Methods for Distributed Information Retrieval

Nicholas Eric Craswell

A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

May 2000

Typeset in Palatino by TEX and LATEX 2ε.

Except where otherwise indicated, this thesis is my own original work.

Nicholas Eric Craswell
23 May 2000

This thesis is dedicated to my original PhD supervisor Paul Thistlewaite, who passed away in February 1999.

Acknowledgements

Thanks to the many people who contributed to this work, either directly by discussing the work or reviewing drafts, or indirectly by giving their time and friendship.

Thanks to my supervisor David Hawking for providing advice and support in so many areas. His research knowledge, technical understanding, common sense, good attitude and friendship improved my work and my PhD experience.

Thanks to the late Paul Thistlewaite for guiding me through my initial experiments in document filtering, my dabbling with aglets and on to the development of the work presented here. He guided me towards the principles of practicality, effectiveness and generality which I hope are embodied in this thesis.

Thanks to Peter Bailey, who joined our research group in late 1998. He has variously been my thesis advisor, research collaborator, temporary house-mate and friend. Thanks also to Chris Johnson for being on my panel and providing sound thesis advice.

Thanks to the postgrad discussion group attendees, including at various stages Steve Blackburn, Bill Clarke, Roland Goecke, Zhen He, Raj Nagappan, Rochelle O'Hagan, Sam Taylor, Linda Wallace and John Zigman.

Thanks to various other friends, collaborators and partners in crime: Francis Crimmins, Mark Grundy, Jason Haines, Steve Lawrence, Andy Macfarlane and Robert Umphelby.

Thanks to the Department of Computer Science for keeping me in offices, stationary and coffee, and for funding my Berkeley conference trip. Thanks to Richard Walker for TEX support and lending me useful books. For good Linux systems, the loan of a large monitor and miscellaneous advice, particularly during writeup, thanks go to Bob Edwards and the rest of the technical support group.

Thanks to the Cooperative Research Centre for Advanced Computational Systems (ACSys) for funding my travel to several conferences. Thanks to Mark Grundy and John O'Callaghan of ACSys for organising my AltaVista internship at Digital's Gold Coast labs, and to Greg McCane for having me there. Thanks to Jan Bitmead for organising the ACSys Student Meetings, Professional Development Courses and everything else.

Thanks finally for support from family: Mum, Dad, Penny, the Craswells, the Gillespies and the Woltmanns.

This research was supported by an Australian Postgraduate Award PhD Scholarship. Additional financial support was provided in the form of a supplementary scholarship by the ACSys Cooperative Research Centre.

Presentational Conventions

A number of presentational conventions have been adopted in this thesis:

- Italics are used when a term is defined, but not thereafter.
- Spelling is according to the (Australian) Macquarie Dictionary, in particular the version which is searchable on the Web (`http://www.dict.mq.edu.au/`).
- References to archival publications are used in preference to Internet URLs.
- In cases when URL references are necessary, a URL is inserted in parentheses like this (`http://pastime.anu.edu.au/nick/pubs/`). Each URL refers to a Web page as it appeared on 23 May 2000.
- Equations from other studies are presented here using a standard notation, which is described in Table 2.2. While every effort has been made to avoid introducing errors, readers should consult the original studies for authoritative versions.

Abstract

Published methods for distributed information retrieval generally rely on cooperation from search servers. But most real servers, particularly the tens of thousands available on the Web, are not engineered for such cooperation. This means that the majority of methods proposed, and evaluated in simulated environments of homogeneous cooperating servers, are never applied in practice.

This thesis introduces new methods for server selection and results merging. The methods do not require search servers to cooperate, yet are as effective as the best methods which do. Two large experiments evaluate the new methods against many previously published methods. In contrast to previous experiments they simulate a Web-like environment, where servers employ varied retrieval algorithms and tend not to sub-partition documents from a single source.

The server selection experiment uses pages from 956 real Web servers, three different retrieval systems and TREC ad hoc topics. Results show that a broker using queries to sample servers' documents can perform selection over non-cooperating servers without loss of effectiveness. However, using the same queries to estimate the effectiveness of servers, in order to favour servers with high quality retrieval systems, did not consistently improve selection effectiveness.

The results merging experiment uses documents from five TREC sub-collections, five different retrieval systems and TREC ad hoc topics. Results show that a broker using a reference set of collection statistics, rather than relying on cooperation to collate true statistics, can perform merging without loss of effectiveness. Since application of the reference statistics method requires that the broker download the documents to be merged, experiments were also conducted on effective merging based on partial documents. The new ranking method developed was not highly effective on partial documents, but showed some promise on fully downloaded documents.

Using the new methods, an effective search broker can be built, capable of addressing any given set of available search servers, without their cooperation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A large proportion of Web content is not indexable by centralised Web search engines. Even when considering only the indexable documents, engines have partial coverage and infrequently update their indexes. For this reason, it is common for a document provider to run its own search server, which provides the most comprehensive (and perhaps only) index of its documents. This is the case, for example, at `britannica.com` and `amazon.com`. Tens of thousands of these smaller search servers are available on the Web.

A person who wishes to use these numerous smaller servers faces several problems. They must somehow discover which servers exist. For a particular query they must select which to search, use varied server query interfaces and view servers' non-comparable results lists, one after the other. These difficulties are faced on the Web, but also in any environment of distributed search servers.

A search broker acts as an intermediary between a user searching for information and a set of search servers. It may perform automatic server selection, choosing servers which are likely to be most useful. It may also concurrently query the selected servers and present their results to the user in a single merged list. It does so by interfacing with the various servers to retrieve their results then applying a results merging method. A broker which responds to queries with a merged results list provides the query-response interface of a centralised search server, while actually performing a distributed search.

The effectiveness of a broker over a given set of servers depends on the effectiveness of its server selection and results merging methods. Its selection method must choose servers which return relevant documents. Its merging method must rank the combined results, which might number in the tens or hundreds, such that the relevant results are in top ranks.

A generally applicable broker can address any set of available search servers without requiring explicit cooperation. Examples of explicit cooperation are exporting term occurrence statistics or running uniform search software. Generally applicable brokers can address a wider range of information needs by addressing a wider range of existing search servers. If a new search server arises which is of interest to users, a generally applicable broker can search it "as is". Other brokers would require the new server to either cooperate or be excluded.

Making a generally applicable broker raises interesting research problems. For

example, many server selection methods are based on the assumption that search servers export statistics describing term occurrences in their documents. Based on these statistics, the methods tend to select servers with more query term occurrences. If search servers do not cooperate by exporting term occurrence information, how can the broker perform selection?

This thesis shows that a search broker can be generally applicable without sacrificing effectiveness.

1.1 Research approach

This thesis considers the problems of effective server selection and results merging in an environment where not all servers cooperate. It proposes new methods and evaluates them against numerous existing methods including those which require cooperation. The question is whether methods which do not require cooperation can be as effective as state of the art methods which do.

Evaluation experiments are based on methodology from the Text Retrieval Conference (TREC) [Harman 1999], but with some innovations.

The new selection and merging methods are not yet implemented in a broker. However, they are simple extensions of methods already used by the Web search broker Inquirus [Lawrence and Giles 1998], so their addition to a broker would be straightforward.

The thesis does not explicitly report all joint research conducted throughout the author's thesis project. It excludes work on Web search evaluation (Hawking, Craswell, Thistlewaite, and Harman [1999], Craswell, Bailey, and Hawking [1999] and Hawking, Craswell, Bailey, and Griffiths [2000]), aglets (Craswell, Haines, Humphreys, Johnson, and Thistlewaite [1997]) and miscellaneous TREC efforts (Hawking, Thistlewaite, and Craswell [1997], Hawking, Craswell, and Thistlewaite [1998a], Hawking, Craswell, and Thistlewaite [1998b], Hawking, Craswell, and Bailey [1999] and Hawking, Voorhees, Craswell, and Bailey [1999]). All this work contributes to the author's understanding of the field, but is not crucial to effective, generally applicable distributed information retrieval. The two large evaluation experiments in this thesis have also been reported in two conference papers which are the original work of the author: Craswell, Hawking, and Thistlewaite [1999] and Craswell, Bailey, and Hawking [2000].

Chapter 2 surveys distributed information retrieval, including its problems, methods and evaluation methodology. Chapter 3 introduces new server selection and results merging methods, and states hypotheses concerning their effectiveness. Chapter 4 discusses experimental methodology, explaining decisions made in the design of selection and merging experiments. Chapters 5 and 6 present evaluation experiments in selection and merging respectively, including results, discussion and conclusions. Chapter 7 presents overall thesis conclusions and implications for the field of distributed information retrieval. Appendix A describes the distributed information retrieval terminology which is applied consistently throughout this thesis.

Chapter 2

Distributed Information Retrieval

This chapter introduces the field of distributed information retrieval. It begins by describing the relationship between past work in the field and new work presented in this thesis. Then its three remaining sections each deal with a different aspect of that past work. Section 2.2 describes problems and terminology in distributed information retrieval, Section 2.3 describes solutions, including previously published selection and merging methods as well as currently available search brokers. Section 2.4 surveys past effectiveness evaluation experiments in the area, including experimental methods and results.

2.1 Relationship with previous research

The study of distributed information retrieval lies at the intersection between information retrieval and distributed systems. From the former comes the goal of effectiveness, that a person who enters a query in the broker should find relevant information. This thesis attempts to develop effective methods, and evaluates the effectiveness of new methods against numerous other methods. Several previous studies have also concentrated on effectiveness and performed effectiveness evaluation experiments (Section 2.3 describes methods and Section 2.4 describes evaluation experiments).

From distributed systems comes the issue of breadth of applicability. A broker and search server are incompatible if one requires communication which the other does not support. For example, a STARTS [Gravano et al. 1997] search broker requires that all servers be compatible with the Stanford Proposal for Internet Meta-Searching. Because most Web search servers are not STARTS compatible, such a broker is not generally applicable in a Web environment. Many of the studies concerned with effective distributed information retrieval have introduced methods which are not generally applicable. Conversely, other studies have introduced methods which are generally applicable, and implemented in a Web search broker, but have never undergone effectiveness evaluation. This thesis attempts to develop methods which are generally applicable.

This thesis draws on generally applicable distributed information retrieval methods in three ways. First, it evaluates previously unevaluated generally applicable merging methods, to see how they compare to merging methods which rely on server

cooperation. Second, the methods it introduces are based on the functionality of the operational Web search broker Inquirus [Lawrence and Giles 1998]. Because Inquirus is generally applicable, and the new methods require no further cooperation from search servers, the new methods are also generally applicable.

Third, its evaluation experiments attempt to model real Web search servers. Previous experiments in the area (see Section 2.4) have often split documents from a single source, for example a single TREC sub-collection, amongst multiple search servers and applied homogeneous retrieval systems at the server. By contrast, Web servers such as those listed by InvisibleWeb (`http://www.invisibleweb.com/`) and CiteLine (`http://www.citeline.com/`) use varied retrieval systems and seldom split documents from a source amongst several servers. Most are site-search servers, indexing a single source. Both evaluation experiments in this thesis model heterogeneous retrieval systems and avoid splitting a single source amongst multiple search servers. The latter choice leads to a greater server size variation and topic skew than in some previous experiments.

The thesis also draws on methods which require cooperation. The new methods, probe queries for selection and reference statistics for merging, are enabling technologies for existing server ranking and document ranking algorithms. For example, probe queries allow CORI server ranking [Callan et al. 1995] to take place without server cooperation. Reference statistics allow Okapi BM25 [Robertson et al. 1994] to be applied in results merging, again without server cooperation.

New methods are applicable to information retrieval environments beyond the client/server/broker environment. The selection and merging methods introduced here would be equally applicable in an environment of distributed, interacting, intentional agents [Danzig et al. 1991; Clark and Lazarou 1997]. Merging methods could be applied in parallel information retrieval (see [Rasmussen 1991]), where multiple processing nodes are used to speed up retrieval. Reference statistics (see Chapter 3) have already been used to merge results from eight nodes in TREC [Hawking et al. 1998a]. This saved each node from collating full statistics from the 100 gigabyte VLC2 [Hawking et al. 1998b]. The new merging methods could equally be applied in data fusion (for example [Vogt and Cottrell 1998]), where multiple retrieval methods, multiple query variants or multiple document representations are combined to enhance retrieval effectiveness.

Chapter 4 introduces innovations in evaluation methodology, attempting to improve on experimental methods described in Section 2.4. Merging evaluation is based on simulated input rankings, which are combined to simulate many heterogeneous server configurations. The selection evaluation is less novel, but still is the first to use Web data and one of the first to use heterogeneous retrieval systems and document partitioning by source for modelling distributed servers. Also in selection, this thesis challenges evaluation methodologies based on ideal server ranks.

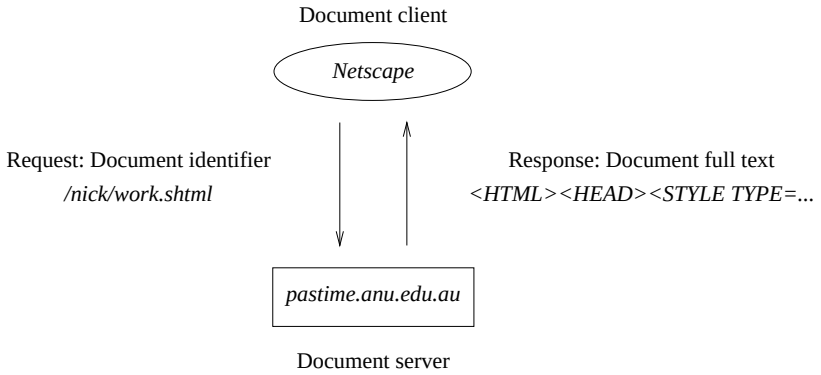


Figure 2.1: Document request. The client request contains a document identifier and the server's response contains the full text of the document in question, if available. In this case the Netscape Navigator client is requesting the document with `http://pastime.anu.edu.au/nick/work.shtml` as its Uniform Resource Locator (URL) [Berners-Lee et al. 1994]. The client sends an HTTP [Berners-Lee et al. 1999] request to `http://pastime.anu.edu.au` for the document identified (within the server) as `/nick/work.shtml`. The server's response contains the full text of the document in question (which is in the HyperText Markup Language (HTML) [W3C 1998]).

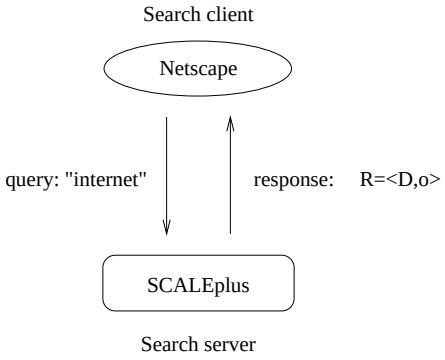
2.2 Problem description

2.2.1 Clients, servers, brokers and users

A *distributed system* is a collection of autonomous computers which cooperate in order to achieve a common goal. They do so without sharing memory or clock, and communicate by passing messages over a communication network. Ideally, the person using such a system is not aware of the different computers, their location, storage replication, load balancing, reliability or functionality. Instead the system should appear as though it runs on a single computer (see "distributed system" in [Howe 1999]). Practical systems succeed to varying degrees in providing such transparency, as was put best by Leslie Lamport: "A distributed system is one in which I cannot get something done because a machine I've never heard of is down."

This thesis considers the problem of distributed information retrieval, the basic unit of which is the electronic *document*. Documents may be full-text, bibliographic, sound, image, video or mixed-media records, although methods studied here operate over the text contained in or associated with each document. A *document server* is set up by some individual or organisation wishing to publish a set of electronic documents. The publisher is referred to loosely as a *document source*. A person views such documents using a *document client*, for example a simple Web browser. To view a document the client sends a request containing a *document identifier*, such as an Internet URL, and the document server returns the document in question if available. This document-pull process is familiar to any person who has used the Web (see Fig-

a) Simple search



b) User's view when entering query

c) User's view of results

Figure 2.2: Simple search. This figure illustrates a simple search of the SCALEplus Web search server. The simple search client sends a query to the search server, which returns a list of results $R = \langle D, o \rangle$ (see explanation in text). Also illustrated are the query (b) and the response (c) as they would appear in the Netscape client.

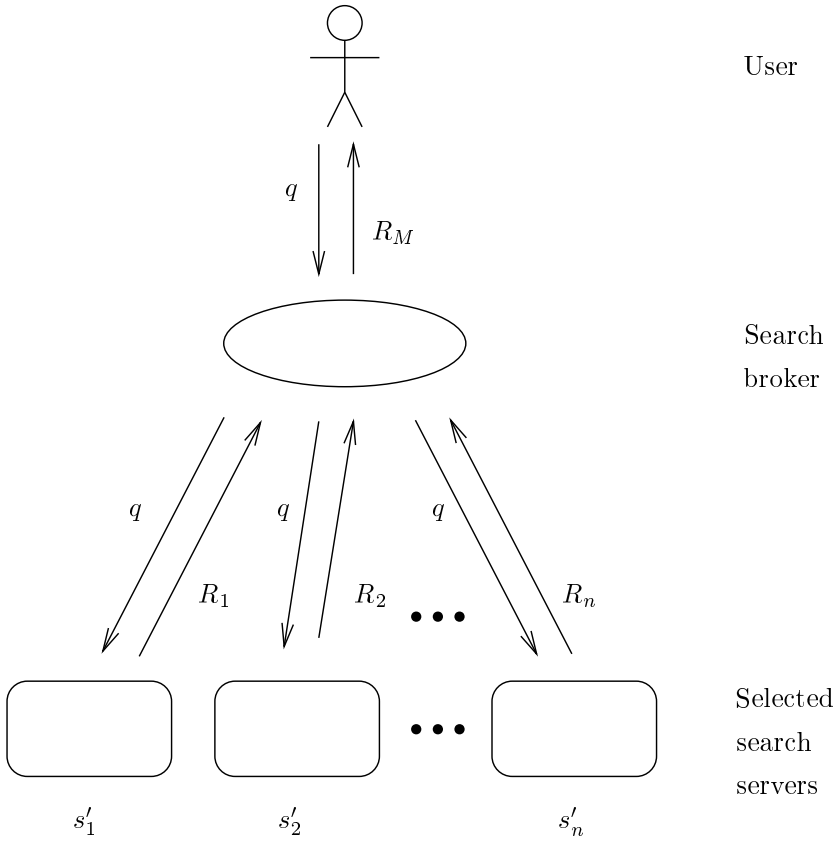


Figure 2.3: Search broker network communication. The broker conducts several concurrent simple searches (see Figure 2.2). The user first provides the broker with a query q, either through the broker's graphical user interface (illustrated here) or by connecting to the broker using a client such as a Web browser. The broker selects servers $(s'_1, s'_2, \ldots, s'_n)$, retrieves their results $(R_1, R_2, \ldots, R_n)$ and generates a merged results list R_M. The broker provides the same query-response interface as a single search server (Figure 2.2), taking a query and returning ranked results, but bases its results on those of multiple distributed search servers.

ure 2.1).

An information retrieval problem arises when a person has access to many documents and requires some systematic organisation or search facility to find relevant information. A common form of information retrieval system is one which takes a query from the person who wishes to find information, and returns a list of documents which are estimated likely to be relevant. Retrieval of relevant information may also be aided by browsing amongst document hyper-links or some category/directory hierarchy.

A distributed information retrieval problem arises when the documents are spread across many document servers. In such a situation it may be possible for a single information retrieval system to request every document from every document server, and perform its search task over the combined document set. Alternatively, various search servers may be set up on the network each covering documents from one or more document servers. In any case such networked information retrieval systems usually provide their search service to clients across the network (as opposed to restricting their service to a single machine). An information retrieval system available across the network is called a *search server* (Figure 2.2), and it is accessed using a *search client*.

Systems which return search results, such as search servers and other information retrieval systems, usually return to the user a ranked *results list* R. The minimal content of $R = \langle D, o \rangle$ is a set of document identifiers D and some ordering o over D. Other information may also be present in results, for example each result in Figure 2.2 has a relevance score (such as 0.98) and a document summary. A system is more *effective* if its results document set D contains more relevant documents, or the same number of relevant documents ranked more highly (o). A system is more *efficient* if it has reduced the costs involved in finding R. The cost of search includes several factors. Computation or storage resources may be expended at client or server. Network resources such as bandwidth may be expended in their communication. Monetary network usage or per-search charges might also apply. Users want a system which is both effective and efficient, in the latter case particularly minimising the costs which apply to the user. However this thesis focuses on effectiveness.

If the system is a search server, its effectiveness depends on the documents it indexes and its *retrieval system*. A retrieval system implements several retrieval algorithms, for ranking, stemming, case folding, relevance feedback and other functions. Several examples of such algorithms are described by Frakes and Baeza-Yates [1992].

One type of search client is a *simple client*, such as Netscape Navigator in Figure 2.2. Users of a simple client face a number of problems. First, they may be have difficulty finding new servers and selecting which to search, particularly in an environment such as the Web where there is no exhaustive list of servers and servers do not export descriptions of their documents. Further, if useful results are spread across multiple search servers, the user must query each in turn after learning the query language and interface conventions of each. This process of learning and querying sequentially is time consuming. The simple client also fails in terms of transparency, because the user is aware of search server heterogeneity, delays and down time. Finally, a simple

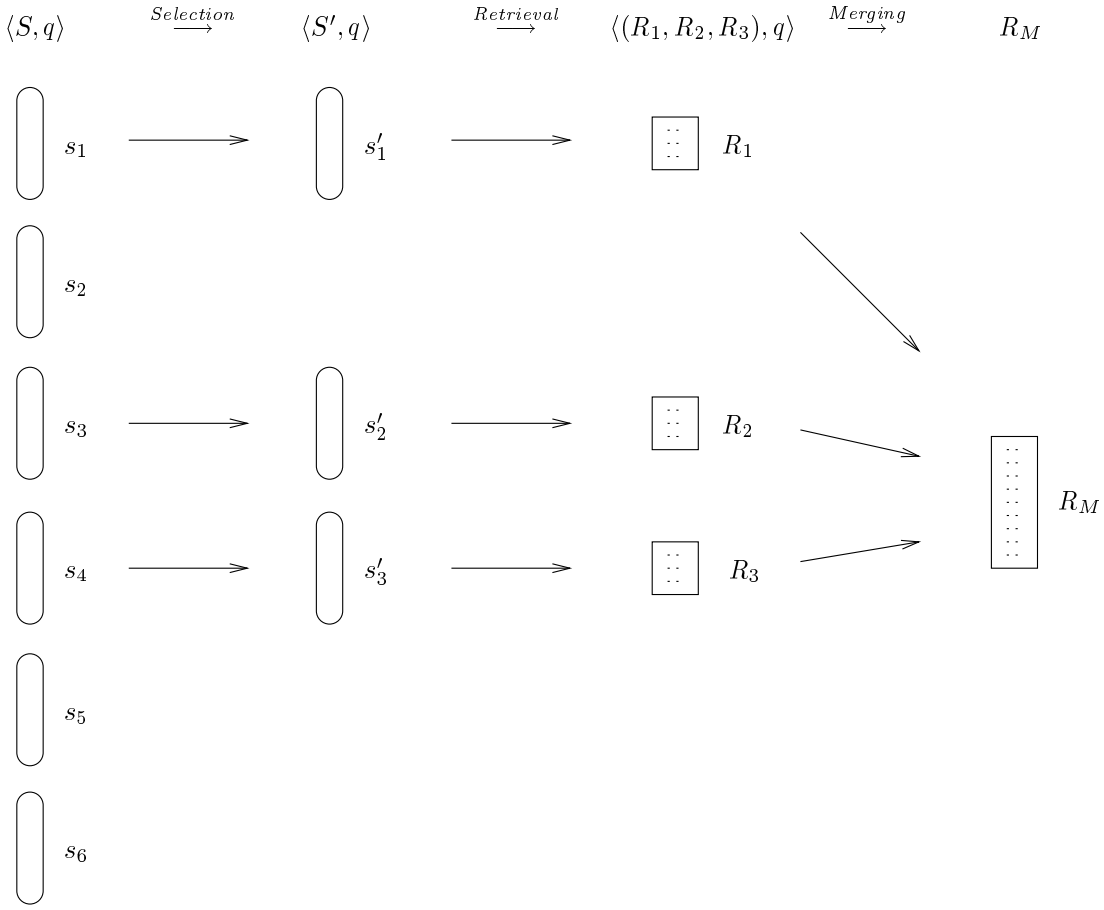


Figure 2.4: Search broker information flow. Given S and q the broker selects a subset $S' \subseteq S$, retrieves R_1, R_2, R_3 from those servers and builds the merged list R_M. The query q usually guides each stage of the process, although in certain cases it may be ignored, for example if the broker's policy is to always select all its servers $S' = S$.

client does not provide a unified view of results from different servers. The user has no indication of how results from one list compare to those of another, or even how each document matches their query. For example, one server given the query "david hawking" might return only documents containing the phrase, while others might return documents containing one word or the other, or even documents containing words with the same stem such as "hawk" and "hawker".

A *search broker* is a more sophisticated search client. Given a query and a set of search servers, it selects a set of servers likely to return relevant documents, queries them concurrently and produces a single ranked results list (see Figures 2.3 and 2.4):

$$\langle S, q \rangle \xrightarrow{\textit{Selection}} \langle S', q \rangle \xrightarrow{\textit{Retrieval}} \langle (R_1, \ldots, R_{|S'|}), q \rangle \xrightarrow{\textit{Merging}} R_M$$

The broker's task begins with a set of search servers S and a query q. A broker is set up to *address* servers S, analogously to a search server set up to search some document set. Identification of servers S is usually performed manually, as noted by Hawking and Thistlewaite [1999] and Fuhr [1999] who calls it the problem of database detection. IntelliSeek Inc (`http://www.invisibleweb.com/`) claim to have a semi-automatic server identification technique, but this is only available internally to its editorial team. The query q is provided by the user, and describes their information need.

During *server selection* the broker selects a subset S' of servers S which are best for answering the user's query q. Choice of best servers might depend on both effectiveness and efficiency considerations. This thesis makes a simplifying assumption that all servers have the same search cost and the user wishes to maximise the quality of results based on bounded costs $|S'| \leq 10$. As will be shown in Chapter 4 the contribution of a server to final results quality depends on the documents it indexes, its retrieval system, the documents and retrieval systems of other servers selected so far and the broker's merging method (see Sections 4.1.1 and 4.1.2). Finding the best selection is therefore a complex optimisation problem. Section 2.3 describes previously published selection methods.

During *retrieval* the broker applies the query q at servers S' to obtain results lists $R_1, \ldots, R_{|S'|}$. As described previously, each results list $R_i = \langle D_i, o_i \rangle$ consists of a document set D_i and an ordering o_i. The broker must employ the appropriate retrieval methods — communication protocol, query language and results parser — to retrieve each list R_i. However, for a given set of servers S', these methods have little influence over final broker effectiveness. Rather, the retrieval system and document set at server s'_i determines the quality of R_i. In an environment such as the Web the broker designer usually has no control over server effectiveness. Instead the broker's retrieval methods either succeed or fail in retrieving R_i. Currently operating Web search brokers (described later) demonstrate that it is possible to succeed, so this thesis concentrates on the broker's methods which do influence its effectiveness: selection and merging.

