Discover: A Resource Discovery System based on Content Routing

Mark A. Sheldon[1], Andrzej Duda[2],
Ron Weiss[1], David K. Gifford[1]

[1] Programming Systems Research Group, MIT Laboratory for Computer Science,
Cambridge, MA 02139, USA
[2] CNRS-IMAG 38610 Gières, France

Abstract. We have built an HTTP based resource discovery system called *Discover* that provides a single point of access to over 500 WAIS servers. Discover provides two key services: *query refinement* and *query routing*. Query refinement helps a user improve a query fragment to describe the user's interests more precisely. Once a query has been refined and describes a manageable result set, query routing automatically forwards the query to the WAIS servers that contain relevant documents. Abbreviated descriptions of WAIS sites called *content labels* are used by the query refinement and query routing algorithms. Our experimental results suggest that query refinement in conjunction with the query routing provides an effective way to discover resources in a large universe of documents. Our experience with query refinement has convinced us that the expansion of query fragments is essential for using a large, dynamically changing, heterogenous distributed information system.

1 Introduction

Locating and accessing information in the rapidly growing, heterogeneous, distributed collection of data sources available via the World Wide Web [4] is a difficult problem of growing importance. The servers on the Web represent a vast potential resource. Unfortunately, neither the organization of the data nor the available tools for associative access to the data is adequate for quickly discovering relevant information.

A general, usable system that provides associative access to all this data must meet certain criteria: It must offer a single point of contact that provides uniform access to all the information available on the Web. It must have reasonable performance in processing user queries. It must address issues of scalability in terms of storage requirements at any of the system's distributed components. It must also address scalability issues in terms of network communications, by efficiently and selectively accessing the large and rapidly growing number of information servers. Finally, it must help the user locate relevant information. In order to do this, the system should provide recommendations for refining user queries and help the user manage and understand the complexity of the information space.

In a large, rapidly evolving network of information providers, there are no expert users because any user's knowledge is quickly out of date. Users will inevitably submit queries with enormous result sets that are likely to adversely impact system performance and inundate the user with unwanted material. Any system that provides global information access must therefore help the user formulate queries that will return more useful results and that can be processed efficiently. A mechanism that recommends such query modifications is called *query refinement*.

Because the set of available information servers is continuously changing, a user requires a single point of contact. Such a system must help the user identify the current set of relevant information providers. It must further allow the user to browse the organization of the information space, and learn about information providers. Finally, the system should be capable of efficiently searching the set of relevant information providers when it is feasible to do so, and merge the results for the user. This function is called *query routing*. Query Routing masks the distribution and heterogeneity of the information space.

The Content Routing System architecture combines all of the above characteristics. It uses compact descriptions of information servers, called *content labels* to accomplish this in an efficient and scalable

This work was supported by the Defense Advanced Research Projects Agency and the Department of the Army under contract DABT63-92-C-0012.

manner [19]. A previous prototype was the subject of [6] (a companion paper to [19]). This prototype was based on the Semantic File System protocol [10].

Discover[1] is a new prototype Content Routing System that provides associative access to over 500 Wide Area Information Servers (WAIS) [13] via the World Wide Web. Discover is the first tool on the Web known to the authors to offer query refinement and query routing. Our work also represents the first attempt known to us to build a transparent, user-guided distributed information system on top of WAIS servers. Our experience with Discover over the last year and a half suggests that Content Routing Systems are practical to implement and useful.

In the remainder of this paper, we review related work (Section 2), describe the content routing architecture and query refinement (Section 3), present the Discover implementation (Section 4), and offer our conclusions and directions for future work (Section 5).

2 Related work

Related work can be broken down into the following categories: Web robots, subject directories, network-based information retrieval systems, and network-based resource discovery systems.

Query refinement is a unique component of our design that draws on our past experience with the Community Information System [9]. In the Community Information System, a simple theorem prover assisted the user in choosing query terms that guaranteed the query could be satisfied at available newswire databases.

Web robots like the Web Crawler [17], WWW Worm, ALIWEB [14], and the RBSE Spider [7], gather information about resources on the web for query-based access. However, these systems are not scalable because they use a global indexing strategy, *i.e.,* they attempt to build one database that indexes everything. They do not provide any organization of the search space, and they do not give the user any guidance in query formulation. These systems overburden network resources by transmitting entire documents, rather than the index data, or better still, content labels. Furthermore, a content routing system allows greater autonomy to each information provider and content router to tailor its indexing mechanisms and facilities using local knowledge. Veronica [12] is a discovery system that maintains an index of document titles from Gopher [1] menus, and it suffers from the same limitations as the Web search systems.

Subject-based directories of information, *e.g.,* Planet Earth[2] and the NCSA Meta-Index[3], provide a useful browsable organization of information. These would be useful paradigms for organizing particular hierarchies in a Content Routing System. As the information content grows, it becomes cumbersome to browse them and discover relevant information in these systems without query routing and refinement. At present, these systems are rather *ad hoc* and static, requiring manual update and maintenance.

Network-based information retrieval systems provide associative access to information on remote servers. WAIS [13] combines full text indexing of documents with network-based access. However, there is no facility for routing queries to relevant databases and merging results, nor is there any mechanism for suggesting query terms to users. The Archie system [8] polls a fixed set of FTP sites on the Internet. A query yields a set of host/filename pairs which is then used to retrieve the relevant files manually. The Conit system [15, 16] provides a uniform user interface to a large number of databases accessible from several retrieval systems. User queries are translated into commands on the different systems. However, there is no support for the automatic selection of relevant databases for a user's query.

Network-based resource discovery systems like Harvest and GLOSS, gather information about other servers and provide some form of query-based mechanism for users to find out about servers relevant to a request. Harvest [5] builds on our work on content routing by providing a modular system for gathering information from servers and providing query-based browsing of descriptions of those servers. A broker is a kind of content router, and a SOIF (Structured Object Interchange Format) object is a kind of simple content label. However, Harvest has no clear architecture for composing brokers. The Harvest Server Registry is like the WAIS directory of servers: It does not appear to support interaction with multiple servers by query routing and merging of result sets, rather it supports the ability to click on a broker and

[1] `http://discover.imag.fr/discover.html`

[2] `http://white.nosc.mil/info.html`

[3] `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/MetaIndex.html`

query it. There is no query refinement capability or even enumeration of field names or values. The user must browse through the SOIF objects to get hints on how to construct queries.

