Search Engines – A Survey Report

Submitted by: Nirmala Bikkannavar

Instructor: Dr. Jerry Gao

CMPE 296U

Date: October 7, 1999

Fall 1999
Abstract

The ability to search and retrieve information from the Web efficiently and effectively is an enabling technology for realizing its full potential. Search engine is a good tool to help users locate information of interest on the Internet. It is among the most useful and high profile resources on the Internet. Search engines are powerful tools for assisting the otherwise unmanageable task of navigating the rapidly expanding World Wide Web. A variety of search engines are available from large-scale and domain specific search engines to directories and spiders to online and offline search engines. The four major elements of a search engine are crawling, indexing, search engine and ranking. Different search engines deploy different algorithms for crawling, indexing, and searching the links; thus each returns different results for similar queries.

In this survey report, architectures of some of the search engines like Google, Harvest, WebCrawler, Infoseek, Lycos, Excite, and Yahoo are studied. These search engines are compared with respect to their features such as crawling, indexing, ranking methods, math commands, power searching commands, search assistance features, display features, Boolean commands, size, number of Web pages indexed, and percentage of Web indexed.
Table of Contents

ABSTRACT................................................................................................................................. II

1 INTRODUCTION .................................................................................................................... 1
  1.1 DIRECTORIES AND SPIDERS ...................................................................................... 1
  1.2 ONLINE AND OFFLINE SEARCH ENGINES ............................................................ 2
  1.3 LARGE-SCALE AND DOMAIN-SPECIFIC SEARCH ENGINES .................................... 3

2 COMPONENTS OF A SEARCH ENGINE .............................................................................. 3
  2.1 CRAWLING ..................................................................................................................... 4
  2.2 INDEXING ....................................................................................................................... 4
  2.3 SEARCH ENGINE .......................................................................................................... 4
  2.4 RANKING ....................................................................................................................... 4

3 ISSUES WITH THE DESIGN OF SEARCH ENGINES ...................................................... 5
  3.1 SCALING WITH THE WEB .......................................................................................... 5
  3.2 SEARCH QUALITY ....................................................................................................... 6

4 WEBCRAWLER .................................................................................................................. 6
  4.1 INTRODUCTION ............................................................................................................ 6
  4.2 ARCHITECTURE OF THE WEBCRAWLER ............................................................... 7
  4.3 WEBCRAWLER’S SEARCH ENGINE ......................................................................... 8
    4.3.1 Indexing mode ...................................................................................................... 9
    4.3.2 Real-time search mode ....................................................................................... 9
  4.4 WEBCRAWLER’S AGENTS ........................................................................................ 10
  4.5 WEBCRAWLER’S DATABASE ..................................................................................... 10
  4.6 WEBCRAWLER’S QUERY SERVER ........................................................................... 11
  4.7 WEBCRAWLER SEARCH FEATURES ...................................................................... 12

5 HARVEST ............................................................................................................................ 12
  5.1 INTRODUCTION ............................................................................................................ 12
  5.2 ARCHITECTURE .......................................................................................................... 13
  5.3 GATHERER .................................................................................................................... 15
  5.4 BROKER ........................................................................................................................ 16
  5.5 INDEX/SEARCH SUBSYSTEM ................................................................................... 17
  5.6 OBJECT CACHE .......................................................................................................... 18
  5.7 REPLICATOR ................................................................................................................. 18
  5.8 SEARCHING ................................................................................................................ 19
  5.9 FLEXIBILITY ................................................................................................................ 19

6 ALTAVISTA ........................................................................................................................ 20
  6.1 CRAWLING .................................................................................................................... 20
  6.2 RANKING ....................................................................................................................... 20
  6.3 INDEXING ....................................................................................................................... 21
  6.4 SPAMMING ................................................................................................................... 21
  6.5 LANGUAGE DETECTION ............................................................................................ 22

7 EXCITE .............................................................................................................................. 22
  7.1 CRAWLING .................................................................................................................... 22
  7.2 RANKING ....................................................................................................................... 22
  7.3 INDEXING ....................................................................................................................... 23
  7.4 SPAMMING ................................................................................................................... 23
  7.5 REGIONAL EDITIONS ................................................................................................. 24
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: Architecture of WebCrawler</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Figure 2: Architecture of Harvest</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Figure 3: Architecture of Google</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Figure 4: Google Query Evaluation</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Figure 5: Millions of Web Pages Indexed</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Figure 6: Search Engine Sizes</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Figure 8: Percentage of Web Indexed</td>
<td>56</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

Table 1: Comparison of Crawling Features ................................................................. 40
Table 2: Comparison of Indexing Features ................................................................. 43
Table 3: Comparison of Ranking Features ............................................................... 45
Table 4: Comparison of Spam Features ................................................................. 46
Table 5: Comparison of Search Engine Math Commands ........................................ 48
Table 6: Comparison of Power Searching Commands ............................................. 49
Table 7: Comparison of Search Assistance Features ............................................... 50
Table 8: Comparison of Display Features ............................................................. 51
Table 9: Comparison of Boolean Commands ......................................................... 52
1 Introduction

World Wide Web presents a huge amount of information sources, and is growing at an unprecedented rate. To make effective use of this wealth of information, users need ways to locate information of interest to them. In the past few years, a number of knowledge navigation, resource discovery tools, and search engines have been created, and have gained wide acceptance on the Internet.

The ability to search and retrieve information from the Web efficiently and effectively is an enabling technology for realizing its full potential. Search engine is a good tool to help users locate information of interest on the Internet. There are many search engines such as Alta Vista, Infoseek, and Excite, which offer an extensive coverage of HTML documents on the Internet. Search engines try to index all documents on the entire World Wide Web. Search engines are powerful tools for assisting the otherwise unmanageable task of navigating the rapidly expanding World Wide Web.

1.1 Directories and Spiders

The search industry has evolved two dominant ways to find things –

1. Directories

Directories depend on humans for their listings. The Webmasters submit a short description of their Web sites to the directory. The editors of the directory can also write a short description for the site they review. Directories order all knowledge into some structure and classify individual Web pages with respect to that structure. A search looks for matches only in the descriptions submitted. Changing a Web site has no automatic effect on the listing.

The problems with directories are as follows:
1. The classification is a labor-intensive activity, and
2. If the information that the user interested in is not reflected by the classification structure, then the user is out of luck.

Yahoo is a good example of a directory.

2. **Spiders**

Spiders are computer programs, which can find and organize the Web documents automatically. Such systems have three important parts:

1. A *robot* that explores the Web and retrieves pages.
2. A *database* of information refined about those pages, including the ability to search that database against queries and order the resulting matches; and
3. A *user interface* for obtaining queries and presenting the results.

These search engines create their listings automatically. If a Web page is changed, search engines eventually find these changes.