During *results merging* the broker combines results $R_1, \ldots, R_{|S'|}$ into a merged results list $R_M = \langle D_M, o_M \rangle$, such that $D_M = D_1 \cup \cdots \cup D_{|S'|}$ and o_M is an effective ranking. Merging may be based on properties of $R_1, \ldots, R_{|S'|}$, downloaded documents D_M or information provided by cooperating servers.

A broker may apply very simple methods for selection and merging. For example, it may select $S' = S$ for every query as does MetaCrawler [Selberg and Etzioni 1997]. It may also merge results lists by simply concatenating the incoming lists as does DogPile (`http://www.dogpile.com/`). Such selection and merging is likely to be ineffective in an environment of many search servers, some of which return no relevant documents. Selecting all servers is also inefficient, again because it may lead to querying servers which contribute no useful information. This thesis considers more effective selection and merging methods.

2.2.2 Practical considerations

Having described in the abstract the entities in a distributed information retrieval system and their interactions, this section ends by describing practical considerations in distributed search.

Why not build a central index?

If the goal of a search broker is to provide a search service over the documents of search servers S, why not download the documents in question and provide a centralised search service?

One reason is cost. In early 1999 a search broker covering 12 large search servers could cover 42% of the estimated 800 million Web pages, while the largest individual search server covered only 16% [Lawrence and Giles 1999]. Each of the large search servers was expensive to run, downloading a large number of documents (Web crawling) and maintaining the required hardware and personnel to provide fast search results to the world. By contrast, a search broker could cover more than double the number of documents with minimal expenditure. Making use of existing search servers is less expensive than building a central index.

The other reason is access permissions. For example, CareData and InvisibleWeb claim their thousands of search servers cover documents "hidden" or "invisible" to other indexers. Documents are invisible because of access restrictions such as:

- A robot exclusion request [Koster 1994] restricts access by outside indexers. Document providers make an exclusion request if they wish to avoid crawling by outside indexers, which can increase the provider's server load, network traffic and network bill. Providers may also request exclusion on documents which are updated much more frequently than the outside indexer updates its index. For example, a news site might update its documents every day but an outside indexer might only update its index once a week. This means that soon after every index update the search server is likely to begin returning documents which have changed or no longer exist.

- Most outside indexers on the Web rely on links to find documents (when crawling). In some cases a document set may be provided without links through a search-only interface.

- Intranet documents or password-protected documents are not available for central indexing. Even if they were made available for central indexing, their inclusion in a generally-available search servers would be a potential security hole, possibly allowing users without access permissions to find out about restricted documents.

Search servers over invisible documents are usually set up by the provider of those documents or with the explicit permission and cooperation of that document provider. This allows any access restrictions to be bypassed. A prominent example of documents with access restrictions are the MEDLINE abstracts, searchable only through

the PubMed search server (`http://www.ncbi.nlm.nih.gov/PubMed/`) due to robot exclusion and a lack of hyper-links. Whenever access restrictions apply to documents, building a central index may not be an option, while distributed search may.

Why build a generally applicable broker?

A server which is not generally applicable might, for example, rely on servers to be STARTS [Gravano et al. 1997] or Z39.50 [ANSI/NISO 1995] compliant. There are tens of thousands of search servers on the Web, according to InvisibleWeb (`http://www.invisibleweb.com/`), the vast majority of which do not support such standards. It is also rare for search server software, such as the products listed at SearchTools (`http://www.searchtools.com/`), to support such standards. The advantage of a generally applicable broker is that it can address one or more servers which do not explicitly cooperate, while a broker which relies on cooperation can address none.

Will search server administrators find search brokers acceptable?

By performing concurrent search and results merging, a search broker allows users to quickly find documents from various servers. However, in doing so it also bypasses individual search server results pages. Some search server administrators will find this acceptable, others will not.

A common situation on the Web is for a document provider to set up a search service with the sole purpose of improving access to its documents. This is the case with many servers listed by InvisibleWeb. In such cases, bypassing the search results should not be a problem. Being associated with a broker should increase, or at least not reduce, user access to a search server's results.

However, not all search servers administrators would be pleased to be searched by a broker. For example, a search engine company provides results with related branding, advertising, services and links. It derives no benefit from increased traffic to documents it returns, because it has no strong association with document providers. A broker in effect "steals" and "rebadges" the product of a search engine company, its results page, simply by reformatting it into a merged list. It is possible for the merged list to maintain some of the search engine's branding and advertising. However, if a search broker were to become highly popular, search engines still might take technical or legal action to block it.

This is one reason why this thesis concentrates on the tens of thousands of InvisibleWeb search servers rather than the handful of advertising funded search engine company servers.

What about server discovery?

This thesis presupposes that a search broker administrator knows of some useful set of search servers S, then configures the broker to address those servers. This is analogous to a Web site administrator knowing of some document set and then configuring

a search server to search them. However, on the Web it is possible to discover new documents automatically based on hyper-links using a Web crawler. The crawler allows a Web search engine to index documents which the engine's administrators never explicitly instructed should be indexed. An analogue for this is a search broker which can find its own servers S using automatic means. As stated previously, IntelliSeek claim to have a semi-automatic method for discovering search servers, but few details of it have been released. Automatic server discovery is a largely unexplored problem area.

What about retrieval methods, particularly query translation?

This thesis is concerned with building an effective broker, which operates over some existing set of search servers. In designing such a broker, choice of selection and merging methods is the major determinants of broker results quality. The broker's methods for retrieving results R_i from server s_i' have very little influence over effectiveness, since the broker has no control over servers' documents and retrieval system.

However, the problem of retrieval from search servers on the Web is difficult. The broker must deal with a variety of CGI search interfaces. It must also handle changes in the interface, either by adapting to minor changes in the HTML form or gracefully ignoring a changed server until the broker is manually updated to handle it. Results presentation may also change over time, and the broker should also adapt to those changes. Finally, there is some small scope for a broker's design to influence the quality of results returned by a server. Query translation, changing the query entered by the user into a format acceptable to the search server, can be performed well or badly by the broker. If performed well it can improve the server's effectiveness and therefore the broker's. However, none of these questions are considered further in this thesis, because their impact on broker effectiveness is small.

What about document volatility?

There is a temporal dimension to distributed information retrieval, with two major effects. First a search server's index becomes out of date as documents change. Its index is based on the contents of documents at some point in time. If documents have changed since index time, the server might return documents which no longer exist or no longer match the query. Also, a cooperating server might export other information which has become out of date. Second, as documents and search servers change a search broker can become out of date. For example, if the broker is performing selection based on server performance on past queries, it is using information which may no longer be applicable.

These problems stem from document volatility. A perfect solution would depend on cooperation. A document server could notify the search server and broker every time a document changes, to allow for updating of information. However, such an approach might be highly inefficient.

None of the previously published methods fall into this perfect category. How-

ever, some methods are better than others. Selection based on lightweight probes (the methods are described in full later) is better than selection based on STARTS, because probes get fresh information before each query, while STARTS information updates are less regular. However, lightweight probes are not perfect since the search server responding to the probe might have an out of date index with respect to current document contents.

Merging based on document downloads is better than merging based on server cooperation, since documents might have changed since the cooperating search servers last updated their index. Downloading the latest version of documents D_M from document servers allows the merged list to precisely reflect current document contents.

Methods which get the most up to date information, lightweight probes and document download, both require extra communication at query time. When implementing a broker, a tradeoff must be made between up to date information and efficient operation.

2.3 Solutions

2.3.1 Server selection methods

Many server selection methods are in fact methods for server ranking. A server ranking method ranks servers S with respect to the query q. After ranking, some server thresholding method may be applied — for example taking the top ten ranked servers — to select servers S'. Server ranking methods described in detail here are CORI, CVV, bGlOSS, vGlOSS and lightweight probes. These methods were originally intended to rank cooperating servers which export ranking information (see Table 2.1), although probe queries (see Chapter 3) allow ranking of non-cooperating servers. Server selection methods which do not rank servers include query clustering and modelling relevant document distributions. These instead predict the optimum number of documents to take from each server. Other non-ranking methods are based on Boolean matching against manually assigned meta-data such as that used by InvisibleWeb (`http://www.invisibleweb.com/`). All these methods are described below.

CORI

The CORI [Callan et al. 1995] method ranks search servers as document surrogates, consisting of the concatenation of the server's documents. Using the formulation from the 1995 paper, the belief $p(r_1, \ldots, r_M|s_i)$ in server s_i due to observing terms $r_1, \ldots, r_M$ is determined by:

$$\begin{aligned} T &= d_t + (1 - d_t) \cdot \frac{\log(DF_{i,k} + 0.5)}{\log(DF_i^{max} + 1.0)} \\ I &= \frac{\log\left(\frac{|S|+0.5}{SF_k}\right)}{\log(|S| + 1.0)} \\ p(t_k|s_i) &= d_b + (1 - d_b) \cdot T \cdot I \end{aligned}$$

	Required Information				
Method	*df*	*Size*	*Vector*	*Co/prox*	*Manual*
CORI	o				
Cue validity variance	o				
bGlOSS	o	o			
vGlOSS	o		o		
Lightweight probes	o			o	
Manual/semi-manual methods					o

Table 2.1: Information required for server ranking. Each server ranking method is based on some information which the broker must extract from servers, such as server *df* document frequencies, *Size* number of documents indexed, *Vector* document vector information, *Co/prox* term co-occurrence and proximity information or *Manual* manual input such as relevance judgments or server content summaries. Extraction of the first four without explicit server cooperation is now possible using probe queries, described in Section 3.2.

Notation	Meaning
s_i	The ith search server
S	The set of servers known to a search broker $(s_1, s_2, \ldots)$
s'_i	The ith selected search server
S'	Subset of S selected to answer a query $(s'_1, s'_2, \ldots)$
t_k	The kth query term in q
q	A query asked of the broker $(t_1, t_2, \ldots)$
Q	The number of unique query terms
R_i	The results of s'_i in response to a query q
R_M	The broker's merged search results
$DF_{i,k}$	The number of documents in s_i containing t_k
DF_i^{max}	The maximum DF of any term in s_i
$DFco_{i,k,l}$	The number of documents indexed by s_i which contain both t_k and t_l
$DFprox_{i,k,l}$	The number of documents indexed by s_i in which t_k and t_l occur within a certain proximity of each other
SF_k	The number of servers $s_i \in S$ where $DF_{i,k} > 0$
SS_i	The number of documents indexed by s_i
$TF_{i,k}$	The number of times term t_k occurs in server s_i
TF_j^q	The number of times term t_j occurs in the current query q
$\|V_i\|$	The vocabulary size of server s_i

Table 2.2: **Notation.** Notation used throughout this thesis.

$$p(t_1, \ldots, t_Q|s_i) \quad = \quad \frac{\sum_{k=1}^{Q} p(t_k|s_i)}{Q} \tag{2.1}$$

$DF_{i,k}$ is a document frequency statistic, the number of documents in server s_i containing term t_k. DF_i^{max} is the maximum DF of any term in s_i. $|S|$ is the total number of servers and SF_k is a server-frequency statistic, the number of servers with one or more documents containing term t_k. Q is the number of unique query terms. Servers are ranked in descending order of belief. This notation is also described in Table 2.2. Experiments in this thesis use the same default term frequency value (d_t) and default belief (d_b) as Callan, Lu, and Croft [1995], setting both to 0.4. It also uses the #SUM operator, which is one of the supported CORI operators.

The formula is a modification of an INQUERY document ranking formula, using DF instead of TF (term frequency, the number of times a term occurs in a document) and SF instead of DF. To move far from the default belief d_b, a term needs a high document frequency with respect to the maximum document frequency and a low server frequency with respect to the number of servers. For example, server promise scores are not calculated using CORI in Chapter 6, because high server frequency statistics lead to very little variation from d_b.

bGlOSS and vGlOSS

Gravano and Garcia-Molina proposed the text-source discovery algorithms bGlOSS [Gravano et al. 1994] and vGlOSS [Gravano et al. 1999] (vGlOSS was first described as gGlOSS in [Gravano and Garcia-Molina 1996]). The bGlOSS method ranks servers which each run a Boolean retrieval system, using document frequency and server size information and according to some assumption about the distribution of query terms in documents. For example, assuming that query term distributions are independent, it is possible to estimate the number of documents matching a two term conjunctive query as follows. Given a server with 1500 documents, with 200 containing term A and 50 containing term B, the estimated number of documents satisfying the query $A \wedge B$ is $\frac{200}{1500} \times \frac{50}{1500} \times 1500 = 6\frac{2}{3}$. bGlOSS then ranks servers in descending order of estimated number of relevant documents. Gravano and Garcia-Molina admitted the assumption of independence is questionable, but claimed good experimental results using it.

The vGlOSS [Gravano and Garcia-Molina 1996] server ranking methods $Max(l)$ and $Sum(l)$ rank servers whose retrieval systems are based on a vector space model. To understand the method it is first necessary to give a brief overview of vector space retrieval. Each dimension in a simple vector space for document retrieval corresponds to a term which occurs in the document collection being searched. A document d_j may then be represented as a vector in that space, with its weight in a dimension $wt_{j,k}$ proportional to the term frequency of the corresponding term t_k in the document and perhaps inversely proportional to the document frequency of term t_k in the collection. Document vectors in the space are very sparse, containing non-zero weights for only a fraction of the collection's total vocabulary. To perform retrieval, the query is cre-

ated as a vector in the document space and documents with the closest match, for example using vector inner product, are ranked highest. Document vectors are often normalised to retrieve documents of all lengths, rather than favour long documents which may have greater weights due to verbosity rather than greater query match.

vGlOSS selection is based on the vector sum of the server's normalised document vectors as well as its document frequency statistics. The vector sum for server s_i and term t_k is:

$$cwt_{i,k} = \sum_{j=0}^{j<SS_i} wt_{j,k}$$

where $wt_{j,k}$ is the weight of term t_k in the normalised document vector representing document d_j of server s_i. So if two documents on server s_n contain t_m ="craswell", and the documents have t_m weights 0.01 and 0.07, then $cwt_{n,m} = 0.08$.

$Max(l)$ and $Sum(l)$ estimate summed scores of all documents which score above l, for a particular query and server. They are based on a high correlation scenario and a disjoint scenario respectively (see Figure 2.5). Both are estimates, based exported weights and document frequency statistics, of $Ideal(l)$ the actual summed document scores for documents scoring above l.

The following examples use sample statistics from Table 2.3. Each estimator assumes weights are spread equally across all documents containing a term, so term A contributes a score of $40 \div 200 = 0.2$ to each of 200 documents, B contributes 0.1 to 50 documents and C contributes 0.25 to 20.

Under the high correlation scenario, $Max(l)$, terms occur together as often as possible. $Max(l)$ estimates that 20 documents contain query terms A, B and C, and each document has a score of 0.55. A further 30 documents contain only A and B with a score of 0.3, and 150 documents only contain A scoring 0.2. If the threshold l for inclusion in the merit sum is 0.25, an estimated 50 documents score above the threshold, and the merit sum is $Max(0.25) = 20 \times 0.55 + 30 \times 0.3 = 20$. After calculating scores for other servers in S, the broker then ranks in descending score order.

Under the disjoint scenario $Sum(l)$, terms occur in different documents where possible. With a threshold of $l = 0.25$, the documents containing only A or only B do not qualify, scoring only 0.2 and 0.1 respectively. The documents containing C do meet the score threshold, and vGlOSS estimates the merit of the server to be the total scores of those documents $20 \times 0.25 = 5$. Servers are again ranked in descending order of estimated merit.

For $l = 0$, all three produce the same ranking, so $Max(0) = Sum(0) = Ideal(0)$. The score S_i achieved by a server s_i when $l = 0$ is:

$$S_i = \sum_{j=1}^{Q} cwt_{i,j} \tag{2.2}$$

This formulation is used in Chapter 5, and was also used by French, Powell, Viles, Emmitt, and Prey [1998] and French, Powell, Callan, Viles, Emmitt, Prey, and Mou

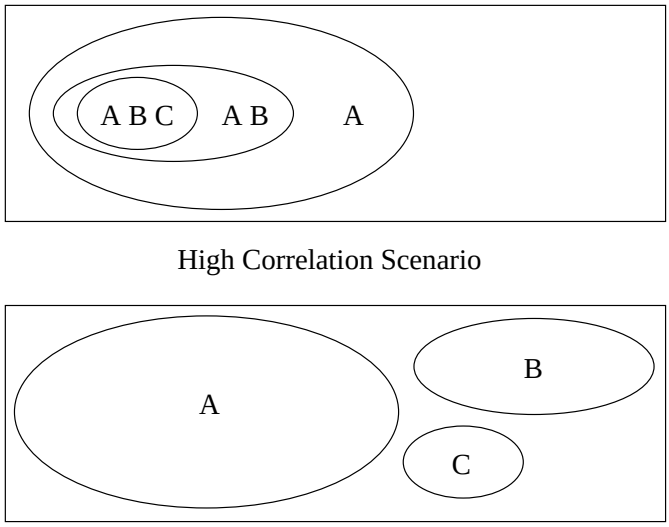


Figure 2.5: vGlOSS scenarios. Under the high correlation scenario, query terms occur together in documents as often as possible, so documents containing term *C* also contain *A* and *B*. Under the disjoint scenario, terms do not occur together where possible, so sets have minimum overlap.

t_k	$DF_{i,k}$	$cwt_{i,k}$
A	200	40
B	50	5
C	20	5

Table 2.3: vGlOSS example statistics. Example for a server s_i. t_k is the term. $DF_{i,k}$ is the document frequency of t_k in s_i. $cwt_{i,k}$ is the sum of weights for t_k over all documents in s_i.

[1999], although under a different evaluation framework.

CVV

The CVV [Yuwono and Lee 1997] server ranking method is based on the Cue validity variance (CVV) of query terms. Terms which can better discriminate between servers have a higher CVV and therefore contribute more to the suitability score S_i:

$$\begin{aligned} CV_{i,j} &= \frac{\frac{DF_{i,j}}{SS_i}}{\frac{DF_{i,j}}{SS_i} + \frac{\sum_{k \neq i}^{|S|} DF_{k,j}}{\sum_{k \neq i}^{|S|} SS_k}} \\ CVV_j &= \frac{\sum_{i=1}^{|S|} (CV_{i,j} - \overline{CV_j})^2}{|S|} \\ S_i &= \sum_{j=1}^{Q} CVV_j \cdot DF_{i,j} \end{aligned} \tag{2.3}$$

SS_i is the size, in documents, of server s_i. $\overline{CV_j}$ is the population mean of $CV_{i,j}$ over all servers. Servers are ranked in decreasing order of scores S_i. A term needs high CVV to influence selection, which corresponds to a high variation in DF. The formula lacks server size normalisation. For example, if two servers have DF of ten for a given term it does not matter if one server indexes a total of ten documents and the other ten million, they will have the same S_i. This means that CVV selection tends to favour large servers. Evaluation shows that CVV selection improves if size variation is masked (see Chapter 5).

Yuwono and Lee also proposed two other server ranking methods, but described them in less detail than CVV. The $df \cdot isf$ method prefers servers which have a high document frequency (df) for a query term which is not present in many search servers (inverse server frequency isf). The score of a server S_i is:

$$S_i = \sum_{j=1}^{Q} DF_{i,j} \cdot \log\left(\frac{|S|}{SF_j}\right) \tag{2.4}$$

using notation described previously.

The cluster centroid selection method represents a server's documents in a vector space and the server itself as the centroid of those vectors. Server ranking is then based on the inner products of the centroid vectors to the query vector, analogously to document vector ranking.

Lightweight probes

Hawking and Thistlewaite [1999] perform server selection using lightweight probes. Given a large query q the search broker generates a two term probe query and sends it to servers S. Probe results from server S_i for terms t_k and t_l include:

$DF_{i,k}$	Document frequency of t_k in documents indexed by s_i,
$DF_{i,l}$	Document frequency of t_l in documents indexed by s_i,
$DFco_{i,k,l}$	The number of documents indexed by s_i which contain both t_k and t_l, and
$DFprox_{i,k,l}$	The number of documents indexed by s_i in which t_k and t_l occur within a certain proximity of each other.

These results are combined in a linear function to generate a server score S_i:

$$S_i = c_1 DF_{i,k} + c_2 DF_{i,l} + c_3 DFco_{i,k,l} + c_4 DFprox_{i,k,l} \quad (2.5)$$

Values of parameters $c_1 \ldots c_4$ were set after experimentation over a training set of documents, with $0 \leq c_1 \leq 1$, $0 \leq c_2 \leq 1$, $c_3 = 10$ and $c_4 = 100$. Probes were automatically generated using 1-3 word queries, taken from TREC topic titles (TREC is described in Section 2.4.1). For single term titles the term was repeated twice. For titles with more than two terms, a set of *DF* statistics was taken from a separate document set, and the two terms with *DF* closest to some ideal value were used in the probe. Servers are sorted in descending order of score S_i.

This thesis proposes a system of probe queries (Section 3.2) which is quite different from Hawking and Thistlewaite's lightweight probes. Lightweight probes are performed at query time, and require servers to cooperate by returning term occurrence statistics. By contrast, probe queries are performed before query time and are compatible with any server capable of returning a results list.

Query clustering and modelling relevant document distributions

Voorhees [1995] proposed two server selection methods, each based on relevance judgments of the top 100 merged results for each of several training queries. Rather than ranking servers, the methods predict the optimum number of results to take from each server, making S' the set of all servers from which one or more results are retrieved. The methods allow a broker to address any search server, since they do not require servers to cooperate by exporting meta-data. Voorhees described methods which do not require cooperation as "isolated", and those which do as "integrated". The selection methods are the query clustering method and the modelling relevant document distributions (MRDD) method.

In the Query Clustering method, the broker represents training queries for which relevance judgments are available in a vector space. It then clusters query vectors and calculates cluster centroids. It creates the current query q in the same vector space and finds the closest cluster centroid. The number of documents retrieved from a server is then proportional to its average number relevant for training queries in that cluster.

In the MRDD method, the broker finds the K training query vectors closest to the current query's vector. It then calculates for each server, the number of relevant documents returned at each point in a ranking, on average over the K results lists. For example, at rank 10 server s_1 might have on average 2.7 relevant documents while server s_2 might have 2.0. However, server s_2 might have a better average at rank five. MRDD then chooses a level of retrieval from each server that maximises the expected

number of relevant documents.

Other methods

Xu and Callan [1998] suggested the broker could improve CORI selection based on phrase occurrences in documents or using query expansion. Xu and Croft [1999] suggested methods for manufacturing topic skew — assigning documents to search servers so that relevant documents for a given topic tend to be concentrated at one or a few search servers. Natural topic skew does exist in Web servers, see Section 5.1.1, but central control over which documents are indexed by which server is not realistic on the Web. However, Xu and Croft did improve on the effectiveness of CORI selection using the Kullback-Leibler divergence measure to determine how well a topic language model predicts the current query q.

Baumgarten proposed a probabilistic model for distributed information retrieval over a multi-layer hierarchy of servers and brokers [Baumgarten 1997; Baumgarten 1999], which includes server selection. Under the model, brokers select servers so as to maintain the top l merged results, without querying servers which would not contribute to the top l. The selection method, is based on the broker's estimation of server retrieval status value distributions. Estimates are in turn based on statistics exported by cooperating servers.

Zobel [1997] proposed a number of server ranking methods based on collection statistics provided by cooperating servers. The most effective was an inner product similar to Yuwono and Lee's $df \cdot isf$ method. D'Souza, Thom, and Zobel [2000] suggested an n-term indexing scheme, where the broker keeps an index of the distributed documents based on a limited term vocabulary. Because of the smaller vocabulary, the n-term index is smaller than a full central index. A different approach is for the broker to keep the full vocabulary but index documents in groups of 10 [de Kretser et al. 1998]. This central index is half the size of a per-document index. The broker identifies the top ranked 100 or 1000 groups using the index, then queries the appropriate servers, listing the document identifiers of interest. This document identifier list allows each selected server to process the query by consulting only a fraction of its index.

The Web search broker SavvySearch [Dreilinger and Howe 1997] performs server selection based on past "visit" and "no results" events. If the current query contains a term t_k and the user visits a results document, the future selection score of the server which returned the result is boosted, for queries containing t_k. If a server returns no results for a query containing t_k its future selection score is reduced for queries containing t_k. The Web search broker ProFusion also performs selection based on user visits to results pages and query categorisation [Fan and Gauch 1999]. Since visit events are not available with standard test collections, these methods have not yet been included in large-scale evaluation and are not evaluated here.

A number of selection methods based on server classification have also been suggested. CiteLine Professional (`http://www.caredata.com/`) and InvisibleWeb (`http://www.invisibleweb.com/`) categorise and summarise servers manually. Selection is

then based on the presence of query terms in category names and summaries. It is a Boolean match, servers are not ranked. Chakravarthy and Haase [1995] classified servers using WordNet [Miller 1995] style structures. Kirk, Levy, Sagiv, and Srivastava [1995] proposed use of knowledge representation technology for characterising servers and queries, along with methods for matching the two. Dolin, Agrawal, Dillon, and El Abbadi [1996] used Library of Congress hierarchical meta-data descriptions in their system Pharos.

Server selection methods based on server categorisation are also not evaluated here. Those which rely on manual classification can not be implemented because documents used (the WT2g TREC test collection described later) have not been manually classified. Due to time constraints even methods based on automatic classification, based on manually derived training data, such as Pharos [Dolin et al. 1999] are not evaluated here.

2.3.2 Results merging methods

A results merging method gathers information about documents D_M and then generates a ranking o_M with respect to query q. The two broad classes of merging method are cooperative merging methods, which rely on cooperation from search servers, and generally applicable merging methods, which are based on information such as server-assigned document ranks. This section describes methods in both classes.

Note, no merging method is guaranteed to produce precisely the same results as would a centralised search over the same documents. This is because only documents returned by the selected servers make it into the merged list. If a server is not selected, a top ranking document might be missed. If each selected server's top ten results are merged, a document might be missed because it is ranked at eleven or greater. Missing a document which would be highly ranked in the merged list becomes more likely if a server uses different collection statistics or document ranking algorithms than are used in merging.

Merging with server cooperation

With the cooperation of servers, a broker can rank documents R_M in the same order as they would be by a non-distributed system. Such *consistent* merging has been known as perfect merging [Hawking and Thistlewaite 1999] and merging with normalised scores [Callan et al. 1995]. It orders the small document set R_M based on a single ranking algorithm and on consistent collection statistics. These statistics may be collected from all the broker's servers S or servers selected for the current query S'.

One method for achieving a consistent merged ranking, the comparable scores method, has two requirements:

1. Servers all generate document relevance scores using the same retrieval algorithms (document ranking algorithm, stemming method, stopwords list) and return these scores to the broker.

