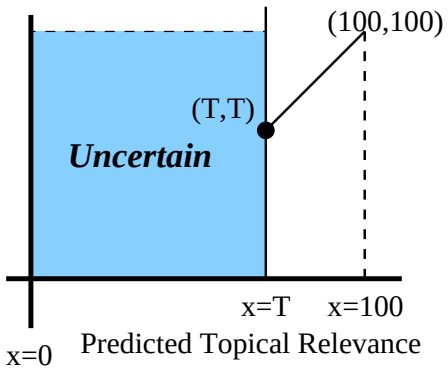

Figure 4.10: Summary topical relevance function is uncertain if the the predicted topical relevance is less than the threshold

have accuracy very close to the full document classifiers, and as a result, the summary classifier can be always believed. Chapter 6 describes how summary classifiers are learned, and the relative accuracy as compared to the full-page classifiers for each of the categories used in the user study.

Uncertainty also exists for predicting topical relevance. A document that mentions user query terms in the title, summary, and URL, is likely to be topically relevant; a document not mentioning the query terms anywhere in the summary, title, or URL is not necessarily not topically relevant. In addition, topical relevance is not binary; a document that is somewhat topically relevant should be scored differently than a document that is strongly topically relevant.

Inquirus 2 resolves this by implementing a summary topical relevance function based on the title and summary. If a search engine returns an abstract, or a topical relevance score, those may be considered instead of or in addition to the summary and title. The summary topical-relevance function produces a numerical score from 0 (not topically relevant) to 100 (completely topically relevant). If the score is over the threshold, currently set at 75, it is considered acceptable; otherwise, the document must be downloaded to resolve the score. Figure 4.10 shows the two regions for

the topical relevance summary function. The scores less than the threshold are always considered "UNCERTAIN"; scores above the threshold are believed. We have experimented with using neural networks and other methods for returning a probability distribution function (PDF) for the topical relevance scores specific to each search engine, although these methods are difficult to train and require more resources to evaluate.

Section 5.2.2 presents user study data on the effectiveness of the selective download module. No results from the special-purpose search engine ResearchIndex (CiteSeer) were downloaded, since the abstract produced a believable topical relevance score.

4.3.4 Scoring Module

Inquirus 2 utilizes need-based scoring. Each user can utilize her own scoring function for each search. To implement need-based scoring, the user preference category that determines the sources and query modifications also specifies the scoring function.

Originally Inquirus 2 allowed any PERL function as a scoring function. Code was added to allow access to the built-in variables for each result. This method allowed for very complex scoring functions, but security concerns make it unwise to permit users to create their own functions. In addition, since any function could be used, the task of the selective download was especially difficult.

Currently, Inquirus 2 supports the original style functions as well as a newer, restricted, format. The new scoring functions are restricted to a summation of piecewise linear functions. To improve flexibility for dealing with different sources and to improve the task of the selective download module, meta-attributes are permitted.

The format of the new functions is a list of attributes, weights, and a list of points to be used as the piecewise linear function: (w_k, a_k, P_k)

Where: w_k is the weight, a_k is the attribute name, and P_k is a list of points for the piecewise linear function.

For example, the scoring function for the category "Research papers" used in the user study was:

$(.5, M{:}new\_researchpapers,'(-1,0)(.5,100)')$ $(.5, quicktr,'(0,0)(75,75)(93,100)')$

In this case there are two attributes, *M:new_researchpapers* and *quicktr*; each is assigned a 50% weight and has a corresponding piecewise linear function, as shown in Figure 4.11. The attributes are both defined to permit the use of the summary functions. The attribute *M:new_researchpapers* is custom-defined to specify that for the general-purpose search engines, a summary classifier can be applied if the document is not downloaded, and a full classifier is used if the document is down-

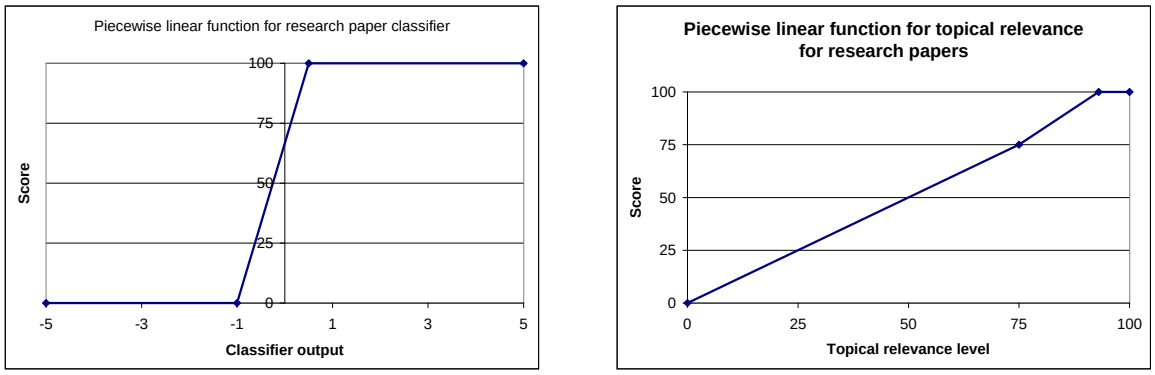

Figure 4.11: The piecewise linear functions for the research paper classifier (left) and topical relevance (right).

loaded. For the special-purpose search engine ResearchIndex, the attribute *M:new_researchpapers* is fixed at about 0.5, since every result from ResearchIndex is assumed to be a research paper. The selective download module is able to determine from which source a result comes from. If the result is from ResearchIndex, the attribute *M:new_researchpapers* is defined, and need not trigger a download or apply any classifiers. The attribute *quicktr* is defined by default to use the summary classifier for general-purpose search engines. If there is a full abstract, such as the case for results from ResearchIndex, the summary classifier is always believed; otherwise, a threshold is used as is described in Section 4.3.3.

The new style of scoring functions has no limit on the number of attributes that can be used, although for Inquirus 2, most scoring functions use only two attributes. Meta-attributes can be defined for each scoring function as desired. A meta-attribute is evaluated based on the search engine that returned the result. These rules are in plain text and contained in the same files that specify the scoring functions. A meta-attribute can be defined for a set of search engines, an individual search engine, or as a default. A meta-attribute can also specify only for downloaded or not-downloaded documents. As a simple example, a document that is downloaded should use the full-page classifier, while a document that is not downloaded should use an appropriate summary classifier for the source search engine.

The scoring module for Inquirus 2 is designed to compute a score for each result as soon as sufficient information is available to score it. Scores returned from the scoring module do not currently depend on any other result found, providing the property of independent result scoring.

4.3.5 Parallel Processing

The current design of Inquirus 2 is not multi-threaded; instead, it utilizes the freely available ParallelUserAgent libraries to permit the result processing loop to be called each time a web page is returned. All other modules are called accordingly, as described in the implementation of the Result Processor, Section 4.3.3. The current termination condition is based on the number of results processed and there is an explicit timeout in place for each web request (both search engine requests and result download requests). Since the interface is incremental, the user is always able to obtain the top ranked results before the search completes.

Future work will include implementing the code as distinct threads and permit distribution of these threads across multiple processors. The property of independent result scoring combined with an incremental interface facilitates parallel processing.

4.3.6 Other Tools and Libraries

In addition to the basic architecture, we implemented several specific CGI scripts. Table 4.4 lists the extra scripts we implemented.

Logging scripts

To facilitate analysis of the user data, special logging code was implemented. All three interfaces support use of an alternate click URL. The special URL calls our logging CGI script to record the click event. Each click event is associated with the specific search and result item to permit detailed analysis.

In addition to logging user clicks, logging libraries were designed to log all search events, user requests, authentication failures, and final search ranks. To comply with the requirement that the user study be anonymous, the logging code was adapted to separate potentially identifiable information (the user's IP address) from the search information (query and preferences).

User study scripts

The user study was built on top of the Inquirus 2 system. An alternate interface was added, and custom HTML start pages were created. When a user participated in the user study, the interface was programmed not to list any results, but instead to call a special vote program once ten or more results were found. The special vote CGI script interfaces with the currently running (or completed) search associated with that user. This was accomplished by parsing the log files generated by the search itself. When a user voted, a new vote logfile was created to ensure that the user was not presented with the same result more than once.

The user study scripts were designed to ensure that users must remain anonymous, but usage

by the same user could be followed. They were also designed to operate while the search was still progressing to reduce the wait time for a user wishing to vote. After a user completed voting for seven results, a special CGI script would show her the final sorted list, allowing her to click on results that might not have been presented during the user study.

User-authentication scripts

Inquirus 2 supports a wide range of authentication and resource-management options. Special tools were created to allow users to log in, save or change their password, or to log out. User passwords were stored in a local Perl DBM file, in such a way that they could never be recovered, only compared against a provided token. Not even the system administrator of Inquirus 2 could regenerate a user's password from the database file without breaking MD5 hashes or performing a brute force attack. Each user was also assigned a semi-random extra bit string to improve security of their hash. All authentication tokens were generated based on a variety of parameters including the user's IP address.

In addition to strong authentication code, the system provides the administrator strict control of the usage of the system. The authentication libraries allow specification of individual or group user usage and resource limits. The simple configuration files could limit "internal users" to a dozen simultaneous connections, while outside users could only make five, unless they are registered, in which case they are permitted to make six. The authentication module code also allows restricted access only to the system administrator for testing purposes.

Our user study was required to be anonymous. Shortly before the user study began, the system was modified to remove all authentication tokens automatically and prevent users with accounts from logging in. Instead, all users were assigned an anonymous user ID and required to click on an agreement document. The authentication system ensured that only users who agreed to the terms would be able to access any of the system scripts. The system allows for easily updating the terms, and ensuring that all users have seen the most recent terms.

All authentication, both before and after the user study, operated by using a single cookie sent to the user's web browser. Special CGI scripts were created to allow removal of any system set cookies.

Customization scripts

Prior to the user study, any authenticated user could customize her input user interface. This customization was performed through a CGI script that allowed editing of the associated options file. Each user would have her own options file to describe the specific mapping of button names to preferences. Deletion of all preferences would set the default options file for that user, otherwise changes would be written to her personal options file. Although this feature is not yet implemented,

we intend to explore the ability for authenticated users to create custom preferences, either based on learning of new custom categories or a combination of or re-weighting of existing preferences.

CHAPTER 5

User Study

To understand user judgments of usefulness and the effectiveness of our architectural improvements, we performed a user study. Our user study was designed to collect data for three purposes: analyze user judgments of usefulness and topical relevance and the effects of category classifiers and query modifications, collect data about the effectiveness of each of the four architectural improvements, and collect of data related to metasearching in general.

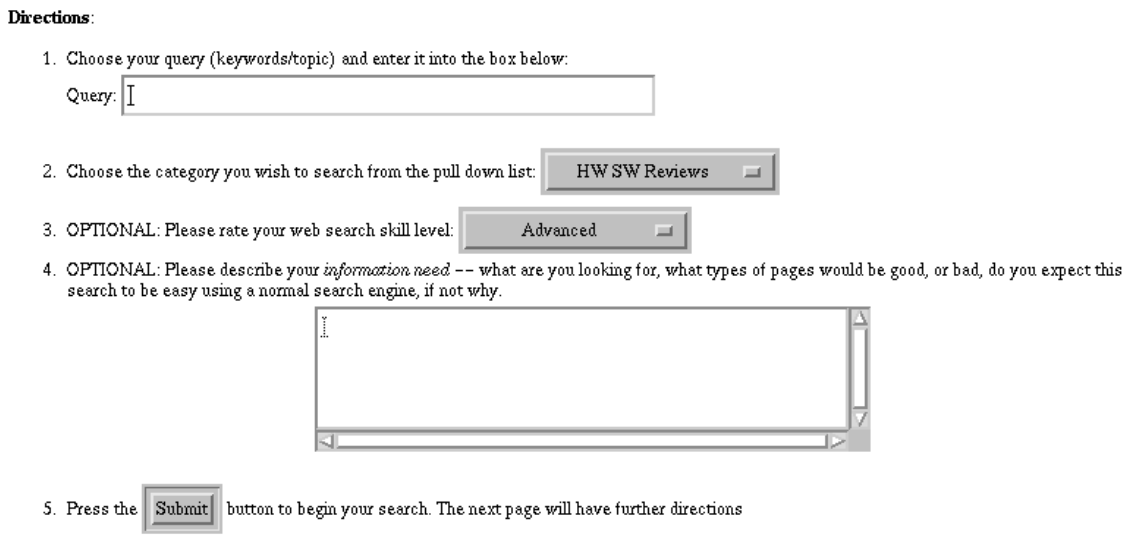

Figure 5.1: A screen shot of the user study initial query form

Our user study was built on top of Inquirus 2, asking users to vote on up to seven documents per search, where a vote consisted of optional comments plus a judgment of topical relevance and usefulness. In addition to saving all user votes, we also stored internal processing data useful for analyzing the effectiveness of each of our four architectural improvements: incremental interface, source and category-specific query modifications, selective download and need-based scoring.

Field	Description
Query	This is a short write in area where the user enters their topical query
Category	A pull down list of four categories to help describe the user's need. The four categories available were: Research papers, Reviews, Personal homepages, and "none of the above" that indicated to use topical relevance only
Web search skill level	User's were asked to optionally provide an indication of their web search experience, there were five options: Novice, Limited experience, Good, Advanced, and Expert
Detailed description	Users were asked to optionally provide a detailed description of their information need

Table 5.1: Fields for the user study input form

Item	Description
User queries	Every user search, chosen category and any options. These data were recorded for both the special user-study interface and regular users of the Inquirus 2 search interface
Search-engine requests	All search-engine requests were logged, including both modified and unmodified and special purpose search engine requests
Individual results	Every result processed by the system was logged. For each result, the time (relative to the start of the search), the relevant attributes (and classifier scores), the total predicted score (and final rank), the ranks of each search engine reporting the URL, and the title and summary, and if the document was downloaded. From the search engine and rank, we could determine if a particular result was found as a result of a modified query
User actions	All user votes, optional comments from the user study are logged. In addition, all user clicks for regular search results were logged

Table 5.2: Factors recorded or logged for analysis as part of the user study

Our anonymous web-based user study ran from December 14, 2000, through February 16th, 2001, and collected 684 user document votes. There were 82 unique users, with 117 total searches that had at least one document vote.

Our user study data demonstrated several important discoveries about user's judgments of usefulness, including a bounding relation, where the level of the topical relevance judgment forms an upper bound on the level of the usefulness judgment. The study also provided strong evidence that considering categories helps to identify useful documents, and query modifications significantly increased the chance that a random result was of the desired category. The user study data also demonstrated that the selective download module can significantly reduce the perceived latency, by passing highly useful documents quickly to the scoring module and then to the incremental interface.