The GLOSS system [11] also provides a mechanism for finding servers relevant to a query, but it uses a probabilistic scheme. GLOSS characterizes the contents of an information server by extracting a histogram of its words occurrences. The histograms are used for estimating the result size of each query and are used to choose appropriate information servers for searching. GLOSS compares several different algorithms using the histogram data. The Discover prototype does not fit into any of GLOSS's categories, because it does not use document frequency as a measure of server relevance. Moreover, GLOSS's estimates of query result set sizes are not at all accurate. GLOSS does not search remote databases, it only suggests them to the user who must search them and retrieve documents manually.

3 Content Routing

The goal of a Content Routing System is to provide efficient associative access to the contents of a very large, heterogeneous, distributed set of information providers. Such a system must be designed to scale as the network of information servers grows and must provide two key services: query refinement and query routing.

This section reviews the architectural structure and the user interface operations of a Content Routing System. We then discuss some general approaches to query refinement and finish with a description of query routing.

3.1 System organization

A Content Routing System combines a large, distributed set of information providers into a single coherent framework. A Content Routing System consists of a hierarchical network (a directed acyclic graph) with conventional information providers at the leaves and *content routers* as mediating nodes. (See Figure 1.) The end-point information providers support query-based access to their documents. At a content router, a user may browse and query the information about providers registered at that content router as well as make use of the router's facilities for query refinement and routing. A content router may also provide access to conventional documents.

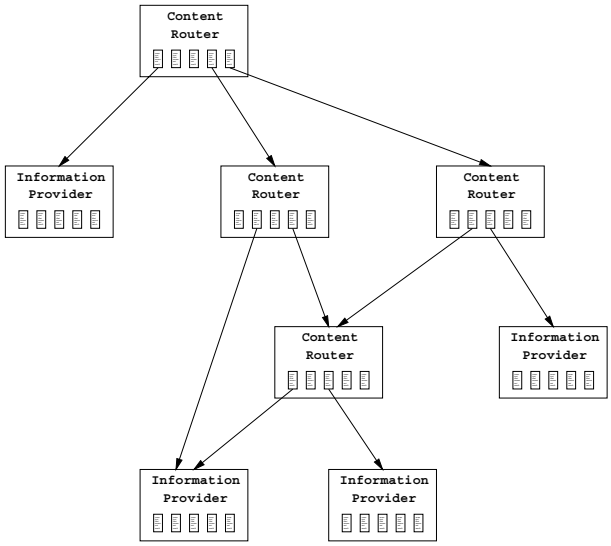

Fig. 1. System Organization

A content router registers an information provider, which may be another content router, using a compact description of that information provider's contents called a *content label*. A content label contains a query that is true of all documents available from the information provider. It also contains

information that can be used for query refinement (see below). For example, the following is a simple content label query that says that every document at its server has a collection type of `techreports`, is from an institution named `mit`, and is from the department of `electrical engineering and computer science`:

```
(institution: mit) and (collection-type: techreports) and
(department: electrical engineering and computer science)
```

The precise structure of content labels is not specified by the Content Routing architecture, but is left to the information providers and content routers themselves. A content label may therefore be a simple predicate consisting only of the host name of the computer acting as the information provider. It may also be a disjunction of all the terms in all the documents at the information provider. We expect typical content labels to consist of a subset of the attributes derived from the indexed documents as well as *value-added* attributes that describe the collection of documents but may not occur in the documents themselves. A content router may restrict the form and size of the content labels it accepts from registering information providers. Constraining the size of content labels allow the system to scale, and may require an information provider to eliminate some terms from its content label.

3.2 User interface operations

A user may browse and search the information available via a content router, retrieve documents, and request suggestions for additional query terms when formulating more specific queries. A content router may provide access to conventional documents, *e.g.,* text files, bibliographic entries and video footage. In addition, a content router supports a document type called a *collection*. A collection document is a set of other documents, some of which may be collections, together with the set's content label.

The operations available at a content router are enumerated in Table 1. The intention is that a user first attaches to a content router with an *open* operation. The user may browse that system's contents, and may use *show-fields* and *show-values* to formulate a query to be used in a *select*. The result set shrinks and grows as the user performs successive *select* and *expand* operations, possible using *refine* along the way. The user invokes the *search* operation when the cost of doing so is acceptable. The determination of whether the cost of a particular *search* operation is acceptable is left to the implementation of the content router. The user may *retrieve* documents that seem relevant. Note that a *retrieve* operation on a collection document returns a human readable form of the collection's content label. This enables a user to get a sense of what is in the a collection and what terms it indexes.

To simplify the user's interaction with the system, the implementation described here uses HTML forms and the notions of a *current result set* and a *current query*. (See the example in Section 4.1.)

3.3 Query Refinement

Query refinement recommends terms related to a particular query. These terms can be used for formulating new queries or for refining a query to focus it on documents of interest.

There are two axes along which to measure the effectiveness of term recommendations in the context of content routing. Suggested terms should partition the space of all relevant leaf documents reachable from the given result set. We refer to this set of documents as the *document space*. Suggested terms should also partition the set of information providers registered at the content router and relevant to the query. These are the servers to which the query would be routed. We refer to this set as the *route set*.

There are a number of methods for deriving recommended terms to add to a user's query. The approach considered here is based on conditional probability of term collocation. We feel that this approach is more suited to the more heterogeneous environment of today's internet than that of our previous work [9]. We have explored using a thesaurus for automatic server clustering and for term suggestions. The traditional use of a thesaurus, however, is to expand the result set by replacing a term with a disjunction of terms from its thesaurus entry [18]. This has the opposite effect of our query refinement mechanism, which reduces the result set size.