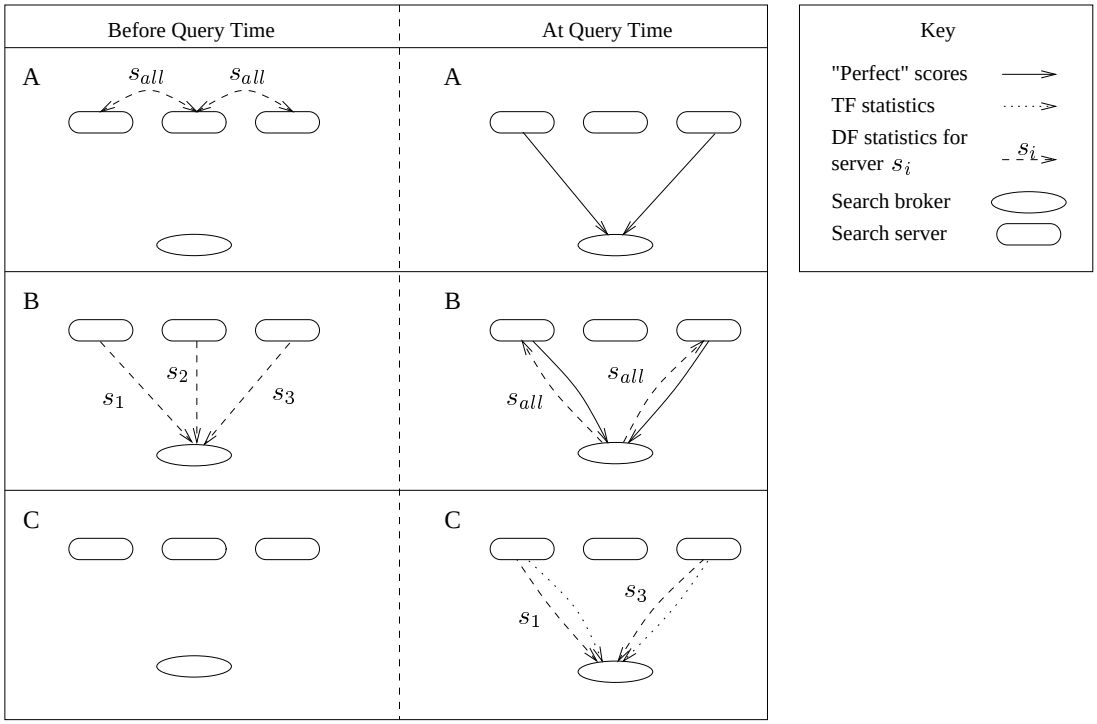


Figure 2.6: Propagating statistics for merging Under scheme A [Viles and French 1995], servers propagate their document frequency (DF) statistics amongst themselves before query time, then provide results with comparable scores for merging. Under scheme B [Callan et al. 1995], servers also return comparable scores, but document frequency statistics are collated at the broker and supplied with the query. Under scheme C [Kirsch 1997] there is no communication before query time. Instead the servers return the necessary term frequency and document frequency statistics with their results, and the broker uses the statistics for merging. Notice that consistent scores in A and B are based on DF statistics for s_{all} while those in C are based only on statistics from s_1 and s_3. This is indicative of the problem of defining the "collection" when collating collection statistics (addressed in Appendix A).

2. If the ranking algorithm requires collection statistics, as the most effective algorithms do [Harman 1992], all servers use the same collection statistics. This consistency may be achieved by either:

   - propagating and aggregating statistics between the servers S before query time [Viles and French 1995], or
   - aggregating statistics at the broker before query time, and then at query time providing servers with statistics on query terms [Callan et al. 1995; de Kretser et al. 1998; Mazur 1994].

   For an illustration of the communication required for such aggregation, see Figure 2.6.

If the above requirements are met, servers will return comparable scores and the broker merges by simply sorting documents in order of descending score.

The other method which produces a consistent R_M, the statistics provision method [Walczuch et al. 1994; Gravano et al. 1997; Kirsch 1997], ranks documents with respect to collection statistics collated from servers S' at query time. The servers return statistics pertaining to q with their results. The broker uses these statistics to generate a consistent ranking. For example, if the broker's merging is based on a simple $tf \cdot idf$ document ranking, each server s'_i would return with its results R_i query-term frequency statistics for documents D_i and query-term document frequency statistics for its entire index. By summing document frequency statistics and using term frequency statistics, the broker may then generate each document's $tf \cdot idf$ score. The merging is again consistent because documents D_M are ranked in the same order as they would be by a single index, except this time covering the documents of servers S' and servers need not all run the same retrieval system, because the consistent scores are generated by the broker.

Baumgarten's probabilistic model for distributed information retrieval [Baumgarten 1997; Baumgarten 1999] suggests a different merging method, also based on server cooperation. Under the model, search servers must export statistics and employ some probabilistic ranking algorithm. The broker modifies document retrieval status values from various servers according to server information, to obtain a merged ranking consistent with the probability ranking principle [Robertson and Sparck Jones 1976].

Generally applicable merging methods

On the Web, where lack of server cooperation makes cooperative merging unfeasible, brokers must employ different merging methods. Rank based merging methods use the ordinal ranks returned by servers to generate o_M. Ranks may be simply interleaved [Callan et al. 1995; Smeaton and Crimmins 1996] such that the merged ranking consists of all documents of rank one, followed by all documents of rank two, and so on. Documents which appear in multiple results lists can have their rank improved under this scheme. Alternatively the gap between a server's documents in R_M may be

made inversely proportional to server promise (server selection score) [Yuwono and Lee 1997]. This is similar to a method by Voorhees [1995], which chooses documents from each incoming list using a weighted $|S'|$ sided die. On that die, weights begin at levels proportional to server promise, then decrease to zero as documents are chosen.

Score based merging methods use the relevance scores returned by servers. Note, scores from different servers may be non-comparable due to differences in ranking algorithms or collection statistics. When ranking algorithms differ, the magnitude of scores may vary, for example ranging between $0-1$, $0-100$ or with no fixed maximum. Even using the same ranking algorithm, different servers return non-comparable scores because their scores are based on local collection statistics (rather than statistics for all servers S' or S as in cooperative merging). The solution suggested by Selberg and Etzioni [1997] is to scale the scores from each server to range between two set values, then sort by scaled scores. A further refinement is to weight scaled scores more highly if they originate from a server with a higher server promise. For example by multiplying the scaled score by the server promise score [Gauch et al. 1996]. Callan, Lu, and Croft [1995] did this, but did not need to scale the scores first, because they were already based on a uniform document ranking (scoring) algorithm.

Content based merging methods may be applied in situations where document servers allow the broker to download document full texts. The Inquirus [Lawrence and Giles 1998] search broker downloads documents, then employs the following document ranking algorithm:

$$R \;=\; c_1 N_p \;+\; \left(c_2 - \frac{\sum_{i=1}^{N_p-1} \sum_{j=i+1}^{N_p} min(d(i,j), c_2)}{\sum_{k=1}^{N_p-1} (N_p - k)} \right) \Big/ \frac{c_2}{c_1} \;+\; \frac{N_t}{c_3} \qquad (2.6)$$

where R is the document's score, N_p is the number of query terms present in the document (each term is counted once), N_t is the number of query term occurrences in the document, $d(i,j)$ is the minimum distance (number of characters) between the ith and jth query terms, and c_1, c_2 and c_3 are constants. Inquirus did not use a ranking algorithm of proven effectiveness, because such algorithms require collection statistics, which are unavailable from Web search servers.

2.3.3 Search brokers

This section gives examples of currently operating search brokers of various types, emphasising their differences. It then describes current brokers which give a hint of the next generation.

Example brokers

Brokers which run on a central server, to which a user connects with a Web browser, are called central brokers. Those which run on the user's computer are called client brokers. General brokers are those which search search servers such as AltaVista (`http://www.altavista.com/`) which cover documents from many sources. Other

brokers have a narrow focus. For example, a shopping broker searches shopping servers and a medical search broker searches medical servers.

- Some central, general search brokers have very basic merging and selection, such as DogPile (`http://www.dogpile.com/`). DogPile always queries all its servers then presents a concatenation of their results lists.
- Central, general brokers with merging but selection $S' = S$ include MetaCrawler [Selberg and Etzioni 1997], C4 (`http://www.c4.com/`), Mamma (`http://www.mamma.com/`) and InferenceFind (`http://www.ifind.com/`). InferenceFind also performs clustering on its results.
- Central, general brokers with merging and some manual server selection include Inquirus [Lawrence and Giles 1998] and The BigHub (`http://www.thebighub.com/`).
- Central, general brokers with merging and selection include SavvySearch [Dreilinger and Howe 1997] and ProFusion [Gauch et al. 1996].
- At least one central, general broker without automatic retrieval or merging but with selection over many search servers, is available on the Web: InvisibleWeb.com (`http://www.invisibleweb.com/`).
- Central shopping search brokers include eBoodle (`http://www.eboodle.com/`) and DealPilot.com (`http://www.dealpilot.com/`). These allow product searches across multiple shopping search servers.
- Client, general brokers include Connectix Surf Express Deluxe (`http://www.connectix.com/`), SearchBuddy (`http://www.searchbuddy.com/`) and Fusion [Smeaton and Crimmins 1996].
- One client medical search broker, CiteLine Professional (`http://caredata.com/`), performs selection based on manually generated meta-data but no concurrent retrieval or merging.

The remainder of the section notes interesting features of Inquirus and of server selection brokers CiteLine Professional and InvisibleWeb.com.

Next generation brokers

Inquirus [Lawrence and Giles 1998] runs on a server at NEC Research Institute at Princeton. It has manual selection amongst a number of fixed server subsets: Web, News, Images and Web and News. For example, the Web set has 17 servers.[1] Merging is based on downloaded document contents and a novel document ranking algorithm (Equation 2.6). Inquirus is intended to search more of the Web than any single

[1] Inquirus Web search servers: HotBot, Direct Hit, Google, Infoseek, AltaVista, Excite, Northern Light, Snap, Fast, Voila, WebTop , SearchEdu, Open Directory, Thunderstone, Yahoo Inktomi, EuroSeek and Yahoo.

server can. The indexable Web includes more than one billion non-duplicate indexable documents (`http://www.inktomi.com/webmap/`). Since large Web search servers have surprisingly low overlap, Inquirus covers more pages by combining the results of multiple engines [Lawrence and Giles 1999].

CiteLine Professional (`http://www.caredata.com/`) specialises in medical data, with $|S| > 2000$ specialist medical search servers. InvisibleWeb.com (`http://www.invisibleweb.com/`) covers a broader range of search servers, with $|S| > 10000$. Neither broker is capable of automatic retrieval or merging, but both perform selection based on manually assigned meta-data. Examples of coverage include the PubMed (referenced previously), Internet Movie Database (`http://imdb.com/`) and New York Times (`http://www.nytimes.com/`) search servers. The search servers often have a comprehensive and up to date index, which covers a single document source.

This thesis explores the idea of building a generally applicable search broker similar to Inquirus and incorporating highly effective selection and merging methods. To that end later chapters introduce and evaluate methods for effective Web merging and selection which are extensions to Inquirus. The broker would cover search servers such as those listed by CiteLine Professional and InvisibleWeb. The number of servers $|S|$ would depend on the application. In some cases a broker over ten or one hundred specialist servers would be useful. In any case, the evaluation experiments model single source, heterogeneous search servers such as those which exist on the Web.

2.4 Published evaluation experiments

This section first describes general effectiveness evaluation methodology in information retrieval. Then it summarises previously published effectiveness evaluation experiments in server selection and results merging.

2.4.1 Overview of information retrieval evaluation

Many systems provide a search service by responding to an incoming query with a list of search results, usually ranked in order of likely relevance to the query. These include research systems [Voorhees and Harman 1998], search servers (as described in Section 2.2) and search brokers. The goal of any search service is to return results which satisfy the user, and because the goal is the same in both the distributed and non-distributed case, similar experimental methods may be used in each. For this reason this section now discusses the two main designs for system evaluation: the user study and evaluation using a test collection.

In a *user study*, users spend time interacting with different search systems. During the interaction the experimenter records data such as the user's behaviour, the user's stated levels of satisfaction, system inputs, system outputs and time taken. Then, through analysis of this data, the experimenter draws conclusions about the relative effectiveness of different systems. Although a user study experiment closely

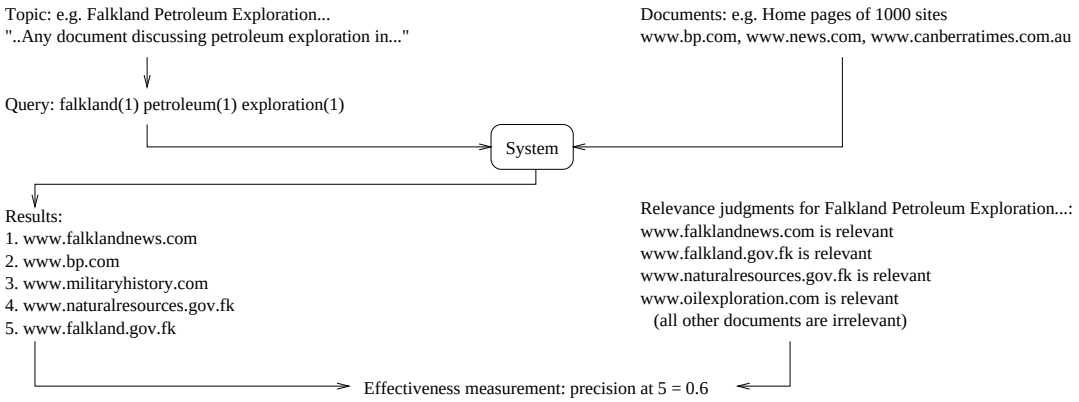


Figure 2.7: Test collection example. For a given topic, in this case one about offshore oil exploration near the Falklands, a query is generated in the query language of the system. Given the query and test collection documents, a system produces a ranked results list. Using relevance judgments — judgments are based on whether a human relevance assessor finds each document relevant — the system's effectiveness can be measured. In this case its precision at five documents, the proportion of the top five judged relevant, is $\frac{3}{5} = 0.6$.

models real interaction between user and system — differing only when experimental requirements influence the user's behaviour — it has some disadvantages. Every user is different and user needs change over time, so it is impossible to give two or more systems exactly the same task. In the act of querying one system, the user's state of knowledge changes. This in turn changes their future needs, queries and levels of satisfaction with future results. Consequently the experimenter must record enough data in a counterbalanced experimental design, allowing statistical analysis to reveal general trends in effectiveness. Getting enough data can require a large experiment. Also, to test a new facet of a system, the whole experiment should be repeated even if the baseline system stays the same. Because user studies involve so much user input and start each time from scratch, user study methodology is not applied in this thesis.

A *test collection* in this context is an abstract laboratory model — comprising documents, topics and relevance judgments — of a user's behaviour when searching. Each topic is a fixed statement of a user information need, usually in natural language, which is used both as a basis for generating queries and for judging the relevance of documents. For each topic, a relevance assessor views documents to judge whether they are relevant. Real information needs change over time, but the topic and judgments form a fixed picture of user preferences. Using this, the experimenter may evaluate a system's effectiveness by: (1) generating a query based on the topic (query generation may be automatic or manual), (2) querying the system over the test collection's documents and (3) measuring the effectiveness of the system in returning judged-relevant documents in high ranks. The whole process is illustrated in Figure 2.7.

Commonly employed effectiveness measures are *precision* at N, the proportion of the top N documents judged relevant, and *recall* at N, the proportion of all judged-relevant documents in the top N. The experimenter should only use the latter measure if sufficient relevance judgments have been made to identify practically all relevant documents (see the description of pooling below). For applications where both recall and precision matter, it is common to graph one against the other, to see how precision degrades as recall increases. A single-number measure commonly used in research is *average precision*, the area under the non-interpolated precision-recall graph. In experiments concerning Web data, it is becoming more common to focus on early precision [Hawking et al. 1999] rather than precision and recall. The reason for this is that users searching the Web seldom look at a large number of documents for a single query [Spink et al. 1999].

A test collection is an abstraction. To model a single user's behaviour for a single topic, input from as many as three individuals may be required: for topic generation, manual query generation and relevance judging. The topic description and relevance judgments are fixed despite the fact that a real user's needs constantly change, even during their viewing of a results list. However, this simplifying assumption of unchanging information need allows two or more systems to be evaluated on precisely the same task. In addition, once a test collection has been created it may be reused in a number of different experiments, which require no new user data, as long as the experimenter is confident that practically all potentially relevant documents have been judged. Because of the latter condition, only test collections with sufficiently complete relevance judgments are used in this thesis.

A major expense in creating a test collection is in paying relevance assessors for their time in making judgments. In the first test collections, all documents in the collection were judged with respect to each topic, for example see work from 1967 reprinted as [Cleverdon 1997]. Although this guaranteed the identification of all documents that the assessors found relevant, the cost of the judging limited the number of documents in test collections. Since then newer and larger test collections have been built using the *pooling* method, notably in the context of the NIST Text REtrieval Conferences (TREC) [Voorhees and Harman 1999]. Pooling involves combining top results from many systems to build an assessment pool. If the pool contains enough documents and the systems have been successful in returning documents which are likely to be relevant, it may be assumed that the pool contains all relevant documents. By judging only documents in the pool, assessors can identify most relevant documents while judging a fraction of the available documents. In TREC, several two gigabyte test collections have been built with complete judgments, and documents have already been collated for a ten gigabyte Web-data collection with complete judgments.

In summary, an existing test collection with complete judgments may be used in evaluation without any new user input. Evaluation by user study or use of a collection without complete judgments requires new user input. Because judging resources are scarce and TREC test collections are adequate for the task, this thesis performs evaluation using a two gigabyte TREC test collection with Web documents and two TREC test collections each containing two gigabytes of news and government docu-

ments. Each test collection has sufficient judgments, through pooling, for repeated experiments with no new judging.

2.4.2 Selection evaluation

There are two main models for evaluating the effectiveness of server selection methods. *System level evaluation* includes the whole process depicted in Figure 2.4: selection, retrieval and merging. An effective selection is one which provides effective results R_M to the user, for a given server configuration and merging method. In the other model, *server merit evaluation*, each server is assigned a "merit" value. Server merit evaluation measures the proportion of all available merit captured in selected or top ranked servers. Server merit evaluation is always based on some merit baseline, which captures the notion of "server goodness".

Baselines in server merit evaluation

Gravano and Garcia-Molina [1996] and Yuwono and Lee [1997] defined the merit of a server using $Ideal(l)$ (Equation 2.2), being the summed scores of documents scoring above l for the current query. A better server under $Ideal(l)$ is one which has more documents scoring above l for the query or the same number with higher scores.

Rather than tying merit to a single document scoring algorithm, as does $Ideal(l)$, Zobel [1997] defined merit according to multiple algorithms. He defined desirable documents as those returned in the top 100 by at least one of a number of retrieval systems. Under this definition, a better server is one which indexes a greater number of desirable documents.

Callan, Lu, and Croft [1995], Hawking and Thistlewaite [1999] and French, Powell, Viles, Emmitt, and Prey [1998] defined server merit according to the number of documents indexed which are relevant to the current query. Under this relevance baseline, a better server is one which indexes more relevant documents. This baseline requires a test collection in which practically all relevant documents have been identified.

Measures in server merit evaluation

In system level evaluation, effectiveness of R_M may be measured using precision, recall or average precision measures described previously. However, server merit evaluation requires different measures. Callan, Lu, and Croft [1995] built an ideal ranking, in descending order of merit, and used mean squared errors between ideal and calculated ranks as an effectiveness measure:

$$MSE = \frac{\sum_{i=0}^{i<|S|}(O_i - R_i)^2}{|S|}$$

where O_i is the ideal rank of server s_i and R_i is the rank assigned to s_i by the server selection method. Mean squared errors vary arbitrarily if multiple servers have equivalent merit, since in such cases O_i can also vary arbitrarily. For example, if the top three

servers have identical merit, any one of them might have the ideal rank of one, but the choice of which $O_i = 1$ influences MSE.

Gravano and Garcia-Molina [1996] proposed recall and precision analogues, for use in selection evaluation.

$\mathcal{R}_n$ The proportion of merit captured by the top n servers.

$\mathcal{P}_n$ The proportion of servers in the top n which have non-zero merit.

French, Powell, Viles, Emmitt, and Prey [1998] proposed additional measures:

$\mathcal{R}_*$ The proportion of merit captured by the top n^* servers.

$\mathcal{P}_*$ The proportion of servers in the top n^* which have non-zero merit.

They defined n^* as the number of servers known to have non-zero merit. The following descriptions of selection evaluation experiments indicate which measures were used.

Evaluation experiments

Callan, Lu, and Croft [1995] performed both system level and server merit evaluation, using a relevance based merit definition and MSE. However, they did not compare CORI selection to any other selection method. Xu and Callan [1998] followed up on this, improving effectiveness of CORI selection using phrase occurrence information and, in particular, query expansion.

Xu and Croft [1999] compared CORI and Kullback-Leibler selection, finding the latter to be more effective. They used system level evaluation, dividing TREC documents into 100 search servers, each of which uses INQUERY retrieval with global collection statistics allowing consistent merging based on document scores. In the experiment, document assignments to search servers were based in some cases on clustering. As mentioned previously, assigning documents to search servers in this way is unlikely on the Web.

Voorhees and Tong [1997] evaluated query clustering and MRDD selection in combination with weighted Voorhees merging. They performed system level evaluation over five servers corresponding to five TREC subcollections. The servers had heterogeneous retrieval systems, of two different types. Results showed that query clustering and MRDD perform well, regardless of retrieval system. However, best results were obtained using ten search servers, with all combinations of five sub-collections and two retrieval systems.

Gravano and Garcia-Molina [1996] performed server merit evaluation of vGlOSS using the $Ideal(l)$ merit baseline, and $\mathcal{R}_n$ and $\mathcal{P}_n$ measures. They found that $Max(l)$ is a good estimator to balance $\mathcal{R}_n$ and $\mathcal{P}_n$ while $Sum(l)$ maximises $\mathcal{P}_n$. They also found that for $Ideal(0)$, $Max(0)$ and $Sum(0)$ give perfect answers.

Yuwono and Lee [1997] evaluated CVV, centroid vector selection, vGlOSS, CORI, $df \cdot isf$ and bGlOSS selection. They used the evaluation methodology of Gravano and Garcia-Molina, based on the same ideal ranking. They found that CVV was the most

effective selection method, and document centroid selection was the second most effective although only marginally better than vGlOSS. They found CORI, $df \cdot isf$ and bGlOSS selection to be least effective.

Zobel [1997] performed a server merit evaluation of several proposed selection methods. Evaluation was with two separate sets of test data, partitioning TREC disk two into 43 servers and TREC disk three into 91. Two measurement methods were used, one measuring $\mathcal{R}_n$ at every level of n, using a relevance baseline the other using Zobel's new baseline described previously. This was based on the number of server documents which would be returned by one or more of a number of retrieval systems. Results showed the inner product server ranking method to be the most effective. A similar experiment, comparing n-term indexing, CVV and inner product selection, also found inner product to be most effective [D'Souza et al. 2000].

French, Powell, Viles, Emmitt, and Prey [1998] raised concerns over the merit definition suggested by Gravano and Garcia-Molina. They evaluated $Ideal(0)$ server ranking against a relevance baseline and found that it did not match well. In other words, Gravano and Garcia-Molina's ideal servers were not always the ones containing relevant documents. Their testbed split documents from TREC discs 1–3 into 236 search servers. French, Powell, Callan, Viles, Emmitt, Prey, and Mou [1999] followed up on this work, using the same testbed to compare vGlOSS and CORI. They found that CORI selection is best under relevance based merit baseline. They also found that vGlOSS approximates a size based ranking, giving preference to servers with more documents.

Hawking and Thistlewaite [1999] performed both system level and server merit evaluation, in the latter case using a relevance baseline. They compared several realistic and control selection methods including their own lightweight probes method and the query clustering method by Voorhees. Their 98 search servers indexed TREC documents divided first into six by source, such as Associated Press and US Congressional Record, then further divided into chunks of approximately equal size (in bytes). Results showed lightweight probes to be superior to query clustering.

In summary, experiments which compare more than two selection methods have been rare. Experiments on methods from two studies have been performed by Hawking and Thistlewaite [1999], French, Powell, Callan, Viles, Emmitt, Prey, and Mou [1999] and [Xu and Croft 1999]. Given the controversy over merit baselines and concerns raised in Chapter 4, it is not clear how much weight to give to the results. Further, most studies published so far have assumed search server cooperation, possibly including the running of homogeneous retrieval systems. By contrast, Craswell, Bailey, and Hawking [2000] and Chapter 5 compare vGlOSS, CORI, CVV and a modification of CORI. The experiment models an environment where servers do not cooperate and have heterogeneous retrieval systems.

2.4.3 Merging evaluation

Merging evaluation methods are much less controversial, although experiments comparing multiple methods are rare.

Evaluation methods

Merging evaluation is simply based on the quality of the merged list R_M. If relevant documents are ranked highly, the merging method has been effective. This may be evaluated in terms of early precision, for example the precision at 10 of a merged list of 100 indicates whether the merging method has succeeded in promoting relevant documents. It may also be measured using average precision at any point in the merged ranking, since average precision rewards both the presence and the rank of relevant documents.

Evaluation experiments

Callan, Lu, and Croft [1995] compared four merging methods:

- Interleaving ranks,
- Raw scores from INQUERY based on local statistics,
- Comparable scores from INQUERY based on global statistics and
- Weighted scores according to server promise.

Measures were average precision and precision. Precision was measured at various ranks, various levels of recall and point R in the ranking, where R is the number of relevant documents for a query. They found that comparable scores and weighted scores were equivalent and that, since comparable scores require collation of global collection statistics, weighted scores seemed most promising in practice. Raw scores were 10–20% worse, and interleaving was highly ineffective.

Other studies have introduced and evaluated new merging methods without comparing them to other methods. The experiments reported in [Craswell et al. 1999] and again in Chapter 6 are the first to compare a large number of merging methods (all those listed in Section 2.3.2).

2.5 Summary and conclusion

This chapter introduced several key ideas. It described how a search broker interacts with multiple search servers and the user. It noted that selection and merging methods determine the effectiveness of a broker over a given set of servers. It surveyed selection and merging methods, and a number of existing brokers. It also covered evaluation of selection and merging methods.

Prior to this work no generally applicable, fully automatic methods for server selection had been published. The few published merging methods of this type had not been evaluated. Consequently, it has been unclear how to perform selection in a fully automatic, generally applicable search broker. In addition it has been unclear which merging method is best or even whether generally applicable merging methods are effective at all. This thesis addresses both these problems by proposing new methods and evaluating against numerous existing methods.

Chapter 3

New Methods and Hypotheses

This chapter introduces new selection and merging methods, then states hypotheses concerning their effectiveness. These hypotheses are tested in two large effectiveness evaluation experiments in Chapter 5 and 6.

3.1 Using downloaded documents

The new methods assume that the broker downloads and analyses document texts. Since search results consist of document identifiers, not documents, this involves a separate stage after retrieval (see Figure 2.1). Document servers respond to requests from brokers as they would to any other requester. For example, requests from a user's document viewing client or from a search broker downloading documents for indexing (crawling), are no different.