5.1 Study details

Our user study was set up to collect user judgments of topical relevance and usefulness of web documents. The selected documents were found by Inquirus 2, in response to a user query and user-specified category. User's were not restricted for their queries, but were limited to four choices for their category, one of which was a "none of the above" option. The study was anonymous in that no personally identifying information was collected[1], but through the use of cookies, repeat queries by the same user (computer) were identified.

Over the nearly two months, the study was running, 82 unique users participated (only counting those that provided at least one vote). Each user volunteered and was not paid or otherwise compensated for participation. Volunteers probably heard about the study from either word-of-mouth, personal e-mails requesting they participate (although participation was anonymous), or from going to the Inquirus 2 main page and clicking on the user study link. It is believed many of the volunteers were my friends or family, although unless a user contacted me through out-of-band communication (phone, e-mail or in person), his or her identity was not known for certain.

To participate in the user study, all users were first required to agree to the consent form, a copy of the text is shown in Figure 5.7. The consent form was necessary to comply with the requirements of the Institutional Review Board of the University of Michigan. A user's agreement allowed us to set a single cookie that permitted both tracking of that user's actions and verification they agreed to the consent form. Once the cookie was set, future searches by that user would not require agreement to the original consent form.

The first screen seen by a study participant is shown in Figure 5.1. Table 5.1 describes each field presented to the user for the user study input form. Although optional, users provided a web-search skill level for 94% of the vote bearing searches. Also, users provided optional comments for about 68% of the vote-bearing searches.

Level	Topical relevance text	Usefulness text
1	Not topically relevant	Not useful
2	A little topically relevant	A little bit useful
3	Somewhat on topic	Somewhat useful
4	Very topically relevant	Very useful
5	Exactly the right topic	Exactly what I wanted

Table 5.3: Text associated with each level of topical relevance and usefulness

[1]The user's IP address is always recorded by the web server, but the logging scripts separated this information from user votes or queries.

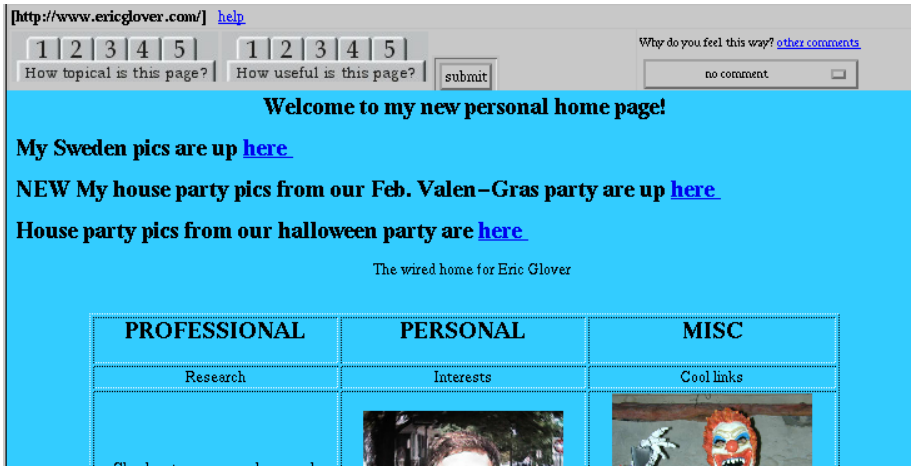

Figure 5.2: A screen shot of a vote window for a query of "eric glover" and a category of "personal homepages"

After specifying search criteria, Inquirus 2 located results. If fewer than ten results could be found, the user would be presented an error and have the option of going back to change his search. Once ten results were processed, the system would automatically pop up a vote window. Figure 5.2 shows a screen shot of a sample vote window.

The "vote window" presented a user with a document for which to vote, and provides three options for entering information: topical relevance, usefulness, and optional comments. There are five options for both topical relevance and usefulness as described in Table 5.3. In addition to voting for topical relevance and usefulness, a user could provide optional comments. The user could click on "other comments" to pop up a free-form write in window, and or could pull down and choose one of eleven default comments. Table 5.4 lists each of the fixed optional comments.

Table 5.5 lists the number of times each of the optional comments occurred:

Each user was asked to vote for up to seven documents. To enhance the data, documents to be voted for were not picked randomly. Each user vote was associated with the specific method used to pick the voted for URL. Four different pick methods were applied to choose a document to be voted for from all the remaining unvoted documents. Table 5.6 describes the four methods used. All searches used the "top-ranked" method for the first document. For the first month of the study, every document after the first was selected using the biased method. After the first month, the other methods were used with a 40% chance of either random or "high ranked" and 20% chance of biased. It should be noted that there was a bias against picking a result from a special purpose search engine (applied to all methods). If a special purpose search engine was picked, 70% of the time a new choice would be made.

Number	Optional comment
0	No comment
1	Wrong meaning of my terms
2	Page had an error
3	Page was too old
4	Page was too short
5	Contents already seen
6	Page was not of sufficient quality
7	Page was too specific
8	Page was too general
9	Page did not have the correct type of content
10	Page did not contain any of my keywords

Table 5.4: The pull-down list of fixed optional comments

5.2 Hypotheses and Analyses

Our user study was designed to analyze three things: usefulness, architectural effectiveness, and general metasearch data. For each of these areas, we present several hypotheses and the relevant user data and analysis. Results should be viewed in light of non-uniform methods of document selection.

5.2.1 Usefulness

Users provided judgments of both topical relevance and usefulness. Usefulness is related to, but not the same as, topical relevance. Usefulness is defined by both topical relevance and other factors, as stated in Hypothesis 1, such as the category of the document.

Hypothesis 1 *Usefulness is strongly dependent on topical relevance, but usefulness is also dependent on other factors. Topical relevance is necessary but not sufficient for usefulness.*

To test Hypothesis 1, we compared paired user judgments of topical relevance and usefulness as shown in Table 5.7. To test the relative dependence, we performed a correlation between the judgments. Topical relevance judgments and usefulness judgments of all documents showed a correlation of 0.779. This relatively high correlation suggests a strong relationship, but also suggests there may be other factors.

The data from Table 5.7 suggests a bounding relationship, where the level of the topical relevance judgment appears to provide an upper bound on the level of the usefulness judgment. Of the 684 judgments, only 25 were to the lower left of the diagonal. Of those 25 documents, 22 were

Optional comment	Number of votes
No comment	374
Wrong meaning of my terms	48
Page had an error	10
Page was too old	6
Page was too short	4
Contents already seen	17
Page was not of sufficient quality	17
Page was too specific	32
Page was too general	51
Page did not have the correct type of content	93
Page did not contain any of my keywords	32

Table 5.5: Frequency of each optional comment

Pick method	Description
Random	A document was randomly chosen from all remaining unvoted for documents
Biased	A document was chosen based on its predicted usefulness score with the following score range and probability: (0-50) - 20%, (50-70) - 10%, (70-85) - 7.5%, (85-95) - 7.5%, (95-100) - 45% – random - 10%
Top ranked	A document ranked as either first or second by at least one search engine (could be modified or unmodified)
High ranked	A document ranked in the top five by at least one search engine (could be modified or unmodified)

Table 5.6: Four methods for selecting document to be voted for

immediately next to the diagonal and could be explained by minor differences in the interpretations of the associated text corresponding to each vote.

There were two documents judged as “Not topically relevant”, but “Somewhat useful.” These documents appear to be serendipitous finds. The first one was a research paper about market-based mechanisms for planning, found for a query of “thread scheduling lotteries” and a category of “research papers”. The second was likely judged useful because the user had interests other than their query. The user was searching for “dreamcast review” with a category of “reviews”, and judged a document about a U2 CD as “somewhat useful”. Although serendipitous finds may not follow the bounding relation, it appears to be a good general rule, given more than 96% of the user votes followed the rule, with only five out of 684 more than one away from the diagonal.

The bounding relation has several implications for IR research. First, it suggests that a binary assumption about the judgments of topical relevance and usefulness are not sufficient; there is a richer set of information by using more than two levels. Second, it demonstrates both that topical

Usefulness judgments \ Topical relevance judgments	1	2	3	4	5
1	203	42	38	14	12
2	3	34	34	21	15
3	2	5	30	32	26
4	0	3	9	30	39
5	0	0	0	3	89

Table 5.7: User topical relevance verses user usefulness, across is topical relevance judgments, down is usefulness judgments. Each square is the number of votes.

relevance is important and that topical relevance is not the only factor involved for usefulness (since there were a substantial number of highly topically relevant documents that were not judged as highly useful). Third, it suggests that a document that is partially topically relevant may be more useful than one that is highly topically relevant.

To verify that the user judgments of topical relevance and usefulness were distinct, we performed a paired, two-tail T-TEST, producing a P of $9.39 * 10^{-48}$. This extremely low P value suggests that the two data sets are from different distributions, supporting the supposition that users interpreted topical relevance and usefulness differently (as they were instructed).

Hypothesis 2 *Usefulness is dependent on the category, such that documents that are of the desired category on average will be more useful than documents not of the desired category.*

Hypothesis 2 addresses the relationship between user usefulness judgments and their specified category. A document not of the desired category should be unlikely to be useful. From the bounding relation, the level of topical relevance provides an upper bound on the level of the usefulness judgment, suggesting a document that is of low topical relevance, independent of the category, is not going to be useful. To test Hypothesis 2, we analyzed the average usefulness of documents judged as highly topically relevant (four or five), as shown in Table 5.8. *Category* refers to the documents from searches where a user specified either personal homepages, reviews, or research papers. Documents found for searches when the user specified "none of the above (topical relevance)" did not have a category-classifier applied, and hence were excluded. *Class+* and *class-* refers to the results of the appropriate category classifier as being positive (document predicted to be in the given category) and negative respectively. Documents found from CiteSeer, the special-purpose research-paper search engine, were assumed to be classified as positive for research papers, even though no classifier was actually applied. CiteSeer was the only special-purpose search engine used for this

user study.

-	category	class+	class-	category and $TR >= 4$	class+ and $TR >= 4$	class- and $TR >= 4$	category and TR=5	class+ and TR=5	class- and TR=5
Average Usefulness	2.26	2.41	2.09	3.67	3.86	3.41	4.22	4.52	3.79
Average Topical relevance	2.74	2.85	2.63	4.58	4.59	4.57	5	5	5
Num docs	459	245	214	168	99	69	97	58	39

Table 5.8: Effect of category on average user judgments of topical relevance and usefulness. *category refers to a user chosen category other than "none of the above", class+ means classified as positive, class- means classified as negative.

Table 5.8 shows the average usefulness of all documents that had a classifier applied, broken down by level of topical relevance (4, 5 or both) and classifier result. Documents classified as positive were slightly more useful on average (2.41 vs 2.09) and slightly more topically relevant. However, we are interested in the documents most likely to be highly useful (4 or 5), so we restricted the set to those judged as highly topically relevant. For topical-relevance judgments of 4 or 5, the average usefulness for documents classified as positive was 3.86 vs. 3.41, though their average topical relevance was almost the same. For documents judged as 5 for topical relevance, the difference was much more significant, with an average usefulness of 4.52 vs. 3.79. An average of 4.52 requires that at least half of the documents were judged as 5 for usefulness. An unpaired, 2-tail, equal variance T-Test comparing the usefulness of the documents judged as 4 or 5 for topical relevance, broken down by classification (positive vs. negative) was 0.0149, very significant. Considering only the documents judged as 5 for topical relevance, the probability improves is less than 0.0001. However, for the documents judged as 1 or 2 for topical relevance the T-Test produced a P value of 0.835, suggesting category has almost no effect for documents judged as not topically relevant (the bounding relation prevents category from having any effect on documents judged as 1 for topical relevance).

Table 5.8 and the T-Tests on the various classes strongly suggests that category is a significant factor of the usefulness judgment when a document is judged as highly topically relevant.

Hypothesis 3 *Results found from an appropriately modified query are more likely to be classified as positive than results found from an unmodified query.*

Hypothesis 4 *Results found from modified queries will have a higher average usefulness than results found through unmodified queries.*

-	category	modified	not modified	both	category and TR>=4	modified and TR>=4	not modified and TR>=4	category and TR=5	modified and TR=5	not modified and TR=5
Average Usefulness	2.26	2.04	2.52	2.04	3.67	3.71	3.66	4.22	4.6	4.16
Average Topical relevance	2.74	2.37	3.14	2.52	4.58	4.51	4.6	5	5	5
Num docs	459	170	252	23	168	49	116	97	25	70

Table 5.9: Effect of query modifications on average user judgments of topical relevance and usefulness.

-	**Modified**	**Unmodified**
Class+	127	100
Class-	44	152
Percent +	75	40
Percent -	25	60

Table 5.10: Query modifications and the classification of results.

Hypothesis 3 states that query modifications will increase the probability a result is classified as positive. Hypothesis 4 states that results found from modified queries are more useful on average than results found from unmodified queries. Given hypothesis 2, if we assume hypothesis 3 is true, then hypothesis 4 should also be true, because more results classified as positive should increase the average usefulness. Unfortunately hypothesis 4 is not supported by our data, and the previously stated logic ignores the possibility that modified queries, although increasing the chance of positive classification, may have other detrimental effects, such as lowering the average topical relevance. However, documents found from a modified query that were judged as 5 for topical relevance demonstrated a higher average usefulness than documents judged as 5 for topical relevance, but found from an unmodified query.

There were 170 documents, in a category (other than “none of the above (topical relevance)”) found as a result of a modified query, and 252 documents found as a result of an unmodified query.

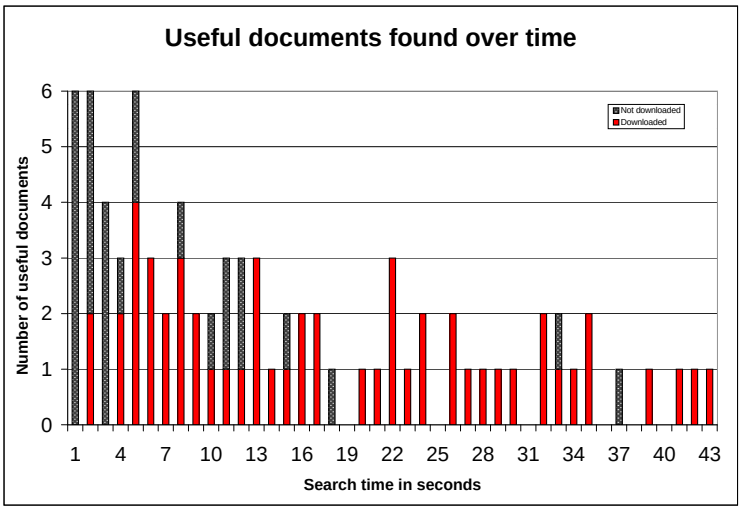

Figure 5.3: Useful documents found at various times throughout the search

A document found by both a modified query and an unmodified query was counted as both. Only 23 documents, less than 15% of the smaller set, were found by both a modified and unmodified query. A document found only from a special purpose search engine, or from one of the "test" query modifications was excluded from this data. Table 5.10 shows the effect of query modifications. Results found from modified queries were nearly twice as likely to be classified as positive as results found from unmodified queries (75% vs 40%). The effectiveness of the query modifications was significant with respect to category, but according to Table 5.9, our query modifications significantly lowered both the average topical relevance and the total number of documents judged as useful. However, a document that was judged as highly topically relevant (5) and found by a modified query was on average more useful than a document that was highly topically relevant, but found from an unmodified query.