We adopt a predicate data model in which user queries consist of boolean combinations of terms. Terms may be either keywords or attributes (field name, value pairs). A user query Q is a predicate that

open	`open` *collection-name* initializes a connection to *collection-name* and returns a result set that contains all the documents at that collection.
select	`select` *result-set query* returns a new result set containing the elements of *result-set* that match *query*.
expand	`expand` *document query* if *document* is a collection, then returns a result set consisting of the documents in *document* that match the query. Otherwise, returns a result set containing just *document*.
expand	`expand` *result-set query* returns the union of the result sets obtained by applying `expand` to every document in *result-set*.
search	`search` *result-set query* returns a new result set containing all conventional documents in *result-set* that match the query and all documents that result from a recursive `search` of each collection document in *result-set*.
show-fields	`show-fields` *result-set* returns a list of available attribute field names.
show-values	`show-values` *result-set field-name* returns a list of potential values for *field-name* attributes.
refine	`refine` *result-set query additional-args* returns a list of recommended query terms that may be used to reduce size of the search space.
retrieve	`retrieve` *document* returns the contents of *document*. In the case of a collection, returns a human readable description of the collection's contents (content label).

Table 1. Content Router Operations

is either true or false of a document, *i.e.*, $Q(d)$ is true if and only if document d matches Q. The *document space* for a query Q is

$$\mathcal{D}(Q) \equiv \{d \mid Q(d)\}$$

where d is any (conventional) document reachable by traversing the network from the current content router.

We use the conditional probability of term collocation to recommend terms that are related to a given query and that efficiently partition the document space. The conditional probability p_i that term t_i occurs in a document that satisfies Q is the size of the document space for Q conjoined with t_i divided by the size of the document space for Q:

$$p_i \equiv \frac{\|\mathcal{D}(Q \wedge t_i)\|}{\|\mathcal{D}(Q)\|}$$

If a term has a high conditional probability p_i, then it is statistically related to the query, and therefore likely to be semantically related as well. A term with a low conditional probability will more dramatically reduce the size of the document space when it is conjoined with Q.

The prototype described in this paper simply offers the user the 40 terms with the highest conditional probabilities. See Section 4.3 for details of the algorithm for computing these terms as well as a description of how well it performs.

In general, a user may want to learn about terms closely correlated with the query (synonyms), terms inversely correlated with the query (antonyms), or terms moderately correlated with the query. Synonyms and antonyms provide feedback on what sorts of terms are indexed and help the user formulate better queries. Moderately correlated terms are especially useful in reducing the document space by a reasonable

amount while not eliminating documents so quickly that highly relevant items are lost. The system uses a *ranking function* to order terms and suggests to the user a manageable set consisting of the highest ranked terms.

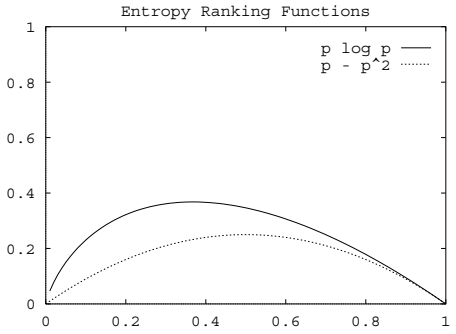

Fig. 2. Entropy Functions

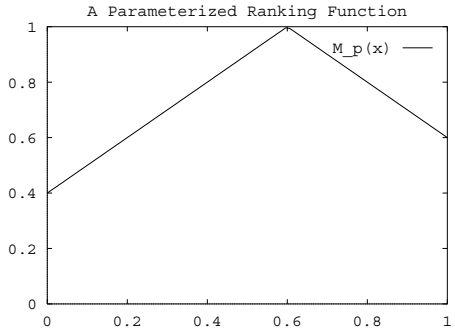

Fig. 3. $\mathcal{M}_{0.6}$ Function

Entropy functions give more favorable ranks to terms that are moderately correlated with the query terms, that is it prefers terms that reduce the result set by a moderate amount. A generalized entropy ranking function has the form

$$\mathcal{E}(t_i) = p_i F(p_i)$$

There are a variety of choices for F. One standard information theoretic entropy function, which we shall call $\mathcal{E}_S$, is

$$\mathcal{E}_S(t_i) = -p_i \log p_i$$

Another useful entropy function, which we shall call $\mathcal{E}_V$, is

$$\mathcal{E}_V(t_i) = p_i(1 - p_i)$$

Figure 2 illustrates the behavior of these entropy ranking functions. As the figure shows, these functions favor terms with with moderate probabilities. $\mathcal{E}_V$ prefers terms nearer $p_i = 0.5$.

It is useful to have a ranking function that is parameterized by the most favored probability (or result set reduction size). This would allow the user or the system to rank terms differently depending on the context. For example, the ranking could suggest synonyms or antonyms. A simple ranking function $\mathcal{M}_{\overline{p}}$ that is parameterized by the favored conditional probability, $\overline{p}$ is

$$\mathcal{M}_{\overline{p}}(t_i) = 1 - |p_i - \overline{p}|$$

Figure 3 illustrates the behavior of $\mathcal{M}_{\overline{p}}$ for $\overline{p} = 0.6$.

The system also should consider the expected time it will take to evaluate a query. This in turn depends on the number of remote servers involved in servicing the query, *i.e.,* the size of the route set. We are currently investigating techniques for producing cost estimates. We can apply exactly the same technique as above simply by replacing the document set with the route set. So far, our experience is that query refinement based on the document space produces semantically meaningful suggestions and also effectively reduces the size of the route set.

3.4 Query routing and forwarding

To process a user query, *i.e.,* to implement the *expand* or *search* operations, a content router must identify servers relevant to a query (the route set), forward the query to those servers, and merge the results.

When a route set has only a single collection, then the client should be given a pointer directly to that collection so the current server does not participate in future transactions. Similarly, clients should be given direct pointers to documents so that retrieval operations bypass mediating servers.