MetaCrawler [Selberg and Etzioni 1995] was the first search broker to use document texts, optionally downloading documents for "verification". Web documents are volatile and a search server's index is always based on document contents at the time of crawling. If documents have changed since the last crawl, it is possible for Web search servers to return the identifiers of documents which no longer match the query or no longer exist. MetaCrawler verification simply removes such bad results from the merged results list.

Inquirus [Lawrence and Giles 1998] makes extensive use of downloaded documents. It performs verification, but also performs results merging based on document texts as described in Section 2.3.2. In addition, from each document it generates a query biased summary to assist users in deciding which documents to view. A query biased summary consists of passages extracted from the document, chosen to include occurrences of query terms [Tombros and Sanderson 1998]. Finally, when a user chooses to view a document, Inquirus intercepts the request and returns its cached copy. The caching might mean the document arrives more quickly, but also ensures the merged ranking is based on precisely the same document that the user sees. In the cached copy, Inquirus also highlights occurrences of query terms, to help the user in finding the relevant portion of the document.

Although one operational broker uses document texts for results merging, none use them for server selection. However, server selection methods which use document

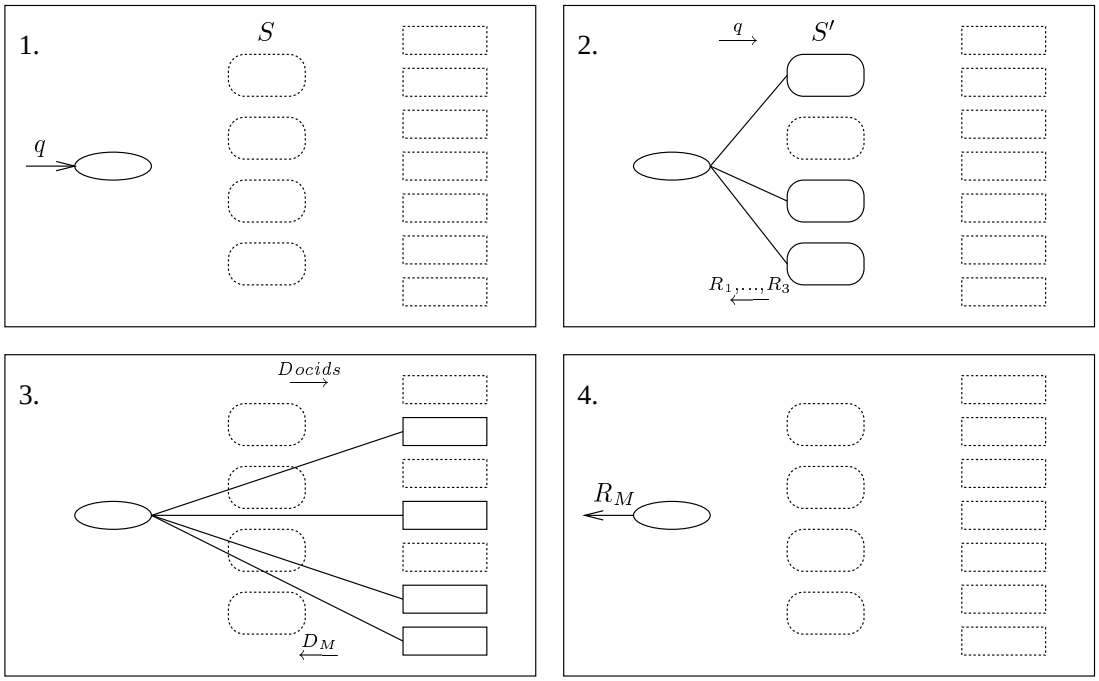


Figure 3.1: Search broker network communication with document download. Document download adds another stage of network communication to that depicted in Figure 2.3. (1) The user queries the search broker. (2) The broker queries selected search servers S' to obtain their search results $R_1, \ldots, R_3$. (3) The broker requests the full-texts of documents D_M from the appropriate document servers. (4) The broker uses document texts for verification, summarisation, caching or merging, and returns merged results to the user.

texts have been proposed. Query clustering and MRDD [Voorhees 1995] applied in this context would rely on training query results documents being downloaded and judged (see Section 2.3.1).

A study by Callan, Connell, and Du [1999] suggests query based sampling of search servers. This involves constructing queries and sending them to all servers *S*, then downloading the resulting documents. The broker can then perform selection based on statistics from the sampled documents. Documents sampled this way from a server are a non-random sample of its full document coverage. However, the study found that document frequency statistics from a few hundred downloaded documents can accurately reflect the statistics of a million documents. The study used single term queries based on various query selection strategies, such as selecting terms from documents downloaded so far or from a separate document set. Regardless of the strategy, query based sampling produced reasonably unbiased statistics. The study did not include effectiveness evaluation of selection based on sampled documents.

3.2 New server selection methods

This section describes the probe query method and a method for estimating the effectiveness of heterogeneous servers.

Probe queries

The broker sends probe queries to all servers *S*. For each results list it downloads the top *n* documents (Figure 3.2). From the downloaded documents, the broker may extract statistics and apply server ranking methods such as vGlOSS, CORI, CVV and others server ranking methods. This is possible without relying on the cooperation of servers. Probe queries were developed independently from the very similar query based sampling method of Callan, Connell, and Du [1999]. That study was published before probe queries were reported.

Probe queries are multi-term queries taken from some available query log, rather than single term queries selected on the fly as used for query based sampling. This query selection strategy is chosen because (1) Choice of strategy seemed not to bias the statistics extracted from a server [Callan et al. 1999] and (2) Multi-term queries allow effectiveness estimation (see below). Later experiments evaluate whether probe queries allow effective, generally applicable server selection, and if so how many probes are necessary.

Probe queries are quite different from the lightweight probes proposed by Hawking and Thistlewaite [1999].

- Lightweight probes:
  - Are sent to every search server, for every user query.
  - Are a two term subset of the user's current query.

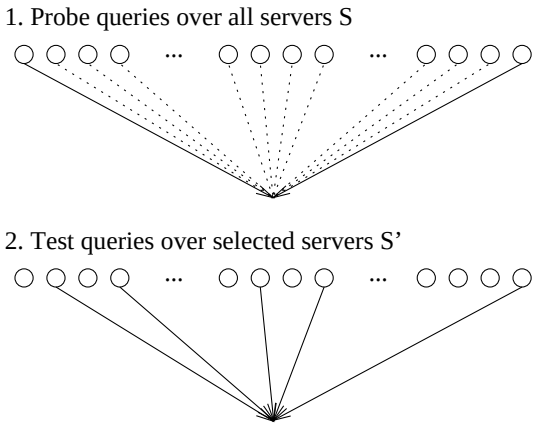


Figure 3.2: Probe queries and test queries. (1) Before query time, the broker sends each probe query to all search servers downloading their top results. The broker can then extract statistics from the documents for use in server ranking, and can also use the results to estimate server effectiveness. (2) Given a new query, the broker can apply server selection methods.

  - Retrieve statistics describing term occurrences from search servers, using a special protocol.
  - Allow selection based on the retrieved statistics.

- Probe queries:
  - Are sent to every search server, in a batch of perhaps 50. This happens periodically, perhaps once per month, not at user query time.
  - Are past user queries, taken from a query log.
  - Retrieve results lists from search servers, then top ranked documents from document servers. Neither of these require special protocols or other forms of server cooperation.
  - Allow selection based on term occurrence statistics, after extracting them from downloaded documents.

Lightweight probes use more up to date information, but require extra communication at query time. Probe queries can potentially become out of date, but require no additional communication at query time and are compatible with any available search/document server.

For some applications, communication costs of the methods might be quite similar. Each lightweight probe retrieves four different statistics amounting to 16 bytes of data. If the broker receives 100 queries per day and sends lightweight probes to 956 servers per query, the communication (not counting network packet headers etc) is 44 megabytes per month. For 1000 queries per day, the requirement would be about 440 megabytes per month.

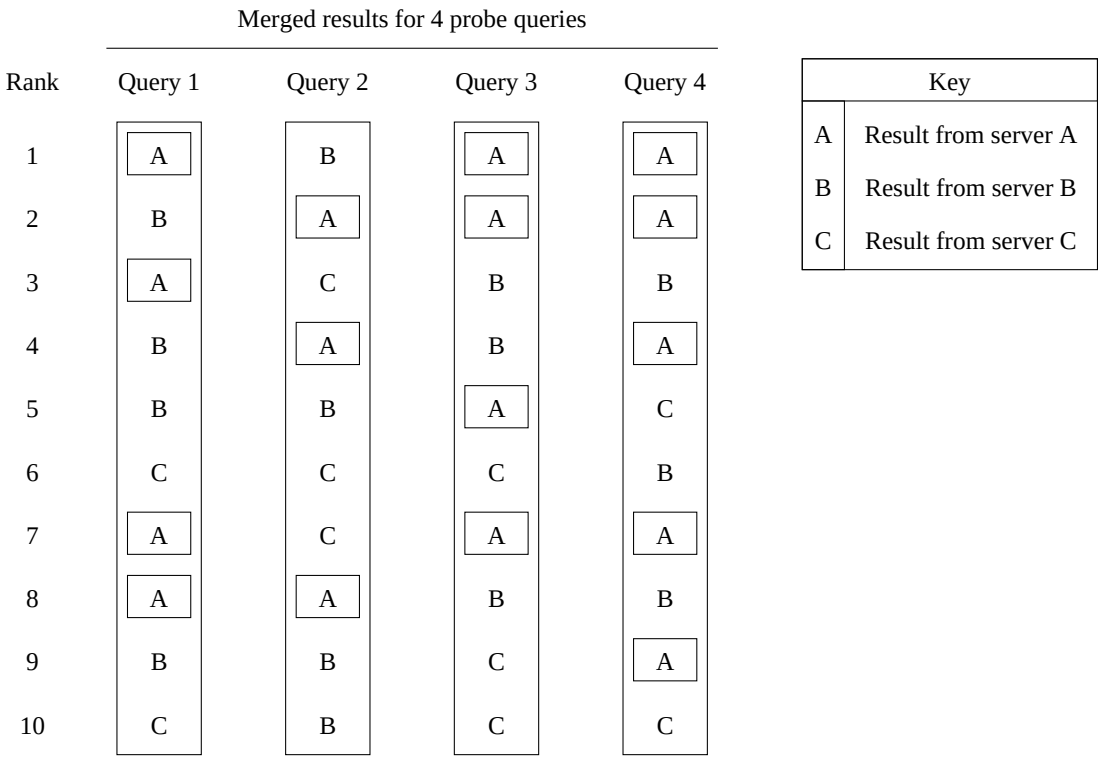


Figure 3.3: Predicting effectiveness. Over four probe queries, server A tends to have more top-ranked documents in the merged list. Therefore it is estimated to be more effective. (The top ten merged results are marked relevant in this example, but experiments in Chapter 5 mark the top twenty.)

In contrast, using probe queries involves downloading documents. When experiments in Chapter 5 simulate 50 probe queries over 956 servers, the broker downloads ($50 \times 956 =$) 47 800 documents containing search results from the search servers and discovers 60 000 unique documents which it downloads from document servers. So the broker downloads 107 800 documents in total. If documents are on average 5120 bytes, and probe queries are carried out once per month, the communication is 526 megabytes per month. This number remains the same regardless of how many user queries are processed per day.

By comparison, central indexing of the documents in the Chapter 5 experiments would require a full crawl/download of the documents to a central search server. If this occurred once per month, the communication would be about 2 000 megabytes per month. By the end of the month, many documents may have been added, changed or removed, and this would not be reflected in the central index. By contrast, with distributed solutions, retrieval is always based on the 956 distributed search servers, which would in many cases be updated daily or weekly.

Effectiveness estimation

Server ranking methods are usually based on term occurrence statistics within a server's documents. However, such methods do not take into account the effectiveness of heterogeneous retrieval algorithms. For example, if two servers have the same documents, but one runs a crude Boolean retrieval algorithm while the other employs a highly effective state of the art retrieval algorithm, the latter server is a better selection. Given information on servers' past effectiveness, it should be possible to modify server rankings accordingly and increase effectiveness. However, in an environment where relevance judgments are not available on past results, a more practical approach is to estimate the server's effectiveness.

A first attempt at automatic effectiveness estimation is as follows. The broker sends the multi-term probe query to all servers, then merges the results using a highly effective merging method based on downloaded document contents and reference statistics. Reference statistics, described below, are shown in Chapter 6 to allow highly effective merging without cooperation.

The broker then marks the top 20 results in the merged list as relevant. Across a number of probe queries the broker can calculate E_i, the expected number of marked relevant documents from server s_i. For example, consider ten probe queries sent to the server $s_n =$ `www.cpac.org`. If for each probe the top ten results were merged and a total of eight documents were marked relevant, then $E_n = \frac{8}{10} = 0.8$. See also Figure 3.3. In a top ten list from this server, the expected number of marked-relevant documents is 0.8.

Marking the top 20 merged probe query results as relevant is like a technique often used during automatic relevance feedback, where the top ranked documents from an initial query are assumed to be relevant and used for query expansion.

A CORI ranking, for example, could be modified by adding E_i times some constant c to $p(r_1, \ldots, r_M|s_i)$. This modified belief value can then be used to build a modified

server ranking and, after thresholding, a selection modified by estimated effectiveness. This formulation, of weighted addition, was chosen during initial testing.

Probe queries and effectiveness estimation are simple extensions to a search broker such as Inquirus, which has the machinery necessary to issue queries and download documents. The only extra code required is that which extracts statistics and performs selection.

3.3 New merging methods

This section introduces reference statistics and the feature distance document ranking algorithm.

Reference statistics

Many of the most effective document ranking functions are based on both document information and collection information [Harman 1992]. Document information is that which may be extracted from a document's text, such as a term's frequency of occurrence, the distance between term occurrences and the size of the document. Collection information is extracted from the corpus being searched. The most commonly used forms of collection information are document frequency statistics — the number of documents containing a given term — and the total number of documents in a collection.

In general a search broker can access document information, by downloading document full-texts, but can not access the collection information of distributed servers. True collection information, from servers S or S', is only available if servers cooperate (see Figure 2.6). For example, a search broker can not apply a $tf \cdot idf$ weighting in merging without cooperation because, although it may extract the required tf information from downloaded documents, it has no df information. For this reason the novel Inquirus ranking function (on page 25) does not use collection information.

Because true collection information is in general impossible to obtain, a new *reference statistics* approach is suggested here. The approach is to use collection information extracted from a more accessible collection as a reference point, rather than true collection information. For example, Inquirus can not access collection information from non-cooperating Web search servers, but could instead use collection information extracted from a different collection such as a readily available TREC collection or even extracted from documents downloaded during probe queries.

Such reference statistics allow the broker to employ a highly effective document ranking method, such as the Cornell variant of the Okapi BM25 probabilistic ranking function [Singhal et al. 1995; Robertson et al. 1994]:

$$w_k = TF_k \times \frac{\log(\frac{SS_i - DF_{i,k} + 0.5}{DF_{i,k} + 0.5})}{2 \times (0.25 + 0.75 \times \frac{DL}{AVDL}) + TF_k} \tag{3.1}$$

where w_k is the weight ascribed to a document due to occurrences of term t_k and

for purposes of maintaining consistent notation it is assumed that the document is on some server s_i. Notation only used in this equation (therefore not in Table 2.2) is TF_k the number of times t_k occurs in the document, *DL* the length of the document and *AVDL* the average document length. Document weights due to query terms are summed to give an overall score. Documents are then ranked in decreasing order of score. Reference statistics required are *DF*, *SS* and *AVDL*. Document downloads provide the document information, *TF* and *DL*.

Feature distance

In practice, the search broker might not wait until the last, slowest document has arrived before presenting merged results to the user. The system is more likely to apply a timeout on downloads, and if several of perhaps 100 documents do not make it, they are simply left out of the merged list. Another measure for saving time and bandwidth in document downloads is to base merging on partial document downloads: perhaps only using the first four kilobytes of each document, cutting down the network traffic if some documents are very large. Particularly for users with a slow connection, for example modem users, effective merging with partial download would be attractive.

Feature distance merging is based on the postulate that a ranking algorithm which gives higher weight to term occurrences near the start of each document will degrade gracefully with partial document downloads. Naturally the algorithm would also have to be highly effective on full documents. Hence hypotheses concerning both full and partial download are stated in Section 3.4.

A feature is defined as an occurrence of a query term within a document. The score achieved by a document is the sum of the scores achieved by its features. A feature score is based on four intuitions:

- Features which occur near the start of a document are more important than those near the end, so score decreases with the distance (l) of the feature from the start of the document.
- Features which occur close together are more important than those which occur far apart, so score decreases with the distance (d) between the current feature and the previous feature.
- The first time a feature occurs is more interesting than later times, so score decreases as the number of previous occurrences (n) increases.
- Terms which occur often in a collection are less important than rare terms, as captured in a simple $tf \cdot idf$ weighting. For this reason a feature score is inversely proportional to *DF*, the document frequency of the term within the collection.

Preliminary experiments [Craswell, Hawking and Thistlewaite 1999] using feature distance ranking suggested the following two feature distance ranking functions w_A and w_B. The former was most effective in general, while the latter had interesting

effectiveness results using reference statistics and partial document download:

$$w_A = \frac{1}{n \cdot \sqrt{d \cdot DF} \cdot \ln(l)} \tag{3.2}$$

$$w_B = \frac{1}{n^{1.1} \cdot \ln(d) \cdot \ln(DF+1) \cdot \ln(l)} \tag{3.3}$$

Feature scores, based on downloaded texts and reference statistics (see above), are summed to obtain document scores, which are used to generate the merged list in decreasing-score order. Both w_A and w_B are heuristic in nature, and were derived through tuning on 1% of the 3125 experimental configurations in the TREC-6 testbed described in Chapter 6 and in Craswell, Hawking, and Thistlewaite [1999]. They are not based on an underlying retrieval model. The comparative usefulness of different "intuitions" has not been analysed. For example, it might be the case that the l term does not enhance effectiveness and can be left out.

3.4 Hypotheses

The following hypotheses concern the effectiveness of the new methods: probe queries, effectiveness estimation, reference statistics and feature distance merging. Those related to selection are given an "S", and those related to merging an "M". The hypotheses marked with an asterisk (*) are central to the thesis: if they hold then a search broker can be generally applicable without sacrificing effectiveness.

* S1 Probe query based selection is as effective as selection based on information from cooperating servers.

S2 Incorporation of estimated effectiveness E_i improves selection effectiveness.

* M1 Reference statistic based merging is as effective as cooperative merging.

M2 Feature distance merging is as effective as merging based on the highly effective document ranking algorithm Okapi BM25.

M3 Feature distance effectiveness degrades more gracefully under partial download conditions than that of other document ranking algorithms.

The next chapter discusses methodology for testing these hypotheses.

3.5 Summary and conclusion

This chapter introduced new methods for server selection and results merging. Probe queries allow the broker to apply a method such as CORI, vGlOSS and CVV without cooperation from search servers. Effectiveness estimation is an attempt to improve selection effectiveness over heterogeneous search servers. Reference statistics allow the broker to apply algorithms such as Okapi BM25 in results merging without server

cooperation. Feature distance merging might improve merging effectiveness on partially downloaded documents.

The probe query and reference statistic methods allow a search broker to address non-cooperating servers, but it remains to be seen if they are effective. The other methods are suited to realistic environments. The effectiveness estimation method attempts to avoid a situation where a highly ineffective server is selected on the basis of its documents' term occurrence statistics. The feature distance method attempts to avoid a large drop in merging effectiveness, which might arise if the broker partially downloads documents for efficiency reasons.

Chapter 4

Evaluation Methodology

The preceding chapters introduced methods for distributed in formation retrieval and the following chapters evaluate them. This chapter bridges the gap by addressing the design of evaluation experiments. It describes previously used and new methodological alternatives. It then explains evaluation experiment design decisions made here.

Section 4.1 highlights problems with server merit evaluation, indicating why this thesis uses system level evaluation of selection methods. It then describes the necessary steps for system level evaluation. Section 4.2 describes issues in merging evaluation, including a novel method for generating simulated input rankings $R_1, \ldots, R_{|S'|}$.

4.1 Selection evaluation

There are two main approaches in server selection evaluation (see Section 2.4.2): system level evaluation and server merit evaluation. Although there has been some discussion of merit definitions [Gravano and Garcia-Molina 1996; French et al. 1998], there has been little comparison between system level and server merit evaluation.

This section argues for system level evaluation, for the time being, based on a simple assumption and two small experiments. These indicate that merit definitions are not yet well enough understood to perform merit evaluation with confidence.

The assumption is that a good selection S' is one which leads to good merged results R_M. For example, given the choice between a selection which maximises the quality of R_M and one which selects the servers containing the most matching documents, the former is optimal.

The first small experiment investigates whether the relevance based merit definition does maximise the quality of R_M and compares it to two new baselines.

The second experiment shows that the additional merit gained by selecting a server depends on which other servers have also been selected. So a server can not have a fixed merit number that is uninfluenced by other servers.

These experiments do not mean that server merit evaluation should never be performed. On the contrary, since merit experiments do not require simulation of retrieval and merging they are smaller than system level experiments. However, more work would be required to determine whether server merit evaluation closely approximates system level evaluation. Such work is beyond the scope of this thesis.

	Precision at 20		
Baseline	Mean	Standard deviation	Significant difference from Baseline 1.
1. Effectiveness (Precision at 10)	0.471	0.230	N/A
2. Density relevant	0.439	0.230	Yes (0.05)
3. Number relevant	0.419	0.246	Yes (0.05)
4. *Ideal*(0)	0.161	0.209	Yes (0.05)

Table 4.1: Comparing baselines (homogeneous retrieval). This table reports the measured effectiveness of various baselines, using the 956 server testbed described in Section 5.1. Measurement is of the precision at 20 of the merged result R_M after selecting each baseline's top 10 servers. The relevance based merit definition (baseline 3) does not describe the true contribution of a server as well as the definition based on server effectiveness. Significance is tested using a paired t-test, with the significance level in brackets (0.05).

	Precision at 20		
Baseline	Mean	Standard deviation	Significant difference from Baseline 1.
1. Effectiveness (Precision at 10)	0.406	0.223	N/A
2. Density relevant	0.378	0.213	Yes (0.05)
3. Number relevant	0.365	0.222	Yes (0.05)
4. *Ideal*(0)	0.131	0.200	Yes (0.05)

Table 4.2: Comparing baselines (heterogeneous retrieval). If servers run various retrieval algorithms, the effectiveness based ranking is still the best baseline by a statistically significant margin. See also Table 4.1.

4.1.1 Merit definitions may be incorrect

The merit baseline by Gravano and Garcia-Molina [1996] was *Ideal*(l), the summed scores of documents with a vector space score above l for the current query. Gravano and Garcia-Molina argued that there is no point in a server having relevant documents if they are not returned to the user, so having high-scoring documents is paramount and *Ideal*(l) is correct. French, Powell, Viles, Emmitt, and Prey [1998] argued that there is no point in a server returning documents if they are not relevant, and so use a relevance based merit baseline.

Both arguments are correct. It is both pointless for a server to have relevant documents if they are not returned and to return documents if they are not relevant. This suggests a couple of new baselines:

Relevance Density A server's ability to return relevant documents might be indicated by its ratio of relevant documents to total documents. An extreme example is two servers each with ten relevant documents, but one indexes a total of ten documents while the other indexes ten million. If merit is based on the number of relevant documents alone, these servers have equal merit. Under this definition one would have a merit of 1 and the other a merit of $\frac{1}{1000000}$.

Effectiveness A server's ability to return relevant documents is perfectly captured by its effectiveness. For example, if the broker is to merge a server's top ten documents, it should select servers with the best precision at 10.

Using the system level evaluation framework described in Section 5.1, it is possible to evaluate the four baseline merit definitions. See that section for a full description, but in summary the framework simulates 956 search servers using the TREC WT2g [Hawking et al. 1999] Web-data test collection. Server retrieval systems are either homogeneous, all running Okapi BM25, or heterogeneous, using three different algorithms. Merging is based on document download, BM25 ranking and reference statistics. Results in Table 4.1 show that if all servers run BM25, the relevance baseline is statistically significantly inferior to the effectiveness based ranking. Results in Table 4.2 show that with heterogeneous retrieval algorithms the same is true.

These results indicate that a relevance based merit definition is not the best way of capturing a server's desirability in server selection. As mentioned previously, further investigation would be required to determine whether this mismatch can produce misleading evaluation results.

4.1.2 Server merit depends on other selected servers

Although the previous subsection showed that an effectiveness based ranking was the best of four baselines, it did not show it to be optimal. This subsection shows using two examples that, if the goal is to maximise the quality of results R_M, there is no (simple) optimal merit definition.

First, consider merging through simple interleaving of ranks. As illustrated in Figure 4.1, once servers B and C have been selected, adding A can harm effectiveness. So perhaps A has negative merit. But if no servers are yet selected, servers A and C have equivalent, positive merit, contributing equally to overall effectiveness. Because the contribution to effectiveness depends on merging, and merging depends on other servers selected so far, merit depends on which servers have been selected so far.