 AltaVista, Infoseek, and Lycos are some of the examples of spiders.

1.2 **Online and Offline Search Engines**

There are two approaches used by the search engines –

1. **Online Search Engines**

Online search engines find relevant documents on the Web by launching a Web robot (also called a wanderer, worm, walker, spider, crawler, or knowbot). The robot receives a user query, then systematically explores the Web to locate documents, evaluates their relevance, and returns a rank-ordered list of documents to the user. The vastness and exponential growth of the Web makes this approach impractical for every user query.
2. Offline Search Engines

Offline search engines search a precompiled index built and updated periodically by Web robots. The index is a searchable archive that gives reference pointers to Web documents. This is obviously more practical, and most existing search engines are based on this approach.

1.3 Large-scale and Domain-specific Search Engines

Two types of search engines have been developed –

1. Large-scale Search Engines:

In large-scale search engines, all documents on the Web are indexed into one huge database. They try to be comprehensive; as a result, a search may return an abundance of related and unrelated information. They exemplify the trade-off between breadth and quality.

2. Domain Specific Search Engines:

All documents on the Web are clustered, classified, and partitioned into a set of domain specific information spaces based on a set of well-defined domains. These are inadequate to most topics, but are more likely to quickly focus a search in their area.

2 Components of a Search Engine

The four major elements of a search engine are crawling, indexing, search engine and ranking, which are as follows.
2.1 Crawling

Crawler visits a web page, retrieves it, and then follows links to other pages and retrieves them. Also, it returns to the sites on regular basis to look for changes. There are many challenges in designing a crawler. Externally the crawler must avoid overloading web sites or network links as it goes about its business. Internally the crawler must deal with huge volumes of data. Unless it has unlimited resources and time, it must carefully decide what URLs to scan, in order to keep its client informed of changes to the Web. It’s important for the crawler to visit important pages first. Many crawling strategies and metrics are deployed. If the search engine is building a specialized database on a particular topic, then pages that refer to that topic are more important and should be visited as early as possible.

2.2 Indexing

The information retrieved by a crawler is indexed based on the keywords, title, author etc. and stored in a database, which is also called a catalog. It is like a giant book containing a copy of every Web page that the crawler finds. If a Web page changes then this index is updated with new information.

Generating a comprehensive index requires systematic traversal of the Web to locate all documents. The Web’s structure is similar to that of a directed graph, so it can be traversed using graph-traversal algorithms.

2.3 Search Engine

The search engine is a program that sifts through millions of pages recorded in the index to find matches to a search and rank them in order of what it believes is most relevant.

2.4 Ranking
Search engines generally use location and frequency of search keywords to rank a page. Following are some of the rules that search engines use for ranking:

1. The pages with search keywords appearing in the title are assumed to be more relevant to the topic than others.
2. The search engines check to see if the keywords appear near the top of a web page, such as in the headline or in the first few paragraphs of text. They assume that any page relevant to the topic will mention those words right from the beginning.
3. A search engine will analyze how often keywords appear in relation to other words in a web page. Those with a higher frequency often deemed more relevant than other web pages.
4. The proximity of query words in the document.
5. Search engines give Web pages a boost for certain reasons such as link popularity. It tells which of the pages in its index have a lot of links pointing to them. These pages are given a slight boost during ranking, since a page with many links to it is probably well regarded on the Internet.
6. Some hybrid search engines, those with associated directories, may give a relevancy boost to sites they’ve reviewed. The logic is that if the site was good enough to earn a review, chances are it’s more relevant than an unreviewed site.
7. Meta tags are also used to find relevancy of a Web page.

Different search engines use different approaches to crawl, index, search, and rank a Web document.

3  Issues with the Design of Search Engines

3.1  Scaling with the Web
Creating a search engine which scales even to today's web presents many challenges. Fast crawling technology is needed to gather the Web documents and keep them up to date. Storage space must be used efficiently to store indices and, optionally, the documents themselves. The indexing system must process hundreds of gigabytes of data efficiently. Queries must be handled quickly, at a rate of hundreds to thousands per second. These tasks are becoming increasingly difficult as the Web grows.

3.2 Search Quality

The quality of a search engine is an important issue in designing search engines. The completeness of the index is not the only factor in the quality of search results. "Junk results" often wash out any results that a user is interested in. One of the main causes of this problem is that the number of documents in the indices has been increasing by many orders of magnitude. Because of this, as the collection size grows, we need tools that have very high precision (number of relevant documents returned, say in the top tens of results). This very high precision is important even at the expense of recall (the total number of relevant documents the system is able to return).

In the subsequent sections the architectures of some of the search engines like Google, AltaVista, Infoseek, Harvest, WebCrawler, Lycos, Yahoo, and Excite are discussed and compared with respect to crawling, indexing and, ranking features.

4 WebCrawler

4.1 Introduction

The WebCrawler project began as Brian Pinkerton's research project at the Department of Computer Science and Engineering at the University of Washington in Seattle. Later it was acquired by America Online. In November of 1996, Excite, Inc acquired WebCrawler.
The WebCrawler design has the following characteristics:

- It uses a *content-based, full-text indexing* system to provide a high-quality index. In a Web robot, there is no additional network load imposed by full-text indexing; the load occurs only at the server.
- It uses a *breadth-first search* strategy to create a broad index, spreading the load among servers and ensuring that every server with useful content has at least several pages represented in the index.
- It tries to include as many Web servers as possible. It does so in a friendly manner, such as not overloading Web servers with rapid-fire requests. It also respects the Robot Exclusion Standard, which is a way for Webmasters to communicate to compliant robots which areas of the Web are off-limits.

WebCrawler starts with a known set of documents, examines the outbound links from them, follows one of the links that leads to a new document, then repeats the whole process. In other words, WebCrawler simply explores the Web space as a large *directed graph* using a graph traversal algorithm that performs the following sequence of actions over and over:

1. Discovers a new document.
2. Marks the document as having been retrieved.
3. Deciphers any outbound links.
4. Indexes the content of the document.

### 4.2 Architecture of the WebCrawler
As illustrated in figure 1, the WebCrawler’s software architecture is made up of the following four components:

1. **Search engine**: This directs the WebCrawler’s activities and is responsible for deciding which new documents to explore and for initiating their retrieval.
2. **Database**: This handles the persistent storage of the document metadata, the links between documents, and the full-text index.
3. **Agents**: These are responsible for retrieving the documents from the network at the direction of the search engine.
4. **Query Server**: This implements the query service provided to the Internet users.

### 4.3 WebCrawler's Search Engine

The WebCrawler’s search engine determines which documents and what types of documents to visit. Non-indexable files, such as pictures, sounds, PostScript, or binary data, are not retrieved. In addition, erroneously retrieved files are ignored during the indexing step. This sort of file-type discrimination is applied to both **indexing** and **real-**
time search modes. The search engine uses different discovery strategies when running the WebCrawler in these two search modes.