An unpaired, two-tailed, equal variance T-Test comparing the usefulness judgments of the set of documents judged as 5 for topical relevance and found from an unmodified query was compared with the set of documents with a 5 for topical relevance and found from a modified query, producing a P of 0.035. This was not as significant as category, but was less than 0.05.

5.2.2 Architectural Components

The second focus of the analysis of the data from our user study was on the effectiveness of the architectural improvements. There were four architectural improvements we wished to quantify: the incremental user interface, source selection and query modification, selective download, and

need-based scoring.

Incremental interface

To reduce the effects of the *temporal search problem*, we utilized an incremental interface. With our incremental interface, results are displayed, in the correct relative rank, as they are found. Hypothesis 5 states that useful results are found throughout the search process. If all useful results were found only at the beginning, or at some fixed point in time, then there is minimal value to using an incremental interface. If useful results are distributed throughout the search process, then an incremental interface allows presentation of some useful results to users, while permitting the search to continue to find more.

Hypothesis 5 *Useful results are found throughout the search process, not all at the beginning, middle or end.*

Figure 5.3 shows the number of useful documents (judged as 5 for usefulness) found and the corresponding search time when they were scored by the system.[2] In our user study, users were not presented with scored results as found, but rather presented with specifically selected results and asked to vote. The time a result was found and scored was saved for analysis, even though the vote may have occurred several minutes after the result was first scored. Figure 5.3 clearly shows that useful results were found throughout the search process. Of all the documents users voted for and judged as 5 for usefulness, 41% (38 out of 92) were found in the first ten seconds of the search. The relatively high percentage is likely due to the selective download module. In the second ten seconds, 20% (18 out of 92) of the useful results were found. About 38% of the useful results were found after the first 20 seconds. It is important to note that the specific times taken are a function of the implementation, and a different implementation may operate faster or slower. However, any metasearch engine that performs information gathering actions will require time. For our implementation, with less than half of the useful documents found in the first ten seconds, an incremental interface seems justified.

Source selection and query modification

Inquirus 2 chooses search engine, query modification pairs for each search. To improve consistency, the same four search engines were used for every search, except for research papers, where a special purpose search engine, CiteSeer [Lawrence et al., 1999] was added. We did not study in detail source selection. Figure 5.4 shows the percentage of useful documents with respect to each of the four general-purpose search engines used in our user study: AllTheWeb, AltaVista, Google, and Northern Light. This graph shows that Google (this includes both modified and unmodified

[2] Search engine latency will affect the times results are first identified.

queries) contributed the most or about 49% of the total number of useful documents. Although Google contributed an overwhelming percentage of useful documents, using only Google would miss about half of the total useful documents.

Other research has studied source selection, and we did not do any in-depth analysis. For search-engine-specific query modifications, we studied the percent chance a random result returned from a modified query would be judged as useful. Section 5.2.1 describes Hypothesis 3, which can be used as one measure of effectiveness of our query modifications. The query modifications as described in Section 5.2.1 demonstrated on average 75% on-category results, significantly more than the 40% of the results found from unmodified queries, although QMLP, our method for learning the query modifications, seemed to reduce the average topical relevance, reducing the total number of useful documents found. However, highly topically relevant documents found from modified queries were more useful on average than highly topically relevant documents found from unmodified queries.

In addition to our work, related work by Agichtein demonstrated a significant improvement for source-specific query modifications when used for question answering [Agichtein et al., 2001].

Although the data from our user study is only for three categories, the results are promising, and suggest that query modifications can be a useful tool in category-specific metasearching.

Selective download

To improve performance and to reduce perceived latency, we utilized a selective download module, as described in Section 4.3. Figure 5.3 shows the distribution of useful documents over time. Documents that were not downloaded make up a significant proportion of the useful documents (29%) and were found in a fraction of the time. In the first five seconds, more than 63% of the documents judged as 5 for usefulness that were not downloaded were found.

The design of the selective download module used in Inquirus 2 was such that only documents scoring high for topical relevance (at least 75 out of 100 using the Inquirus 2 measure) and classified as positive would be not downloaded. Although our predictions of topical relevance were not perfect, Inquirus 2's topical relevance function demonstrated a correlation of 0.49 between our predicted topical relevance and user's actual judgments. Our summary classifiers demonstrated high accuracy for each of the three categories (more than 85%); using a positive threshold, further improved the accuracy. Given the relatively high accuracy for the summary classifiers and the reasonable predictiveness of our topical relevance function, we predict that the average usefulness of results not downloaded by the selective download module should be higher than results downloaded. The average usefulness judgments of all downloaded documents was 2.24, while average usefulness for documents that were not downloaded was 2.75.

Of the 11,279 total documents processed by the system (including those not voted for), 1844

were not downloaded, producing a significant savings in resources and a reduction in perceived latency.

Need-based scoring

To improve the ability to identify useful documents, the user's chosen category was used to determine the scoring function, as opposed to only considering topical relevance. The correlation between our predicted usefulness and actual usefulness was 0.344. This correlation might be so low because of the relatively low accuracy for our predictions of topical relevance, a primary component of our usefulness score. In addition, our scoring functions were fairly simple, and placed an equal weight on topical relevance and category, while our data suggests other functions may be more effective.

Search engine	Total	"None of the above"
AllTheWeb	28	11
Alta Vista	9	4
Google	49	38
Northern Light	15	14

Table 5.11: Results judged as 5 for usefulness by search engine.

Figure 5.5 shows the distribution of actual usefulness judgments for different ranges of predicted usefulness. Although our predicted usefulness score demonstrated a low correlation with actual usefulness judgments, the distribution of user usefulness judgments shows that a predicted usefulness of less than 50 was strongly suggestive of a document not being useful. A predicted usefulness of 95 or higher demonstrated a significantly higher percentage of highly useful documents than the three scoring ranges with lower predicted usefulness.

Table 5.8 shows the effect of category on average usefulness. The significant improvement (3.79 to 4.52) suggests that considering both category and topical relevance can improve the ability to predict usefulness, over topical relevance alone. In our future work we wish to explore improved functions for usefulness, based on what was learned from our user study.

5.2.3 General Metasearch

In addition to collecting data for analysis of our architecture and data on usefulness judgments, we have collected data related to parameters relevant to general metasearch. Two arguments for metasearch engines are that individual search engines have relatively low coverage, and combining results from multiple search engines provides an improvement over any one individually, and no

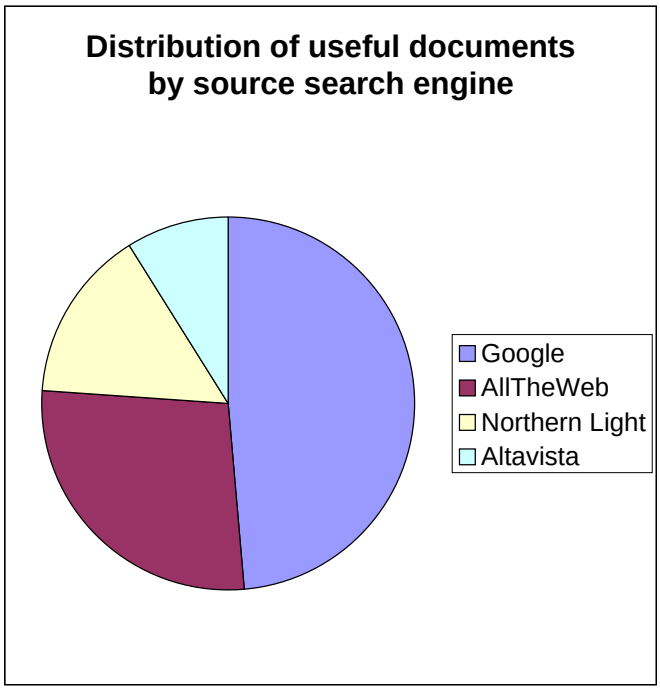

Figure 5.4: Distribution of useful documents by search engine

single search engine provides all the useful results.

To measure these factors we examine result overlap and useful result distribution. Several other studies have concluded very low result overlap, supporting the theory that it is beneficial to combine results from different search engines [Lawrence and Giles, 1998a, Lawrence and Giles, 1998c, Lawrence and Giles, 1998b, Lawrence and Giles, 1999a, Selberg, 1999, Gordon and Pathak, 1999]. A metasearch engine that retrieves all of the useful results from only one source suggests that there is no need to consider other sources, and lowers the utility of a metasearch engine. Selberg discusses the distribution of user clicks as a proxy for judgments of usefulness. He concluded that no single search engine dominated, justifying the need for combining results from multiple sources [Selberg, 1999].

We analyzed result overlap, both between all sources, and overlap as distinct for modified and unmodified queries. Of the 2661 results found as a result of a modified query[3], and the 7043 results found from an unmodified query, only 139 were duplicated. Of the 11,279 total documents (this includes the special purpose search engine CiteSeer and the "test" query modifications) there were 867 found by more than one source (modified or unmodified), or about 7.7%. Of the 2661 results found by a modified query, 59 were found by two (of four), and two results were found by

[3]We only consider results found by one of the first four modifications, we did not count documents from special purpose search engines or "test" modifications.

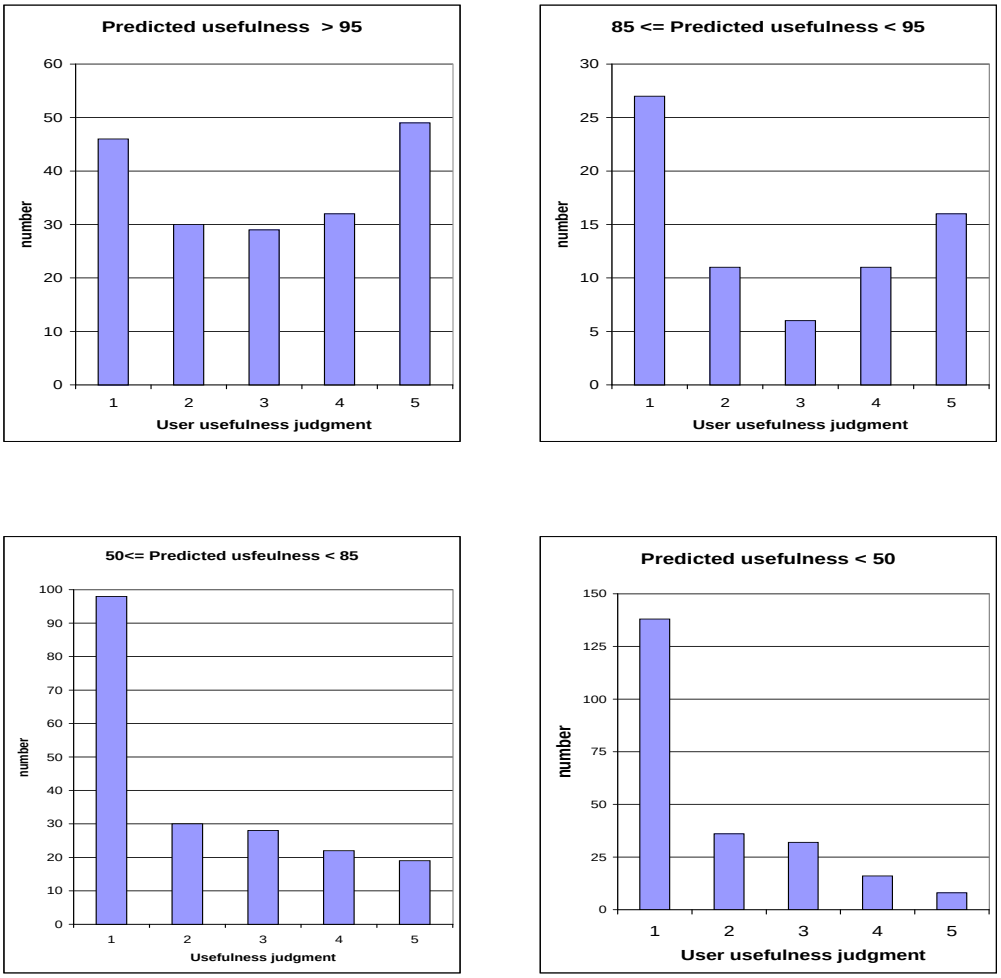

Figure 5.5: Distribution of user usefulness judgments as a function of predicted usefulness

three search engines, with no results returned by all four of the query modifications (excluding test modifications).

Overlap for the unmodified queries was also very low with 385 of the 7043 results found by exactly two sources, 59 by three, and seven results by all four sources. The overlap of results was less than 8%. Of the 446 results that were of a user category other than "none of the above (topical relevance)" that were voted for, only 23, or about 5.2%[4] were found by both a modified and unmodified query.

Table 5.11 shows the distribution of useful results for each of the four search engines we studied. Although Google seems to be a major contributor of useful results, no single search engine

[4]We did not consider the 59 results that were voted for, of a real category and found from either a special purpose search engine, or a "test" query modification.

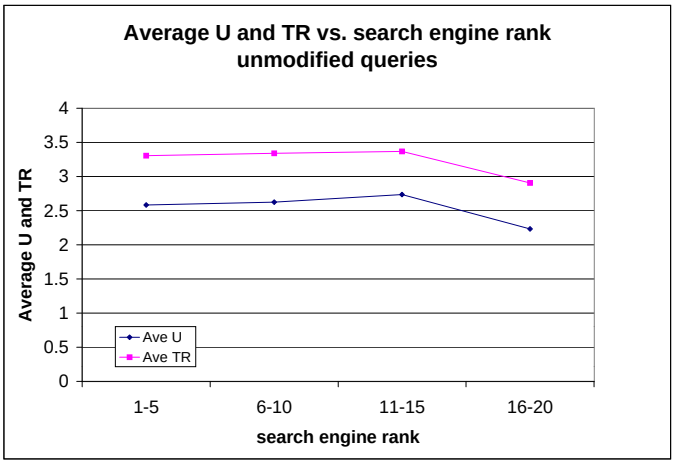

Figure 5.6: Average usefulness and topical relevance judgments as a function of search engine rank

contributed more than half of the total useful documents. For the case of "none of the above" (no real category), Google contributed slightly more than half (57%) of the useful results. We had expected a more equal distribution of useful documents, and it is not clear that one search engine will dominate in the future. The queries we submitted are not representative of typical queries sent to these search engines, and may be biased because of the subjects of this study expecting a "different" search engine. Even though there is a dominant search engine, considering only Google would have removed about half of the useful results.