Second, using the experimental environment described in Section 5.1 it is possible to find an ideal selection of N servers by trying all selections of N and seeing which maximises effectiveness. Using the measure average precision at 100 and topic 435, such an exploration yields interesting results. The best selection of one server for topic 435 is to select the server `nko.org`, whose top ten has relevant documents at ranks one and four. No other server has a better top ten. Yet, `nko.org` does not feature in the best selection of ten servers. Because of interactions in merging, documents which

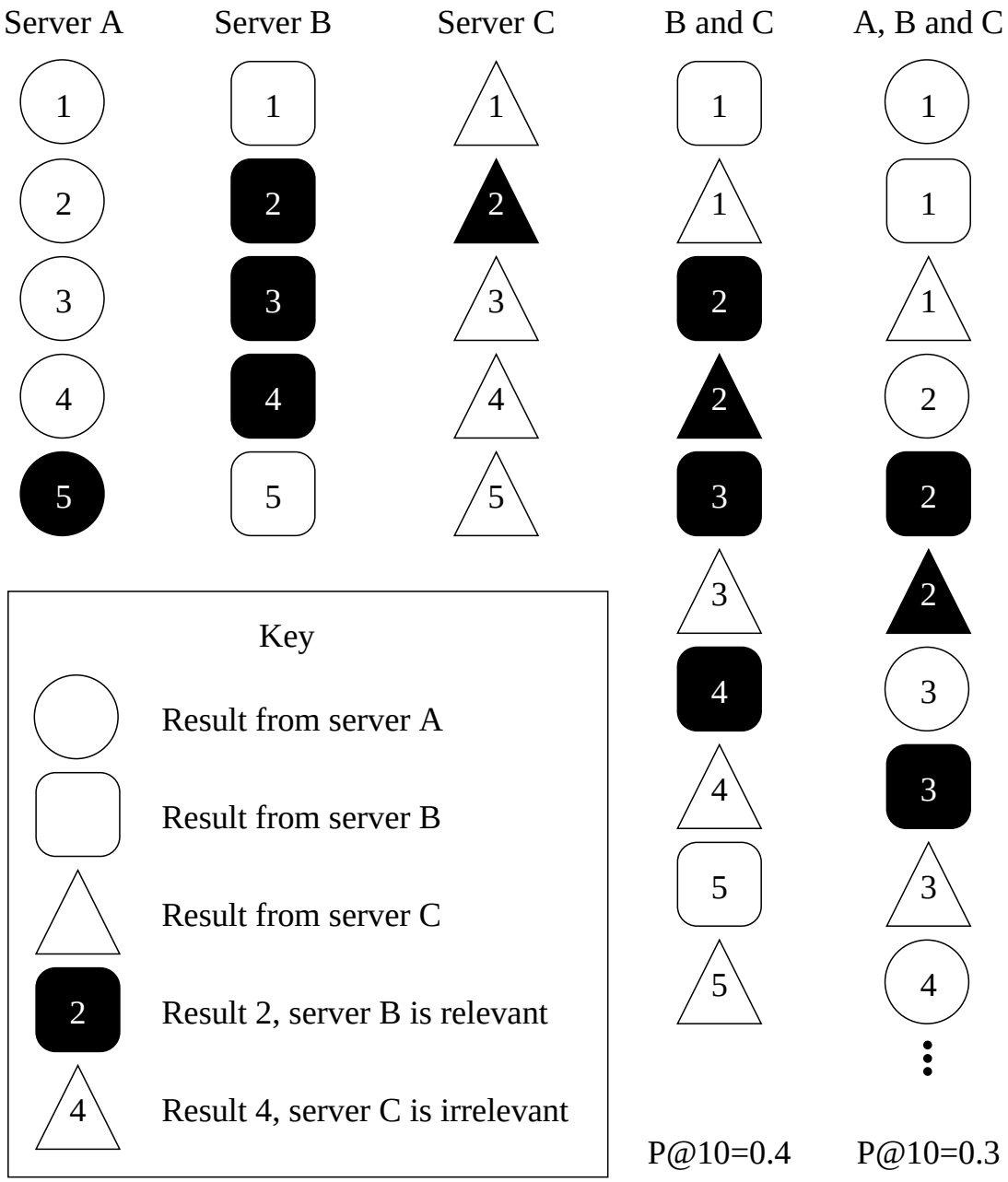


Figure 4.1: Retrieving relevant documents may reduce broker effectiveness. Defining merit as effectiveness (baseline 1 in Table 4.2) servers A and C are equivalent, each contributing one relevant document. If the broker is to select at most one server, A and C have equal and positive merit. However, if the broker is to select at most three servers — in order to maximise precision at 10 having merged using simple rank interleaving — it should select only B and C. Adding A actually decreases effectiveness. This simple example illustrates that (1) a server may have negative merit, even if it contributes a relevant document and (2) a server's merit depends on which *other* servers are already selected.

rank highly in `nko.org`'s top ten do not rank highly in the merged list of 100. There are ten servers better for the top 100 than the single most effective server.

Further study is required to determine whether merit baselines, although inexact, allow evaluation results which agree with system level evaluation. It certainly depends on the baseline. For example, Yuwono and Lee [1997] found that vGlOSS was superior to CORI using the $Ideal(l)$ baseline, while French, Powell, Callan, Viles, Emmitt, Prey, and Mou [1999] found the opposite using a relevance baseline. System level results presented here agree with the latter.

If server merit and system level evaluation agree sufficiently, server merit evaluation is preferable, because it allows experimentation without implementation of retrieval and merging methods. However, currently it is safer to perform system level evaluation, until merit definitions are better understood.

4.1.3 System level evaluation

System level evaluation compares the effectiveness of systems in some environment. The systems are search brokers which perform selection and merging. The environment models distributed search servers, which partition and perform retrieval over the documents of a test collection, and users of the broker. This section discusses methodological choices in modelling brokers and environment.

In the experiment, different broker designs are evaluated — different combinations of selection and merging methods — to determine which is the most effective. The most effective broker design can then be applied in the real world.

However, the experiment does not compare different environments. Instead, the modelled environment should closely resemble some real environment. In particular, the modelled environment should be as close as possible to the environment in which the best designed broker might be deployed. In this case, the chosen environment is of search servers such as those listed by InvisibleWeb (`http://www.invisibleweb.com/`). These servers are small and tend to be associated with a single Web document provider. They are unlikely to sub-partition a source amongst several servers.

The steps to modelling broker and environment are:

1. Choose a test collection, whose documents the simulated search servers will index, which provides:
   - A natural partitioning of the documents into search servers,
   - Enough search servers, and
   - Enough documents per search server,
2. Model selection, including extraction of selection information,
3. Model search server retrieval systems over document partitions, and
4. Model merging including collation of information, for example full or partial document download.

The test collection chosen must model characteristics of the search servers of interest. Otherwise methods might be developed which rely on all servers being the same size, covering similar subject areas or using homogeneous retrieval algorithms. Search servers of interest in this case have the following characteristics. Many servers exist. Most have an association with one or a handful of document providers, so it is preferable to partition a test collection by source rather than by date. For example, it is common for search servers to index Associated Press documents, for example AltaVista (`http://news.altavista.com/`) indexes them. Agreement amongst multiple servers to each index a different date slice of Associated Press is far less common. Single source servers vary in size more than servers which evenly partition a collection. They also tend to have high topic skew. That is, relevant documents for a topic are concentrated in a few servers rather than spread randomly.

To have enough servers and enough documents per server, a large collection is preferable. This suggests TREC [Harman 1999] ad hoc and Web track test collections as good candidates. Ad hoc document collections usually come from five sources. By contrast, the WT2g Web track test collection covers 956 document servers. A partitioning of WT2g by source yields 956 Web-data search servers of varying size with significant topic skew, each covering documents from a single document provider.

In simulating selection, past studies have assumed that servers cooperate with the broker by exporting statistics describing their documents. While such cooperation allows efficient and effective selection, it is not common in Web search servers. The alternative is to simulate probe queries and extraction of statistics from downloaded documents. Having assigned a simulated retrieval method to a server, running probe queries and extracting statistics from only returned documents is not difficult.

A retrieval system must be assigned to each search server, allowing the server's search results to be generated for probe and user queries. Past studies have assigned homogeneous retrieval methods to servers, for example Xu and Callan [1998] assume each search server runs INQUERY. It is equally possible to assign varied algorithms to servers, and this is indeed more realistic when modelling a Web environment.

Finally it is necessary to simulate merging. Cooperative merging may be simulated using a consistent retrieval algorithms and statistics from servers S or S'. However, this is again unrealistic in the Web environment, where true global collection statistics are not collated by the broker. Instead, the broker may use a generally applicable merging method. For example, it can use reference statistics with document download and Okapi BM25.

4.2 Merging evaluation

There is only one approach to merging evaluation, exhibited in [Callan et al. 1995]. The experimenter applies multiple merging methods to the same input rankings $R_1, \ldots, R_{|S'|}$, then evaluates merged lists to see which method has produced the most effective output. However, simulating the entire selection and retrieval process to generate $R_1, \ldots, R_{|S'|}$ requires non-trivial experimental effort and computation. It might

include heterogeneous retrieval over a large number of servers, selection with probe queries then processing of test queries over selected servers.

The innovation introduced here is to simulate input rankings. Fast simulation allows more inputs to be generated with less experimental set-up and computation. This allows merging methods to easily be evaluated in a greater variety of conditions.

4.2.1 Simulating input rankings

Past runs are available for some test collections. For example, official TREC runs are available to conference participants at the conference's web site (`http://trec.nist.gov/`). The experimenter may use such runs, which are over an entire test collection, to simulate runs over document partitions. For example, if one partition contains only Federal Register documents, a run over that partition may be simulated by removing all non-Federal Register documents from a full, official run.

Rankings simulated in such a way may not be identical to the rankings which the system would have produced over a single partition. Most systems base their output on collection statistics, and the simulated ranking will be with respect to global statistics. However, the results will be similar to those produced by a real system, and obtained with much less effort in implementing or installing retrieval systems and less computation time. TREC runs originate from diverse, complex retrieval systems, much more so than could be easily set up by a single experimenter. Finally, generating input rankings from official runs is repeatable by other experimenters in a way which is not strongly subject to mistakes.

There are two dangers in simulating input rankings. First, if a test collection has many partitions, even an official TREC run listing 1000 documents per topic may not contain enough documents to make up the required partition runs. For example, partitioning WT2g by document server yields 956 search servers, making it unlikely that a sizable run for any partition will come from an official top 1000. The second problem is simulating selection, a time consuming process.

Both problems can be solved by partitioning a test collection into a small number of sub-collections, for example partitioning TREC-6 ad hoc documents by source into five sub-collections. Since each of these usually contains some query term occurrences for every TREC topic, it is plausible that the five are selected servers S' from some larger set S. With only a handful of partitions it is likely that most official top 1000 lists will have more than enough results to simulate $R_1, \ldots, R_5$ of size at least 30 documents and usually much larger.

Evaluation with simulated input rankings is suited to a test collection with few partitions and large available results lists, such as TREC ad hoc test collections. System level evaluation, as argued earlier, is suited to a test collection with many more natural partitions. Despite requiring different test collection characteristics, both types of evaluation are valid, and both are used in this thesis.

4.3 Summary and conclusion

This chapter argued that despite the simplicity of server merit evaluation experiments, it is not clear that selecting servers with the highest merit always leads to the best search results. It then described methodological choices in selection and merging evaluation experiments. This included a simulation of servers using WT2g partitioned by source and with heterogeneous retrieval algorithms, which models Web search servers more closely than previous testbeds. It also included a method for reducing the size of merging experiments by simulating the retrieval results of servers S' in a simple and repeatable fashion.

Based on server merit discrepancies, it can be concluded that, for the moment, it is safest to perform system level evaluation. This may change if future experiments show server merit evaluation results to closely approximate system level results. Methodological choices made here are carried through to evaluation experiments in Chapter 5 and Chapter 6.

Chapter 5

Server Selection Experiments

The primary question is whether selection based on probe queries is as effective as selection based on full server information (hypothesis S1, Section 3.4). Also evaluated is selection augmented by estimated server effectiveness (hypothesis S2). In addition the evaluation includes a comparison of various server ranking algorithms, and a comparison of distributed and centralised search within the testbed.

Section 5.1 describes the experimental methods employed, including implementation details of the evaluated methods. Section 5.2 presents results of the experiments. Section 5.3 discusses the implications of these results, particularly with respect to hypotheses S1 and S2. Section 5.4 presents further experiments, which test the generality of results. Section 5.5 outlines conclusions.

5.1 Method

The previous chapter argued that the effectiveness of a selection method is best evaluated according to the final results R_M presented to the user. Therefore this chapter's experimental methodology is to simulate the whole distributed retrieval process:

$$\langle S, q\rangle \overset{Selection}{\longrightarrow} \langle S', q\rangle \overset{Retrieval}{\longrightarrow} \langle (R_1, \ldots, R_{|S'|}), q\rangle \overset{Merging}{\longrightarrow} R_M$$

This process requires partitioning of a test collection into servers, then for each selection S' the application of servers' retrieval systems and the broker's merging method.

The three following subsections describe the modelled search servers, broker and user.

5.1.1 Search servers: documents and retrieval

This experiment uses the WT2g [Hawking et al. 1999] TREC [Voorhees and Harman 1999] test collection. WT2g contains two gigabytes of Web documents, from 956 document servers, for example `http://greenpeace.org/` and `http://www.londontransport.co.uk/`. The documents are partitioned on document server boundaries, making 956 search servers, each indexing all and only documents from one document server. The selection methods evaluated here do not rely on a one to

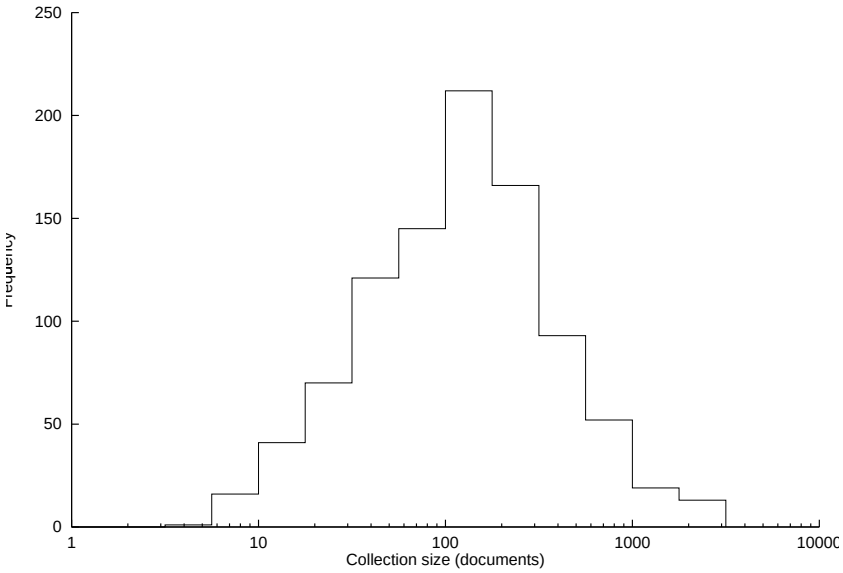


Figure 5.1: Server sizes. Histogram of WT2g server sizes in documents, partitioning by source.

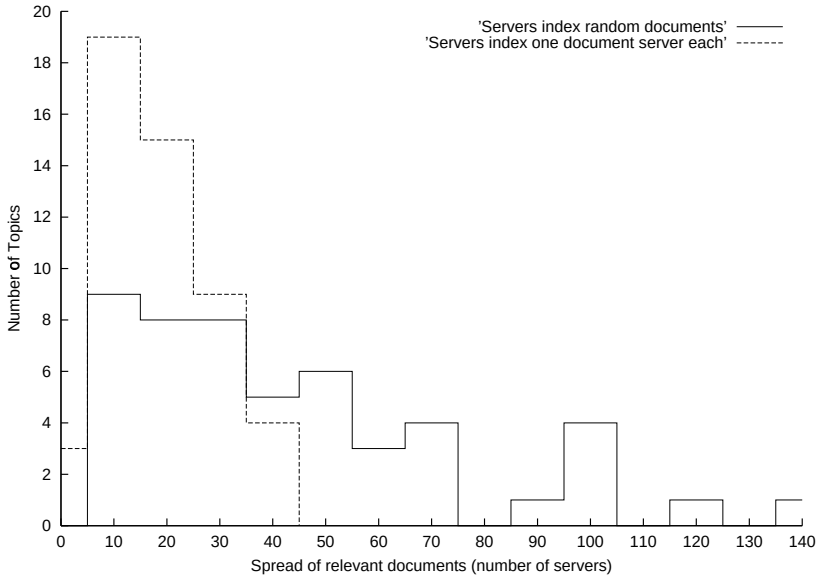


Figure 5.2: Topic skew. Relevant documents for a topic tend to be concentrated on a few servers when documents are partitioned by source, rather than spread randomly across servers. This histogram illustrates the difference between allocating documents by source and allocating documents randomly, for TREC topics 401–450. This indicates that Web servers have natural topic skew.

Study	Document type	Partitioned by	Servers
Callan et al. [1995]	TREC news/gov.	Source, disk	17
Gravano and Garcia-Molina [1996]	Newsgroups	Group	53
Yuwono and Lee [1997]	SMART	Source	4
French et al. [1998]	TREC news/gov.	Source, date	236
This chapter	TREC Web (WT2g)	Source	956

Table 5.1: Test collections used in published experiments. It is easy to find search servers which index one particular document server (source). Partitioning WT2g by source simulates such servers. By contrast it is less common for multiple search servers to index one newsgroup each or to subpartition news documents by date. Such scenarios are less realistic. In WT2g the relevant documents are spread across 956 servers, giving low density of relevant documents per server and high topic skew (Figure 5.2). However, this reflects topic skew in real Web servers. Also, the collection-wide density of relevant documents in TREC-8 WT2g experiments was the same as that in the main TREC-8 task (about 0.9%) [Hawking et al. 1999]. (Note: Yuwono and Lee used collections supplied with the SMART system, called CACM, CISI, CRAN and MED.)

one mapping between search and document servers. The one to one mapping simply models a common situation on the Web, where a search server covers documents from one source.

The 956 servers have a wide range of sizes, from a handful of documents to several thousand (see Figure 5.1). Average server size is 259 documents, close to the figure of 289 documents reported for live Web servers by Lawrence and Giles [1999]. Missing from the server set are large servers, for example those covering an online news archive. One reason for this is that the test collection is based on a Web crawl, but many such collections are "invisible" to crawlers. See the discussion on page 11 for more information. Also missing are the very large search servers run by search engine companies.

Larger test collections might, in future, allow realistic experiments with larger servers. For example, using WT10g it might be possible for some search servers to index multiple, related document servers. However, a test collection based on Web crawling will never include Web documents that are truly "invisible" to crawlers. Inclusion of such documents would require cooperation from the document providers.

The 956 servers here also have considerable topic skew, with relevant documents for a given topic tending to be concentrated at a few servers rather than spread across many servers (see Figure 5.2). Such skew is a property of search servers which cover one source, a common situation on the Web. This is in contrast to previous published experiments, where servers often subdivided a larger, homogeneous collection (Table 5.1).

Each server runs one of three document ranking algorithms. The first is highly

effective, the Cornell variant of the Okapi BM25 probabilistic ranking function (Equation 3.1). The function was modified slightly for use in retrieval here. In the usual BM25 function, the presence of a term which occurs in more than half the server's documents will actually reduce a document's relevance weight. Such occurrence statistics are unlikely in a two gigabyte TREC ad hoc collection, but are much more likely on a small server with high topic skew. For example, the `womenspace.com` server indexes 114 pages, all of which contain the word "women". However, this does not mean that for the query "women clergy" a document should be penalised for containing the word "women" more times. Similarly for the query "u.s. investment in africa" the server `bayes.econ.umn.edu` has the word "investment" in 32 of its 64 documents, but this does not mean the word is not important for ranking with respect to the query. A partial solution to this problem, implemented here, is to allow each term to have only non-negative effects on BM25 scores, by setting any negative effects to zero.

The second ranking algorithm is intended to be much less effective, ranking servers based on their summed query term frequencies. This simulates a Web search server which gives documents a bonus for having more term occurrences but does not use more sophisticated models or statistics. Its formulation is:

$$Count_{d_i} = \sum_{k=1}^{M} TF_{i,k} \tag{5.1}$$

where M is the number of query terms and $TF_{i,k}$ is the number of times term t_k occurs in document d_i. Documents are ranked in descending order of *Count* values.

The third "ranking" algorithm simply requires a Boolean conjunction of the query terms, listing matching documents in arbitrary order.

$$\begin{aligned} AND_{d_i} &= 1 \quad \textit{if } \forall\, t_k \in q :\ TF_{i,k} > 0 \\ &= 0 \quad \textit{otherwise} \end{aligned} \tag{5.2}$$

q is the user's (possibly multi-term) query. Returning only documents with $AND = 1$ provides results consistent with a simple Boolean search server.

Because the three algorithms vary in effectiveness and approach, they are broadly consistent with the variation in Web search server retrieval systems [Hawking, Craswell, Thistlewaite and Harman 1999]. All three retrieval algorithms are implemented in PADRE [Hawking et al. 1998a].

For topics 401–450 there are 888 server-topic pairs where it was possible for a relevant document to be returned. This is to be expected given the natural (and realistic) topic skew exhibited by the partitioning. Mean precision at 10 in those cases was 0.147 for BM25, 0.120 for count and 0.084 for Boolean retrieval. BM25 does not achieve great effectiveness, however a perfect retrieval algorithm in the same situation would only achieve 0.232. Low precision is due to the low number of relevant documents per server. Searching these servers, which can only return relevant documents in 888 out of $956 \times 50 = 47\,800$ possible server-topic combinations, it is interesting that a realistic

search broker can achieve a mean precision at twenty of around 0.2 (see Figure 5.3).

With 956 servers each running one of three algorithms there are 3^{956} different possible server configurations, a very large number. This is realistic. On the Web the number of search servers and variety of available retrieval systems is greater, so the potential number of configurations is far larger.

The resources available for this experiment allowed for only a few configurations. Randomly choosing several configurations, then repeating the whole experiment for each, would have made the experiment far too large. If too few configurations were chosen, they might not be representative of the whole set. Instead a single, uniform, heterogeneous configuration was constructed, by listing the servers in increasing order of size (in documents), then assigning algorithms in alternating order. This ensured that a roughly even number of servers used each algorithm, and that some very small and some very large servers used each.

One might think that in the real Web more systems run bad algorithms than good, given the number of servers which return no results or seem to use search systems hand-coded by amateurs. However, without studying the effectiveness of such systems, an even distribution seemed to have the benefit of simplicity, fairness and evenness.

A homogeneous configuration with BM25 at each search server is also used in Section 5.4 for further experiments.

5.1.2 The broker: selection and merging

This experiment does not assume cooperation from search servers. Instead, merging and selection are based on downloaded document contents as described in Chapter 3.

Probe queries allow the broker to apply server ranking methods such as CORI. However, since they involve downloading the top 10 documents from each server for each query, sending too many probe queries can be expensive. Because of this several levels of probing — several choices of how many probe queries to execute — are evaluated. A set of 200 probe queries come from titles of TREC topics 151–200 and 251–400 (201–250 have no titles). Levels of probing are at 10, 25, 50, 100, 150 and 200 probe queries, choosing newer queries. For example the 10 probe query level used topics 391–400.

Probe queries have some interesting implications in implementation. For example, it is possible for probe queries to retrieve no results from a server. For the two term query "*A B*", a BM25 server will return no results if neither *A* nor *B* is present in any indexed document. A Boolean server will return no results if no document contains both *A* and *B*. Ten probe queries found some information on 702 servers, while 200 found information on 948 servers. The 8 missing servers ranged in size from 9 to 217 documents (on average 63) and all were Boolean servers. Amongst the implications of having no information on a server are that it will never be selected, and that server size (SS_i) is 0. When computing server rankings in such cases, for example with CVV selection, division by zero was handled by making the result $0/0 = 0$.

The second stage of selection is server ranking. Four server ranking methods are

evaluated here:

1. CORI selection as described in Equation 2.1.
2. Cue validity variance (CVV) as described in Equation 2.3.
3. The vector space variant of GlOSS (vGlOSS), described on page 16.
4. CORI selection modified according to servers' expected effectiveness, as described on page 40.

The first three server ranking methods have been evaluated together before [Yuwono and Lee 1997], although not in system level evaluation over Web documents with heterogeneous retrieval algorithms. The fourth method is new.

Modified CORI has a tuning constant c. This was set by using the ideal form of expected effectiveness E' (which is actual mean effectiveness). Hand tuning was applied to provide optimal effectiveness with E', giving a constant of $c = 0.03$. This gives a sensible value for c without tuning against the real data E_i. This value of c was used in applying both the "CORI plus E_i" and "CORI plus E_i'" methods.

The third stage of selection is thresholding. The number of servers to query is governed here on grounds of efficiency. It is assumed all servers have equivalent search costs and the user is willing to bear the cost of selecting only 10 servers per query. This number reflects the number of servers queried by existing Web search brokers. MetaCrawler queries 13 servers, Profusion queries 3–9 and Inquirus queries 17.

Merging of selected servers' top 10 lists is based on document download, reference statistics and the Okapi BM25 document ranking algorithm (Equation 3.1). This combination is shown to be highly effective in the next chapter. Reference statistics in this case are taken from a 10% sample of the 100 gigabyte TREC VLC2 Web-data collection [Hawking et al. 1999]. WT2g is a subset of VLC2, so it might be thought that the 10% VLC2 sample and WT2g might have very similar statistics. However, a real broker using probe queries would have an even better picture of the statistics, having downloaded tens of thousands of documents during probing. Experiments here use the 10% VLC2 sample for simplicity.

For each selection of 10 servers the broker merges up to 100 documents, because servers return their top 10 documents, but may return fewer if they do not have enough which match the query.

Central indexing

The main purpose of this experiment is to compare selection methods over the 956 search servers. However, a small side-experiment is also conducted. This models a central indexer over the 956 document servers.

There is one problem with simulating a central indexer, which is simulating its coverage of the documents in question. A central search engine which covers tens or hundreds of millions of documents still only covers a small fraction of the indexable

Web [Lawrence and Giles 1999]. So one way of simulating central indexing would be to simulate a 10% coverage found to be common by Lawrence and Giles, by not indexing 90% of WT2g documents. However, central engines have limited coverage due to scalability problems faced when indexing millions of document servers. These limitations would not apply in a central index of 956 servers.

Limited coverage of a 956 server index would come from another source. Some Web document servers exclude some or all of their documents from indexing (see discussion on page 2.2.2). If InvisibleWeb is to be believed, there are tens of thousands of Web document servers whose documents can not be centrally indexed, but can be searched using local search servers. So a central indexer trying to cover the 956 document servers would be excluded from indexing some proportion of them.

Here, evaluation under three levels is considered: all servers, 50% of servers and 25% of servers. As with the assignment of retrieval systems, the set of servers which are unavailable for indexing is selected by listing servers in decreasing size order and indexing every second or every fourth. For all levels of central indexing, the retrieval method is BM25, with the same queries used in distributed retrieval, no stemming, case folding and no relevance feedback.

5.1.3 User model

The test collection used here, from the TREC-8 small Web task [Hawking et al. 1999], comprises:

- Two gigabytes of Web documents which are a subset of VLC2, containing all VLC2 documents from each of 956 document servers. Servers were chosen on the basis of having relevant documents in the TREC-7 Very Large Collection track,
- 50 research-style topics (TREC topics 401–450) and
- Pooled relevance judgments for each topics, made by NIST.

In this experiment, the user's query is based on the titles of TREC topics, usually consisting of 2–3 words which succinctly describe the information need. Short queries are used because real Web users tend to use few query terms [Spink et al. 1999].

For each test query, results are merged then evaluated according to precision at 20. Precision at 20 was chosen because Web users do not often look beyond the first page of search results [Spink et al. 1999] and most Web brokers return 20–50 results on their first page [Lawrence and Giles 1998; Selberg and Etzioni 1997; Fan and Gauch 1999]. Precision at 20 has also been used in official TREC Web-data experiments [Hawking et al. 1999].