4.3.1 Indexing mode

In this mode the goal is to build an index of as much of the Web as possible within limited storage space. WebCrawler believes that the Web documents used to build the index should come from as many different servers as possible. It uses a modified breadth-first algorithm to ensure that every server has at least one document represented in the index. These steps show how the algorithm works:

1. When a document on a new server is found, that server is placed on a list of servers to be visited right away.
2. One document from each of the new servers is retrieved and indexed before visiting any other documents.
3. When all known servers have been visited, indexing proceeds sequentially through a list of all servers until a new one is found, at which point the process repeats.

4.3.2 Real-time search mode

In this mode, the goal is to find documents that are most similar to a user's query. The WebCrawler uses a different search algorithm. The intuition behind the algorithm is that following links from documents that are similar to what the user wants is more likely to lead to relevant documents than following any link from any document. The algorithm works like this:

1. WebCrawler runs the user's query against its index to first come up with an initial list of similar documents.
2. From the list, the most relevant documents are noted, and any unexplored links from those documents are followed.
3. As new documents are retrieved, they are added to the index, and the query is re-run.
4. The results of the query are sorted by relevance, and new documents near the top of
the list become candidates for further exploration.

5. The process is iterated either until the WebCrawler has found enough similar
documents to satisfy the user or until a time limit is reached.

4.4 WebCrawler's Agents

The search engines invoke agents for the purpose of retrieving Web documents. Because
waiting for servers and the network creates a search bottleneck, agents run in separate
processes, and the WebCrawler employs up to 15 agents in parallel. For each new Web
document to be retrieved, the search engine finds a free agent, and asks the agent to
retrieve the URL representing the document. The agent either responds to the search
engine with an object containing the document content or an explanation of why the
document could not be retrieved. After the agent has responded, it becomes free again
and may be given new work to do.

The agent program uses the CERN WWW library (libWWW), which supports access to
several types of content through different protocols, including HTTP, FTP, and Gopher.
As a practical matter, running agents in separate processes helps isolate the main
WebCrawler process from memory leaks and errors in the agent and in libWWW.

4.5 WebCrawler's Database

The WebCrawler's database holds both the full-text index and the representation of the
Web as a graph. The database is stored on disk and is updated as documents are added.
To protect the database from system crashes, updates are made under the scope of
transactions that are committed every few hundred documents.

WebCrawler uses NeXTStep's IndexingKit to build its full-text index, which is inverted to
make queries fast: looking up a word produces a list of pointers to documents that contain
that word. More complex queries are handled by combining the document lists for several
words with conventional set operations. The index uses a *vector-space model* for handling queries.

Words from a document are run through a *stop list* to prevent common words from being indexed, and they are weighted by their frequency in the document divided by their frequency in a reference domain. Words that appear frequently in the document and infrequently in the reference domain are weighted most highly, while words that appear infrequently in either are given lower weights. This type of weighting is commonly called *peculiarity weighting*.

The remainder of the database stores data about servers, documents, and links. Entire URLs are not stored; instead, they are broken down into objects that describe the server and the document. A link in a document is simply a pointer to another document. Each object is stored in a separate Btree on disk: documents in one, servers in another, and links in the last. Separating the data in this way allows the WebCrawler to scan the list of servers quickly to select unexplored servers or the least recently accessed server.

### 4.6 WebCrawler's Query Server

The query server implements the WebCrawler’s search service. The query model it presents is a simple *vector-space query model* based on the full-text database. Users enter keywords as their query, and the titles, and URLs of documents containing some or all of those words are retrieved from the index and presented to the user as an ordered list sorted by relevance. In this model, relevance is the sum (over all words in the query) of the product of the word's weight in the document and its weight in the query divided by the number of words in the query.

It does not place an undue burden on individual servers while building its index. WebCrawler adopts the standard for robot exclusion standard and identifies itself as WebCrawler in the HTTP User-Agent request header field when traversing the Web.
4.7 WebCrawler Search Features

1. **WebCrawler makes searching easy**: WebCrawler employs advanced search and retrieval technology from Personal Library Software (PLS) so that even novice users will get the results they want. WebCrawler is programmed to "do the right thing" even when your searches are described in a less than expert way.

2. **WebCrawler supports natural language searching**: WebCrawler supports "natural language searching" so that users can type their searches in plain English without worrying about mastering complex search syntax. Advanced users will be happy to know that WebCrawler also supports a wide range of Boolean search operators.

3. **WebCrawler matches any or all or your search terms**: When a series of search terms are typed in, WebCrawler is programmed to find results that match any or all of those words.

5 Harvest

5.1 Introduction

Professor Michael Schwartz of the University of Colorado at Boulder was the team leader for the Harvest project. Harvest was deployed on the Internet in November 1994. It is an integrated suite of customizable tools that provides a scalable, customizable architecture for gathering, indexing, caching, replicating, and accessing Internet information. The philosophy behind the Harvest system is that it gathers information about Internet resources and customizes views into what is "harvested". The creators of the Harvest system recognize three types of problems with most current Internet information systems:
- Most World Wide Web robots use expensive object retrieval protocols to gather indexing information and do not coordinate information gathering among them. Each World Wide Web robot gathers all the information it needs, without trying to share overlapping information with other robots.
- Little support exists for customizing how different information formats and index/search schemes are handled.
- Internet data and indices often become very popular and cause serious network and server bottlenecks.

Harvest provides a very efficient means of gathering and distributing index information (with Gatherers), and supports the easy construction of many different types of indexes customized to suit the peculiarities of each information collection (with Brokers). In addition, Harvest also provides caching and replication support to alleviate bottlenecks.

5.2 Architecture

A Harvest Gatherer collects indexing information, while a Harvest Broker provides an incrementally indexed query interface to the gathered information.
Figure 2: Architecture of Harvest

As illustrated in figure 2, Harvest offers a flexible scheme consisting of Gatherers and Brokers that can be arranged in various ways. This flexibility enables efficient use of network and server resources.

The Harvest architecture consists of the following subsystems –

- **Gatherer** collects indexing information.
- **Broker** provides a flexible interface to gathered information.
- **Index/Search subsystem** allows the information space to be flexibly indexed and searched in a variety of ways.
- **Object Cache** stores contents of retrieved objects to alleviate access bottlenecks to popular data.
- **Replicator** mirrors index information of Brokers to alleviate server bottlenecks.

5.3 Gatherer

The Gatherer provides an *efficient* and *flexible* way to collect indexing information. It solves two major problems that plague most current Web indexing systems:

- Data collection inefficiencies
- Duplication of implementation effort

Most current indexing systems cause excessive load on remote sites and generate excess network traffic. Retrieving via HTTP/Gopher/FTP requires heavyweight operations, like forking separate processes for each object, and entire objects often are retrieved when only a small part of the information actually is needed (for example, retaining only HTML anchors in an index).