4 Experimental System

This section presents an HTTP content router for WAIS servers called Discover. Our goal was to exploit the large information space contained in available WAIS servers and to explore how content labels can be generated automatically for large collections of documents. We wanted to validate the content routing architecture using a large scale, real life example and gain experience using content labels for query formulation and for discovering resources in a large information space. We wanted to determine whether our goals could be achieved with adequate performance. We also hoped to provide a useful service to the WWW community.

Our prototype incorporates several simplifications to the Content Routing Architecture. Because our experiment involved using over 500 WAIS servers over which we have no control, it was not possible to enlist the aid of WAIS administrators to produce value-added attributes. Nor was it possible to do detailed statistical analyses to choose the content label attributes because we did not have access to the WAIS index structures (and browsing all the documents was not feasible). In a real content routing system, the individual servers would be responsible for this analysis. We therefore chose to experiment with very simple content labels that consisted of disjunctions of keywords obtained from the individual WAIS source and catalog files.

The remainder of this section presents an example of a user session, describes the structure of the Discover WWW content router for WAIS, and reports on our experience with the system and its performance.

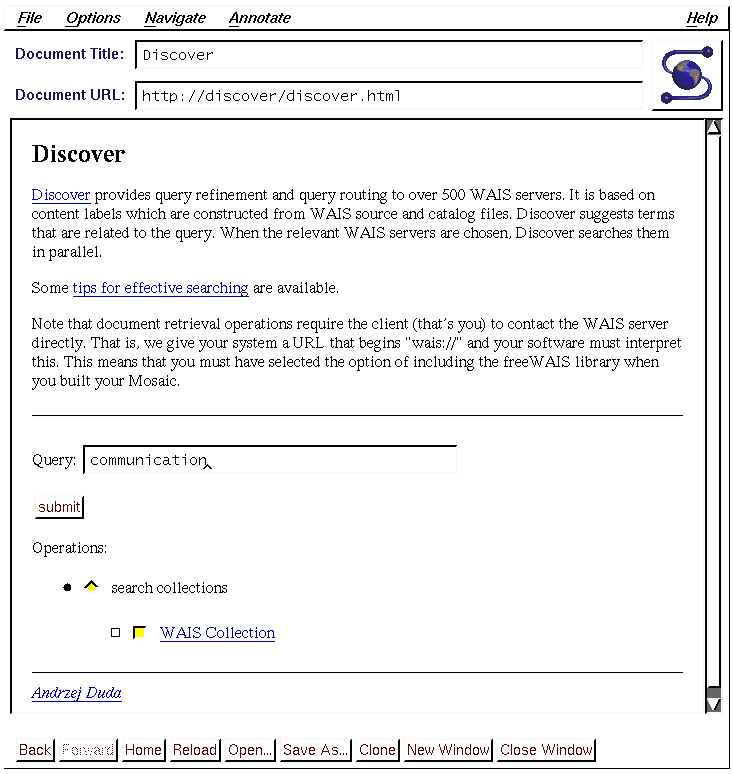

Fig. 4. Initial HTML Form document of the router

4.1 Example Session

A typical user session consists of iterative applications of *select*, *expand*, *refine*, *search*, and *retrieve* operations. A user's initial contact with Discover is shown in Figure 4. This start page contains a brief

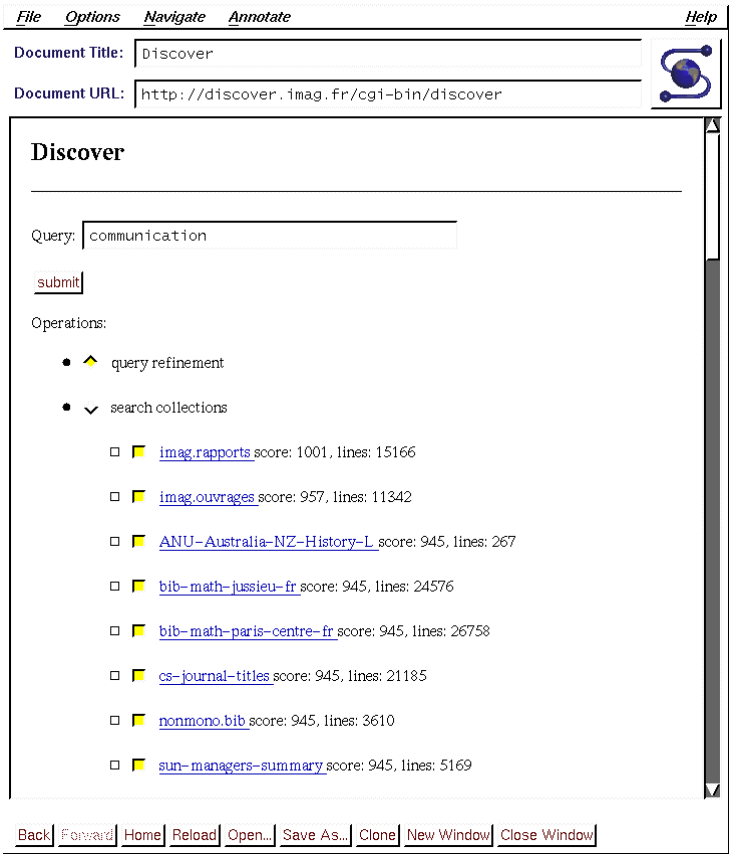

Fig. 5. List of relevant servers

description of the system and links to additional information about the architecture and about formulating queries. At the top level, the Discover content router contains a single collection consisting of all 504 WAIS servers currently registered with our system. Initially, the user may *expand* the WAIS collection document by submitting a query. A query in this prototype is an ordinary WAIS query, *i.e,* keywords combined with the boolean operators `and`, `or`, and `not`. Nesting of operations is not supported.

In this example, the user submits the query `communication`. The result is shown in Figure 5. At this stage, Discover presents the 69 collections (WAIS servers) relevant to the query. The result is computed using a local routing database. The user may *retrieve* a collection document to determine its relevance, connect directly to the collection, perform an exhaustive *search*, or modify the query possibly using the *refine* operation. The user may also manually eliminate elements of the current result set by clicking in the appropriate check boxes.