5.2 Results

As shown in Figure 5.3, CORI outperforms vGlOSS and CVV, particularly with many probe queries or full information obtained from cooperating servers. The performance

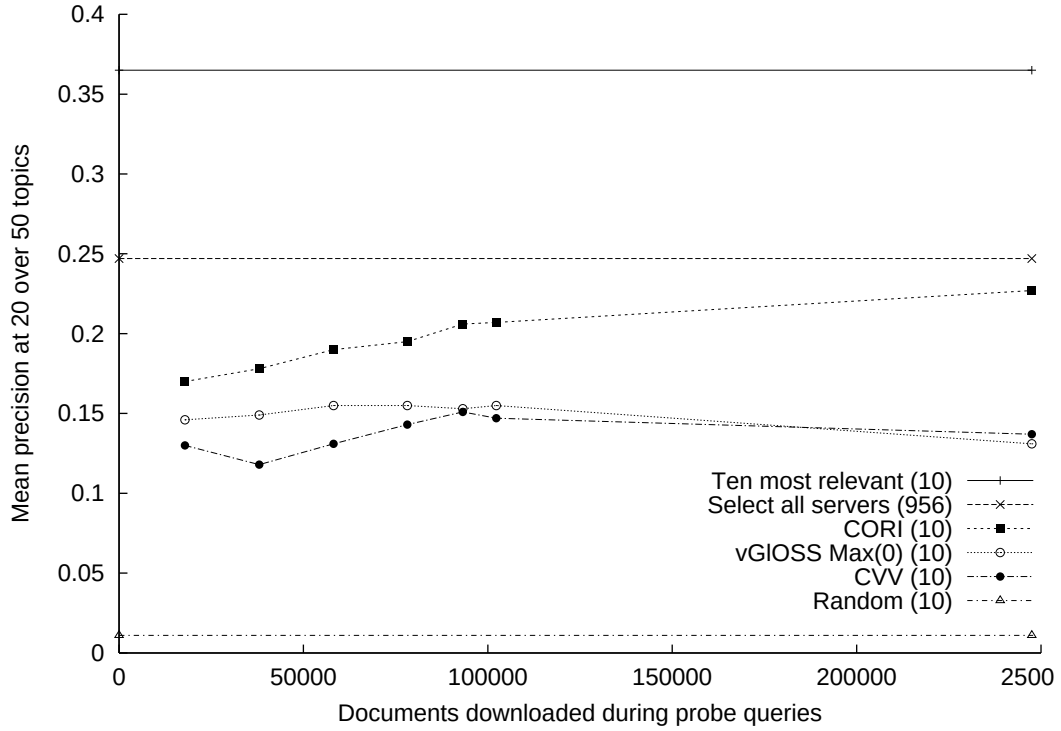


Figure 5.3: Selection effectiveness with probe queries and heterogeneous servers. This figure presents evaluation results over various levels of probe queries for test topics 401–450. CORI, vGlOSS and CVV have data points for (from left to right) 10, 25, 50, 100, 150 and 200 probe queries. The rightmost points are based on full collection information, as would be available from cooperating servers, not probe queries. For reference, the figure also includes three results from Table 5.2, based on selecting 10 servers with the most relevant documents, all 956 servers and 10 random servers. Numbers in parentheses are the number of servers selected per query. Note, a discussion of efficiency issues appears, starting on page 37.

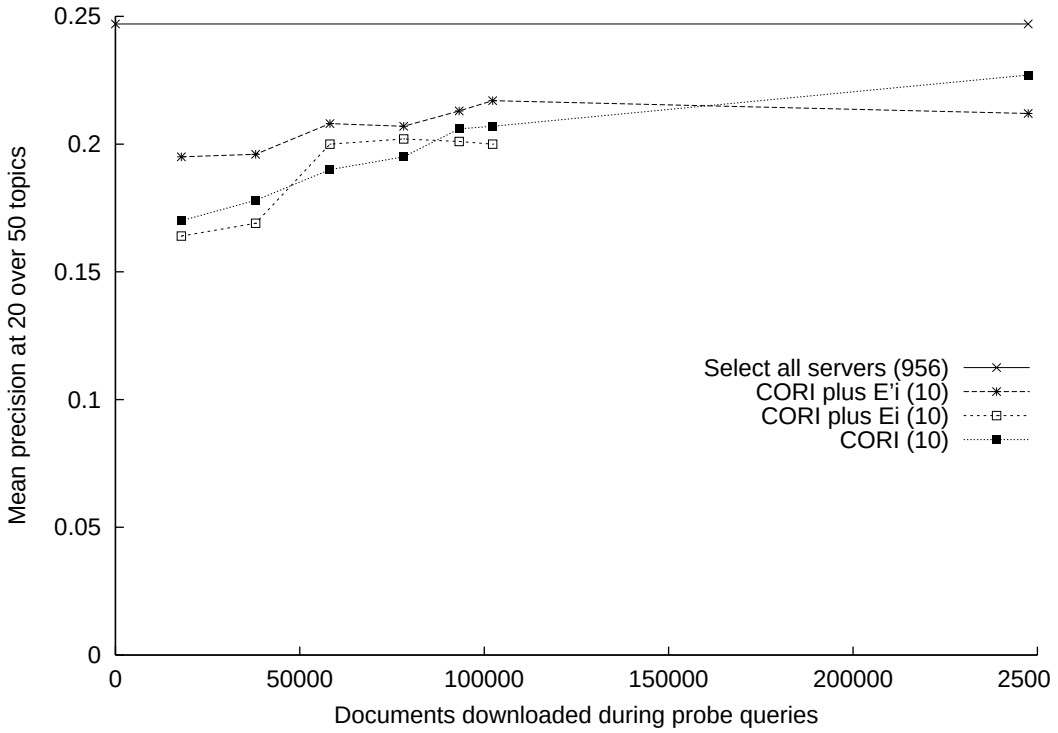


Figure 5.4: Selection effectiveness based on effectiveness estimation (heterogeneous servers). This figure is the same as Figure 5.3, with the same probe query levels and topics 401–450, but shows modification of CORI by E_i, estimated effectiveness, and E_i', true effectiveness. There is no rightmost point for CORI plus E_i, because the method for effectiveness estimation requires probe queries and no probe queries are conducted in the full cooperation case. Only E_i' achieves a improvement over CORI which is statistically significant, and then only at 10 and 25 probe queries.

of vGlOSS and CVV with full information is close to their performance with only 10 probe queries. CORI steadily degrades with fewer probes, falling by as much as 0.05 in the 10 probe query case, equivalent to one fewer relevant document per results list on average. Two tailed, paired t-tests at a 0.05 level show that CORI with full information is equivalent to CORI with 150 or 200 probe queries but superior to CORI with 100, 50, 25 and 10 probe queries.

With fewer probes vGlOSS effectiveness improves, while CVV varies. No realistic selection of 10 servers is superior to a selection of all 956. However, a relevance based selection of 10 is significantly superior. All realistic selections are far superior to a random selection of 10.

Figure 5.4 shows that CORI modified by E_i disappointingly provides no consistent improvement over plain CORI, so statistical tests were not conducted. However, using known expected effectiveness E_i' provides a consistent improvement, which is significant at the 10 and 25 probe query points (using t-tests as described previously).

Note, these results rely on six t-tests of full information against probe queries and six t-tests of CORI against CORI plus E_i'. Section 5.4 performs more tests. With 12 tests, each with a 95% confidence of being correct, there is a $1 - (.95^{12}) = 45\%$ chance that at least one test is incorrect. However, the chance that several particular tests are incorrect is much less. For example, full information CORI is statistically equivalent to 150 and 200 probe queries. The probability of precisely those tests both being wrong is low. For this reason, the results may be presented with some confidence.

Table 5.2 shows results for a central index with 100%, 50% and 25% coverage. Interestingly, distributed retrieval over a very effective selection of 10 servers is superior to a 100% index. The 50% index achieves 0.227 equivalent to CORI selection based on full information (the rightmost CORI point in Figure 5.3). The 25% index, achieving only 0.135 is less effective than most selections of 10 servers.

5.3 Discussion

The most important test is of hypothesis S1. Results show that selection based on 200 or 150 probe queries is as effective as selection based on full information, at a 0.05 confidence level. With fewer probe queries there is a significant drop in effectiveness, although it is not clear at what point a loss would be noticeable to the user. For example, there is a statistically significant difference between the full cooperation and 100 probe query cases, but it only involves a 14% fall in mean effectiveness. In any case, at 150 and 200 probe queries results are achieved which are statistically indistinguishable from results in the full cooperation case. So hypothesis S1 holds for high levels of probe queries.

It is interesting that selection based on fewer than 50 000 documents can have effectiveness so close to that based on all 250 000 documents. This indicates that probe queries are a good way of obtaining information about Web search servers without relying on their explicit cooperation.

Hypothesis S2 was that effectiveness estimation E_i can improve selection effec-

Selection method	Proportion selected	Precision at 20	
		Mean	Standard Deviation
Ten most relevant	10/956	0.365	0.222
Centralised 100% index	N/A	0.343	0.239
Select all servers	956/956	0.247	0.266
Centralised 50% index	N/A	0.227	0.222
CORI plus E_i'	10/956	0.208	0.205
CORI plus E_i *	10/956	0.200	0.208
CORI *	10/956	0.190	0.202
vGlOSS Max(0) *	10/956	0.155	0.213
Centralised 25% index	N/A	0.135	0.170
CVV *	10/956	0.131	0.168
Random selection	10/956	0.011	0.043

Table 5.2: Distributed vs centralised. This table presents results over test topics 401–450 and the heterogeneous server configuration. It lists results for four realistic selection scenarios, each selecting 10 servers based on 50 probe queries (marked with *). It also presents results for central BM25 retrieval over 100%, 50% and 25% of the servers (using title only queries and no relevance feedback as used in the distributed case). Finally, it presents results based on selecting all 956 servers, selecting the 10 with the most judged-relevant documents and selecting 10 random servers. All methods have large spread (standard deviation).

tiveness. It is not necessary to perform statistical tests to see that E_i provides no improvement that is statistically significant or noticeable to the user (Figure 5.4). This is disappointing. However, this lack of improvement may be due to a low potential for improvement. Using the perfect indication of server effectiveness E_i', which is the mean effectiveness of a server across test queries using official relevance judgments, only provides a significant improvement at the 10 and 25 probe query levels. Even these might be too small to be noticeable to the user.

This might mean that in the current testbed differences in effectiveness are not useful in selection. Alternatively the method of modifying CORI according to effectiveness, by adding the estimate times a constant $c = 0.03$, might be the problem. The testbed explanation is plausible. With extreme topic skew illustrated in Figure 5.2 the presence of relevant documents might be a much more important influence on selection than the effectiveness of server retrieval algorithms. Effectiveness estimation might be more important in situations where more servers (of varying effectiveness) have relevant documents to offer. For example, it might be useful in situations where multiple servers cover the same document collection. In addition, the small server sizes in this testbed might mask differences in effectiveness, as was suggested earlier when it was found that BM25 and "count" retrieval had similar effectiveness. Although effectiveness estimation failed, the possibility of statistically significant improvement using E_i' confirms that selection can be improved by taking server effectiveness into account.

In Figure 5.3, the initial upward slope of CVV and vGlOSS from the full information to the 200 probe query case may indicate some problem with server size normalisation. Without size normalisation, the broker might be confused by size variation in servers and tend to select larger servers. Probe queries mask server size variation, perhaps explaining the improvement of CVV and vGlOSS. This fits with the observation by French et al [French et al. 1999] that vGlOSS correlates well with a size-based server ranking.

A notable feature of CVV is that it gives a bonus to query terms with high document frequency variance, assuming that they will be good discriminators between servers. However, this may not be a good assumption, at least in this testbed. For example, the test queries included the terms *fluids* and *mirjana*. In document ranking the latter term would be considered more important than the former, because it is rare. Instead, in the CVV method, *fluids* has the highest CVV of all query terms and *mirjana* has the lowest. This is because the document frequency of *mirjana* is usually 0 and sometimes 2 or 3. By contrast, *fluids* appears in 297 servers in between 1–232 documents in each. The intuition that terms with varying document frequencies are good discriminators may not be correct, particularly for terms which only appear a few times in a few servers such as *mirjana*.

In Figure 5.3 a relevance based selection of 10 servers outperforms a selection of all 956. This shows that querying fewer servers can improve broker effectiveness. While CORI selection and others do not outperform the selection of 956, it would be interesting to vary the number of servers selected and the number of documents retrieved per server to see how effectiveness might vary. Even with current parameters,

10 documents per server from 10 servers, it is still worthwhile to perform selection for reasons of efficiency. A broker querying a CORI selection of 10 achieves almost the same effectiveness as a selection of 956 with $\frac{10}{956} \approx 1\%$ of the retrieval effort.

Another interesting result is that a highly effective selection of only 10 servers, based on knowledge of relevance judgments, outperforms 100% central retrieval using BM25 (Table 5.2). Centralised BM25 results are crude, using no relevance feedback or other techniques, however distributed results also use simple techniques. Further, distributed results include only a fraction of the WT2g documents (those of only 10 servers) and are based on ineffective, heterogeneous retrieval algorithms.

A likely explanation for 10 servers outperforming a central index stems from topic skew. The 10 servers with the most relevant documents contained on average 83% of a topic's relevant documents. For 15 topics, the 10 contained all relevant documents. Particularly for the latter topics, retrieval over the 10 servers is likely to be effective. Centralised 100% retrieval has the same relevant documents, but adds a huge number of irrelevant documents which may contain query terms, which might explain its lower effectiveness.

The results, that CORI outperforms vGlOSS, agree with French, Powell, Callan, Viles, Emmitt, Prey, and Mou [1999]. They disagree with Yuwono and Lee [1997], perhaps indicating a problem with the *Ideal*(l) merit baseline.

The results presented here are for a specific type of user in a specific configuration of distributed environment. The user is most interested in the top 20 search results, and is only prepared to wait for the broker to select and search 10 servers, merging at most 100 results. The configuration of the distributed environment includes a specific assignment of heterogeneous retrieval algorithms to search servers. Central 50% and 25% indexes are over a specific server subset each. Therefore, the results describe what happens in one realistic Web search configuration — more realistic than any previous experiments in terms of document type, topic skew and server heterogeneity — but results might vary using different parameters, a different Web-like testbed or even on the live Web. Only more experiments of this type will test the generality of these results.

5.4 Further experiments

This chapter has evaluated 38 broker designs. 35 came from five server ranking algorithms by seven levels of information (six levels of probe query, plus the full information case). The other three were brokers which select all, the 10 most relevant and 10 random servers.

The experiment explored 38 broker designs at the expense of exploring the space of server configurations. This is because the primary experimental questions are best answered by multiple algorithms over multiple levels of information. Adding another dimension by including a representative sample of the 3^{956} possible server configurations would blow out the experimental size beyond manageable levels.

However, it is desirable to confirm experimental results by testing against a second

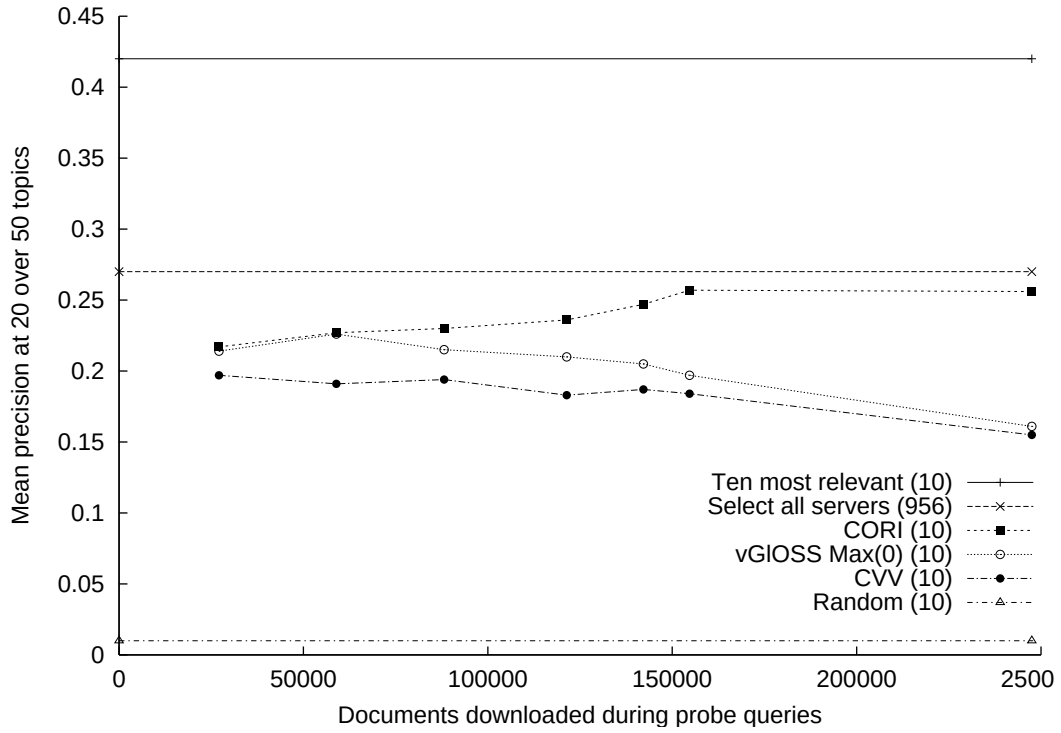


Figure 5.5: Selection effectiveness with probe queries and homogeneous servers. This is the same as Figure 5.3 except that it reflects an environment where every search server runs BM25.

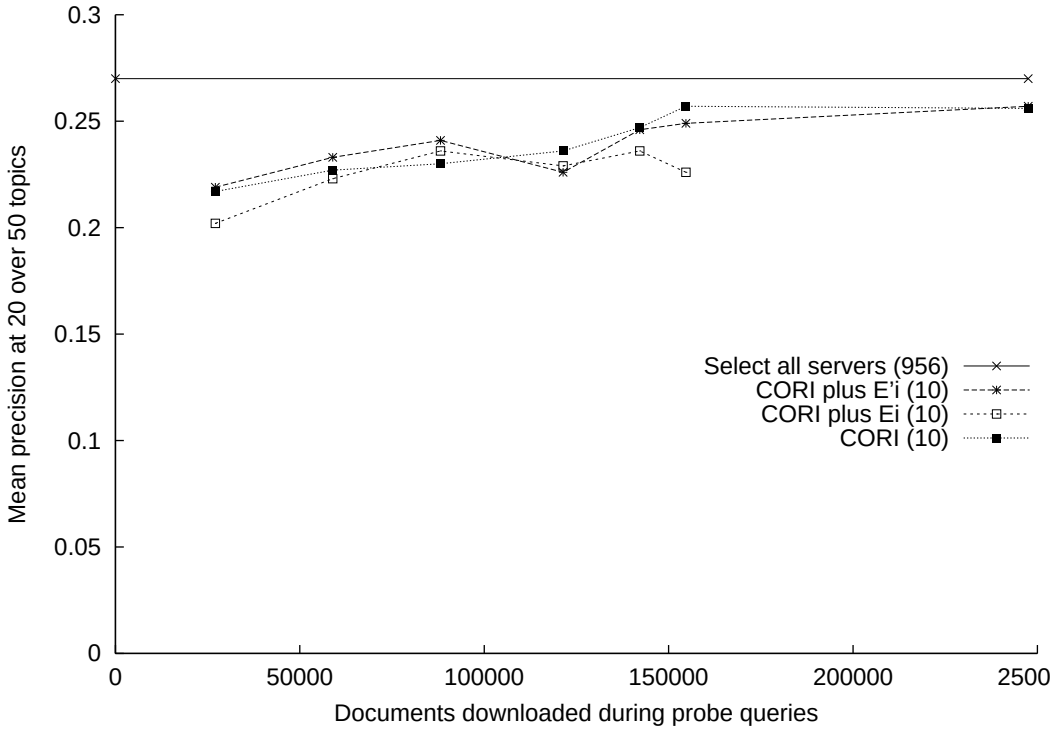


Figure 5.6: Selection effectiveness based on effectiveness estimation. This is the same as Figure 5.4 except that it reflects an environment where every search server runs BM25.

representative server configuration. The homogeneous configuration represents a situation where all 956 search servers run the same effective retrieval algorithm, Okapi BM25.

Figure 5.5 and Figure 5.6 show results much similar to Figure 5.3 and Figure 5.4. Data points move to the right because switching from Boolean to BM25 retrieval yields more probe query documents. Data point move up because BM25 servers are more effective, and the extra relevant documents make their way into the merged top 20.

CORI selection degrades slightly less sharply with fewer probe queries. This is borne out by statistical tests, which only show a significant difference between full information and the 10 probe query case. At all other levels of probing, effectiveness is indistinguishable from that in the full information case.

In the homogeneous configuration modification by effectiveness E_i or E_i' unsurprisingly yields no improvement. Effectiveness estimation was intended for differentiating between heterogeneous servers. vGlOSS and CVV improve more steadily with fewer probe queries, although vGlOSS dips at the 10 probe query level.

A very good selection of 10, based on relevance judgments, is still better than a selection of 956. Realistic selections of 10 still do not outperform a selection of all 956, although the best CORI points are statistically indistinguishable from the 956 server case.

These additional results reinforce the finding that sufficient probe queries and full cooperation allow similar levels of selection effectiveness. They also indicate that effectiveness estimation is best attempted in an environment where effectiveness differs between servers.

5.5 Conclusion

Hypothesis S1 holds: there is no significant effectiveness difference between selection based on sufficient probe queries and selection based on full information. This means that a broker can perform effective selection without relying on server cooperation, in an environment such as the World Wide Web.

Hypothesis S2 does not hold: effectiveness estimation does not improve effectiveness. This may be due to the method of estimation, the method for modifying CORI or the testbed. Even with true effectiveness figures E_i', based on relevance judgments, only a small (but statistically significant) improvement in effectiveness was achieved.

Other results:

- CORI is best evaluated server ranking method, outperforming vGlOSS and CVV.
- All selection methods maintain a reasonable level of effectiveness based on probe queries, even on as few as 10 or 25. With fewer probe queries, CORI effectiveness degrades but vGlOSS and CVV do not always.
- Distributed retrieval over 10 servers per query is not as effective as retrieval over a central index with 100% coverage. However, in the Web case such an index is

unrealistic. For a 50% central index the distributed case fares better. In the 25% case, distributed search is superior.

When adding server selection to a broker such as Inquirus, it might even be possible to eliminate the probe query stage, instead basing selection on results returned during user queries. Building such a scheme which maintains effectiveness while being fair to all servers — it would be unfair if a server is never queried so never selected — is a good subject for future work. Other obvious future work would be to evaluate some of the other selection methods listed in Section 2.3.1 against CORI.

Chapter 6

Merging Experiments

The primary question in this chapter is whether merging using reference statistics is as effective as merging based on true statistics (hypothesis M1, Section 3.4). If so a search broker can address non-cooperating servers without sacrificing effectiveness. Another question is whether feature distance ranking outperforms other ranking algorithms such as Okapi BM25 over full document download (hypothesis M2) and partial document download (hypothesis M3). Results also show which rank, score and download based merging methods are most effective.

Section 6.1 describes experimental methods, divided into three subsections on modelling of search servers, the broker and the user. Section 6.2 presents results of the experiments. Section 6.3 discusses the implications of these results, particularly with respect to stated hypotheses, and compares the results to other published findings. Section 6.4 presents further experiments, to test results generality. Section 6.5 outlines conclusions.

6.1 Method

Experiments here evaluate a number of merging methods using TREC ad hoc test collections. Section 6.1.1 describes search servers and their retrieval results. Section 6.1.2 describes the broker's selection and merging methods. Section 6.1.3 describes the user model.

6.1.1 Search servers: documents and retrieval

Instead of partitioning test collection documents amongst servers S and simulating retrieval, these experiments use methods described in Chapter 4 to simulate input rankings from S'. In this case $|S'| = 5$, and the five servers are the TREC-6 test collection documents partitioned by source:

- The Financial Times (1991–1994),
- Federal Register (1994),
- Congressional Record (1993),

- Foreign Broadcast Information Service (early 1990's) and
- LA Times (1989–1990).

The five have between 27 992 and 210 158 documents each, much larger than the 956 servers of the previous chapter which average 259 documents. Therefore while the previous chapter modelled site-search servers, this chapter models larger search servers, perhaps on the Web, which each cover content corresponding to some publication over the course of a few years.

Retrieval is of the top 30 results from each server, giving merged lists of at most 150 documents. This is enough to make the merging task interesting, without going beyond a size which would be practical in real document download based merging. Since official TREC runs contain 1000 documents per topic, it is almost always possible to obtain a top 30 for each server.

For generating simulated retrieval results, five runs were chosen to represent a wide range of systems and strongly test the performance of merging methods under server heterogeneity. The runs were from:

- University of California Berkeley, based on logistic regression (`Brkly22`),
- Cornell University, based on the vector space model (`Cor6A3cll`),
- City University, London, based on the Okapi probabilistic model (`city6at`),
- MDS RMIT, using a limited-context vector space model (`mds602`), and
- Queens College CUNY, using a modified probabilistic model trained using a spreading activation network (`pirc7At`).

For example, MDS over LA Times documents is simulated by removing all non-LA Times documents from the MDS run.

Five systems with five servers gives 25 input rankings, and $5^5 = 3125$ possible server configurations, assuming any combination of retrieval algorithms is possible. The 3125 include five fully homogeneous configurations, where all servers use the same algorithm, and 120 configurations with all five different algorithms. No configuration seems more likely than another. It is also not realistic to assume some central coordinator decides on the configuration. Therefore a good merging method should be able to deal with any configuration. For this reason these experiments evaluate merging methods across all 3125 possible configurations.

6.1.2 The broker: selection and merging

Selection is not required since the five modelled servers are assumed to have already been selected from a larger set. However, some merging methods require an indication of server promise, to give preference to good server's results in merging. Here such methods are supplied with weights generated using a simple server selection method, a $df \cdot isf$ weighting (Equation 2.4). The selection experiments in the previous

chapter made use of the best merging method described here, coming chronologically after these experiments. These experiments do not use CORI selection, because at the time it was not clear which selection method was best. However, see Section 6.4 for investigation of CORI server promise in this testbed.

Merging methods were described in Section 2.3.2 but those evaluated are summarised again here:

Interleaving Documents D_M are sorted in increasing order of server-assigned rank. In cases where incoming ranks are equal, order is arbitrarily determined by Perl's sort routine.

Yuwono and Lee interleaving Documents are interleaved unevenly such that the gap between documents from a server is inversely proportional to server promise.

V interleaving The Voorhees method for interleaving requires the input rankings to have length proportional to server selection score, then for documents to be removed from lists at random with probability proportional to the list's remaining size. Here an approximation called V Interleaving is used. The largest server promise score is scaled to 30, and smaller scores are scaled proportionately, then scores are assigned to the appropriate incoming top 30 lists. Documents are removed from each list with a probability proportional to its remaining score. The score is reduced by one for each document removed, unless the reduction would be to zero for a non-empty list.

Raw scores Raw scores produced by different systems are not comparable. For example, MDS returns a vector match score ranging between zero and one, while Okapi returns a retrieval weight with no fixed maximum used to give a probabilistic ordering. In general scores are not even comparable if all servers use the same algorithm, because they will be based on local collection statistics. However, merging according to such unmodified scores is included here as a point of comparison for the methods based on score scaling.

Scaled scores The broker scales each server's document scores to range between two fixed values, then sorts according to scaled scores.

Weighted scaled scores A server's scaled scores are multiplied by its server promise score.

Random Documents D_M are ordered randomly to generate R_M.

Document download If the broker downloads documents D_M, it can generate its own ranking based on their content and some ranking algorithm. The following ranking algorithms are applied:

Inquirus Lawrence and Giles [1998] proposed a ranking algorithm which does not require collection statistics, (Equation 2.6).