Although the Gatherer can access an information Provider from across the network using the native HTTP, Gopher, or FTP protocols, this arrangement is primarily useful for interoperating with systems that do not run the Harvest software. The following are two important ways for Gatherers to achieve efficient use of network and server resources:

- A Gatherer can be run at the Provider site, saving a great deal of server load and network traffic.
- A Gatherer can feed information to many Brokers, saving repeated gathering costs.

The Harvest Gatherer provides efficient data collection through Provider site-resident software optimized for indexing. The Gatherer scans objects periodically, maintains a cache of indexing information (so that separate traversals are not required for each request), and allows a Provider's indexing information to be retrieved in a single stream (rather than requiring separate requests for each object). It minimizes network traffic by
pre-filtering the contents and sending only incremental updates of indexing information in compressed form over the network.

The Gatherer avoids duplication of implementation efforts by providing enough flexibility to allow different indexes to be built. It uses a customizable content extraction system that allows users to customize what data are gathered, whether data are gathered locally (which is more efficient but requires site cooperation), or remotely (which allows data to be gathered via the standard HTTP/Gopher/FTP protocols). The Gatherer extracts information in different ways depending on the file types. It can, for example, find author and title lines in Latex documents, and symbols in object code.

5.4 Broker

The broker is very flexible and provides an indexed query interface to gathered information. Periodically, the Broker retrieves information from one or more Gatherers or other Brokers, and incrementally updates its index. The Broker's interface is independent of the indexer, and can be customized to include new indexers with minimal effort. The Broker also can be configured to expire and re-collect information at varying intervals from the specified Gatherers.

The Broker collects objects directly from another Broker using a bulk transfer protocol. The Broker keeps track of the unique identifiers and time-to-live's for each indexed object. When a query or update is received, it invokes the Index/Search Subsystem. A Broker can collect information from many Gatherers to build an index of widely distributed information. Brokers also can retrieve information from other Brokers, in effect cascading indexed views from one another, using the Broker's query interface to filter/refine the information from one Broker to the next.

Harvest provides a distinguished Broker instance called the Harvest Server Registry (HSR), which registers information about each Harvest Gatherer, Broker, Cache, and Replicator in the Internet. The HSR is useful when searching for an appropriate Broker
and when constructing new Gatherers and Brokers, to avoid duplication of effort. It can also be used to locate Caches and Replicators.

### 5.5 Index/Search Subsystem

Harvest defines a general Broker-Indexer interface that can accommodate a variety of back-end search engines to accommodate diverse indexing and searching needs. The backend is required to support Boolean combinations of attribute-based queries, and incremental updates. A variety of different backends can thus be used inside a Broker. Currently, Harvest supports WAIS, Glimpse, and Nebula; they all are optimized for different uses.

Glimpse supports space-efficient indexes and flexible interactive queries. Glimpse uses pointers to occurrence blocks of adjustable sizes, instead of pointing to the exact occurrence. It can thus achieve very space efficient indexes, typically 2-4 percent the size of the data being indexed compared with 100 percent in the case of WAIS. As a concrete example, indexing the Computer Science technical reports from 280 sites around the world requires 9 GB with a standard WAIS index but only 270 MB using Glimpse. Glimpse also supports fast and incremental indexing, as well as queries involving Boolean combinations of keywords, regular expression pattern matching and approximate matches.

In contrast to Glimpse, Nebula focuses on providing fast searches and complex standing queries at the expense of index size. Each object in Nebula is represented as a set of attribute/value pairs. Nebula supports the notion of a view, which is defined by standing queries against the database of indexed objects. This allows information to be filtered based upon query predicates, effectively constraining the search to some subset of the database. Within the scope of a view that contains computer science technical reports, for example, a user may search for networks without matching information about social networks. Because views exist over time, it is easy to refine and extend them, and to observe the effect of query changes interactively.
5.6 Object Cache

To alleviate bottlenecks that arise from accessing popular data, Harvest implements an Object Cache that stores the content of HTTP, Gopher, and FTP objects that have been retrieved. The Object Cache runs as a single, event-driven process. For ease of implementation, the Cache spawns a separate process to retrieve FTP files, but retrieves HTTP and Gopher objects itself. The Cache separately manages replacement of objects on disk and objects loaded in its virtual address space. It also keeps all metadata for cached objects in virtual memory, to eliminate access latency to the metadata.

Multiple Object Caches can be arranged hierarchically for scalability. The Object Cache allows sites to customize hierarchical relationships between caches at multiple levels of the network (for example, at a campus, regional, and backbone network). Different caching parameters, such as timeouts, maximum object size, cache storage size in memory and disk, as well as caching policies, also can be customized.

5.7 Replicator

The Harvest Replicator provides a weakly consistent, replicated wide-area file system for mirroring the information that the Brokers have. This alleviates bottlenecks that arise from heavy demand on particular servers. Each file system occasionally "floods" its closest neighboring file systems with complete state information to ensure consistency, and to allow its neighbors to detect updates that for some reason have failed to propagate. The weak form of consistency used in the Replicator is called eventual consistency; if all new updates ceased, the replicas eventually converge.

The Replicator also can be used to divide the gathering process among many servers (for example, allowing one server to index each U.S. regional network) by distributing the partial updates among the replicas. The Replicator also allows sites to customize the degree of replication, topology of updates, and the frequency of updates.
5.8 Searching

With the help of a spider to collect Web pages, Harvest can index the Web information space. It uses WAIS as its backend searching and indexing engine. The Harvest Home Pages Broker has a very flexible and powerful interface, providing Boolean search queries based on author, keyword, title, or URL reference.

5.9 Flexibility

The Harvest system was designed using a component-based approach. Each component performs a critical function in the searching and retrieval task. By keeping the jobs of the components generalized, we could easily use the components to build a totally different system. Additionally we could also experiment with component designs, swapping them in and out as necessary.

The real power comes from having the search engine split into two portions, the broker and the gatherer. Many users find it very helpful to have multiple gatherers running on multiple machines. This allows for the distribution of the load on the network. Each gatherer could be set to index a certain portion of the network space. This process clearly increases the overall speed of the indexing, but it also provides another benefit. Each gatherer creates its own database, and it is very easy to limit searches to just one of these databases. Doing so allows users to target their search much more efficiently.

Likewise we could also have multiple brokers running on one machine or multiple machines. One reason for having multiple brokers is to give users more searching options. We could also have one broker that provides a quick search for users who need speed, and we could a second broker that provides a very thorough or even a phonetic search for users who are more patient. The machines running the brokers and the machines running the gatherers could be the same ones, but they don’t necessarily have to be!
6 AltaVista

Alta Vista is another commercial search engine, which claims to have indexed 150 million web pages, which is approximately 18.8% the total web space.