The distribution of useful documents demonstrates a wide variation of each search engine, but suggests that using a metasearch engine can double the total number of useful results over searching only one search engine. It should be noted that we only considered the top 20 or 30 results from each search engine, so it is possible that requesting more results may have increased the percentage overlap. We feel it is reasonable to consider such a small web search horizon because increasing the number of requested documents will significantly increase resource costs, and risk altering the attitude of the search engine towards hostile.

Figure 5.6 shows the average usefulness judgment and average topical relevance judgment as a function of search engine rank. To smooth out the data, the search engine ranks used for Figure 5.6 were divided into four blocks of five, 1-5, 6-10, 11-15 and 16-20. The correlation between search engine rank and user judgments of usefulness was -0.21, suggesting a very weak relationship.

Web skill level	Number of searches
(0) No answer	7
(1) Novice	9
(2) Limited experience	8
(3) Good, but not advanced	43
(4) Advanced	37
(5) Web search expert	13

Table 5.12: Distribution of user skill levels

5.3 General Observations

In addition to providing a vote, users were asked to provide optional comments explaining their judgments. Of the 684 user votes, 310 votes had an accompanying "canned" optional comment. Table 5.5 shows the distribution of comments. "Page did not have the correct type of content" was the largest complaint with 93 occurrences. The second most frequent canned comment was "Page was too general" with 51 occurrences. A close third was "Wrong meaning of my terms" with 48. In addition users had the option of providing a detailed description of their information need (once per search). 79, or about 67.5% of the 117 vote bearing searches had an associated detailed comment. The vast majority of users (110 out of 117 vote bearing searches) provided an optional web skill level. Table 5.12 shows the distribution of web skill levels. The skill level (3) "Good, but not advanced" was the most common, with 43 occurrences.

Although our user study did not study the generalizability of category-based searching from a user perspective, some user comments suggest that a better interface may be needed. One user stated that the category chosen was not exactly what they wanted, and suggested an new category be added. The majority of comments were more detailed descriptions of the user's information needs, as opposed to comments on the interface or system.

Please read the informed consent form below

If you wish to use Inquirus 2, you must agree to the terms of this document.

Other related documents include: the privacy statement and the detailed project description.

The Inquirus 2 search tool is being used as a testbed to collect data for my thesis. The study is titled: *Analysis of user value assessments of web documents with respect to both a query and a category* and involves collecting anonymous user judgements of document value. The goal of this work is to study the ability of the Inquirus 2 sytem (and its underlying algorithms) to predict *value* of results. Unlike regular search engines, Inquirus 2 attempts to utilize the category you pick as well as the query terms you enter, and we wish to examine how we can utilize the category information to provide better search results for you.

In order to use any data collected from Inquirus 2 for my thesis, I am required to have permission from the University of Michigan IRB Behavioral Sciences Committee. To get this permission, I am required by them to post an informed consent form, this document, for all prospective users of the system to see. Users who do not agree to the terms of this document will not be permitted to perform searches on Inquirus 2 while the logged data could be used for my thesis.

Use of Inquirus 2 is purely voluntary and is intended to be anonymous, unless you voluntarily provide personal information. The system will set a cookie on your computer to track your actions, but will not ask for your personal information, except for an optional e-mail address if you choose to register. At several points during your search you will have the option of providing value (and topical relevance) judgements for search results, as well as optional comments. As with any use of the system, providing of your judgements is voluntary. You must be aware that your use of the system, and any judgements you provide will be logged. For more details on what is logged please see the detailed project description. In short, all queries you perform, and the results you click on are logged. In addition, the links on a result page have been modified to enable me to log which links you click on, if you wish to avoid the logging of clicks on result pages you can simply click on the url at the top of the result page to see the original result page.

As with all web services the source IP address of all web requests is logged. If you are concerned that your IP address may reveal your personal identity, you might wish to consider using a proxy. If you use AOL and you use AOL's browser, you are automatically using a proxy. If you use a dial-up ISP, in general only the geographic location you dialed is revealed. Most ISP's provide a free proxy and assistance on using it. While writing my thesis, if I discover information that I feel could reveal someone's identity, I will attempt to block it out, or obsecure it. IP addresses will not be correlated with specific queries, and in general will be printed in aggregate form only -- i.e. 50% of the requests came from AOL users. If you have any questions you can feel free to e-mail us at inquirus2@ericglover.com.

If you are under the age of 18, you must get your parent's or guardian's permission before clicking the agree button, or using this system. Inquirus 2 sends queries to regular search engines and does not guarantee that only good results will be found, although unlikely, it is possible results found (from the underlying search engines) may be offensive to users. The system does not intentionally show objectionable results, and is designed to find results that are closer to what you are searching for than by a simple text query alone.

Normal use of Inquirus 2 is just like any other search engine, except you have the option of picking a category, such as "personal homepages," or "research papers" to improve the search, both to find better results and to better rank the results which are found. Sometimes Inquirus 2 might appear slower than a regular search engine. This is because Inquirus 2 sometimes pre-reads (downloads) potential documents to better predict how good they are for you.

Data collection for this experiment is done in three ways: First normal system usage, where users enter queries and click results, provides basic implicit data. Second, every result page contains at the top a toolbar where you are asked to "vote" for the topical relevance and the value of each page. Third, users who wish to provide more data, that will be used for my thesis and to improve the system, are asked to utilize *survey mode* where you are asked to vote for seven documents prior to seeing the ranked this. In survey mode, the system chooses what documents (from those found for your query) to present, allowing me to collect data on both good and bad documents. In general voting should take just a few seconds, with maybe a minute for you to read the page to decide how good it is.

All data collected is stored on servers at the NEC Research Institute Inc. ("NEC") and may be used for my thesis, and possibly other related research projects. As stated users of the system are anonymous, and as such, even a full public release of the queries is unlikely to permit someone to determine your true identity. IP addresses can be protected if you use a proxy, which is recommended, and is the default for many ISPs.

All use of this system is voluntary by you, and as such you may stop using the system at any time without negative consequences to yourself.

As required by the University of Michigan IRB, I must provide the following information:

My name is Eric Glover, and I am the Primary Investigator for this study. I am a Ph.D. student at the University of Michigan and an intern at the NEC Research Institute: If you need to contact me I can be reached at: (XXX) XXX-XXXX (my cell phone number) or by e-mail at: compuman@research.nj.nec.com, or compuman@eecs.umich.edu.

Should you have any questions about the consent form (in general), or research subject's rights you may contact the IRB directly. The contact for the IRB Administrator is Kate M. Keever: keever@umich.edu, and the phone number for the IRB is: (734) 936-0833

By clicking AGREE below, you agree to the terms discussed in this consent form: Specifically you agree that your use of the system may be logged, and if you are under 18 you have permission from a parent or guardian. If you do not agree, you should not click AGREE below, and should not use this system.

AGREE

---

inquirus2 @ ericglover.com
Last modified: Tue Aug 1 19:06:16 EDT 2000

Figure 5.7: User consent form to use Inquirus 2.

CHAPTER 6

Query Modification Learning Procedure (QMLP)

Typical web search engines index millions of pages across a variety of categories, and return results ranked by *expected topical relevance.* Only a small percentage of these pages may be of a specific category, for example, personal homepages, or research papers. A user may examine large numbers of pages about the right topic, but not of the desired category.

Our method uses a classifier to recognize web pages of a specific category and discovers modifications to queries that bias results toward documents in that category. The results of the learning procedure were used to generate several categories in Inquirus 2, and were studied in the user study described in Chapter 5.

Query Modification Learning Procedure automatically produces a set of query modifications and search engine pairs that have several desirable properties, and requires only an initial set of training documents and a small number of test queries. In this Chapter, we describe a methodology for learning search-engine, query-modification pairs to facilitate category-specific web search.

6.1 The Problem

Given a category, a user may guess a reasonable query modification to enhance the search. For example, a user searching for personal homepages about people who like to "yo-yo" may guess that adding "home page" to the query of "yo-yo" will enhance the results. Unfortunately, naive query modifications are not always useful, and in some cases may be detrimental. Likewise, simply guessing does not account for differences between search engines, further reducing the effectiveness. The problem is to automatically learn a set of effective query modifications, that when used, produce results consistent with the user's intentions.

Section 3.8.2 describes the properties of query modifications used in Inquirus 2: search engine specific, category specific, query independent and sometimes non-obvious. Any method involving

human judgment will be unlikely to discover non-obvious query modifications. In addition, there may be features that individually appear to work, but when combined do not, or vice versa.

The definition of a *category* may not always be easy to express in simple terms. Instead of expecting a textual description of a category to train, we begin with a set of web pages that are representative of the desired category. For example, a set of homepages of researchers, a set of research papers, or a set of documents about a particular company. The concept of category is not limited by subject, but may be based on some property, such as poetry pages, or short stories.

The problem is given the initial representative set, discover a set of reasonable search engine and query modification pairs. The learned search engine and query modification pairs should ultimately reflect the category component of a user's information need.[1]

6.2 The Method

For a specific category, our first step is to train a support vector machine (SVM) [Platt, 1998] to classify pages by membership in the desired category. Classifier accuracy is improved by considering, in addition to words and phrases, the documents' HTML structure and simple word location information (e.g., whether a word appears near the top of the document).

We then learn a set of query modifications. For this experiment, a *query modification* is a set of extra words or phrases added to a user query to increase the likelihood that results of the desired category are ranked near the top.[2] Since not all search engines respond the same way to modifications, we use our classifier to automatically evaluate the results from each search engine, and produce a ranking of search engine, query modification pairs. This approach compensates for differences between performance on the training set and the search engine, which has a larger database and unknown ordering policy.

6.3 SVMs and Web Page Classification

Categorizing web pages is a well researched problem. We choose to use an SVM classifier [Vapnik, 1995] because it is resistant to overfitting, can handle large dimensionality, and has been shown to be highly effective when compared to other methods for text classification [Joachims, 1999, Kwok, 1999]. A brief description of SVMs follows.

Consider a set of data points, $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}$, such that $\boldsymbol{x}_i$ is an input and y_i is a

[1] In addition to the ability to locate on-category results, it might be desirable to consider search engine performance or other factors when evaluating query modifications.

[2] Our system supports field based modifications, such as a constraint on the URL or anchortext.

target output. An SVM is calculated as a weighted sum of kernel function outputs. The kernel function of an SVM is written as $K(\boldsymbol{x}_a, \boldsymbol{x}_b)$ and it can be an inner product, Gaussian, polynomial, or any other function that obeys Mercer's condition.

In the case of classification, the output of an SVM is defined as:

$$f(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{i=1}^{N} y_i \lambda_i K(\boldsymbol{x}_i, \boldsymbol{x}) + \lambda_0. \tag{6.1}$$

The objective function (which should be minimized) is:

$$E(\boldsymbol{\lambda}) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{N} \lambda_i, \tag{6.2}$$

subject to the box constraint $0 \leq \lambda_i \leq C, \forall_i$ and the linear constraint $\sum_{i=1}^{N} y_i \lambda_i = 0$. C is a user-defined constant that represents a balance between the model complexity and the approximation error. Equation 6.2 will always have a single minimum with respect to the Lagrange multipliers, $\boldsymbol{\lambda}$. The minimum to Equation 6.2 can be found with any of a family of algorithms, all of which are based on constrained quadratic programming. We used a variation of Platt's Sequential Minimal Optimization algorithm [Platt, 1998, Platt, 1999] in all of our experiments.

When Equation 6.2 is minimal, Equation 6.1 will have a classification margin that is maximized for the training set. For the case of a linear kernel function ($K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$), an SVM finds a decision boundary that is balanced between the class boundaries of the two classes. In the nonlinear case, the margin of the classifier is maximized in the kernel function space, which results in a nonlinear classification boundary.

Some research has focused on using hyperlinks, in addition to text and HTML, as a means of clustering or classifying web pages [Chakrabarti et al., 1998a, Flake et al., 2000]. Our work assumes the need to determine the class of a page based solely on its raw contents, without access to the inbound link information.

6.4 QMLP Details

Figure 6.1 shows our main algorithm, QUERY MODIFICATION LEARNING PROCEDURE (QMLP). This algorithm first trains an SVM classifier on labeled data. The algorithm then automatically generates a set of good query modifications, ranked by expected recall. Finally, using the learned classifier to evaluate the query modifications on real search engines, a rank ordering of query-modification, search-engine tuples is produced. The classifier and the tuples are incorporated into Inquirus 2 to improve category-specific web search. In addition to learning the full-page classifier, a *summary classifier* is also learned for use by the selective download module of Inquirus 2.

QMLP($\mathcal{A}$, $\mathcal{B}$, SE, Q, n)

INPUT: Training examples $\mathcal{A}$ (positive) and $\mathcal{B}$ (negative)
Set of search engines SE, test queries Q
The number of results to consider n

OUTPUT: Ranked list of search engine,
query modification tuples $\{se, qm\}$

1. Generate set of features F from $\mathcal{A}$ and $\mathcal{B}$
2. Using F train SVM classifier
3. F^* is the top 100 features from F, by expected entropy loss
4. Select set of possible query modifications,
   $QM = \{F^*\} \cup \{F^* \times F^*\}$
5. Remove duplicate or redundant modifications
6. $QM' =$ PRE-PROCESS-QMOD$(QM, \mathcal{A}, \mathcal{B})$
7. The set of *tested* modifications $QM'' \subseteq QM'$
8. **return** SCORE-TUPLES (QM'', SE, Q, n)

Figure 6.1: Query Modification Learning Procedure.

6.5 Training the Classifier

First we train a binary classifier to accurately recognize positive examples of a category with a low false-positive rate. To train the classifier, it is necessary to convert training documents into binary feature vectors, which requires choosing a set of reasonable features. Even though an SVM classifier may be able to handle thousands of features, adding features of low value could reduce the generalizability of the classifier. Thus, dimensionality reduction is performed on the initial feature set.

Unlike typical text classifiers, we consider words, phrases and underlying HTML structure, as well as limited text location information. A document that says "home page" in bold is different from one that mentions it in anchor text, or in the last sentence of the document. We also added special features to capture non-textual concepts, such as a URL corresponding to a personal directory. Figure 6.2 describes the representation of document features. Appendix B describes the symbol features in more detail.

Initial Dimensionality Reduction

We perform a two step process to reduce the number of features to a user specified level.