BM25 The well known and highly effective Okapi BM25 document ranking algorithm (Equation 3.1). This may only be applied using reference statistics.

BM25 (no *df*) A modification of the Okapi BM25 document ranking algorithm with no document frequency information, and length normalisation according to a constant rather than the true average document length:

$$w_i = TF_i \times \frac{1}{2 \times (0.25 + 0.75 \times \frac{DL}{4096}) + TF_i}$$

where w_i is the relevance weight assigned to a document due to query term t_i (this weight is multiplied by the query weight of t_i), TF_i is the number of times t_i occurs in the document and DL is the length of the document in bytes.

tf · *idf* A simple ranking algorithm, with weighting based on a term's frequency within the document and the document frequency of that term within the collection (a collection statistic):

$$w_i = \frac{TF_i}{\log(CS/DF_i)}$$

which may only be applied using reference statistics.

Feature distance Ranking algorithms designed to degrade gracefully under partial document download, described in Section 3.3 with Equations 3.2 and 3.3. Feature distance merging may only be applied using reference statistics. The two equations were a result of hand tuning on 1% of the 3125 server configurations.

Every effort has been made to correctly implement the merging methods evaluated. This includes a reimplementation from first principles, described in Section 6.4, to uncover any bugs in the code.

Under document download the BM25, *tf* · *idf* and feature distance algorithms all require collection statistics. In this case the reference statistics are used, taken from a random 10% sample of the TREC-6 documents, sampled by taking every 10th file encountered on disk. This is consistent with the broker having access to some of the documents in question, which is not an unreasonable assumption since it is downloading them for merging. If no occurrences of a term were found in the 10% sample, the term's document frequency is assumed to be one.

6.1.3 User model

These experiments use the TREC-6 ad hoc test collection. User needs over 50 queries are described by TREC topics 301–350. A query is generated for a topic using unstemmed terms from the full topic description, with query weights corresponding to the number of times the term occurs in the topic. These queries are more detailed than those in the previous chapter, indicating a more in-depth searcher, willing to spend more time on query construction. This is consistent with the use of news/government documents and longer merged lists in this experiment. Relevance judgments are available for the documents and topics in question.

Method	Mean average precision: Reference Statistics	Mean average precision: True Statistics	Configurations where reference statistics are superior
$tf \cdot idf$	0.127	0.144	0%
BM25	0.186	0.188	0%
Feature distance w_B	0.189	0.182	100%
Feature distance w_A	0.191	0.191	58%

Table 6.1: Reference statistics (TREC-6). Document download based methods with and without reference statistics. Mean average precisions are over 3125 observations.

Since the output of the merging process is a ranked results list (R_M), standard information retrieval evaluation techniques can be applied in merging evaluation. The measure is average precision, particularly average precision at 150.

$$\textit{Average precision at } n = \frac{\sum_{i=1}^{num\_rel\_ret(n)} \frac{i}{rank(i)}}{num\_rel}$$

where $rank(i)$ is the rank of the ith relevant document, $num\_rel\_ret(n)$ is the number of relevant documents in the top n results, and *num_rel* is the number of relevant documents in the collections being searched.

Average precision was chosen because it is a standard TREC measure and because it allows the whole of R_M to be evaluated. By contrast, using precision at 150 would have resulted in all merging methods having the same effectiveness score, since in each case D_M is always the same. Early precision, for example precision at 20, would be a viable measure, but was not used here.

6.2 Results

Table 6.1 compares document download based methods with reference statistics and true collection statistics, over all 3125 configurations. It only includes methods which use some form of collection information. The rightmost column shows the proportion of the 3125 configurations where reference statistics are better than true statistics. For $tf \cdot idf$ and BM25, reference statistics provide worse performance, but performance is equivalent or better for both feature distance methods. For the three most effective methods, the effect of reference statistics is small.

Table 6.2 presents results for all merging methods over all 3125 configurations. The rightmost column lists the proportion of configurations where feature distance w_A merging was superior to other merging methods. Since most numbers are very high, usually 85% or above, no tests of statistical significance need be applied. The table

Method	Information	Average Precision Mean	Average Precision Standard Deviation	Configurations where w_A is superior
Random	None	0.062	0.005	100%
Inquirus	C	0.085	0.008	100%
Interleaving	R	0.120	0.011	100%
Raw scores	S	0.123	0.038	97%
$tf \cdot idf$	CX	0.127	0.012	100%
Yuwono Lee interleaving	RW	0.131	0.012	100%
Scaled scores	S	0.134	0.016	100%
V interleaving	RW	0.135	0.013	100%
Weighted scaled scores	SW	0.151	0.015	100%
BM25 (no df)	C	0.177	0.007	100%
BM25	CX	0.186	0.009	85%
Feature distance w_B	CX	0.189	0.011	68%
Feature distance w_A	CX	0.191	0.010	0%

Table 6.2: All merging methods (TREC-6). Means and standard deviations of average precision are over 3125 observations. The proportion of configurations for which w_A was superior is also listed. Merging information used by the broker; S: Scores, R: Ranks, W: Server promise weight, C: Document content, and X: Reference Statistics.

Method	Mean average precision Full download	Mean average precision First 4 kB	Drop in mean average precision
Inquirus	0.085	0.127	-0.042
$tf \cdot idf$	0.127	0.129	-0.002
BM25 (no df)	0.177	0.166	0.011
BM25	0.185	0.172	0.013
Feature distance w_B	0.189	0.176	0.013
Feature distance w_A	0.191	0.173	0.018

Table 6.3: Feature distance with partial download (TREC-6) Effectiveness of ranking algorithms using full and partial document download. Mean average precisions are over 3125 observations.

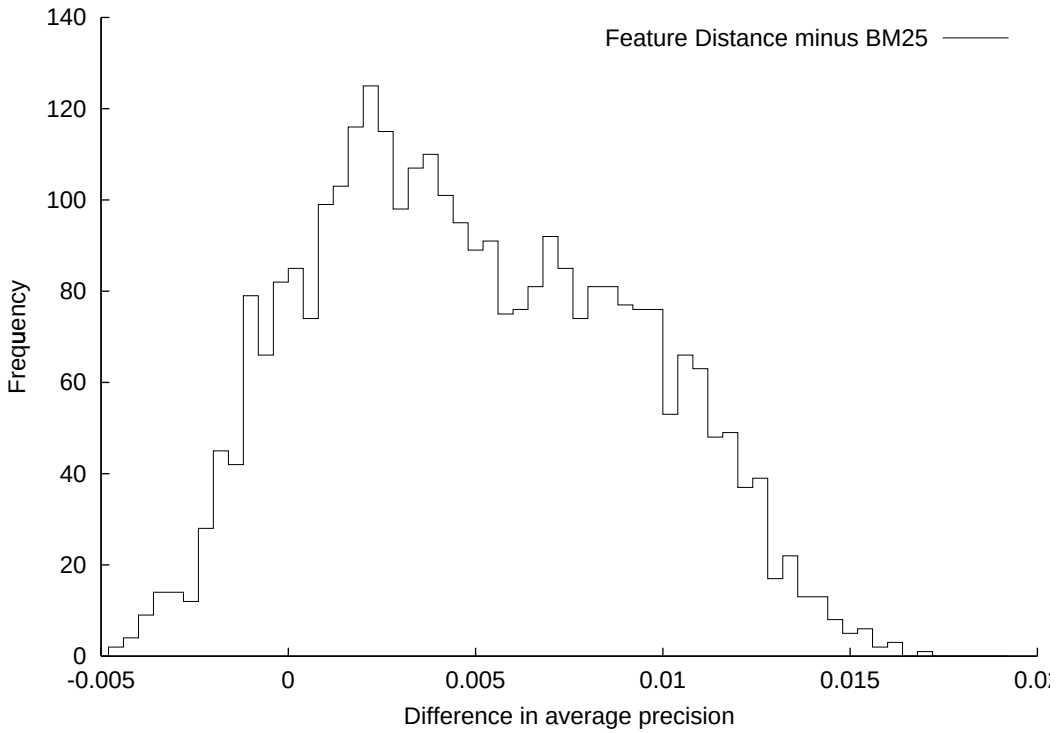


Figure 6.1: Feature distance vs BM25. Histogram of pairwise differences in mean average precision between feature distance w_A and BM25 over 3125 configurations. The improvement due to the feature distance, though consistent, is very small.

also shows a number of other results. Random merging is the worst method. The top three methods are based on document downloads and reference statistics, while the fourth is a download based method without collection statistics. Score based methods outperform rank based methods. Modification according to server selection scores improves score based merging (weighted scaled scores) and rank based merging (V interleaving and Yuwono and Lee interleaving).

Table 6.3 tests document download based methods under full document download and under partial download conditions. Partial downloads are of at most 4k of each document. For the more effective merging methods there is a drop in effectiveness in almost all cases. Furthermore, the drop for feature distance merging is worse than that of Okapi BM25. The Inquirus and *tf.idf* merging methods actually improve or stay the same, but are not highly effective to begin with.

Figure 6.1 is a histogram of pairwise differences between feature distance and BM25 merging over all 3125 configurations. It shows that differences are small but consistently in favour of feature distance.

6.3 Discussion

The most important hypothesis tested here is hypothesis M1, that reference statistics are as effective as true statistics in results merging. Table 6.1 shows merging effectiveness at very similar levels with reference statistics and true statistics. In the case of feature distance w_B, reference statistics actually improve effectiveness, although the reason for this is not clear. For w_A the results are better in 60% of cases and on average there is no noticeable drop in effectiveness. For Okapi BM25 there is a slight drop in effectiveness, in 95% of cases, however, the change in effectiveness is unlikely to be noticeable by a user. Similarly for *tf.idf* there is a fall in effectiveness, but perhaps not one noticeable to the user. In any case, merging using more effective methods is not harmed by switching to reference statistics.

This result means that effective merging may be achieved without collating true collection statistics. Since collation requires cooperation from servers and extra network communication, reference statistics are preferable. Even in cooperative merging, the servers could use a standard set of reference statistics to save on collation effort.

Although propagation of true collection statistics is not necessary, collation of information about documents D_M still is. Collating document information from cooperating search servers, for example using term frequency information provided through STARTS [Gravano et al. 1997], will always be more efficient than document download. This is because transmission of a server's pertinent scores and statistics requires a single small network communication. Downloading a server's documents from the appropriate document servers requires multiple, larger network transmissions. However, the many users of Inquirus find its document download times acceptable. In addition, downloading documents allows results validation, summarisation and caching (see Section 3.1). Document download also guards against dishonest search servers, which might return inflated scores or statistics to promote their documents in the merged list.

Hypothesis M2 was that feature distance merging is superior to Okapi BM25 merging. This is shown to hold both in the rightmost column of Table 6.2 and in Figure 6.1. However, feature distance was tuned to TREC-6 and a 1% sample of configurations in this merging task, while BM25 was not. In addition, the improvement is so slight that it would be imperceptible to a user. The result is interesting only because the improvement is so consistent (Figure 6.1). This suggests two things. First, feature distance might be worth future investigation, perhaps in developing even more effective formulations and some theoretical model for this type of retrieval. Second, it may be that merging and retrieval, although both document ranking problems, are slightly different. This would explain why BM25 is not optimal for merging. It may be possible to develop specialist merging algorithms which are more effective at merging than any document ranking algorithm for retrieval over large document sets. If this were the case, such specialist algorithms for reordering small document lists might also have application in other results presentation tasks.

Hypothesis M3 was that feature distance effectiveness degrades gracefully under partial document download. Partial download allows the broker to present its merged

results more quickly by cutting off long documents at some fixed time or bandwidth limit. In this case a 4k limit was used to test the degradation of different methods. The result was that partial download almost always harms effectiveness, and that it harms feature distance methods more than Okapi BM25, so the hypothesis does not hold. In addition the decrease from 0.191 to 0.173 may be noticeable to a user. To determine whether such a decrease is worthwhile, further experiments would have to be conducted, into the time and bandwidth savings which accompany partial download. For example, if results come twice as quickly, a drop in effectiveness might be justified for certain applications.

The remainder of this section discusses other interesting features and implications of the results. Table 6.2 shows that score based merging is superior to rank based merging (except for raw scores which are ineffective as expected). Modification by $df.isf$ slightly improves both score and rank based methods. The superiority of score based methods might be explained by a score's ability to capture more fine grained comparisons between documents. For example, two documents with adjacent ranks might be revealed by their scores to have identical or very different levels of matching against the current query. The implication is that generally applicable search brokers which do not download documents should use score rather than rank based merging. The improvement using crude $df \cdot isf$ on score based merging suggests that further improvements might be possible using more effective selection scores (again see Section 6.4).

Noticeable in Table 6.3 is that two merging methods improved under partial download, although not enough to overtake other methods. A contributing factor might be their lack of document length normalisation. Without length normalisation, an algorithm can be "confused" by documents of varying length. This could make them ineffective on full documents, and better in a partial download situation where long documents have been truncated. It is also interesting to note that while partial document download harms merging effectiveness, downloading part of a server using probe queries in the previous chapter did not harm selection effectiveness as badly.

Finally, these results in general correlate with the results of Callan, Lu, and Croft [1995]. They found that rank interleaving was the worst method while comparable scores (cooperative merging) was the most effective. The difference is that their weighted scores method was as effective as cooperative merging, perhaps a reflection of the use of a homogeneous retrieval algorithm and a high quality server promise indicator (CORI scores). Their raw scores method was of medium effectiveness, but has no analogue here, being based on homogeneous retrieval methods.

6.4 Further experiments

The experiments presented so far this chapter and in [Craswell et al. 1999], have three potential flaws which are addressed in this section.

The first potential flaw was in the testbed software. The initial version of the software provided the results presented in Craswell, Hawking, and Thistlewaite [1999].

This testbed was reimplemented with the benefit of more experience in early 2000. The goal was to see if the new results match the original results and, if not, to find out why.

Reimplementation confirmed that most results in the original paper were correct. However, it turned up one bug and one bad decision in the old code (plus a a few other very small differences which are not detailed here). The bug was that the effectiveness of interleaving was inflated to roughly equal that of Yuwono and Lee merging. Table 6.2 has been updated with the correct interleaving result. Note, a difference in V interleaving between results reported here and the original paper is not a bug. It is due to the nondeterministic nature of the algorithm.

The bad decision concerned the scaling of scores, affecting results for scaled scores and weighted scaled scores. The original decision was to scale scores from each retrieval system to the range 0–1. However, correct behaviour is to scale scores from each *server* in this way. For example, if two servers run the same retrieval algorithm, with score ranges 0.2–0.5 and 0.3–0.7, the original behaviour was to scale within the range 0.2–0.7. The correct behaviour is to scale one server within 0.2–0.5 and the other within 0.3–0.7. This is because a broker in practice would most likely not know that two servers run the same retrieval system so have semi-comparable scores. In addition, in this testbed, two servers running the same algorithm have precisely comparable scores because they originate from the same official TREC run. Scaling scores in the original way therefore increased the effectiveness of servers on very homogeneous configurations. For example, when all servers run Okapi BM25, the merged list is based on comparable scores. Again, Table 6.2 has the updated results for scaled and weighted scaled scores.

The second potential flaw was that feature distance algorithms were tuned using the test collection (TREC-6) and a roughly 1% sample of the 3125 configurations. This made feature distance weightings specific to TREC-6 and the merging problem, while BM25 is a more general algorithm. The reimplementation of the merging code also increased its generality, making it easy to run the same experiment on the TREC-3 ad hoc test collection. This was done and the experiments are reported in this section.

The third potential flaw was the crude server promise figure ($df \cdot isf$) which might have disadvantaged weighted scaled scores, V interleaving and Yuwono and Lee interleaving. However, initial experiments with CORI server promise (belief values, Equation 2.1, page 16) showed it to be incompatible with the current testbed. Here, most servers contain most query terms, so usually $|S| = SF_k = 5$. In CORI selection this means that T is very small, so the final belief value does not stray far from $d_b = 0.4$. A method such as V interleaving, which bases its merging on relative server promise, will give roughly equal weight to all servers because server promise scores for five servers are in the range 0.40–0.42. This problem stems from the fact that CORI is noncommittal on high-SF terms, combined with the fact that this testbed does not model the broader range of servers S. Because of this, the $df \cdot isf$ server promise figure is retained for these experiments.

The experimental method is the same, but replaces TREC-6 with TREC-3. The servers are:

Method	Mean average precision		Configurations where reference statistics are superior
	Reference Statistics	True Statistics	
$tf \cdot idf$	0.104	0.104	30%
Feature distance w_B	0.136	0.135	93%
Feature distance w_A	0.138	0.138	27%
BM25	0.147	0.147	20%

Table 6.4: Reference statistics (TREC-3). Document download based methods with and without reference statistics. Mean average precisions are over 3125 observations.

- Wall Street Journal (1987–1992),
- Associated Press (1988–1989),
- Ziff-Davis,
- Federal Register (1988–1989), and
- US DOE abstracts.

The retrieval systems are from the same groups, except that to increase the variety of search results the MDS system is replaced by PADRE, using a distance based metric for ranking [Hawking and Thistlewaite 1994].

New results concerning hypothesis M1 are presented in Table 6.4. They support even more strongly than those in Table 6.1 that use of reference statistics does not harm effectiveness. Variation in the TREC-6 testbed results is not reflected here, even though precisely the same code is used in both experiments. The explanation for this is not obvious.

New results concerning hypothesis M2 are presented in Table 6.5. These results confirm that tuning on 1% of TREC-6 server configurations made feature distance formulations w_A and w_B specific to that testbed. On TREC-3 feature distance performs less well than BM25, although is still more effective than rank and score based merging methods.

Merging methods are less effective in the TREC-3 testbed. This is difficult to explain because the TREC-6 input rankings generally contain fewer relevant documents than the TREC-3 input rankings. One explanation is that although there are more relevant documents in TREC-3 runs, they tend to be concentrated in AP and WSJ results lists (Table 6.6). This skew might make the merging task more difficult, particularly for methods based on ranks and scores.

V interleaving and Yuwono and Lee interleaving both perform less effectively than

Method	Information	Average Precision Mean	Average Precision Standard Deviation	Configurations where w_A is superior
Interleaving	R	0.080	0.014	100%
Inquirus	C	0.084	0.014	100%
V interleaving	RW	0.094	0.017	100%
Yuwono Lee interleaving	RW	0.095	0.017	100%
Scaled scores	S	0.097	0.020	100%
Raw scores	S	0.098	0.034	93%
$tf \cdot idf$	CX	0.104	0.017	100%
Weighted scaled scores	SW	0.115	0.023	99%
Feature distance w_B	CX	0.136	0.020	73%
Feature distance w_A	CX	0.138	0.019	0%
BM25 (no df)	C	0.146	0.019	2%
BM25	CX	0.147	0.019	0%

Table 6.5: All merging methods (TREC-3). Means and standard deviations of average precision are over 3125 observations. The proportion of configurations for which w_A was superior is also listed. Merging information used by the broker; S: Scores, R: Ranks, W: Server promise weight, C: Document content, and X: Reference Statistics.

Server	Mean Precision at 30
AP (TREC-3)	0.43
WSJ (TREC-3)	0.41
FR (TREC-3)	0.06
DOE (TREC-3)	0.04
ZF (TREC-3)	0.04
FT (TREC-6)	0.22
FBIS (TREC-6)	0.18
LA (TREC-6)	0.17
CR (TREC-6)	0.07
FR (TREC-6)	0.04

Table 6.6: Effectiveness of input rankings, TREC-3 vs TREC-6. Simulated input rankings in the TREC-3 testbed have greater variation in effectiveness. These figures are a document server's mean precision at 30 across all five retrieval systems and 50 topics.

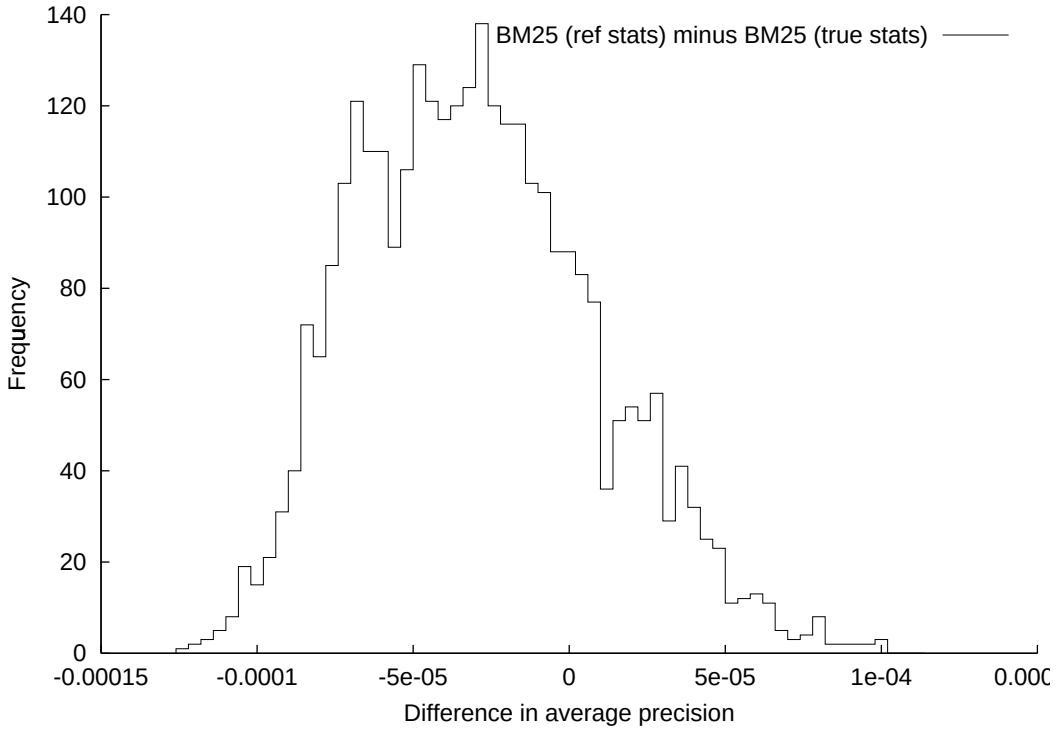


Figure 6.2: BM25 (ref stats) vs BM25 (true stats). Histogram of pairwise differences between BM25 with reference statistics and BM25 with true collection statistics over the TREC-3 testbed. As noted in Table 6.4, reference statistics are only superior 20% of the time. However, the means come out exactly the same, and this histogram shows the differences are very small.

score based methods over TREC-3 documents. Surprisingly, scaled scores do not outperform raw scores in the TREC-3.

6.5 Conclusion

Reference statistics do not cause a drop in effectiveness for the best merging methods, so hypothesis M1 holds. The implication is that an operational search broker need not rely on servers exporting their collection statistics, at least on grounds of effectiveness. Hypothesis M2 does hold for TREC-6, but this is due to tuning on 1% of testbed configurations. Rather than recommending use of feature distance merging in an operational broker, these results suggest that further exploration of feature distance formulations may be fruitful. Hypothesis M3 does not hold, because feature distance's degradation is worse under partial download than BM25's. For both ranking methods, partial download yields a significant drop in effectiveness. Broker designers considering merging based on partial downloads should consider this drop in effectiveness and measure the possible gain in efficiency.

Chapter 7

Conclusions

Methods for distributed information retrieval are usually introduced in the context of either general applicability or effectiveness evaluation, but not both. Studies which concentrate on applicability usually describe a method's implementation in a real Web search broker, but without evaluation. Studies which concentrate on effectiveness usually perform evaluation, but introduce methods which rely on cooperation from search servers. Methods which require cooperation are restricted in application and seldom used in practice.

Probe queries and reference statistics allow a broker to be both generally applicable and effective. General applicability stems from having minimal requirements of servers: search servers need only return search results and document servers need only return documents. Effective selection is achieved with probe queries which allow the application of CORI server ranking. Effective merging is achieved with reference statistics which allow the application of Okapi BM25 document ranking.

7.1 Methods

Probe queries allow a broker to learn about server term occurrence statistics, then apply a server ranking method such as CORI. Evaluation shows that selection based on sufficient probe queries is as effective as selection based on servers' true term occurrence statistics. This indicates that a broker can perform effective selection without relying on cooperation from search servers.

Effectiveness estimation allows a broker to learn about a server's ability to return relevant documents, then modify its server selection to favour more effective servers. Evaluation of an initial estimation method over heterogeneous servers provides no significant improvement in effectiveness. Even use of true (not estimated) effectiveness figures provides only a slight improvement. Further investigation is required.

Reference statistics allow a broker to apply an effective document ranking algorithm, such as Okapi BM25, without relying on server cooperation and collation of true collection statistics. Evaluation shows merging based on reference statistics is as effective as merging based on true statistics. This indicates that a broker can perform effective merging without relying on cooperation from search servers.

Feature distance document ranking is designed to be effective in situations where

merging is based on partially downloaded documents. Evaluation shows its effectiveness on full downloads to be promising. After tuning to a 1% portion of the TREC-6 configurations, it consistently outperformed BM25 on TREC-6 and came close on TREC-3. However, on partial downloads it degrades as much or more than BM25.

Combining probe queries and reference statistics would allow a broker to address a wide range of search servers without relying on their cooperation. The broker would be as effective as a broker which requires and receives full cooperation. So this thesis has successfully shown that a broker can be generally applicable without sacrificing effectiveness.

Effectiveness estimation and feature distance were less successful. They were designed to handle realistic Web conditions — with heterogeneous retrieval systems at servers and document download limits — but were not central to the current investigation of generally applicable, effective distributed search. For this reason, the thesis concentrates on probe queries and reference statistics. However, the initial results reported here indicate that the methods may be worthy of more detailed study in the future.

7.2 Other contributions

This thesis makes number of other contributions to distributed information retrieval research.

An important question when designing a search broker is with what kind of search servers will it interact. Studies have often described a broker interacting with a set of unspecified, homogeneous search servers, or with a small set of Web search engines. This thesis describes an Inquirus-like broker interacting with a large number of InvisibleWeb-listed servers (`http://www.invisibleweb.com/`). Such servers run heterogeneous retrieval algorithms and do not cooperate with brokers. They often index documents from a single document provider, corresponding to document set partitioning by source rather than by date or clustering [Xu and Croft 1999]. These characteristics have influenced the design of new methods and evaluation experiments.

The selection experiments are amongst the first to incorporate heterogeneous retrieval algorithms and use a large number of search servers partitioned by source. Partitioning by source also provides greater server size variation and topic skew than in previous experiments, which is consistent with real single-source Web search servers. Results show that CORI selection is more effective than vGlOSS and CVV. Other results show that a very good selection of ten servers outperforms a centralised index, and that a CORI selection of ten outperforms a central index of 25% of sources. The overall recommendation is that a generally applicable broker should apply CORI selection based on probe queries.

The merging experiments evaluate some generally applicable methods which have never before been compared. These experiments are also the first to incorporate heterogeneous retrieval algorithms and a large number of server configurations. Results show score based methods outperform rank based methods, and that modification of

such methods by server promise improves effectiveness. These results hold true over two test collections with 3125 server configurations each. Overall, it is recommended that a generally applicable broker should download documents. This allows the most effective merging plus document verification, summarisation and caching. If the broker does not download documents for efficiency reasons, it should apply score based merging methods, modified by server promise scores if available.