6.1 Crawling

- Webmaster can submit web pages to Alta Vista for indexing. About a month after submitting a page, AltaVista will visit the web site and look for other pages to add. Then it returns about every four to six weeks, to check for changes and new pages.
- AltaVista will gather many of the pages from the web site. It has no particular limit, and many sites have hundreds or thousands of pages listed.
- If key pages are missing for some reason, then it can be manually submitted again.

6.2 Ranking

Alta Vista uses the following factors to rank the pages:

1. How many times query words appear
2. Whether the query words are in the title or meta tags
3. The proximity of query words in the document

Few other factors are:

1. All search terms appear, rather than just some.
2. Search terms appear in the title, meta tags, and the first few lines of the page
4. AltaVista places a stronger emphasis on root domain home pages, in response to a single word and popular searches.
5. Short, focused titles, use of phrases in the meta keywords tag, and minimal repetition of words in meta tags.
6. Lists the older pages first, when there are a number of pages about equal in relevancy. This is not always the case, but it does appear to happen frequently.
7. AltaVista also has automatic phrase searching. The service has a dictionary of several million phrases, created using an automated process that uses linguistic patterns to determine what is a phrase. As a result, pages where search terms occur in the page in the order entered are more likely to appear than those simply containing the terms, but not in the specified order.

There are exceptions to all of these factors of course.

6.3 Indexing

1. AltaVista doesn’t index punctuation, so “webmaster’s guide” becomes “webmaster’s guide”.
2. Accents matter only if the user enters them.
3. Comments are not indexed.
4. Only the first 100K of text on a page is indexed. After that only links are indexed, up to a maximum of 4MB. Since most web pages are under 100K, these limitations should not be a problem for most webmasters.
5. Pages heavy with text in a small font size may not get listed. Avoid using font size 2 or lower as the dominant size for the page’s body copy.

6.4 Spamming

These are things AltaVista considers spamming:

- Submitting the URL repeatedly on any day.
- Submitting a large number of URLs from the same site on any day.
- Submitting identical pages from the same site.
- Repeating keywords over and over, for no good reason.
- Stuffing pages with keywords unrelated to the page’s actual content.
- Using invisible or text too small to read.
- Use of the meta refresh command.
6.5 Language Detection
AltaVista automatically categorizes web pages by language. Its spider tries to determine the language of a web page at the time it is spidered. The technology is dictionary-based. AltaVista looks at a page to see if the bulk of the words match those of a particular language.

7 Excite

Excite claims to have indexed about 125 million Web pages, which is about 15.6% of the total Web space.

7.1 Crawling

Excite’s mega spider crawls the Web every 21 days. This is the crawler that will visit most people’s web sites after an initial submission, and which will revisit on a regular basis. It pulls back everything it can gather in addition to the submitted page within a “courtesy window” of 30 seconds. After that it’ll return at a later time. Excite also runs a fresh spider that each week crawls two million top web sites, as determined by Excite.

7.2 Ranking

As with most engines, Excite favors:

- Pages with keywords in the title tag, which have repeated frequently in relation to the rest of the document.
- Words contained in complete sentences.
- **Link Popularity**: Pages that have many links pointing at them, especially those linked from popular web sites.

Excite will try to normalize documents when it ranks them. For example, a page with only a few words, including the search terms, might be considered super-relevant.
compared to a longer document where additional words “dilute” the appearances of the search term. Excite tries to adjust for these types of extremes. In mid-1998, Excite also began supplementing its raw search results by prefacing them with matching categories from the Excite directory.

7.3 Indexing

- Excite is a full-text index.
- It is unique among the major players in also providing concept searching. It means that Excite understands that words have synonyms. A search for “fruit pies”, for example, might bring up a page that only says “apple pies”. Excite knows that apple is a type of fruit and so goes beyond searching for only the keywords.
- Excite does not index comment text or ALT text.
- Excite eliminates basic pronouns and common adjectives as stop words.
- Excite supports the meta description tag. This support was added in mid-1998. If no tag exists, Excite will create a description by selecting sentences that it believes best represent the page’s content. In either case about 395 characters of text is displayed.
- Excite will display 70 characters of title tag.

7.4 Spamming

Excite tries to screen out spamming before adding to its catalog. For example if it finds a string of words such as:

money money money money money money money money

It’ll replace the excess repetition, so that essentially, the string becomes:

money xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx
If Excite cannot screen spamming, it may penalize a page. In particular the more it detects unusual repetition, the more heavily it will penalize a page. Excite does not penalize for the use of hidden text, but penalties will apply if hidden text is used to disguise spam content.

### 7.5 Regional Editions

Excite operates a series of regional editions. These share a common database, based in the US. When someone performs a country-specific search, Excite employs *domain filtering* to provide relevant results. For example, during an UK specific-search, Excite will filter out sites that are not within the .uk domain.

Obviously there are some regional-specific sites that use non-regional domains, such as those ending in .com or .net. Excite compensates for these by adding them to an inclusion list. Each regional edition also has its own *Add URL* form. These forward the URL information to the Excite spider, which will eventually add the site to the master database. Because the information goes to the same place, there is no need to submit to each of the regional guides individually.

### 7.6 Branded Search Engines

Excite “powers” the services of AOL Netfind in the US and Canada and the Netscape search engines. In both cases, the raw search engine results should be identical to the Excite results. Occasionally small differences may appear, but this is rare. Since both services draw on the main Excite database, there is no need to submit to them separately. If some site is listed in Excite, then it’ll also be listed in them.

Excite also produces the Netscape channel content, which includes directory listings. By and large, this is material from within Excite that is given a Netscape look-and-feel. So, if any site is listed with the Excite directory, the presence is also there within Netscape.
Excite’s database also powers its Magellan search results, though listings there may be noticeably different from the main Excite service, as a slightly different ranking algorithm is used. In contrast Excite’s WebCrawler service uses its own, separate index.

8 Lycos

8.1 Introduction

Dr. Michael Mauldin headed the Lycos project for Machine Translation at Carnegie Mellon University as an experiment in best first search within the Web information space. It helps users locate Web documents containing specific user-supplied keywords. Due to the comprehensiveness of its database, Lycos quickly became very popular with Web users who needed to conduct full-content searches over the space of documents formed by the Web. Lycos is claimed to have indexed 50 million of the Web pages, which is about 6.3% of the total Web space.

Lycos answers the query by searching through its huge catalogue on the web. Groups of programs called spiders search the WWW everyday (including gopher and ftp sites), and build a catalogue. The catalogue is more less a database of web addresses containing information about what is important at that address. The spider programs insure that the most popular programs are indexed first. The index created by the spiders is checked against the catalogue, to check whether the new web pages have to be added, deleted or subtracted. Lycos currently catalogues three type of files - HTTP files, gopher files and FTP file.