First, we remove all features that do not occur in a specified percentage of documents as rare

Code	Description
T	Title word or phrase
TS	Occurs in first 75 terms of the document
F	Occurs anywhere in full-text (except title)
E	Occurs in a heading, or is emphasized
UP	Word or special character (tilde) occurs in the URL path
UF	Word or special character occurs in the file name portion of the URL
A	Occurs in the anchortext
S	Special symbol – Captures non-textual concepts, such as personal directory, top of tree, name in title

Figure 6.2: Document vector types used

words and very frequent words are less likely to be useful for a classifier. A feature f is removed if it occurs in less than the required percentage (threshold) of both the positive and negative sets, i.e.,

$$\left(|\mathcal{A}_f|/|\mathcal{A}| < \mathcal{T}^+\right) \text{ and } \left(|\mathcal{B}_f|/|\mathcal{B}| < \mathcal{T}^-\right)$$

Where:

- $\mathcal{A}$: the set of positive examples.
- $\mathcal{B}$: the set of negative examples.
- $\mathcal{A}_f$: documents in $\mathcal{A}$ that contain feature f.
- $\mathcal{B}_f$: documents in $\mathcal{B}$ that contain feature f.
- $\mathcal{T}^+$: threshold for positive features.
- $\mathcal{T}^-$: threshold for negative features.

Second, we rank the remaining features based on entropy loss. No stop word lists are used.

6.5.1 Expected Entropy Loss

Entropy is computed independently for each feature. Let C be the event indicating whether the document is a member of the specified category (e.g., whether the document is a personal homepage). Let f denote the event that the document contains the specified feature (e.g., contains "my" in the title). The prior entropy of the class distribution is $e \equiv -\Pr(C)\lg\Pr(C) - \Pr(\overline{C})\lg\Pr(\overline{C})$. The posterior entropy of the class when the feature is present is $e_f \equiv -\Pr(C|f)\lg\Pr(C|f) -$

$\Pr(\overline{C}|f)\lg\Pr(\overline{C}|f)$; likewise, the posterior entropy of the class when the feature is absent is $e_{\overline{f}} \equiv -\Pr(C|\overline{f})\lg\Pr(C|\overline{f}) - \Pr(\overline{C}|\overline{f})\lg\Pr(\overline{C}|\overline{f})$. Thus, the expected posterior entropy is $e_f Pr(f) + e_{\overline{f}} Pr(\overline{f})$, and the *expected entropy loss* is

$$e - \left(e_f \Pr(f) + e_{\overline{f}} \Pr(\overline{f})\right).$$

If any of the probabilities are zero, we use a fixed value. Expected entropy loss is synonymous with expected information gain, and is *always* nonnegative [Abramson, 1963].

All features meeting the threshold are sorted by expected entropy loss to provide an approximation of the usefulness of the individual feature. This approach assigns low scores to features that, although common in both sets, are unlikely to be useful for a binary classifier.

One of the limitations of using this approach is the inability to consider co-occurence of features. Two or more features individually may not be useful, but when combined may become highly effective. Coetzee discuss an optimal method for feature selection in [Coetzee et al., 2001]. Our method, although not optimal, can be run in constant time per feature with constant memory per feature, plus a final sort,[3] both significantly less than the optimal method described by Coetzee. We perform several things to reduce the effects of possible feature co-occurence. First, we consider both words and phrases (up to three terms). Considering phrases reduces the chance that a pair of features will be missed. For example, the word "home" and the word "page" individually may be poor at classifying a personal homepage, but "home" followed by "page" might be very useful.

A second approach to reducing the problem is to consider many features, with a relatively low threshold for the first step. The SVM classifier will be able to identify features as important, even if individually they might not be. As a result, considering a larger number of features can reduce the chance that a feature is incorrectly missed due to low individual entropy. For our experiments, we typically considered about a thousand features for each classifier, easily handled by an SVM.

6.6 Choosing Query Modifications

Like the work of Mitra [Mitra et al., 1998], the goal of our query modification is to identify features that could enhance the precision of a query. Unlike their work, we have extra information regarding the user's intent in the form of labelled data. The labelled data defines a category, and the learned modifications can be re-applied for different topical queries that fall in the same category without any re-learning (query independence).

[3] We assume that the histogram required for computation is generated separately, and we assume a constant time to look up data for each feature from the histogram.

Once the training set has been converted to binary feature vectors[4], we generate a set of query modifications. Selected features may be non-textual, or on fields not usable by every search engine, such as anchortext, or the URL. Features or combinations of features are selected based on their predicted precision and recall, independent of whether a human judges them as "relevant". We consider precision as the percentage of the documents that satisfy a given query modification that are "on-category". Recall is the percentage of "on-category" documents that satisfy the given query modifications. Such features or combinations of features may be non-obvious.

To generate the ranked list of possible query modifications, we score all possible query modifications by expected *recall*. We define $\mathcal{QM}$ to be the set of query modifications, or all combinations of one or two features. A user parameter, **P**, is the desired minimum precision. To compute the precision, we must consider the *a priori* probability that a random result from a search engine is in the desired category, as opposed to the probability that a random document from the training set is in the positive set. To compensate for the difference between the *a priori* probability and the distribution in the training set, we add a parameter α, defined below. Figure 6.3 shows our algorithm for ranking the query modifications.

Consider the following definitions:

- $\mathcal{QM}$: Set of all possible query modifications for consideration,
- qm : A single query modification, comprised of a set of one or more features: $qm = \{f_1, f_2, ...f_n\}$,
- $\mathcal{A}_{qm}$: the set of documents from $\mathcal{A}$, the positive set, that contain all the features in qm,
- $\mathcal{B}_{qm}$: the set of documents from $\mathcal{B}$, the negative set, that contain all the features in qm,
- α : factor to compensate for *a priori* probability of the class: $P(A) = \frac{|\mathcal{A}|}{|\mathcal{A}|+\alpha|\mathcal{B}|}$,
- **P**: User provided parameter for minimum desired precision.

The algorithm PRE-PROCESS-QMOD returns a ranked list (by expected recall) for each query modification that meets the specified precision level, **P**. Precision in this context is based on percentage of results classified as positive; there is no query, only a category. We can measure precision on the web, but cannot measure recall without having knowledge of all possible results and their classification. Typical query modification approaches strive to maximize precision; however, a query modification may overly constrain the results causing very high precision, but very low recall. As a result, we feel that ranking query modifications by expected recall is more desirable, as long as they have at least some minimum precision requirement. If a user is searching for a specific page in some category, say an individual's homepage, as opposed to a set of homepages of people

[4]We do not consider term frequency information for this step, only presence or absence of a given feature in a document.

PRE-PROCESS-QMOD($\mathcal{QM}$, $\mathcal{A}$, $\mathcal{B}$)

INPUT: $\mathcal{QM}$, $\mathcal{A}$, $\mathcal{B}$

OUTPUT: A ranked list of all $qm \in \mathcal{QM}$

1. **foreach** $qm \in \mathcal{QM}$
2. Compute predicted precision:
   $\mathcal{P}'(qm) = \frac{|\mathcal{A}_{qm}|}{|\mathcal{A}_{qm}| + \alpha|\mathcal{B}_{qm}|}$
3. **if** $\mathcal{P}'(qm) < \mathbf{P}$
4. $Score(qm) = 0$
5. **else**
6. $Score(qm) = \frac{|\mathcal{A}_{qm}|}{|\mathcal{A}|}$
7. **end foreach**
8. **return** all $qm \in \mathcal{QM}$ sorted by $Score(qm)$

Figure 6.3: Initial ranking of the query modifications.

who have a particular interest, low recall can make it very likely the desired page is never found [Grossman and Frieder, 1998, Van Rijsbergen, 1979]. In general, there is an inverse relation between precision and recall. Our approach allows the user of the training code to control this balance by choosing the minimum precision level.

6.7 Scoring Query Modifications and Engines

Not all search engines respond the same way to a query modification, or contain the same distribution of documents. One user may be looking for homepages of persons who work for a company, while another might be looking for homepages of people who own a Ford truck; in each case, the *best* query modification and search engine may be different. To rank the search-engine, query-modification tuples, we use representative test queries and apply the classifier to the results. Figure 6.4 shows an algorithm that can rank a set of search engines and query modifications starting with a set of sample queries. The ranking is based on the number of valid documents returned that are classified as true by the learned classifier.[5] In addition, a specified parameter n controls how many results are considered for each search-engine query. Considering too many results may harm performance, while too few may not accurately capture the effectiveness of a given tuple. n should be similar to the web search horizon used by the metasearch engine that will use the learned query

[5]The learned classifier considers many features, in addition to words and phrases, reducing the chance that a single feature causes a page to always classify as true. In addition, SVMs are designed not to overfit, further reducing the chance of a single or small set of features dominating.

modifications.

Thus, we define for training and testing:

- QM'' : Set of query modifications for testing, $QM'' \subset \mathcal{QM}$,
- p : A single web page $< url, html >$,
- se : A search engine, $se \in SE$, all search engines,
- q : A query to be submitted to a search engine, Q is the set of test queries,
- $p = DOWNLOAD(u)$: Downloads page p corresponding to URL u, if there is an error then p is defined as nil,
- $CLASSIFY(p, u)$: Function that returns true if page p at URL u is of the desired category.

Although the algorithm could compute scores for hundreds of test queries and tens of thousands of possible query modifications, we caution against running it on a large set, to avoid sending large numbers of queries to the search engines. To further reduce the burden on each search engine, we ran the algorithm serially, alternating search engines between each request. For our experiments, we choose QM'' to be a relatively small subset of the set of all ranked query modifications, plus the query with no modification, and at least one "naive" modification for comparison. For the training of the categories used in the user study, we added the ability to record the number of returned results for each search engine query. Although it is not possible to compute recall, the number of returned results may permit choosing of a more balanced query modification. A query modification with 100% precision, but only returns three total results, is likely too restrictive, and is probably worse than a query modification that results in 90% precision, but returns 10,000 results.

6.8 Results

To test QMLP, several categories were learned and used in Inquirus 2. Earlier work describes some results for using early version of the category of personal homepages and the category of calls for papers, as well as experiments with a gaussian kernel function [Glover et al., 2001]. This dissertation focuses on the three categories used in the user study: Personal homepages, product reviews and research papers.

For each category, we started with a set of training and test documents, a set of sample test queries, and four search engines: AltaVista, AllTheWeb, Northern Light, and Google. In each case a linear kernel function was used, and n was always set to 20, to match the web search horizon used for the user study. For each category, a full-page and summary classifier was learned to identify documents in the given category. In addition, QMLP was used to recommend a set of query

SCORE-TUPLES(QM, SE, Q, n)

INPUT: List of search engines and query modifications to test, test queries, Q, a parameter n

OUTPUT: A ranked list of all $< qm, se >$

1. **foreach** $qm \in QM$
2. **foreach** $se \in SE$
3. $Score_{qm,se}$ = EVAL-TUPLE(qm, se, Q, n)
4. **end foreach**
5. **end foreach**
6. **return** all $< qm, se >$ in QM, SE, sorted by $Score_{qm,se}$

EVAL-TUPLE(qm, se, Q, n)

INPUT: Query modification qm, search engine se, test queries Q, n

OUTPUT: Relative score for the tuple $< qm, se >$

1. INITALIZE $score = 0$
2. **foreach** $q \in Q$
3. $R =$ first n result URLs from search engine se with query CONCAT(q, qm)
4. **foreach** url $u \in R$
5. $p = DOWNLOAD(u)$
7. **if** $p \neq nil$ AND $CLASSIFY(p, u) = TRUE$
8. $score = score + 1$
9. **end if**
10. **end foreach**
11. **end foreach**
12. **return** $score$

Figure 6.4: Functions to score each $< qm, se >$ pair for a given classifier and test queries.

modifications for each search engine. For consistency, we choose the *best* query modification from each search engine for our user study, even though one search engine may have several query modifications appearing to perform better than the best query modification from another search engine. In addition to choosing the best query modification for each search engine, we also choose a second query modification for one search engine (to allow us to measure overlap between multiple query modifications for the same search engine) and a "bad" query modification for later analysis of effectiveness.

For each category, a set of positive pages were found using various methods, including web searching, recommendations, and hub pages. Negative pages were formed from a combination of random pages (found from a pseudo-random pick of a 200 million page web crawl) and negative pages that were "near-misses" found while searching for positive pages.

6.8.1 Personal Homepages

A personal homepage is a difficult concept to define objectively. The definition we used is a page made by an individual (or family) in an individual role, with the intent of being the entry point for information about the person (or family). It is possible a person may have more than one personal homepage, and it is also acceptable for a person to dedicate her homepage to an interest and or hobby, but not to corporate endeavors. Pages that were manual redirects or entry pages (pages that had only an image and a small amount of text) were removed from the training and test sets. We also considered only pages in English, although the symbol and structural features are language independent.

For training we used 598 positive examples, and 1200 negative examples. The initial training was performed using thresholds of 7.5% for both the positive and negative, and the top 600 features by expected entropy were kept. To make the classifier more general, we restricted the training set to a maximum of 5% of any of the documents from any one domain.

Table 6.1 lists the top ten features as scored by expected entropy loss, and Table 6.2 lists the top 10 features after scoring by the SVM (linear kernel function). Many features rank high for both, but some features, such as "my" in the top 75 terms scored high for expected entropy loss, but was not useful for the SVM classifier; probably because other features co-occur with "my". The top ranked feature was "UF./" or the URL ending in a slash, consistent with the URLs of most personal homepages. The high scoring feature "S.dpersonal" states that the URL represents a standard "personal directory"; such as the URL being `http://people.domain.net/user/file` or `http://domain/people/user/file` or `http://domain/~user/file`.

For testing of the personal homepage classifier, we used 335 positive and 994 negative test

1	UF./
2	TS.my
3	S.dpersonal
4	F.my
5	T.s
6	T.page
7	T.home page
8	T.home
9	T.s home
10	TS.i

Table 6.1: The top 10 features for personal homepages as scored by expected entropy loss

#	**Feature**	**Weight**
1	S.dpersonal	1.03
2	UP.geocities	0.98
3	UF./	0.8
4	T.home page	0.8
5	UF.index	0.78
6	T.page	0.65
7	TS.work	0.55
8	TS.this page	0.55
9	F.complete	-0.53
10	UP.user	0.48

Table 6.2: The top 10 features for the personal homepage classifier

documents. The accuracy of the full-page classifier on the positive class was 94.3% and 97.3% for the negative class. The relatively high accuracy compares favorably with the restricted personal homepage classifier of Blum [Blum and Mitchell, 1998].