The result of a *refine* request is shown in Figure 6. After consulting a local query refinement database, Discover offers a sorted list up to 40 terms frequently collocated with the current query terms. The user may select a term from this list to be conjoined with the current query. In this case, the user selects the term `networks`. That term is added to the query, and the system performs a *select* operation with the new query. The smaller result set consisting of 37 collections is shown in Figure 7.

Another iteration of query refinement and *select* appears in Figures 8 and 9. In this example the user selects the term `routing` from the recommended list, and the result set is reduced to only 13 documents. Thus, in just a few operations, the search space has been reduced from 504 possible WAIS servers to 13 without any remote operations.

At this point the user may decide that the result set is satisfactory and opt for an exhaustive search. A separate search process is spawned for each WAIS collection. The results of the parallel searches are displayed in one HTML form as shown in Figure 10. The result set appears as a list of document titles that serve as hypertext links to the actual documents at the WAIS servers. To *retrieve* a document, the

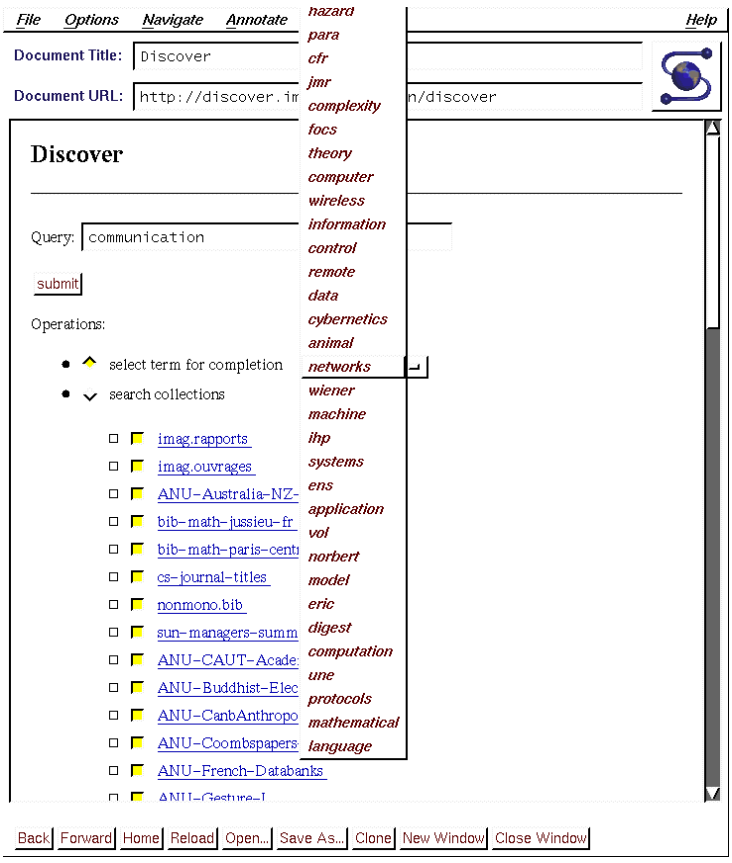

Fig. 6. Query refinement

user clicks on its title as illustrated in Figure 11. Note that the hypertext link is a document URL pointing directly to the document on the WAIS server. This URL is interpreted by the client's browser without the intervention of Discover.

4.2 An HTTP Content Router for WAIS

The Discover implementation is based on the framework provided by the HTTP World Wide Web protocol [2]. Since HTML documents and HTML Forms provide the most widely used multimedia display format in use today [3], we chose to use HTML Forms for Discover's input and output. Figure 12 illustrates the structure of Discover.

Discover uses several features of the WWW framework and form support. It presents to the user HTML form documents that can be retrieved and filled in using browsers such as Mosaic and Netscape. Submitting these forms causes the Discover driver, a CGI POST script, to interpret the parameters entered by the user, compute the results, and present the results in another HTML document. The resulting document is typically a form that allows the user to choose new operations and parameters.

Discover makes use of local WAIS databases accessed by means of the WAIS library for query refinement and query routing based on content labels. The router spawns parallel processes to search remote WAIS servers concurrently. The router and the parallel search processes use the WWW library to search remote databases. The WWW library routines are compiled with the freeWAIS library, and are thus able to contact the remote WAIS servers.

The current Discover prototype implements all the operations in table 1 except for *show-fields* and *show-values*, which are of limited utility given that WAIS indices are keyword based. The system described here implements a two-level content routing system.