8.2 Crawling

The Web-wandering component of Lycos originally was derived from a program called Longlegs, written by John Leavitt and Eric Nyberg at Carnegie Mellon University. Lycos uses an innovative, probabilistic scheme to skip from server to server in Webspace. This
avoids overloading any one server with a barrage of requests, and also allows Lycos to give preference to URLs deemed more informative. The basic steps of the algorithm are as follows:

1. When a URL resource is fetched, Lycos scans its contents for new URL references, which it adds to an internal queue.
2. To choose the next URL to explore, Lycos makes a random choice among the HTTP, Gopher, and FTP references on the queue based upon preferences.

Lycos prefers to seek out popular documents, that is, those that have multiple links into them. Lycos also has a slight preference for shorter URLs, which generally are top-level directories and documents closer to the root of the hierarchy. According to Mauldin, the Lycos philosophy is to keep a finite model of the Web that enables subsequent searches to proceed more rapidly. The idea is to prune the tree of documents and to represent the clipped ends with a summary of the documents found under that node. The 100 most important word lists from several documents can be combined to produce a list of the 100 most important words in the set of documents. Lycos complies with the standard for robot exclusion, and identifies itself as Lycos by setting the HTTP User-Agent field in the request header. In this way, Webmasters can tell when Lycos has hit their server.

Lycos can retrieve documents that it has not searched by using the text in the parent document as a description for the unexplored links (the highlighted text from each HTML hyperlink anchor is associated with the URL for that anchor). Lycos does not, however, search and index ephemeral, time-varying, or infinite virtual spaces. Therefore, Lycos ignores the following spaces:

- WAIS databases
- Usenet news
- Mailto space
- Telnet services
- Local file space
Lycos also ignores files that start with "/dev/tty" or end with these extensions: AU, AVI, BIN, DAT, DVI, EXE, FLI, GIF, GZ, HDF, HQX, JPEG, LHA, MAC, MPEG, PS, TAR, TGA, TIFF, UU, UUE, WAV, Z, or ZIP.

8.3 Indexing

To reduce the amount of information that needs to be stored, Lycos extracts the following pieces of information from each document that it retrieves:

- Title
- Headings and subheadings
- 100 most important words
- First 20 lines
- Size in bytes
- Number of words

The 100 important words are selected using the Tf*IDf weighting algorithm, which considers word placement and frequencies, among other factors. Words, for example, are scored by how far into the document they appear. Thus, hits in the title or first paragraph are scored higher.

In a collection of N documents, the term frequency (Tf) is the number of occurrences of particular terms in the collection, and the document frequency (Df) is the number of documents in the collection in which particular terms occur. The idea of an inverse document frequency (IDf) is to measure how good particular terms are as a document discriminator--that is, to distinguish the few documents in which they occur from the many from which they are absent. A typical IDf factor is given by log(N/Df).
In the Tf*IDf weighting algorithm, the basic idea is that the best indexing terms are those that occur frequently in individual documents but rarely in the remainder of the collection. The importance, or weight, of a term is thus defined as the product of multiplying Tf, the term frequency, by inverse document frequency (IDf). In other words, weight = Tf * IDf = Tf * log(N/Df).

Lycos doesn’t read meta tags at all.

9 Infoseek

9.1 Introduction

Infoseek claims to have indexed 75 million Web pages, which is about 9.4% of the total Web space. The Infoseek search service is composed of two integrated services:

1. **Infoseek directory service**: Infoseek has categorized millions of pages into a topic-oriented directory. This directory is organized into about 15 channels. This is the directory of web sites submitted by the webmasters/users.

2. **Ultraseek search engine and index**: The Ultraseek database contains over 50 million pages.

The unique thing about the Infoseek service is the integration between the directory and the Ultraseek server. When a standard search is performed from the default Infoseek homepage, the search activates both the directory and the Ultraseek engine.

9.2 Crawling

After a Web page is submitted, the Infoseek spider usually will appear at the web site within a week or so to verify those pages and often to add others from the site. After that the spider is supposed to recrawl the site at least every two months. It may return more often if the pages change frequently.
9.3 Ranking

The following are the major ranking factors:

- **Early Keywords:** Page with keywords in titles and at the beginning if the page do well.
- **Meta Tags:** Keywords in meta tags give pages a relevancy bump.
- **Frequency:** Pages with a high frequency of keywords do well.
- **Directory Listing:** A major boost is given to pages from sites that are listed in its directory.
- **Link Popularity:** Infoseek uses link popularity to help boost some sites to the top.

9.4 Indexing

- Infoseek indexes ALT text
- It will find matches for some words that are spelled as single words or double words. For example, a search for “hard disk” might also find “harddisk”, and vice-versa.
- It automatically stems word forms, so a search for “swim” will also find “swims” and “swimming”.
- Descriptions are formed from meta tags or from the first characters on a page. They are generally around 200 characters long, though length can vary between 170 and 240 characters.

9.5 Spamming

Infoseek will penalize pages for using a variety of spamming techniques. The major factors are:

- Overuse or repetition of words
- Use of meta refresh faster than the human eye can see.
- Use of *any* redirection.
• Use of color text on same-color background.
• Use of keywords that do not relate to the content of the site.
• Duplication of pages with different URLs.
• Use of pages that bridge to the same URL.
• Rejects pages for using font sizes that are too small, such as font size of –1.
• Pages submitted more than once in a 24 hour period may also be constituted as spam and be removed from the index.

Spamming review is done both at the spider level and human editors, who run spot checks to on popular topics.

9.6 Infoseek Directory

Infoseek’s search results page usually starts with matching categories from its human-compiled directory. This can cause searchers to detour into the directory, and thus, it is a good reason to be listed within it. Being listed can also boost pages from a site significantly in the web index results.

9.7 User-friendliness

Infoseek has done everything possible to make all aspects of all of its products incredibly user-friendly. Of all the major public search engines, Infoseek has the best thought out interface design. Infoseek’s user-friendly atmosphere extends throughout its Ultraseek Server search software. Installation, maintenance, and usage are all easier and more intuitive than with any of the other commercial search engine packages.

9.8 Flexibility and Feature-richness

Infoseek has balanced flexibility and feature-richness with ease of customization, resulting in a product that takes minimal amount of time to configure, yet provides all of the power of its competitors.
9.9 Benefits

1. We can specify the number of spiders (threads) that are indexing or crawling the Web actively at any given time. We can start and stop the threads individually through the web-based administration screens.

2. Ultraseek also allows for title searching, URL searching, host name searching, image searching, and metatag searching.

3. Ulrtaseek contains support for multiple indexes, called *collections* Ultraseek-speak. Each collection can index a different section of an Intranet. Users can search each collection separately or any grouping of collections at the same time.