In addition to the full-page classifier, we trained a summary classifier to be used with the selective download module. To train a summary classifier, we considered only the title, URL, and the first 25 words of the document. We did not utilize actual search engine summaries for training our summary classifier. The summary classifier defines a new feature type "TN" standing for top n terms, in this case 25. The summary classifier was created using both a positive and negative threshold of 5%, and all 150 features meeting the threshold were kept.

The summary classifier was remarkably accurate at identifying a document as a personal homepage. The positive accuracy was 88.7%, slightly worse than that for the full page classifier. The negative accuracy was 97.5%, effectively the same as the negative for the full-page classifier. The

#	Feature	Weight
1	UF./	3.86
2	TN.this page	3.71
3	UP.people	3.63
4	TN.engineering	3.47
5	UP.geocities	3.44
6	TN.i m	3.01
7	TN.i am	2.95
8	TN.to my	-2.74
9	TN.my	2.65
10	TN.welcome	2.45
11	TN.welcome to my	2.26
12	UF.index	2.05
13	UP.home	2.04
14	TN.search	-1.99
15	TN.home page	1.97
16	TN.phone fax	1.92
17	TN.research	1.88
18	S.dpersonal	1.78
19	S.toptree	-1.66
20	TN.at	-1.59

Table 6.3: The top 20 features for the personal homepage summary classifier

relatively high positive accuracy for the summary classifier was probably because most features identifying a personal homepage appear to occur in the title and URL.

Table 6.3 lists the top 20 scoring features for the summary classifier. None of the top twenty features based on SVM weight are in the title. About one third of the top twenty features are in the URL or are special symbol features.

QMLP also generated a set of query modifications from the training data. Table 6.4 lists the top 15 query modifications sorted by predicted recall with at least 50% predicted precision.

Several of the top predicted query modifications, plus naive query modifications and no query modification were used as the input to $SCORE - TUPLES$. Evaluation of each query modification involves numerous requests to each search engine (one for each test query) so it is impractical to test all of them. Table 6.5 shows several of the best predicted query modifications and the precision of unmodified queries for each search engine. The query modifications were scored by balancing the number of returned results with their precision. For these query modifications seven test queries were used: “information retrieval”, “palm pilot”, jones, smith, “ballroom dancing”, “electrical en-

#	Query modification	Predicted precision	Predicted recall
1	F.my + T.page	54	30
2	TS.my + T.s	58	28
3	F.my + T.home	63	25
4	T.s + T.home	62	24
5	F.my + T.home page	70	23
6	TS.my + T.page	70	23
7	T.s home	82	23
8	T.page + TS.i	60	22
9	T.s + T.home page	60	22
10	T.s home page	81	21
11	T.home page + T.s home	81	21
12	T.page + T.s home	81	21
13	T.home page + F.i	59	21
14	TS.my + T.home	80	20
15	T.s home + F.of	80	20

Table 6.4: The top 15 query modifications for personal homepages based on predicted recall

gineering", and "sony playstation".

The effectivness of query modifications as applied to actual user queries is discussed in Section 5.2.1. The results of QMLP for personal homepages (Table 6.5) demonstrates the importance of choosing appropriate search engines for each query modification. There was a wide variation between the measured precision and approximate recall for each query modification for each search engine. In general the search engine AllTheWeb performed best (the best ranked query modification with reasonable recall and a precision of 86%), with NorthernLight performing worst (the best query modification had less than 60% precision, and lower recall) for this category.

6.8.2 Product reviews

The second category used in our user study was "Product reviews". Originally intended to be pages that provide a review of a hardware or software product, this category was expanded to consider non-computer products, such as lawnmowers, cars, etc. In addition, a document that was a music or movie review was also considered positive. A typical review would mention a product and provide some basic description of its features as well as an opinion on it. Most reviews would provide links to where to buy the product, or pricing information, although for movies this was not applicable.

Search engine	Query modification	Predicted precision	Reported results
FA	+”s home” +page	86	288734
FA	+”s home page”	87	179425
FA	+i +”s home”	82	257417
FA	+title:”s home page”	90	11352
FA	+”welcome to my” +page	83	24324
AV	+title:”s home page”	82	11913
AV	+”s home” +page	69	41279
GO	welcome ”to my”	61	272510
AV	+my +title:home	68	15016
NO	+”about me”	59	86644
NO	+”research interests”	59	72122
GO	”s home” page	57	39908
FA	+”i am” +publications	54	117247
GO	”s home page”	59	21105
FA	no modification	14	14748463
GO	no modification	11	21534800
AV	no modification	11	3338061
NO	no modification	8	10473703

Table 6.5: Several of the top ranked query modifications for the category personal homepages

For training we used 818 negative and 199 positive documents. The training and test sets were found in the same manner as the training and test sets used for the category of personal homepages. For training, we used positive and negative thresholds of 7.5% and kept the top 1000 features based on expected entropy loss. Table 6.6 lists the top 15 features of the product review classifier. As expected, the word “reviews” is very significant. The system discovered that “reviews” in the URL path, the title and the full text were all significant. The phrase “product reviews” occuring in the full text was also important.

To test the full-page classifier, we used a test set of 73 positive and 800 negative documents. The positive set demonstrated roughly 85% accuracy, and the negative set roughly 98% accuracy. All of the false negatives (positives classified incorrectly as negative) scored greater than -1.5, with most scoring greater than -1. The majority of the negative documents scored less than -1 suggesting that a shift in the bias term will reduce the false negative rate at the expense of the false-positive rate. Adding product-review specific features or adding more documents to the positive training set may improve the overall accuracy.

In addition to the full-page classifier, we trained a summary classifier for use by the selective

#	Feature	Weight
1	F.reviews	0.15
2	UP.com	0.14
3	F.which	0.14
4	F.first	0.12
5	F.were	0.12
6	E.reviews	0.11
7	F.but	0.11
8	UP.reviews	0.11
9	T.reviews	0.11
10	F.test	0.1
11	F.overall	0.09
12	F.product reviews	0.09
13	F.tools	-0.09
14	F.digital	0.09
15	F.quite	0.09

Table 6.6: The top 15 features for the product reviews classifier

download module. Table 6.7 lists the top 15 features for the summary classifier. The accuracy of the summary classifier was about 95% for the negative test set, and about 92% for the positive test set. The summary classifier performed slightly worse than the full classifier for the negative set. For the positive set, the relatively small size does not guarantee the summary classifier is more accurate than the full-page classifier, even though it appears significantly more accurate (92% vs 85%).

A positive threshold of 0.9 would eliminate about half of the false positives, while preserving the majority of the positive documents. A negative threshold of -0.9 would eliminate all but three of the false negatives, while preserving more than 80% of the negative documents. Assuming a perfect full-page classifier, such thresholds would produce a significant accuracy improvement over the summary classifier, while permitting the far majority of the documents to be classified using the summary classifier.

QMLP was run to produce a set of ranked query modifications. Table 6.8 lists several of the top ranked query modifications. To generate these modifications seven test queries were used: monitors, “pentium iii”, “sony viao”, smith, “web tv”, “windows 2000”, “diablo ii”. The top ranked query modifications for both Google and AllTheWeb contain the word reviews and the letter T. This may be a result of some special code embedded in the HTML not parsed out and mistaken for text. It should be noted that the results do demonstrate a noticable improvement for query modifications with the added “T”. This is an example of a non-obvious query modification.

#	Feature	Weight
1	UF.rev	1.79
2	TN.downloads	1.62
3	T.product	1.56
4	TN.vision	-1.53
5	T.reviews	1.47
6	UP.co	1.38
7	TN.auctions	-1.32
8	TN.design	1.28
9	TN.p	-1.24
10	UP.com	1.19
11	TN.network	1.17
12	TN.how to	-1.09
13	UP.uk	1.05
14	TN.pc magazine	-1.05
15	TN.reviews	0.99

Table 6.7: The top 15 features for the product reviews summary classifier

6.8.3 Research papers

The third category used in our user study was “Research papers”. A web page that was considered positive was a research paper, or a very detailed document. A web page with several links to postscript files was not considered a research paper, however a short abstract that links to a single pdf or postscript file was considered a research paper. Research papers did not need to be formal, but needed to be detailed. A white paper, or even a detailed essay would be considered positive.

For training, we used 885 negative and 250 positive documents. To produce a more general classifier we limited the positive set to five percent of the documents about any one topic or from any one domain. Table 6.9 lists the top 20 features and their weight as learned by the SVM.

The test set involved 994 negative examples and 35 positive examples. The accuracy for the negative set was 98.5%, and the negative set was about 86%. Due to time constraints there were slightly under 300 positive documents. To improve overall accuracy of the classifier, we decided to increase the number of documents in the training set over the test set.

In addition to training the full-page classifier for research papers, we also trained a summary classifier. Table 6.10 lists the top ten features for the research paper summary classifier. Unfortunately, due to an error when running the QMLP scripts, textual summary features (type of TN) were excluded from the summary classifier. Despite the lack of summary text features, the summary classifier demonstrated 97.7% accuracy for the negative set, and 37.1% accuracy for the positive

Search engine	Query modification	Predicted precision	Reported results
GO	reviews so	72	769841
GO	reviews t	70	919108
FA	+reviews +t	69	496805
GO	feature nice	71	189200
GO	graphics nice	71	160252
FA	+reviews +quality	69	233105
GO	”you can” better	63	1216690
FA	+features +review	65	376456
FA	+”it s” +review	62	410647
GO	more reviews	59	1028508
NO	+graphics +nice	62	172851
AV	+feature +nice	66	58727
GO	review	54	1608491
FA	+review	52	424755
AV	+review	39	108782
GO	no modification	14	15836460
NO	no modification	5	11193733
AV	no modification	5	5960345
FA	no modification	4	1942710

Table 6.8: Several of the top ranked query modifications for the category product reviews

set. The low accuracy for the summary classifier for the positive set is likely due to the fact that a title and a short summary cannot easily capture the format and detail level of a research paper; there are few terms that are common to a significant percentage of research papers that would occur in the title or summary text. In addition, the relatively small positive training and test sets reduce the accuracy and ability to measure the accuracy of the classifier.

If a positive threshold of .9 was used, the false positives are cut in half, but at a reduction of the number of positive documents downloaded. However, using any positive threshold will permit the false negative documents to be downloaded and classified with the significantly more accurate full-page classifier. Despite the relatively low accuracy of the positive classifier, combined with the full-page classifier would result in very high accuracy for both positive and negative documents, and could result in a significant reduction in the number of positive documents downloaded.

To produce a set of query modification and search engine pairs, we ran QMLP on six test queries: ”metasearch engine”, maes, soloway, ”art history”, ”cellular mitosis”, and antenna. Table 6.11 lists several of the best query modifications and the scores for no modification for each of the four search

#	Feature	Weight
1	A.pdf	0.34
2	F.next	0.21
3	F.this	0.19
4	F.introduction	0.16
5	F.at the	-0.16
6	TS.university	0.16
7	F.under	-0.15
8	F.terms	0.15
9	F.is a	-0.14
10	F.part	-0.14
11	TS.abstract	0.13
12	F.is also	0.13
13	E.abstract	0.13
14	F.as a	0.12
15	TS.introduction	0.12
16	F.abstract	0.12
17	F.measure	0.12
18	F.should be	-0.12
19	E.in	-0.12
20	F.pp	0.12

Table 6.9: The top 20 features for the research paper classifier

engines. Table 6.12 lists the predicted precision and recall based on the training data. The original predictions can be compared with the final measured data produced by QMLP.

6.9 QMLP Summary

The algorithm QMLP has been demonstrated in several categories to produce highly effective category-specific, query-independent, search-engine-specific, and sometimes non-obvious query modifications. The algorithm can run using reasonable time and memory resources for training sets of thousands of pages, and can discover query modifications of words or up to three word phrases. These query modifications can be specific to features, such as the title or URL. Section 5.2.1 of our user study demonstrates a significant improvement in the ability to locate on-category documents for user queries. Unfortunately, results that are on category are not always useful. The user study also demonstrated that there was a loss of average topical relevance for modified queries. The QMLP algorithm does not consider topical relevance for returned results. This introduces the

#	Feature	Weight
1	T.abstract	1.02
2	UP.publications	1.01
3	UP.papers	0.97
4	T.model	0.7
5	T.an	0.57
6	T.in	0.49
7	UP.net	-0.49
8	T.a	0.39
9	S.toptree	-0.38
10	T.information	0.05

Table 6.10: The top 10 features for the research paper summary classifier

possibility that a search engine could return on-category documents not about the original query, but still considered effective.

Search engine	Query modification	Predicted precision	Reported results
FA	+introduction +”shown in figure”	80	2763
NO	+”this paper” +”likely to”	63	3569
NO	+introduction +”shown in figure”	64	3166
GO	abstract introduction	54	17704
GO	introduction ”shown in figure”	76	2914
FA	+figure +”in this paper”	66	2290
GO	”can be” ”shown in figure”	72	4856
AV	+”this paper” +”likely to”	68	1486
FA	no modification	6	420477
GO	no modification	5	487320
AV	no modification	2	298864
NO	no modification	1	1053584

Table 6.11: Several of the top ranked query modifications for the category research papers

Query modification	Predicted precision	Predicted recall
F.introduction + F.shown in figure	58	14
F.shown in figure + TS.introduction	100	5
F.this paper + F.likely to	61	16
F.likely to + TS.this paper	100	5
F.abstract + F.introduction	11	64
TS.abstract + F.introduction	22	46
TS.abstract + TS.introduction	15	14

Table 6.12: The predicted precision and recall for some of the top ranked query modifications for the category of research papers

CHAPTER 7

Conclusion, Summary and Future Work

Current general-purpose search engines are one-size-fits all; they do not consider the individual needs of a searching user. As a result, a user may be forced to manually submit her query to multiple search engines, and or manually examine dozens of useless results to find a small number of useful results. This dissertation explored several of the problems that make finding useful results difficult for web search or metasearch engines. We then presented Inquirus 2, a preference-based metasearch engine, that utilizes several architectural improvements to help reduce some of the problems that limit the ability to find useful documents.

This dissertation presented four contributions: First, we described several results from our user study related to user judgments of usefulness and user judgments of topical relevance. Second, we presented Query Modification Learning Procedure (QMLP), an automated algorithm that learns source and category-specific query modifications. Third, we presented several techniques that were implemented and tested that incorporate category into the metasearch engine process. Fourth, we presented selective download as a performance enhancing component for Inquirus 2.

To improve our understanding of usefulness, we studied the relationships between user judgments of topical relevance, document category, and user judgments of usefulness. We discovered a bounding relationship where the level of topical relevance provided an upper bound on the level of the usefulness judgment. We also observed that both category and source and category-specific query modifications improve the average usefulness judgments of documents judged as highly topically relevant.