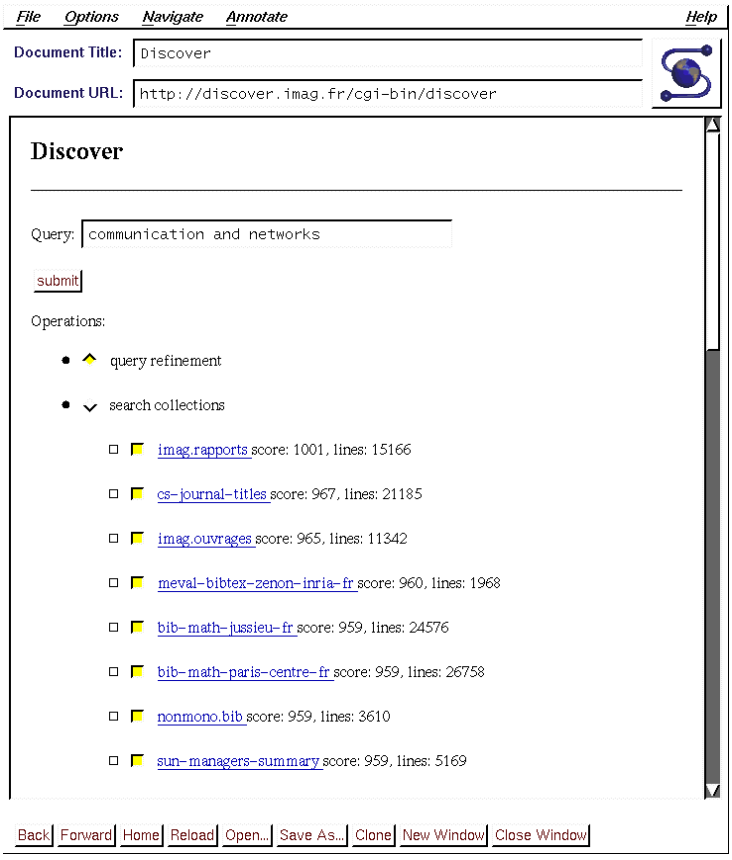

Fig. 7. List of relevant servers after refinement

4.3 Content Labels

In our prototype, we use WAIS source and catalog files to construct content labels. A catalog file consists of one headline for each document in the WAIS database. For example, the headline can be the title of an article. Although a catalog file is much smaller than the entire WAIS database, it describes the WAIS database well. Most WAIS servers return a catalog file in response to an empty query so that it can be retrieved automatically. Here is a sample from a catalog (from the National Science Foundation awards WAIS server):

```
Catalog for database: ./nsf-awards
Date: Sep 23 07:21:33 1994
63920 total documents

Document # 1
Headline: Title: Summer Undergraduate Research Experience Fellowships in the
DocID: 0 2049 /home/pub_gopher/.index/ftp/awards93/awd9322/a9322138

Document # 2
Headline: Title: NYI: Dedicated VLSI Digital Signal and Image Processors
DocID: 0 2682 /home/pub_gopher/.index/ftp/awards92/awd9258/a9258670

Document # 3
Headline: Title: Prediction of Soil Liquefaction in Centrifuge Model Tests
DocID: 0 2681 /home/pub_gopher/.index/ftp/awards91/awd9120/a9120215

Document # 4
```

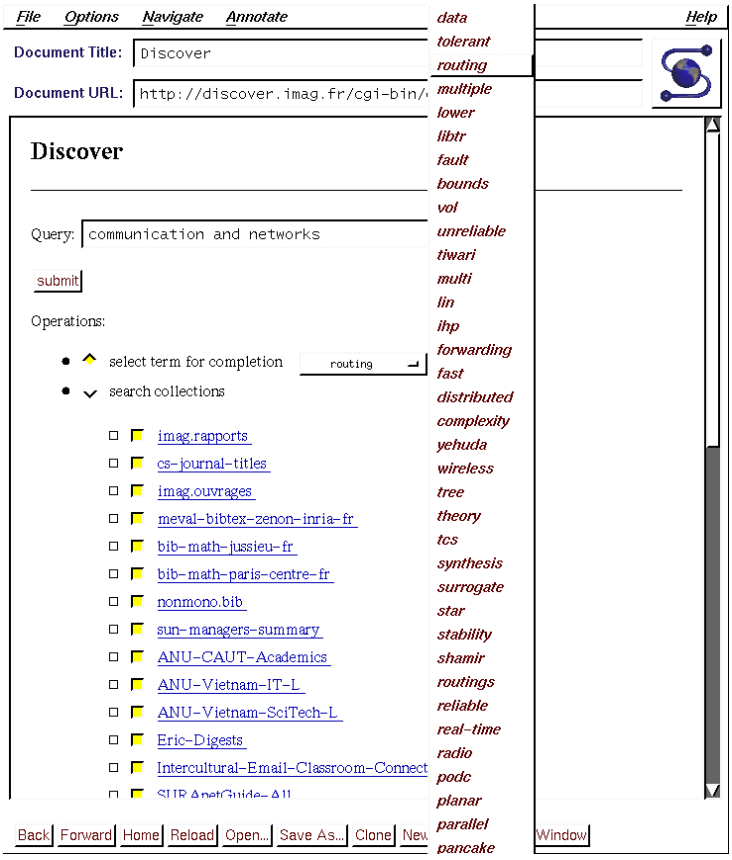

Fig. 8. Further query refinement

```
Headline: Title: Igneous-related Metallogenesis of the Great Basin
DocID: 0 707 /home/pub_gopher/.index/ftp/awards90/awd9096/a9096294
.
.
.
```

A WAIS source file contains a short description of a server, including contact information (*e.g.*, host name, host address, database name, administrator) as well as a short list of keywords and a natural language summary of the server's contents. In our sample, the median size of the source files was under 800 bytes. While they may enable one to categorize a server into a general domain such as biology, source files are not adequate by themselves for query routing or refinement. Here is the entire WAIS source file for the National Science Foundation awards server:

```
(:source
   :version  3
   :ip-address "128.150.195.40"
   :ip-name "stis.nsf.gov"
   :tcp-port 210
   :database-name "nsf-awards"
   :cost 0.00
   :cost-unit :free
   :maintainer "stisop@stis.nsf.gov"
   :description "Server created with WAIS release 8 b4 on
                 Apr 21 09:01:03 1992 by stisop@stis.nsf.gov

   This WAIS database contains award abstracts for awards made by
```