10 Google

10.1 Introduction

Google is a large-scale search engine, which addresses many of the problems of existing systems. It makes especially heavy use of the additional structure present in hypertext to provide much higher quality search results. The goal of the system is to address many of the problems, both in quality and scalability, introduced by scaling search engine technology to such extraordinary numbers. Google is claimed to have indexed about 85 million Web pages, which is about 10.6% of the total Web space.

Both the rate of growth of the Web and technological changes are considered in designing Google. Google is designed to scale well to extremely large data sets. It makes efficient use of storage space to store the index. Its data structures are optimized for fast and efficient access. The main goal of Google search engine is to improve the quality of web search.
10.2 System Features

The Google search engine has two important features that help it produce high precision results:

- It makes use of the link structure of the Web to calculate a quality ranking for each web page. This ranking is called PageRank.
- It utilizes link to improve search results.

10.3 Architecture

Figure 3: Architecture of Google
A high level overview of how the whole system works is given as pictured in figure 3. Most of Google is implemented in C or C++ for efficiency and can run in either Solaris or Linux. In Google, several distributed crawlers perform the Web crawling. There is a URLserver that sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the storeserver. The storeserver then compresses and stores the Web pages into a repository. Every Web page has an associated ID number called a docID, which is assigned whenever a new URL is parsed out of a Web page. The indexer and the sorter perform indexing. The indexer performs a number of functions. It reads the repository, uncompressed the documents, and parses them. Each document is converted into a set of word occurrences called hits. The hits record the word, position in document, an approximation of font size, and capitalization. The indexer distributes these hits into a set of barrels, creating a partially sorted forward index. The indexer performs another important function. It parses out all the links in every web page and stores important information about them in an anchors file. This file contains enough information to determine where each link points from and to, and the text of the link.

The URLresolver reads the anchors file and converts relative URLs into absolute URLs and in turn into docIDs. It puts the anchor text into the forward index, associated with the docID that the anchor points to. It also generates a database of links, which are pairs of docIDs. The links database is used to compute PageRanks for all the documents.

The sorter takes the barrels, which are sorted by docID, and sorts them by wordID to generate the inverted index. This is done in place so that little temporary space is needed for this operation. The sorter also produces a list of wordIDs and offsets into the inverted index. A program called DumpLexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. The searcher is run by a web server and uses the lexicon built by DumpLexicon together with the inverted index and the PageRanks to answer queries.

10.4 Crawling the Web
In order to scale to hundreds of millions of web pages, Google has a fast-distributed 
crawling system. A single URLserver serves lists of URLs to a number of crawlers 
typically ran about 3. Both the URLserver and the crawlers are implemented in Python. 
Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve 
web pages at a fast enough pace. At peak speeds, the system can crawl over 100 web 
pages per second using four crawlers. This amounts roughly to 600K per second of data. 
A major performance stress is DNS lookup. Each crawler maintains its own DNS cache so it does not need to do a DNS lookup before crawling each document. 
Each of the hundreds of connections can be in a number of different states: looking up DNS, 
connecting to host, sending request, and receiving response. These factors make the 
crawler a complex component of the system. It uses asynchronous IO to manage events, 
and a number of queues to move page fetches from state to state.

10.5 Indexing the Web

10.5.1 Parsing

Any parser that is designed to run on the entire Web must handle a huge array of possible 
errors. These errors range from typos in HTML tags to kilobytes of zeros in the middle of 
a tag, non-ASCII characters, HTML tags nested hundreds deep, and a great variety of 
other errors. For maximum speed, instead of using YACC to generate a CFG parser, flex 
is used to generate a lexical analyzer that is outfitted with its own stack.

10.5.2 Indexing Documents into Barrels

After each document is parsed, it is encoded into a number of barrels. Every word is 
converted into a wordID by using an in-memory hash table -- the lexicon. New additions 
to the lexicon hash table are logged to a file. Once the words are converted into wordID's, 
their occurrences in the current document are translated into hit lists and are written into 
the forward barrels. The main difficulty with parallelization of the indexing phase is that 
the lexicon needs to be shared. Instead of sharing the lexicon, Google took the approach
of writing a log of all the extra words that were not in a base lexicon, which it fixed at 14 million words. That way multiple indexers can run in parallel, and then one final indexer can process the small log file of extra words.

10.5.3 Sorting

In order to generate the inverted index, the sorter takes each of the forward barrels and sorts it by wordID to produce an inverted barrel for title and anchor hits and a full text inverted barrel. This process takes place one barrel at a time, thus requiring little temporary storage. Also, the sorting phase is parallelized to use many machines simply by running multiple sorters, which can process different buckets at the same time. Since the barrels don’t fit into main memory, the sorter further subdivides them into baskets that do fit into memory based on wordID and docID. Then the sorter, loads each basket into memory, sorts it and writes its contents into the short inverted barrel and the full inverted barrel.

10.6 Searching

The goal of searching is to provide quality search results efficiently. To put a limit on response time, once a certain number (currently 40,000) of matching documents are found, the searcher automatically goes to step 8 in Figure 4. This means that it is possible that sub-optimal results would be returned.
Google Query Evaluation

1. Parse the query
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at all in the end of any doclist go to step 4.
8. Sort the documents that have matched by rank and return the top k.

Figure 4: Google Query Evaluation

10.7 The Ranking System

Google maintains a lot of information about web documents. Every hitlist includes position, font, and capitalization information. Additionally, hits are factored in from the anchor text and the PageRank of the document. Combining all of this information into a rank is difficult. Ranking functions are designed so that no particular factor can have too much influence.

10.7.1 A Single Word Query

In order to rank a document with a single word query, Google looks at that document's hit list for that word. Google considers each hit to be one of several different types (title, anchor, URL, plain text large font, plain text small font, ...), each of which has its own type-weight. The type-weights make up a vector indexed by type. Google counts the number of hits of each type in the hit list. Then every count is converted into a count-
weight. Count-weights increase linearly with counts at first but quickly taper off so that more than a certain count will not help. The dot product of the vector of count-weights with the vector of type-weights is taken to compute an IR score for the document. Finally, the IR score is combined with PageRank to give a final rank to the document.

10.7.2 Multi-word Query

Now multiple hit lists must be scanned through at once so that, hits occurring close together in a document are weighted higher than hits occurring far apart. The hits from the multiple hit lists are matched up so that nearby hits are matched together. For every matched set of hits, proximity is computed. The proximity is based on how far apart the hits are in the document (or anchor) but is classified into 10 different value "bins" ranging from a phrase match to "not even close". Counts are computed not only for every type of hit but for every type and proximity. Every type and proximity pair has a type-prox-weight. The counts are converted into count-weights and the dot product of the count-weights and the type-prox-weights is taken to compute an IR score. All of these numbers and matrices can all be displayed with the search results using a special debug mode. These displays have been very helpful in developing the ranking system.