In selection evaluation methodology, concerns were raised about server merit evaluation. New merit definitions, based on the density of relevant documents and server retrieval effectiveness, were introduced which are better than the relevance based definition favoured in recent studies. It was also shown that server merit depends on which other servers have been selected so far. For this reason, server merit evaluation is avoided in this thesis.

In merging evaluation methodology, a method for generating simulated input rankings was suggested. The simulated inputs are similar but not identical to runs which would be produced by real search servers. They also make it simple to simulate rankings from widely varying systems without much chance of experimental error, in a way which can easily be repeated by other researchers.

7.3 The future

Given the above conclusions, a few avenues of research now become *less* interesting.

Methods which rely on server cooperation are less interesting. This thesis shows that cooperation is not necessary for effectiveness. Also, if servers are cooperating they could cooperate with a central indexer. By running an rsync [Tridgell and Mackerras 1996] server, a search server can eliminate most crawling costs. A single low-end machine can provide a central search service over millions of documents [Hawking et al. 1999]. Such a solution might be preferable to cooperating distributed search servers, which would require significant cost and effort in maintenance and compliance. It is only when documents are unavailable for central indexing — when cooperation is unavailable — that distributed search becomes interesting.

Efficient aggregation of true collection statistics for use in merging is less interesting. Reference statistics allow effective merging with no aggregation overhead. Also, evaluation of selection methods based on server merit is potentially incorrect. Until merit definitions are better understood, server merit evaluation results are less convincing.

In contrast, several interesting avenues of research have been raised.

Server merit definitions require further investigation. Server merit experiments are less complex than system level evaluation experiments, because they do not require the experimenter to implement retrieval and merging methods. Comparing the results of server merit and system level evaluation could be fruitful. If they agree, future experiments can safely use server merit evaluation. If server merit evaluation is found to be potentially misleading, it would be highly useful to streamline system level evaluation experiments by providing standard results lists, extracted statistics

and other information. Such a framework could be provided to interested TREC participants.

Effectiveness estimation showed some promise. It would be interesting to investigate situations in which such estimation is useful. For example, in cases where many servers index relevant documents, but some less effective servers are less likely to return them. Better methods for estimation and for using estimates in selection could then be developed.

Feature distance merging also showed some promise. It would be interesting to investigate various feature distance formulations, to determine experimentally which are most effective. It would also be useful to develop a more principled model which explains the success of the algorithm, and which might shed further light on its development and application.

Building a real search broker raises a number of interesting problems. Some search servers have volatile content, so the broker would need to not only learn from its probe queries, but adjust over time. It might be possible to eliminate the probe query stage altogether, by learning from ongoing document downloads during merging. However, such rolling learning would pose its own problems, including the problem of fairness, where a server which is never selected can never be selected.

Implementation of a real broker would also validate the broker use case presented in this thesis, that of addressing numerous small Web search servers. It would also raise other, practical problems. For example, in an environment such as the Web, methods which allow a broker to automatically discover and interface with new servers would be useful. When using an effective broker on real search servers, it would become clear how happy the server administrators are to have their server included or excluded from the broker's set. With a real broker, detecting and preventing "cheating" and "spamming" from search and document servers might even become an issue.

7.4 Overall conclusion

The major finding of this thesis is that a search broker can be effective without relying on cooperation from search servers. This is based on evaluation experiments which model Web search servers. The next stage in research is to build and evaluate such a broker.

Appendix A

Variation in Terminology

This brief appendix deals with variation in terminology.

Distributed information retrieval

There has been some variation in terminology in the field of distributed information retrieval. In cases where different terms have been used to describe very similar concepts, this thesis chooses one term and uses it consistently. The chosen terms are defined in Section 2.2.

This thesis uses the following terms:

- *Distributed information retrieval:* A term also used by Xu and Callan [1998], Baumgarten [1997] and French, Powell, Viles, Emmitt, and Prey [1998]. Has sometimes been known as networked information retrieval (for example [Voorhees and Tong 1997; Fuhr 1999]).
- *Server selection:* A term also used by Yuwono and Lee [1997] and Hawking and Thistlewaite [1999]. Has sometimes been known as collection selection (for example [Zobel 1997; Callan et al. 1995]), selection of sub-collections (for example [Baumgarten 1997]), text-source discovery (for example [Gravano et al. 1999]) and database selection (for example [French et al. 1998; Fuhr 1999]).
- *Results merging:* A term also used by Callan, Lu, and Croft [1995] and Hawking and Thistlewaite [1999]. Has sometimes been known as database merging (for example [Voorhees and Tong 1997]), collection fusion (for example [Fuhr 1999]) and subcollection fusion (for example [Baumgarten 1999]).

In each case, terms have been chosen which simply describe the process in question and conform to the use of basic distributed systems (client, server, broker) terminology.

The "collection"

This thesis uses the word *collection* to describe a set of documents over which a ranking algorithm might operate, for example using *collection statistics*. It also describes evaluation using a *test collection*.

Use of the word is avoided in the context of a "distributed collection". This is because the word has already been used in several different ways. It is unclear which is correct:

- All documents indexed by S a broker's search servers [Baumgarten 1997]. Some merging methods use collection statistics from all servers S.
- All documents indexed by S' currently selected servers [Gravano et al. 1997]. STARTS suggests merging based on collection statistics from servers S'.
- Documents indexed by s_i a single search server [Callan et al. 1995]. The broker performs collection selection.

There are several other natural definitions of the collection:

- All documents indexed by any search server. Each broker, by addressing a server set S, covers some subset of the whole distribued collection.
- All documents on any document server. A document need not be covered by a search server to be part of the distributed collection.
- The documents of one document server. "Each document server provides a different document collection."
- The documents of one document provider. For example, `http://news.altavista.com/` searches several news data collections, including Associated Press.

Recognising the difficulty of defining the distributed document collection, particularly in a Web context, this thesis avoids choosing any definition.

Bibliography

ANSI/NISO. 1995. Z39.50-1995 (Versions 2 and 3) Information Retrieval: Application Service Definition and Protocol Specification. (p. 12)

BAUMGARTEN, C. 1997. A probabilistic model for distributed information retrieval. In N. J. BELKIN, A. D. NARASIMHALU, AND P. WILLETT Eds., *Proceedings of the 20th annual international ACM SIGIR conference on research and development in information retrieval* (New York, July 1997), pp. 258–266. ACM Press. (pp. 21, 24, 89, 90)

BAUMGARTEN, C. 1999. A probabilistic solution to the selection and fusion problem in distributed information retrieval. In M. HEARST, F. GEY, AND R. TONG Eds., *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (Berkeley, CA, Aug. 1999), pp. 246–253. ACM Press, New York. (pp. 21, 24, 89)

BERNERS-LEE, T., FIELDING, R. T., NIELSEN, H. F., GETTYS, J., AND MOGUL, J. 1999. Hypertext transfer protocol – HTTP/1.1. Internet Engineering Task Force, RFC 2616.
`ftp://ftp.isi.edu/in-notes/rfc2616.txt`. (p. 5)

BERNERS-LEE, T., MASINTER, L., AND MCCAHILL, M. 1994. Uniform resource locators (URL). Internet Engineering Task Force, RFC 1738.
`ftp://ftp.isi.edu/in-notes/rfc1738.txt`. (p. 5)

CALLAN, J., CONNELL, M., AND DU, A. 1999. Automatic discovery of language models for text databases. In *Proceedings of the 1999 ACM International Conference on Management of Data (SIGMOD 99)* (New York, 1999). ACM. (p. 37)

CALLAN, J. P., LU, Z., AND CROFT, W. B. 1995. Searching distributed collections with inference networks. In E. A. FOX, P. INGWERSEN, AND R. FIDEL Eds., *Proceedings of the 18th annual international ACM SIGIR conference on research and development in information retrieval* (Seattle, Washington, July 1995), pp. 12–20. ACM Press. (pp. 4, 14, 16, 22, 24, 25, 30, 31, 33, 50, 79, 89, 90)

CALVÉ, A. L. AND SAVOY, J. 2000. Database merging strategy based on logistic regression. *Information Processing and Management 36*, 3, 341–359.

CHAKRAVARTHY, A. S. AND HAASE, K. B. 1995. NetSerf: Using semantic knowledge to find Internet information archives. In E. A. FOX, P. INGWERSEN, AND R. FIDEL Eds., *Proceedings of the 18th annual international ACM SIGIR conference on research and development in information retrieval* (Seattle, Washington, July 1995), pp. 4–11. ACM Press. (p. 22)

CLARK, K. L. AND LAZAROU, V. S. 1997. A multi-agent system for distributed information retrieval on the World Wide Web. WETICE97, Collaborative Agents in Distributed Web Applications, IEEE Computer Society Press. (p. 4)

CLEVERDON, C. 1997. The Cranfield tests on indexing language. In K. SPARCK-JONES AND P. WILLETT Eds., *Readings in Information Retrieval*, Chapter 2, pp. 47–59. Morgan Kaufmann. (p. 29)

CRASWELL, N., BAILEY, P., AND HAWKING, D. 1999. Is it fair to evaluate Web systems using TREC ad hoc methods? In *ACM SIGIR Workshop on Web Evaluation* (1999).
http://pastime.anu.edu.au/nick/pubs/sigir99ws.ps.gz. (p. 2)

CRASWELL, N., BAILEY, P., AND HAWKING, D. 2000. Server selection on the World Wide Web. In *Proceedings of the Fifth ACM Conference on Digital Libraries* (2000), pp. 37–46.
http://pastime.anu.edu.au/nick/pubs/dl00.ps.gz. (pp. 2, 32)

CRASWELL, N., HAINES, J., HUMPHREYS, B., JOHNSON, C., AND THISTLEWAITE, P. 1997. Aglets: A good idea for spidering? Proceedings of the 4th IDEA Workshop.
http://pastime.anu.edu.au/nick/pubs/idea.ps.gz. (p. 2)

CRASWELL, N., HAWKING, D., AND THISTLEWAITE, P. 1999. Merging results from isolated search engines. In J. RODDICK Ed., *Proceedings of the 10th Australasian Database Conference, Auckland, NZ* (January 1999), pp. 189–200. Springer-Verlag.
http://pastime.anu.edu.au/nick/pubs/adc99.ps.gz. (pp. 2, 33, 43, 79)

CROFT, W. B., MOFFAT, A., VAN RIJSBERGEN, C., WILKINSON, R., AND ZOBEL, J. Eds. 1998. *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (Melbourne, Australia, Aug. 1998). ACM Press, New York.

DANZIG, P. B., AHN, J., NOLL, J., AND OBRACZKA, K. 1991. Distributed indexing: A scalable mechanism for distributed information retrieval. In *Proceedings of the 14th International Conference on Research and Development in Information Retrieval* (1991), pp. 220. (p. 4)

DE KRETSER, MOFFAT, SHIMMIN, AND ZOBEL. 1998. Methodologies for distributed information retrieval. In *Proc. 18th International Conference on Distributed Computing Systems* (Amsterdam, May 1998), pp. 66–73. (pp. 21, 24)

DOLIN, R., AGRAWAL, D., AND ABBADI, A. E. 1999. Scalable collection summarization and selection. In E. A. FOX AND N. ROWE Eds., *Proceedings of the 4th ACM conference on digital libraries* (Aug. 1999), pp. 49–58. (p. 22)

DOLIN, R., AGRAWAL, D., DILLON, L., AND EL ABBADI, A. 1996. Pharos: A scalable distributed architecture for locating heterogeneous information sources. Technical Report TRCS96-05, Department of Computer Science, University of California, Santa Barbara, CA 93106. (p. 22)

DREILINGER, D. AND HOWE, A. E. 1997. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems 15*, 3 (July), 195–

222. (pp. 21, 26)

D'SOUZA, D. J., THOM, J. A., AND ZOBEL, J. 2000. A comparison of techniques for selecting text collections. In *Proceedings of the 11th Australasian database conference*, Volume 22,2 of *Australian Computer Science Communications* (Canberra, Australia, 2000), pp. 28–32. (pp. 21, 32)

FAN, Y. AND GAUCH, S. 1999. Adaptive agents for information gathering from multiple, distributed information sources. In *1999 AAAI Symposium on Intelligent Agents in Cyberspace* (March 1999). Stanford University. (pp. 21, 59)

FOX, E. A., INGWERSEN, P., AND FIDEL, R. Eds. 1995. *Proceedings of the 18th annual international ACM SIGIR conference on research and development in information retrieval* (Seattle, Washington, July 1995). ACM Press.

FRAKES, W. B. AND BAEZA-YATES, R. 1992. *Information Retrieval. Data Structures and Algorithms*. Prentice Hall, Upper Saddle River, NJ. (p. 8)

FRENCH, J., POWELL, A., CALLAN, J., VILES, C., EMMITT, T., PREY, K., AND MOU, Y. 1999. Comparing database selection algorithms. In M. HEARST, F. GEY, AND R. TONG Eds., *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (Berkeley, CA, Aug. 1999), pp. 238–245. ACM Press, New York. (pp. 17, 32, 49, 64, 65)

FRENCH, J. C., POWELL, A. L., VILES, C. L., EMMITT, T., AND PREY, K. J. 1998. Evaluating database selection techniques: A testbed and experiment. In W. B. CROFT, A. MOFFAT, C. VAN RIJSBERGEN, R. WILKINSON, AND J. ZOBEL Eds., *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (Melbourne, Australia, Aug. 1998), pp. 121–129. ACM Press, New York. (pp. 17, 30, 31, 32, 45, 46, 89)

FUHR, N. 1999. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems 17*, 3 (July), 229–249. (pp. 10, 89)

GAUCH, S., WANG, G., AND GOMEZ, M. 1996. ProFusion*: Intelligent fusion from multiple, distributed search engines. *The Journal of Universal Computer Science 2*, 9 (Sept.), 637–649.
`http://www.tisl.ukans.edu/~sgauch/papers/JUCS96.html`. (pp. 25, 26)

GRAVANO, L., CHANG, C.-C. K., GARCIA-MOLINA, H., AND PAEPCKE, A. 1997. STARTS: Stanford Proposal for Internet Meta-Searching. *SIGMOD Record 26*, 2 (June), 207–218. (pp. 3, 12, 78, 90)

GRAVANO, L., CHANG, K., GARCIA-MOLINA, H., LAGOZE, C., AND PAEPCKE, A. 1997. STARTS - Stanford protocol proposal for Internet retrieval and search.
`http://www-db.stanford.edu/~gravano/starts.html`. (p. 24)

GRAVANO, L. AND GARCIA-MOLINA, H. 1996. Generalising GlOSS to vector-space databases and broker hierarchies. In *Proceedings of the 21st VLDB Conference* (Zurich, Switzerland, 1996). Morgan Kaufmann, San Francisco CA. Also Stanford technical report CS-TN-95-21. (pp. 16, 30, 31, 45, 46)

GRAVANO, L., GARCIA-MOLINA, H., AND TOMASIC, A. 1994. The effectiveness of GlOSS for the text-database discovery problem. In *Proceedings of the 1994 ACM International Conference on Management of Data (SIGMOD 94)* (New York, 1994), pp. 126–137. ACM. (p. 16)

GRAVANO, L., GARCIA-MOLINA, H., AND TOMASIC, A. 1999. GlOSS: Text-source discovery over the Internet. *TODS 24*, 2, 229–264. (pp. 16, 89)

HARMAN, D. 1992. Ranking algorithms. In *Information Retrieval. Data Structures and Algorithms*, Chapter 14, pp. 363–392. Upper Saddle River, NJ: Prentice Hall. (pp. 24, 41)

HARMAN, D. 1999. Proc. eighth text retrieval conference (TREC-8). (pp. 2, 50)

HARMAN, D. K. Ed. 1994. *Proceedings of the Third Text Retrieval Conference (TREC-3)* (Gaithersburg, MD, November 1994). U.S. National Institute of Standards and Technology. NIST special publication 500-225.

HAWKING, D., CRASWELL, N., AND BAILEY, P. 1999. The ACSys TREC-8 experiments. To appear TREC-8. (pp. 2, 87)

HAWKING, D., CRASWELL, N., BAILEY, P., AND GRIFFITHS, K. 2000. Measuring the quality of public search engines. The Fifth Search Engine Meeting. April 10-11, 2000, Boston, Massachusetts.
`http://pastime.anu.edu.au/nick/pubs/SE00/`. (p. 2)

HAWKING, D., CRASWELL, N., AND THISTLEWAITE, P. 1998a. ACSys TREC-7 experiments. In E. M. VOORHEES AND D. K. HARMAN Eds., *Proceedings of the Seventh Text Retrieval Conference (TREC-7)* (Gaithersburg MD, November 1998), pp. 299–312. U.S. National Institute of Standards and Technology. NIST special publication 500-252. (pp. 2, 4, 56)

HAWKING, D., CRASWELL, N., AND THISTLEWAITE, P. 1998b. Overview of TREC-7 Very Large Collection Track. In E. M. VOORHEES AND D. K. HARMAN Eds., *Proceedings of the Seventh Text Retrieval Conference (TREC-7)* (Gaithersburg MD, November 1998), pp. 91–104. U.S. National Institute of Standards and Technology. NIST special publication 500-252. (pp. 2, 4)

HAWKING, D., CRASWELL, N., THISTLEWAITE, P., AND HARMAN, D. 1999. Results and challenges in Web search evaluation. In *Proceedings of WWW8, Toronto* (1999), pp. 243–252. Elsevier.
`http://pastime.anu.edu.au/nick/pubs/www8.pdf`. (p. 2)

HAWKING, D. AND THISTLEWAITE, P. 1994. Searching for meaning with the help of a PADRE. In D. K. HARMAN Ed., *Proceedings of the Third Text Retrieval Conference (TREC-3)* (Gaithersburg, MD, November 1994), pp. 257–267. U.S. National Institute of Standards and Technology. NIST special publication 500-225. (p. 81)

HAWKING, D. AND THISTLEWAITE, P. 1999. Methods for information server selection. *ACM Transactions on Information Systems. 17*, 1, 40–76. (pp. 10, 19, 22, 30, 32, 37, 89)

HAWKING, D., THISTLEWAITE, P., AND CRASWELL, N. 1997. ANU/ACSys TREC-6 experiments. In E. M. VOORHEES AND D. K. HARMAN Eds., *Proceedings of the Sixth Text Retrieval Conference (TREC-6)* (Gaithersburg, MD, November 1997), pp. 275–290. U.S. National Institute of Standards and Technology. (p. 2)

HAWKING, D., VOORHEES, E., CRASWELL, N., AND BAILEY, P. 1999. Overview of the TREC-8 Web Track. To appear TREC-8. (pp. 2, 29, 47, 53, 55, 58, 59)

HEARST, M., GEY, F., AND TONG, R. Eds. 1999. *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (Berkeley, CA, Aug. 1999). ACM Press, New York.

HOWE, E. D. 1999. Free online dictionary of computing. `http://foldoc.doc.ic.ac.uk/`. (p. 5)

KIRK, T., LEVY, A. Y., SAGIV, Y., AND SRIVASTAVA, D. 1995. The Information Manifold. In C. KNOBLOCK AND A. LEVY Eds., *Papers from the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments* (Menlo Park CA, March 1995), pp. 85–91. AAAI Press. Technical Report SS-95-08. (p. 22)

KIRSCH, S. T. 1997. Distributed search patent. U.S. Patent 5,659,732, Infoseek Corporation. `http://software.infoseek.com/patents/dist_search/patents.htm`. (p. 24)

KOSTER, M. 1994. A standard for robot exclusion. `http://web.nexor.co.uk/users/mak/doc/robots/norobots.html`. (p. 11)

LAWRENCE, S. AND GILES, C. L. 1998. Inquirus, the NECI meta search engine. In *Proceedings of the Seventh International World Wide Web Conference* (1998). `http://www7.scu.edu.au/programme/fullpapers/1906/com1906.htm`. (pp. 2, 4, 25, 26, 35, 59, 73)

LAWRENCE, S. AND GILES, C. L. 1999. Accessibility of information on the web. *Nature 400*, 107–109. (pp. 11, 27, 55, 59)

MAZUR, Z. 1994. Models of a distributed information retrieval system based on thesauri with weights. *Information processing & management. 30*, 1, 61. (p. 24)

MENG, W., LIU, K., YU, C., WU, W., AND RISHE, N. 1999. Estimating the usefulness of search engines. In *15th International Conference on Data Engineering* (Sydney, Australia, March 1999).

MENG, W., LIU, K.-L., YU, C. T., WANG, X., CHANG, Y., AND RISHE, N. 1998. Determining text databases to search in the Internet. In A. GUPTA, O. SHMUELI, AND J. WIDOM Eds., *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA* (1998), pp. 14–25. Morgan Kaufmann.

MILLER, G. A. 1995. WordNet. *Communications of the ACM 38*, 11 (Nov), 39–41. (p. 22)

RASMUSSEN, E. M. 1991. Introduction: Parallel processing and information retrieval. *Information Processing and Management 27*, 4, 255–263. (p. 4)

ROBERTSON, S. E. AND SPARCK JONES, K. 1976. Relevance weighting of search terms. *Journal of the American Society for Information Science 27*, 3, 129–146. (p. 24)

ROBERTSON, S. E., WALKER, S., HANCOCK-BEAULIEU, M., AND GATFORD, M. 1994. Okapi at TREC-3. In D. K. HARMAN Ed., *Proceedings of the Third Text Retrieval Conference (TREC-3)* (Gaithersburg, MD, November 1994). U.S. National Institute of Standards and Technology. NIST special publication 500-225. (pp. 4, 41)

SELBERG, E. AND ETZIONI, O. 1995. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 1995 World Wide Web Conference* (1995). `http://www.w3.org/Conferences/WWW4/Papers/169/`. (p. 35)

SELBERG, E. AND ETZIONI, O. 1997. The MetaCrawler architecture for resource aggregation on the Web. *IEEE Expert V*, N (January–February), 11–14. (pp. 10, 25, 26, 59)

SINGHAL, A. AND KASZKIEL, M. 1998. Practical and effective distributed retrieval. ACM SIGIR Workshop: Theory into Practice.

SINGHAL, A., SALTON, G., MITRA, M., AND BUCKLEY, C. 1995. Document length normalization. Technical Report TR95-1529, Department of Computer Science, Cornell University, Ithaca NY. (p. 41)

SMEATON, A. F. AND CRIMMINS, F. 1996. Using a data fusion agent for searching the WWW. Poster presented at the WWW6 Conference, Santa Clara, California, April 1997. (pp. 24, 26)

SPINK, A., BATEMAN, J., AND JANSEN, B. J. 1999. Searching the Web: A survey of EXCITE users. *Internet Research: Electronic Networking Applications and Policy 9*, 2, 117–128. (pp. 29, 59)

TOMBROS, A. AND SANDERSON, M. 1998. Advantages of query biased summaries in information retrieval. In W. B. CROFT, A. MOFFAT, C. VAN RIJSBERGEN, R. WILKINSON, AND J. ZOBEL Eds., *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (Melbourne, Australia, Aug. 1998), pp. 2–10. ACM Press, New York. (p. 35)

TRIDGELL, A. AND MACKERRAS, P. 1996. The rsync algorithm. ANU Computer Science Technical Report TR-CS-96-05. (p. 87)

VILES, C. L. AND FRENCH, J. C. 1995. Dissemination of collection wide information in a distributed information retrieval system. In E. A. FOX, P. INGWERSEN, AND R. FIDEL Eds., *Proceedings of the 18th annual international ACM SIGIR conference on research and development in information retrieval* (Seattle, Washington, July 1995), pp. 12–20. ACM Press. (p. 24)

VOGT, C. C. AND COTTRELL, G. W. 1998. Predicting the performance of linearly combined IR systems. In W. B. CROFT, A. MOFFAT, C. VAN RIJSBERGEN, R. WILKINSON, AND J. ZOBEL Eds., *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (Melbourne, Australia, Aug. 1998), pp. 190–196. ACM Press, New York. (p. 4)

VOORHEES, E. M. 1995. Siemens TREC-4 report: Further experiments with database merging. In D. K. HARMAN Ed., *Proc. Fourth Text Retrieval Conference (TREC-4)* (Gaithersburg, MD, Nov. 1995), pp. 121–130. U.S. National Institute of Standards and Technology. (pp. 20, 25, 37)

VOORHEES, E. M. AND HARMAN, D. K. Eds. 1998. *Proceedings of the Seventh Text Retrieval Conference (TREC-7)* (Gaithersburg MD, November 1998). U.S. National Institute of Standards and Technology. NIST special publication 500-? (p. 27)

VOORHEES, E. M. AND HARMAN, D. K. Eds. 1999. *Proceedings of the Eighth Text Retrieval Conference (TREC-8)* (Gaithersburg MD, November 1999). U.S. National Institute of Standards and Technology. NIST special publication. (pp. 29, 53)

VOORHEES, E. M. AND TONG, R. M. 1997. Multiple search engines in database merging. In *Proceedings of the 2nd ACM international conference on digital libraries* (New York, 1997), pp. 93–102. ACM. (pp. 31, 89)

W3C. 1998. HTML 4.0 specification. W3C REC-html40-19980424. `http://www.w3.org/TR/REC-html40/`. (p. 5)

WALCZUCH, N., FUHR, N., POLLMANN, M., AND SIEVERS, B. 1994. Routing and ad-hoc retrieval with the TREC-3 collection in a distributed loosely federated environment. In D. K. HARMAN Ed., *Proceedings of the Third Text Retrieval Conference (TREC-3)* (Gaithersburg, MD, November 1994), pp. 135–144. U.S. National Institute of Standards and Technology. NIST special publication 500-225. (p. 24)

XU, J. AND CALLAN, J. 1998. Effective retrieval with distributed collections. In W. B. CROFT, A. MOFFAT, C. VAN RIJSBERGEN, R. WILKINSON, AND J. ZOBEL Eds., *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (Melbourne, Australia, Aug. 1998), pp. 112–120. ACM Press, New York. (pp. 21, 31, 50, 89)

XU, J. AND CROFT, W. B. 1999. Cluster-based language models for distributed retrieval. In M. HEARST, F. GEY, AND R. TONG Eds., *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (Berkeley, CA, Aug. 1999), pp. 254. ACM Press, New York. (pp. 21, 31, 32, 86)

YAGER, R. R. AND RYBALOV, A. 1998. On the fusion of documents from multiple collection information retrieval systems. *Journal of the American Society for Information Science 49*, 13, 1177–1184.

YUWONO, B. AND LEE, D. L. 1997. Server ranking for distributed text retrieval systems on the Internet. In R. TOPOR AND K. TANAKA Eds., *DASFAA '97* (Melbourne, April 1997), pp. 41–49. World Scientific, Singapore. (pp. 19, 25, 30, 31, 49, 58, 65, 89)

ZOBEL, J. 1997. Collection selection via lexicon inspection. In *Proceedings of the Australian Document Computing Symposium 1997* (Melbourne, Australia, March 1997). Department of Computer Science, RMIT, Melbourne. (pp. 21, 30, 32, 89)