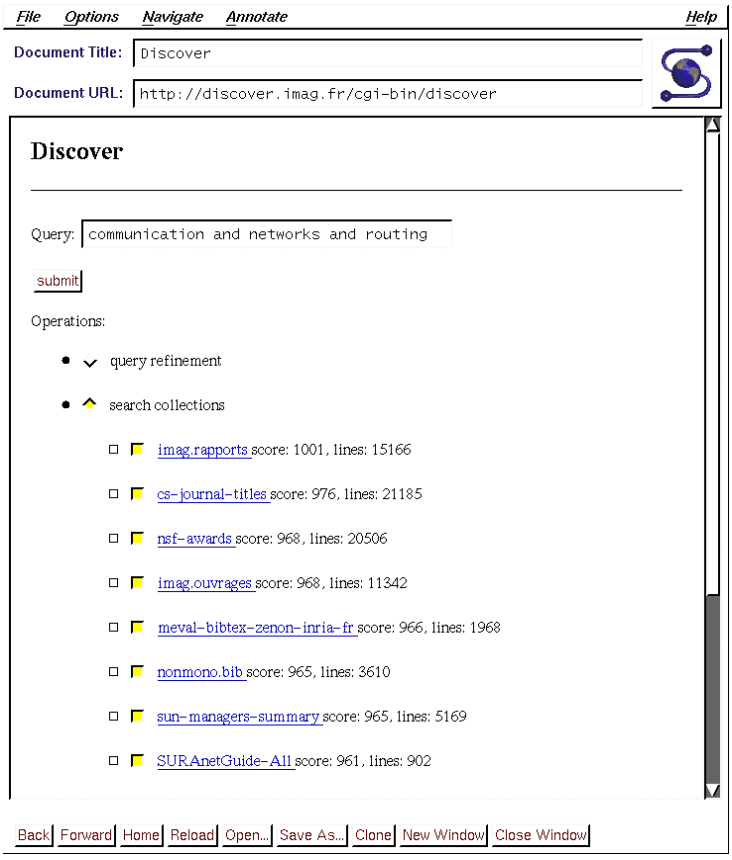

Fig. 9. Final list of relevant servers

```
   the National Science Foundation.  The database covers from the
   beginning of 1990 to the present (no abstracts are available before
   1990).

   If you use WAIS to retrieve these documents, We'd like to hear about
   your experience.  Please write to stis@nsf.gov.

   You might also be interested in the nsf-pubs database which contains
   NSF publications.  It is also on host stis.nsf.gov.

"
)
```

From 504 existing WAIS servers, we have retrieved 386 catalog files which occupy 191.1MB. Some of the catalog files, such as those with article titles, are very useful. Others, such as those with only file pathnames, are not useful. The largest catalog file occupies 16.7MB and contains 113,223 headlines; the smallest one is 79 bytes without any headlines.

Logically, a Discover content label for a WAIS server is the concatenation of the server's source and catalog files. For query routing, we only need to store the set of terms in the document headlines (with duplicates removed). As a result, the total size of the query routing database is only 15.4MB. This information is kept in a local WAIS database.

Recall that a *retrieve* operation on a collection document is defined to return a human readable form of the collection's content label. In response to a retrieval request on a collection, Discover returns a list of the indexed terms from the collection's content label. The terms are sorted by document frequency. This gives the user a good characterization of the contents of a collection. Here is the start of a sample response to a *retrieve* on the nsf-awards collection (each term is preceded by its document frequency):

Fig. 10. Result of the search

```
text:
4916 mathematical
4778 sciences
2520 studies
2453 science
2381 award
2081 presidential
1795 cooperative
1699 development
1619 laboratory
1565 collaborative
1399 study
1355 engineering
1280 physics
1277 molecular
1242 program
1209 systems
1193 dissertation
1170 structure
1136 dynamics
1076 conference
1072 chemistry
1008 computer
```

Query refinement makes use of another local WAIS database containing all headlines from catalog files indexed as one-line entries. The size of the headline file is 62MB and the size of the indexes is 196MB. The

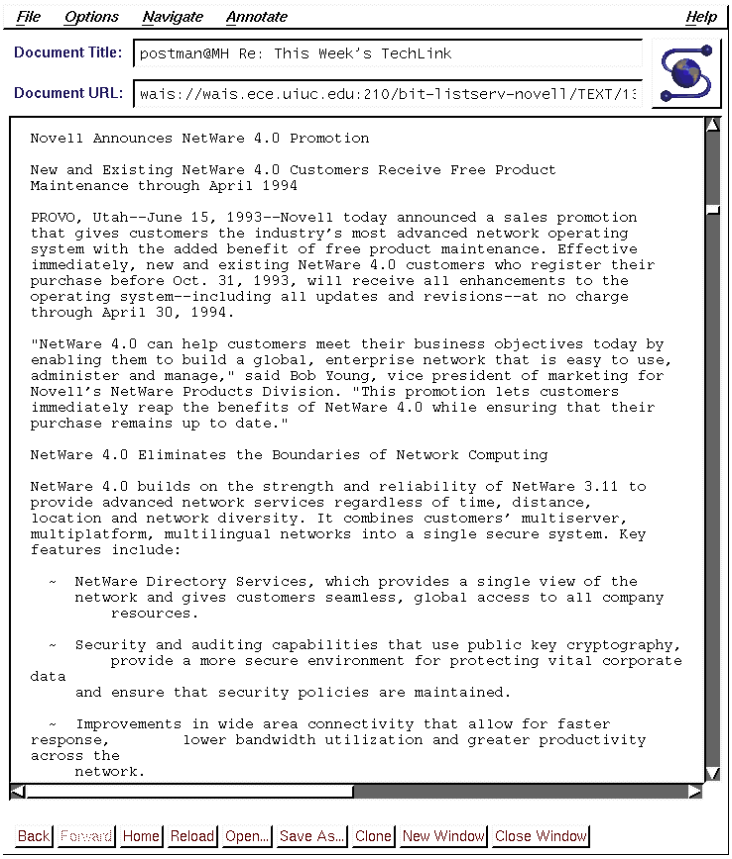

Fig. 11. Retrieved document

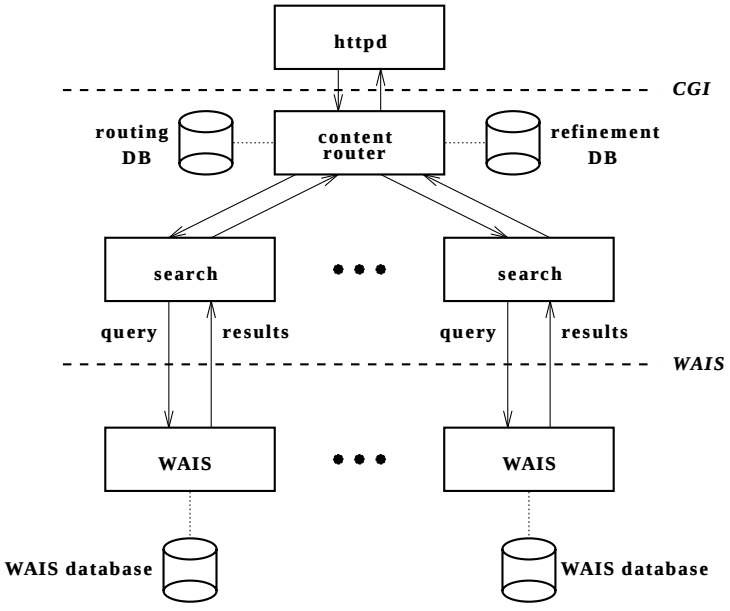

Fig. 12. Structure of the content router

headline database allows the router to find terms related to the terms already specified in the query — the 40 most frequently collocated terms are presented to the user in response to a *refine* request. Discover thus implements a ranking based on conditional probability of term collocation. This is equivalent to $\mathcal{M}_1$ in Section 3.3. Our experience has been that our rankings are similar to that of an entropy function. This is because in the queries we have seen, the highest frequency collocated terms still do not have very high frequencies, typically under 0.5.