QMLP testing returned results that were on-category for no query modification less than 10% of the time for the test queries, with some tests returning on-category results for no modification as little as 2% of the time. In addition to significantly improving the percentage of on-category results, modified queries increased the average usefulness of results judged as highly topically relevant. The average usefulness for modified queries judged as 5 out of 5 for topical relevance was 4.6, while

unmodified queries judged as 5 out of 5 for topical relevance had an average usefulness of only 4.16. However query modifications lowered the average topical relevance resulting in fewer useful results on average.

Our user study confirmed the effectiveness the several of the architectural improvements. Our chosen query modifications were shown effective with 75% of the results from modified queries classified as on-category, and highly topically relevant documents found from modified queries demonstrated a significant improvement in average usefulness.

Selective download was demonstrated to improve performance by not downloading about 16% of the results, and demonstrated an ability to quickly locate useful documents. Documents not downloaded were scored and presented in significantly less time per result and with a higher probability of being judged as useful. 29% of the results judged as useful (judged as 5 out of 5 for usefulness) were found without download, even though significantly less than 29% of the total results were not downloaded. The selective download works well when combined with the incremental interface to permit users to quickly find some of the useful results. Over 63% of the results found without download were scored within the first 5 seconds of the search, compared with about 11% of the downloaded results. The average usefulness for results downloaded was 2.75, while the average usefulness for results that were not downloaded was only 2.24.

Although our particular scoring functions were not shown to be very accurate at predicting usefulness overall, the concept of need-based scoring was demonstrated effective. Considering the category when scoring documents, demonstrated a significant improvement in average usefulness of highly topically relevant documents (4.52 vs 3.79), suggesting that considering the user's category when scoring documents is beneficial.

We did not study source selection or the incremental interface in detail.

Our user study also analyzed several parameters of interest related to metasearch in general. First, we confirmed a very low result overlap, both between all search engines, and between modified and unmodified queries. The low overlap is consistent with the results of several other papers. Second, we demonstrated an imbalance between the useful document contribution of different search engines. Although no search engine contributed more than half of the total useful documents, Google produced significantly more (nearly half of all useful documents found) useful documents then all of the other sources. This data suggests that despite a strong imbalance, metasearching still provides more total useful documents than any one search engine (given only the top 20 results are considered in all cases). Our study was designed to choose which documents to present to users, and asked for an explicit usefulness judgment. Our method is likely more accurate then a method using implicit feedback, such as user clicks or time spent on a page. In addition, since our system

choose the results, each source can have equal representation, and users are less likely to make a mistake than by judging based only on a title or summary.

7.1 Future Work

This dissertation explores several architectural improvements to metasearch engines that can improve the ability to locate and identify useful documents. We identify several areas for future work:

- Improve the accuracy of the topical relevance functions by incorporating other factors such as TFIDF from a proxy collection, as described by Craswell [Craswell et al., 1999].
- Improve the user interface and allow users to train their own categories, using very small training sets.
- Improve QMLP to consider topical relevance, to provide a more effective query modification, balancing both topical relevance and on-category precision.
- Add more categories to Inquirus 2 and better understand the differences between categories on usefulness judgments and effectiveness of the architectural components.
- Improve the scoring functions by both considering the results from the user study, and learning the weights of various attributes.

Our user study demonstrated the effectiveness of several architectural improvements for web metasearch engines. However, it also demonstrated the inadequacy our topical relevance function. Future work should focus on improving topical relevance predictions, both by incorporating extra information, such as proxy statistics to allow computation of TFIDF, and to learn appropriate category-specific topical relevance functions.

QMLP, although shown effective at locating on-category documents, lowered average topical relevance of the results. Future work could be to improve QMLP such that it considers topical relevance when ranking search engine, query modification pairs. One way to do this is to score each document using a combination of a topical relevance function (ideally one that is category-specific) and the category classifier. Other future work can include running QMLP on more categories and a new study that measures consistency of the effectiveness of QMLP on a larger set of categories.

The bounding relation, combined with the data on the effect of category on user usefulness judgments, suggests that the importance of category decreases with lower levels of topical relevance. As

a result, improved scoring functions should be created that adjust the weights as appropriate, possibly using a form other than additive. There are probably many factors other than topical relevance and category that should be considered when scoring documents. Although Inquirus 2 supports the use of arbitrary attributes as part of the scoring function, we did not study the significance of any other attributes.

APPENDICES

APPENDIX A

Categories, Scoring Functions, and Associated Query Modifications

The user study of Inquirus 2 allowed users to choose from four categories: research papers, reviews, personal homepages, and "none of the above (topical relevance)". In addition, Inquirus 2 allowed searching users to choose from several other categories. Each category specified a list of search engines and query modifications, and an associated scoring function. Table A.1 lists the specific scoring functions used by each of the four categories used for the user study. Table A.4 presents a list of several of the categories available in Inquirus 2, their description, the associated search engines and query modification pairs, and the attributes used by each. Table A.2 defines the search engine codes. Table A.3 provides a description of the attributes used in Table A.4.

Category	**Scoring function**
none of the above (topical relevance)	(1,quicktr, '(0,0) (75,75), (100,100)')
research papers	(.5,new_researchpapers, '(-1,0) (.5,100)') (.5,quicktr, '(0,0) (75,75) (93,100)')
reviews	(.5,prod_rev, '(-1,0) (.75,100)') (.5,quicktr, '(0,0) (75,75) (97,100)')
personal homepages	(.5,new_personal_hp, '(0,0) (.5,100)') (.5,quicktr, '(0,0) (75,75) (97,100)')

Table A.1: Scoring functions for the four categories used in the user study.

Table A.1 lists the scoring functions used by each of the four categories in our user study. For each scoring function, there is a list of tuples ($weight$, $attriubute$, $piecewise\ linear\ function$). The attribute $quicktr$ is a built-in attribute representing the topical relevance of the document. The selective download module decides if $quicktr$ is determined based on the downloaded result or the summary. The attributes $new\_researchpapers$, $prod\_rev$, and $new\_personal\_hp$ refer to the output of the classifier as applied to each page. If the document is downloaded, then the full-page classifier is used, if not, the summary classifier is applied. The selective download module is discussed in detail in Section 4.3.3. For results found from the special-purpose search engine CiteSeer, $new\_researchpapers$ is set to .49. Results from CiteSeer are assumed to always be

research papers, and a fixed value of 0.49, when applied to the associated scoring function, will produce a nearly perfect score for $new\_researchpapers$.

Search engine code	Search engine	Search engine code	Search engine
AB	ABCNews	AV	AltaVista
CS	CiteSeer	EE	EE Times
FA	AllTheWeb	GO	Google
HB	HotBot (now a part of Lycos)	NE	News.com (part of CNET)
NO	Northern Light	SN	Snap (now part of NBCI)
YH	Yahoo		

Table A.2: Search engine codes used by Inquirus 2

Attribute	Description
agrade	The average grade-level, determined by processing the number of syllables in each word and average length of a sentence
avworddist	Average distance from the top of the document to the first occurence of each query term
genscore	A handmade function that adds points based on the number of "general like" features
GFOG	A computation of the Gunning FOG score, a predictor of the reading level of "easier" documents
homepage	A handmade function that adds points based on the number of "homepage like" features
latex	A binary attribute where TRUE means the document was created using the tool LaTeX2HTML
pathlength	The number of slashes after the domain name in the URL
researchpaper	A handmade function that adds points based on the number of "research paper like" features present
topicalrelevance	A prediction of the topical relevance of a downloaded document
wordcount	The number of words in the rendered text of the document
wordproximity	A measure of the relative proximity of query terms
wordsinfirst30percent	The percentage of query terms occurring in the first 30 rendered word of the document
wordsinrefspercent	The percentage of query terms occurring in a "references" section of a page
wordspersection	The average of the number of words for each HTML "section"

Table A.3: Built-in attributes used by Inquirus 2

Inquirus 2 supports many search engines. The modular design allows for easy addition of new search engines, as well as easy incorporation into existing or new categories. Unfortunately, the search engine parsers require maintenance to ensure that they are always up to date with the current output user interface of each search engine.

Category	Attributes	Qmods	description
Topical relevance	quicktr	AV:none, GO:none, FA:none, NO:none	This category is a default category when no other category applies.
Research papers	learned research paper classifier, topical relevance	AV:none, FA:none, GO:none, NO:none, FA:introduction "shown in figure", NO:+"this paper" +"likely to", GO:abstract introduction, AV:+"this paper" +"likely to", FA:abstract introduction, NO:+introduction +references, CS:none	The research paper category used in the user study
Research papers about	agrade, researchpaper, wordcount, topicalrelevance, aveworddist, latex, wordspersection	GO:abstract keyword references, AV:, SN:, YH:abstract keyword references, HB, NO, FA:abstract introduction	This category was created manually, and was replaced by the learned category used in the user study
Detailed	agrade, topicalrelevance, wordcount	AV:none, SN:none, YH:none, HB:none, NO:none, GO:none	Searching for a page that is detailed, or advanced, not necessarily a research paper
Research papers which reference X	wordsinrefspercent, topicalrelevance, researchpaper	GO and SN:pp no. vol references bibliography, GO and YH:abstract keywords references, NO:none	Pages that contain the query in the references
Personal homepages	learned personal homepage classifier, topical relevance	AV:none, FA:none, GO:none, NO:none, FA:"s home" page, FA:"welcome to my" page, AV:+title:"s home", GO:"s home" page, NO:+"about me", AV:+"home page"	The personal homepage category used in the user study
Organizational homepage of	pathlength, homepage, terms in title domain or summary	SN:none, GO:none, YH:none, HB:none	The organizational or company homepage of the query
Current events/news	daysold, topicalrelevance, wordcount, agrade	AB: NE: EE: HB:date-constraint	Find recent news stories or current events
General introductory	genscore, topicalrelevance, wordsinfirst30percent, wordcount, GFOG	GO and AV:links resources, AV:"what is", GO:none, SN:none, AV:none, YH:none	Documents that are introductory, or reference or link pages
Reviews	learned review classifier, topical relevance	AV:none, FA:none, GO:none, NO:none, GO:reviews so, FA:reviews t, NO:+more +reviews, AV:+feature +nice, GO:"you can" better, AV:+t +buy	The reviews category used in the user study

Table A.4: Categories in Inquirus 2, their query modifications, scoring functions, and attributes

APPENDIX B

Symbol Features

QMLP described in Chapter 6, describes our procedure for learning source and category-specific query modifications. The first step involves training a document classifier. To improve the ability to classify web pages, we added several types of features. Typical features include words or phrases modified by their associated HTML tags, such as title or heading. We also consider word location (currently we only differentiate between words in the top 75 words of the document and words anywhere in the full text), or words in the URL. Some features may not have a corresponding textual representation, such as if there is a person's name in the title. To handle these features, a type called "symbol" was used and denoted with a "S." as the feature type. Symbol features can be added to improve the accuracy of classifiers for specific categories.

Feature	**Description**
S.tterm$n$	A set of features describing the number of words in the title. S.tterm4 means exactly 4 terms in the title. S.tterm7p means 7 or more terms in the title.
S.wordcount$n$	A set of features describing the number of words in the document. The larger the n the larger the wordcount. n ranges from 0 (0-99 words) to 5 (1600 or more words).
S.toptree	If true, then the document URL refers to a web page at the top of the URL hierarchy, e.g., "http://www.aol.com/privacy.html".
S.dpersonal	If true, then the document URL refers to a file likely to be present in a personal directory, e.g., "http://members.mydomain.com/user123/myfile.html".
S.titleStartsWithName	If true, the title starts with a person's name.
S.dlinks$n$	A set of features describing the number of links pointing to a page below the current page.
S.outlinks$n$	A set of features describing the number of outbound pointing links.

Table B.1: The symbol features used by our full-page classifiers

Table B.1 describes the symbol features that were available for full-page classifiers. Some features such as wordcount (S.wordcount$n$) or title terms (S.tterm$n$) are simple to compute. Features such as S.dpersonal, or S.titleStartsWithName are defined through heuristics. A URL can be considered a "personal directory" based on several common features including a tilde ($\sim$) or common words in the URL such as "people", "homepages", or "members". A person's name in the title is determined based on heuristics that utilize lists of known first and last names (surnames) and a dictionary of known "non-name" words.

The architecture of Inquirus 2 allows for easy addition of new symbol features to help support addition of new categories.

BIBLIOGRAPHY

BIBLIOGRAPHY

[Abramson, 1963] Abramson, N. (1963). *Information Theory and Coding*. McGraw-Hill, New York.

[Agichtein et al., 2001] Agichtein, E., Lawrence, S., and Gravano, L. (2001). Learning search engine specific query transformations for question answering. In *Proceedings of the 10th World Wide Web conference*, Hong Kong.

[Atkins et al., 1996] Atkins, D. E., Birmingham, W. P., Durfee, E. H., Glover, E. J., Mullen, T., Rundensteiner, E. A., Soloway, E., Vidal, J. M., Wallace, R., and Wellman, M. P. (1996). Toward inquiry-based education through interacting software agents. *IEEE Computer*, 29(5):69–76.

[Balabanovic and Shoham, 1997] Balabanovic, M. and Shoham, Y. (1997). Content-based, collaborative recommendation. 40(3):6–72.

[Barry, 1993] Barry, C. L. (1993). *The Identification of User Criteria of Relevance and Document Characteristics: Beyond the Topical Approach to Information Retrieval*. PhD thesis, Syracuse.

[Bates, 1986] Bates, M. (1986). Subject access in online catalogs: A design model. *J. Amer. Soc. Info. Science*, 37(6):357–376.

[Belkin, 2000] Belkin, N. J. (2000). Helping people find what they don't know. *Communications of the ACM*, 43(8):58–61.

[Bharat and Broder, 1998] Bharat, K. and Broder, A. (1998). A technique for measuring the relative size and overlap of public search engines. In *Proceedings of the 7th World Wide Web conference*, pages 379–388, Brisbane, Australia.

[Birmingham et al., 1994] Birmingham, W. P., Drabenstott, K. M., Frost, C. O., Warner, A. J., and Willis, K. (1994). The university of michigan digital library: This is not your father's library. In *Proceedings of Digital Libraries*, pages 53–60. http://www.csdl.tamu.edu/DL94/paper/umdl.html.

[Blum and Mitchell, 1998] Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*.

[Bowman et al., 1994] Bowman, B., Danzig, P., Hardy, D., Manber, U., and Schwartz, M. (1994). The harvest information discovery and access system. In *Proceedings of the Second International Conf. on the World Wide Web*, pages 763–771.

[Boyce, 1982] Boyce, B. (1982). Beyond topicality: A two stage view of relevance and the retrieval process. *Information Processing and Management*, 18(3):105–109.

[Brin and Page, 1998] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *WWW7*, Brisbane, Australia.