4.4 Experience and Performance

Our experience thus far suggests that Discover is a useful tool for content-based access to documents in a large collection of information servers. We are continuing to improve the time and space requirements of the system, but even now the system offers satisfactory performance. For example, the operations in the sample session in Section 4.1 had the following performance when run on a SparcStation IPX:

Operation	Query	Time (min:sec)
expand	`communication`	0:05.07
refine	`communication`	5:35.51
refine	`communication and networks`	1:06.51
search	`communication and networks and routing`	1:29.67
retrieve	`"This Week's TechLink"`	0:09.27

The *expand* operation consulted the local routing database. The *refine* operations consulted the local refinement database. The second *refine* operation was much faster than the first because the document space had been effectively reduced by the previous *refine* and *select* operations. We are still working on improving the query refinement process. Also, based on experience with a prior prototype, we expect a factor of 3–5 speedup when we move the implementation to a SparcStation 10. The *search* operation involved searching 13 remote WAIS servers scattered around the world and merging the results. During the *retrieve* operation, the client contacts the remote WAIS server directly rather than going through Discover. Thus, retrieving this 132K byte document did not involve our server at all.

We have found the power of boolean queries to be very useful. However, a significant portion of WAIS servers do not support full boolean queries. Thus, our server must implement conjunction by invoking a separate query for each conjunct and performing a local set intersection operation. This entails a large performance penalty for queries that use general terms that occur in many documents, even if the result set for the query as a whole is small.

The simple content labels used in this prototype perform remarkably well. Even without using specialized knowledge or detailed data analysis, Discover has proven valuable in exploring a large information space.

5 Conclusion and Future Work

We have build an HTTP-based prototype content router that provides access to the contents of over 500 WAIS servers. We have used this system to test the feasibility of the Content Routing System architecture, to explore techniques for building content labels automatically, and to test the utility and practicality of query refinement. The Discover prototype builds content labels from WAIS source and catalog files. The prototype uses these content labels to support query routing and refinement. We have found that even simple techniques for producing content labels, routing queries, and refining queries are very effective. Discover, though still a prototype, provides these services with adequate performance. Our experience with the prototype suggests that content routing is a useful tool for organizing and accessing a very large, heterogeneous, distributed information system.

We are continuing to investigate alternative approaches to query refinement. We are especially interested in improving its performance. We are extending our system to support other server and document types such as HTML. We are continuing to gather statistics that will enable us to construct better content labels. We are also building more complex content routing system networks and extending Discover's functionality.

References

1. B. Alberti et al. The Internet Gopher protocol: A distributed document search and retrieval protocol. University of Minesota Microcomputer and Workstation Networks Center, Spring 1991. Revised Spring 1992.
2. T. Berners-Lee. Hypertext transfer protocol. Internet Draft, Nov. 1993.
3. T. Berners-Lee and D. Connolly. Hypertext markup language. Internet Draft, July 1993.
4. T. Berners-Lee et al. World-Wide Web: The information universe. *Electronic Networking*, 2(1):52–58, 1992.
5. C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The harvest information discovery and access system. In *Proceedings of the Second International World Wide Web Conference*, Chicago, Illinois, Oct. 1994.
6. A. Duda and M. A. Sheldon. Content routing in networks of WAIS servers. In *International Conference on Distributed Computing Systems Proceedings*. IEEE, June 1994.
7. D. Eichmann. The RBSE spider – balancing effective search against web load. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.
8. A. Emtage and P. Deutsch. Archie – an electronic directory service for the Internet. In *USENIX Association Winter Conference Proceedings*, pages 93–110, San Francisco, Jan. 1992.
9. D. K. Gifford. Polychannel systems for mass digital communication. *Comm. ACM*, 33(2), Feb. 1990.
10. D. K. Gifford et al. Semantic file systems. In *Thirteenth ACM Symposium on Operating Systems Principles*. ACM, Oct. 1991. Available as *Operating Systems Review* Volume 25, Number 5.
11. L. Gravano, A. Tomasic, and H. García-Molina. The efficacy of GLOSS for the text database discovery problem. Technical Report STAN-CS-TR-93-2, Stanford University, Oct. 1993.
12. H. Hahn and R. Stout. *The Internet Complete Reference*. Osborne McGraw-Hill, Berkeley, California, 1994.
13. B. Kahle and A. Medlar. An information system for corporate users: Wide Area Information Servers. Technical Report TMC-199, Thinking Machines, Inc., Apr. 1991. Version 3.
14. M. Koster. ALIWEB – archie-like indexing in the web. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.
15. R. S. Marcus. An experimental comparison of the effectiveness of computers and humans as search intermediaries. *Journal of the American Society for Information Science*, 34(6):381–404, Nov. 1983.
16. R. S. Marcus. Advanced retrieval assistance for the DGIS gateway. Technical Report LIDS R-1958, MIT Laboratory for Information and Decision Systems, Mar. 1990.
17. B. Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.
18. G. Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.
19. M. A. Sheldon, A. Duda, R. Weiss, J. W. O'Toole, Jr., and D. K. Gifford. A content routing system for distributed information servers. In *Proc. Fourth International Conference on Extending Database Technology*, Mar. 1994.