[Buckland et al., 1992] Buckland, M. K., Buttler, M. H., Norgard, B. A., and Plaunt, C. (1992). Oasis: A front-end for prototyping catalog enhancements. 10(4):7–22.

[Budzik and Hammond, 1999] Budzik, J. and Hammond, K. (1999). Watson: Anticipating and contextualizing information needs. In *62nd Annual Meeting of the American Society for Information Science*, Medford, NJ.

[Budzik et al., 2000] Budzik, J., Hammond, K. J., Birnbaum, L., and Krema, M. (2000). Beyond Similarity. In *Working notes of the AAAI Workshop on AI for Web Search*, AAAI Press, Austin, TX.

[Chakrabarti et al., 1998a] Chakrabarti, S., Dom, B., and Indyk, P. (1998a). Enhanced hypertext categorization using hyperlinks. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(2):307–318.

[Chakrabarti et al., 1998b] Chakrabarti, S., Dom, B., Raghavan, P., Rajagopalan, S., Gibson, D., and Kleinberg, J. (1998b). Automatic resource compilation by analyzing hyperlink structure and associated text. *Computer Networks and ISDN Systems*, 30(1–7):65–74.

[Chakrabarti et al., 1999] Chakrabarti, S., van der Berg, M., and Dom, B. (1999). Focused crawling: a new approach to topic-specific web resource discovery. In *WWW8*, Toronto, Canada.

[Cho et al., 1998] Cho, J., García-Molina, H., and Page, L. (1998). Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172.

[Coetzee et al., 2001] Coetzee, F., Glover, E., Lawrence, S., and Giles, C. L. (2001). Feature selection in web applications using ROC inflections. In *Symposium on Applications and the Internet, SAINT*, pages 5–14, San Diego, CA. IEEE Computer Society, Los Alamitos, CA.

[Cole et al., 1996] Cole, C., Kennedy, L., and Carter, S. (1996). The optimization of online searches through the labelling of a dynamic situation-dependent information need: The reference interview and online searching for undergraduates doing a social-science assignment. *Information Processing and Management*, 32(6):709–717.

[Cooper, 1971] Cooper, W. S. (1971). A definition of relevance for information retrieval. *Information Storage and Retrieval*, 7:19–37.

[Cooper, 1997] Cooper, W. S. (1997). On selecting a measure of retrieval effectiveness. In Jones, K. S. and Willett, P., editors, *Readings in Information Retrieval*, Multimedia Information and Systems, pages 87–100. Morgan Kaufmann.

[Cox, 1994] Cox, K. (1994). A unified approach to indexing and retrieval of information. pages 176–181.

[Craswell et al., 1999] Craswell, N., Hawking, D., and Thistlewaite, P. B. (1999). Merging results from isolated search engines. In *Australasian Database Conference*, pages 189–200.

[Croft et al., 1990] Croft, W. B., Krovetz, R., and Turtle, H. (1990). Interactive retrieval of complex documents. *Information Processing and Management*, 26(5):593–613.

[Cyveillance Corporation, 2000] Cyveillance Corporation (July 10, 2000). Internet exceeds 2 billion pages. *Company Press Release*. http://www.cyveillance.com/web/us/newsroom/releases/2000/2000-07-10.htm.

[Davis et al., 1990] Davis, F., Kahle, B., Morris, H., Salem, J., and Shen, T. (1990). Wais interface protocol prototype functional specification. Technical Report version 1.5, Thinking Machines Corporation.

[Davison, 2000] Davison, B. D. (2000). Topical locality in the web. In *Research and Development in Information Retrieval*, pages 272–279.

[Diligenti et al., 2000] Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. (2000). Focused crawling using context graphs. In *26th International Conference on Very Large Databases, VLDB 2000*, Cairo, Egypt.

[Edwards et al., 2001] Edwards, J., McCurley, K., and Tomlin, J. (2001). An adaptive model for optimizing performance of an incremental web crawler. In *The Tenth International World Wide Web Conference WWW10*.

[Flake et al., 2000] Flake, G. W., Lawrence, S., and Giles, C. L. (2000). Efficient identification of web communities,. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD-2000)*, Boston, MA. ACM Press.

[Frants et al., 1996] Frants, V. I., Shapiro, J., and Voiskunskii, V. G. (1996). Development of IR systems: New direction. *Information Processing and Management*, 32(3):373–386.

[Gauch et al., 1996] Gauch, S., Wang, G., and Gomez, M. (1996). ProFusion: Intelligent fusion from multiple, distributed search engines. *Journal of Universal Computer Science*, 2(9).

[Glover et al., 2001] Glover, E., Flake, G., Lawrence, S., Birmingham, W. P., Kruger, A., Giles, C. L., and Pennock, D. (2001). Improving category specific web search by learning query modifications. In *Symposium on Applications and the Internet, SAINT*, San Diego, CA.

[Glover et al., 1999a] Glover, E. J., Lawrence, S., Birmingham, W. P., and Giles, C. L. (1999a). Architecture of a metasearch engine that supports user information needs. In *Eighth International Conference on Information and Knowledge Management (CIKM'99)*, pages 210–216, Kansas City, MO. ACM Press.

[Glover et al., 1999b] Glover, E. J., Lawrence, S., Gordon, M. D., Birmingham, W. P., and Giles, C. L. (1999b). Recommending web documents based on user preferences. In *ACM SIGIR 99 Workshop on Recommender Systems*, Berkeley, CA. ACM Press.

[Gordon and Pathak, 1999] Gordon, M. and Pathak, P. (1999). Finding information on the world wide web: The retrieval effectiveness of search engines. *Information Processing and Management*, 35(2):141–180.

[Gravano et al., 1997] Gravano, L., Chang, C.-C. K., García-Molina, H., and Paepcke, A. (1997). STARTS: Stanford proposal for Internet meta-searching. pages 207–218.

[Gravano et al., 1998] Gravano, L., Garcia-Molina, H., and Tomasic, A. (1998). GlOSS: Text-source discovery over the Internet. *ACM Transactions on Database Systems*.

[Grossman and Frieder, 1998] Grossman, D. A. and Frieder, O. (1998). *Information Retrieval: Algorithms and Heuristics.* Kluwer Academic Publishers.

[Guernsey, 2001] Guernsey, L. (January 25, 2001). Mining the 'deep web' with specialized drills. *New York Times*.

[Harman, 1994] Harman, D. (1994). Overview of the third text retrieval conference (trec-3). In Harman, D., editor, *Proceedings of the third Text REtrieval Conference (TREC-3)*, page 1, Gaithersburg, MD. NIST Text REtrie (National Institute of Standards).

[Hawking et al., 1999] Hawking, D., Voorhees, E., Craswell, N., and Bailey, P. (1999). Overview of the trec8 web track. In *Eighth Text REtrieval Conference (TREC-7)*, Gaithersburg, Maryland.

[Hearst, 2000] Hearst, M. A. (2000). Next generation web search: Setting our sites. *IEEE Data Engineering Bulletin, Special issue on Next Generation Web Search, Luis Gravano (Ed.)*.

[Hoelscher, 1998] Hoelscher, C. (1998). How internet experts search for information on the web. Paper presented at the World Conference of the World Wide Web, Internet, and Intranet, Orlando, FL.

[Howe and Dreilinger, 1997] Howe, A. E. and Dreilinger, D. (1997). SavvySearch: A meta-search engine that learns which search engines to query. *AI Magazine*, 18(2).

[Jansen and Pooch, 2001] Jansen, B. J. and Pooch, U. (2001). A review of web searching studies and a framework for future research. *Journal of the American Society for Information Science and Technology*, 52(3):235–246.

[Joachims, 1999] Joachims, T. (1999). Text categorization with support vector machines: Learning with many relevant features. In *Tenth European Conference on Machine Learning ECML-98*, pages 137–142.

[Kleinberg, 1999] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

[Kochen, 1974] Kochen, M. (1974). *Principles of Information Retrieval.* Melville Publishing Company, Los Angeles, California.

[Koster, 1994] Koster, M. (1994). A standard for robot exclusion. http://www.robotstxt.org/wc/norobots.html.

[Koster, 1995] Koster, M. (April, 1995). Robots in the web: threat or treat? *ConneXions*, 9(4).

[Kwok, 1999] Kwok, J. T.-Y. (1999). Automated text categorization using support vector machine. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, pages 347–351, Kitakyushu, Japan.

[Lawrence, 2000] Lawrence, S. (2000). Context in web search. *IEEE Data Engineering Bulletin*, 23(3):25–32.

[Lawrence and Giles, 1998a] Lawrence, S. and Giles, C. L. (1998a). Context and page analysis for improved web search. *IEEE Internet Computing, July-August*, pages 38–46.

[Lawrence and Giles, 1998b] Lawrence, S. and Giles, C. L. (1998b). Inquirus, The NECI Meta Search Engine. In *WWW7*, pages 95–105, Brisbane, Australia.

[Lawrence and Giles, 1998c] Lawrence, S. and Giles, C. L. (1998c). Searching the World Wide Web. *Science*, 280(5360):98.

[Lawrence and Giles, 1999a] Lawrence, S. and Giles, C. L. (1999a). Accessibility of information on the web. *Nature*, 400(July 8):107–109.

[Lawrence and Giles, 1999b] Lawrence, S. and Giles, C. L. (1999b). Text and image metasearch on the web. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 829–835. CSREA Press.

[Lawrence et al., 1999] Lawrence, S., Giles, C. L., and Bollacker, K. (1999). Digital libraries and Autonomous Citation Indexing. *IEEE Computer*, 32(6):67–71.

[Leake et al., 1999] Leake, D. B., Scherle, R., Budzik, J., and Hammond, K. (1999). Selecting task-relevant sources for just-in-time retrieval. In *AAAI-99 Workshop on Intelligent Information Systems)*, Menlo Park, CA. AAAI Press.

[Lieberman, 1995] Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *International Joint Conference on Artificial Intellignce*, Montreal Canada.

[McBryan, 1994] McBryan, O. A. (1994). GENVL and WWWW: Tools for taming the web. In *Proceedings of the First International World Wide Web Conference*, pages 79–90, Geneva, Switzerland.

[Media Metrix Corporation, 2001] Media Metrix Corporation (March 13, 2001). Jupiter media metrix announces u.s. top 50 web and digital media properties for february 2001. *Company Press Release*. http://us.mediametrix.com/press/releases/20010313.jsp.

[Mitra et al., 1998] Mitra, M., Singhal, A., and Buckley, C. (1998). Improving automatic query expansion. In *ACM SIGIR 98*, Melbourne Australia. ACM.

[Mizzaro, 1997] Mizzaro, S. (1997). Relevance: The whole history. *Journal of the American Society for Information Science*, 48(9):810–832.

[National Information Standards Organization, 1995] National Information Standards Organization (1995). Application service definition and protocol specification (z39.50). NISO Press, 1995. http://lcweb.loc.gov/z3950/agency/.

[Nguyen and Haddawy, 1998] Nguyen, H. and Haddawy, P. (1998). The Decision-Theoretic Video Advisor. In *AAAI Workshop on Recommender Systems*.

[Paepcke et al., 1997] Paepcke, A., Cousins, S. B., Garcia-Molina, H., Hassan, S. W., Ketchpel, S. K., Roscheisen, M., and Winograd, T. (1997). Towards interoperability in digital libraries: Overview and selected highlights of the stanford digital library project. Technical Report CS-TR-97-1581.

[Platt, 1998] Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In Scholkopf, B., Burges, C., and Smola, A., editors, *Advances in kernel methods - support vector learning*. MIT Press.

[Platt, 1999] Platt, J. (1999). Using sparseness and analytic QP to speed training of support vector machines. In *Advances in Neural Information Processing Systems*.

[Salton, 1989] Salton, G. (1989). *Automatic Text Processing.* Addison-Wesley Series in Computer Science. Addison-Wesley.

[Saracevic, 1975] Saracevic, T. (1975). Relevance: A review of and a framework for the thinking on the notion in information science. *(book) Readings in Information Retrieval – Edited by Karen Sparck Jones and Peter Willett, 1997, Morgan Kaufmann Publishers, Inc.*, 26:321–343.

[Schamber et al., 1990] Schamber, L., Eisenberg, M. B., and Nilan, M. S. (1990). A re-examination of relevance: Toward a dynamic, situational definition. *Information Processing and Management*, 26(6):755–776.

[Selberg and Etzioni, 1997] Selberg, E. and Etzioni, O. (1997). The MetaCrawler architecture for resource aggregation on the Web. *IEEE Expert*, (January–February):11–14.

[Selberg, 1999] Selberg, E. W. (1999). *Towards Comprehensive Web Search*. PhD thesis, University of Washington.

[Seltzer et al., 1997] Seltzer, R., Ray, E. J., and Ray, D. S. (1997). *The AltaVista Search Revolution: How to Find Anything on the Internet.* McGraw-Hill.

[Silverstein et al., 1999] Silverstein, C., Henzinger, M., Marais, H., and Moricz, M. (1999). Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12.

[Spink et al., 1999a] Spink, A., Bateman, J., and Jansen, B. (1999a). Searching the web: A survey of excite users. *Internet Research: Electronic Networking Applications and Policy*.

[Spink et al., 1999b] Spink, A., Chang, C., and Goz, A. (1999b). Users' interactions with the excite web search engine: A query reformulation and relevance feedback analysis. In *Proceedings of the 1999 Canadian Association for Information Science (CAIS)*, pages 342–354, Sherbrooke - Canada.

[Tonta, 1992] Tonta, Y. (1992). Analysis of search failures in document retrieval systems: A review. 3(1):4–53.

[Van Rijsbergen, 1979] Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition.* Dept. of Computer Science, University of Glasgow, 2nd edition.

[Vapnik, 1995] Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer Verlag, New York.

[Veerasamy and Belkin, 1996] Veerasamy, A. and Belkin, N. J. (1996). Evaluation of a tool for visualization of information retrieval results. pages 85+.

[Voorhees, 1985] Voorhees, E. M. (1985). The cluster hypothesis revisited. In *Research and Development in Information Retrieval*, pages 188–196.

[Weinstein and Birmingham, 1999] Weinstein, P. C. and Birmingham, W. P. (1999). Agent communication with differentiated ontologies: eight new measures of description campatibility. Technical Report CSE-TR-383-99.

[Zamir et al., 1997] Zamir, O., Etzioni, O., Madani, O., and Karp, R. M. (1997). Fast and intuitive clustering of web documents. In *3rd International Conference on Knowledge Discovery and Data Mining*, pages 287–290.

[Zhu and Gauch, 2000] Zhu, X. and Gauch, S. (2000). Incorporating quality metrics in centralized/distributed information retrieval on the world wide web. In *Proceedings of the 23rd Annual International ACM/SIGIR Conference*, pages 288– 295, Athens, Greece.