10.8 Storage Requirements

Aside from search quality, Google is designed to scale cost effectively to the size of the Web as it grows. One aspect of this is to use storage efficiently. Due to compression, the total size of the repository is about 53 GB, just over one third of the total data it stores. At current disk prices this makes the repository a relatively cheap source of useful data. More importantly, the total of all the data used by the search engine requires a comparable amount of storage, about 55 GB. Furthermore, most queries can be answered using just the short inverted index. With better encoding and compression of the Document Index, a high quality web search engine may fit onto a 7GB drive of a new PC.
10.9 Performance

The Google indexer runs at roughly 54 pages per second. The sorters can be run completely in parallel; using four machines, the whole process of sorting takes about 24 hours. The current version of Google answers most queries in between 1 and 10 seconds. This time is mostly dominated by disk IO over NFS (since disks are spread over a number of machines). Furthermore, Google does not have any optimizations such as query caching, sub-indices on common terms, and other common optimizations. The designers of Google intend to speed it up considerably through distribution and hardware, software, and algorithmic improvements. Their target is to be able to handle several hundred queries per second.

10.10 Scalability

Google is designed to be scalable in the near term to a goal of 100 million web pages. All of the time consuming parts of the system are parallelized and roughly linear time. These include things like the crawlers, indexers, and sorters. Most of the data structures will deal gracefully with the expansion. However, at 100 million web pages Google will be very close up against all sorts of operating system limits in the common operating systems (currently it is run on both Solaris and Linux). These include things like addressable memory, number of open file descriptors, network sockets and bandwidth, and many others. Expanding to a lot more than 100 million pages would greatly increase the complexity of the system.

11 Yahoo

11.1 Introduction

Yahoo is not a search engine, but is a high quality directory that depends on humans for its listings. A short description of a Web page is submitted to Yahoo directory or editors
write one for the site they review. A search looks for matches only in the descriptions submitted. The changes to the Web pages submitted have no effect on the listing.

How the site is ranked for particular terms depends on the listing that is submitted. A yahoo editor may change that listing, or it may not be accepted at all. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics.

11.2 Indexing and Ranking

A search engine visits a web site, then lists each page from a site independently from each other. Ranking depends on the content of the page. In contrast, Yahoo accepts submissions that describe an entire web site, not individual pages. A form is filled with information about the site. This is reviewed by a Yahoo editor and, if approved, is added to the guide. If a site is not submitted, then chances are that the web site will never be listed. Again, the importance the search engines may place on meta tags, page titles, and body copy makes no difference with Yahoo. Everything revolves around information on the form that is submitted, though it is important that the Web site be of good quality and that content there reflect the description that is sent to Yahoo.

When users enter search terms Yahoo checks its catalog of web sites and returns listings in this order:

- Yahoo categories containing the terms.
- Sites with the terms in their titles.
- Sites with the terms in their descriptions.

12 Comparison of Search Engines

The following charts provide comparison of important features of different search engines. This chart is as of September 1, 1999.
12.1 Crawling

This section covers factors related to how well search engines crawl web sites.

<table>
<thead>
<tr>
<th>Crawling</th>
<th>Yes</th>
<th>No</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep crawl</td>
<td>AltaVista, Google</td>
<td>Excite, Lycos</td>
<td>Excite changes to deep crawl 10/99</td>
</tr>
<tr>
<td>Instant Indexing</td>
<td>AltaVista</td>
<td>Excite, Google, Lycos</td>
<td>Pages will appear within a day or two after submission</td>
</tr>
<tr>
<td>Frames support</td>
<td>AltaVista, Google</td>
<td>Excite, Lycos</td>
<td>Lycos provides limited support</td>
</tr>
<tr>
<td>Image Maps</td>
<td>AltaVista</td>
<td>Excite, Google, Lycos</td>
<td></td>
</tr>
<tr>
<td>Robots.txt</td>
<td>All</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Meta Robots Tag</td>
<td>All</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Link Popularity Helps</td>
<td>Lycos</td>
<td>AltaVista, Excite</td>
<td></td>
</tr>
<tr>
<td>Deep Crawl</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learns Frequency</td>
<td>AltaVista</td>
<td>Excite, Google, Lycos</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparison of Crawling Features

- Deep Crawl

The search engines shown doing deep crawls will list many pages from a web site, even if the pages are not explicitly submitted to them. The others will usually list far fewer pages from a site. In general, the larger a search engine's index is, the more likely it will have many pages per site.
• **Instant Indexing**

At an instant indexing search engine, usually any page that is submitted will appear within a day or two after submission.

• **Frames Support**

This shows which search engines can follow frame links. Those that can't will probably miss listing much of the submitted web site.

• **Image Maps**

This shows which search engines can follow client-side image maps. As with frames, those search engines that can't follow image maps will probably miss listing much of the submitted web site.

• **robots.txt**

The robots.txt file is a means for webmasters to keep search engines out of their sites.

• **Meta Robots Tag**

This is a special meta tag that allows site owners to specify that a page shouldn't be indexed. It is ideal for those who cannot create a robots.txt file. To keep spiders out, simply add this text between the header tags on each page we don't want indexed:

```html
<META NAME="ROBOTS" CONTENT="NOINDEX">
```

There is no need to use variations of this tag to help the page get indexed. They are unnecessary. And no need to use this tag if robots.txt file is already used.

• **Link Popularity Helps Deep Crawl**
All search engines can determine the popularity of a page by analyzing how many links there are to it from other pages. Some engines use this as a means to determine which pages they will include in the index. This is NOT the same as ranking a page better for having good link popularity. That is explained further below.

- **Learns Frequency**

A number of search engines can learn how often pages change. Pages that change often may be visited more frequently.
12.2 Indexing

This section explains what gets indexed when search engines spider a page.

<table>
<thead>
<tr>
<th>Indexing</th>
<th>Yes</th>
<th>No</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Body Text</td>
<td>All</td>
<td>N/A</td>
<td>Some stop words may not be indexed</td>
</tr>
<tr>
<td>Stop words</td>
<td>AltaVista, Excite, Lycos, Google</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Meta Description</td>
<td>All but…</td>
<td>Google, Lycos</td>
<td></td>
</tr>
<tr>
<td>Meta Keywords</td>
<td>All but…</td>
<td>Excite, Google, Lycos</td>
<td></td>
</tr>
<tr>
<td>ALT Text</td>
<td>Altavista, Lycos</td>
<td>Excite, Google</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>None</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>Stemming</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Comparison of Indexing Features

- **Full Body Text**

  All of the major search engines say they index the full visible body text of a page, though some will not index stop words or exclude copy deemed to be spam (explained further below).

- **Stop Words**
Some search engines either leave out words when they index a page or may not search for these words during a query. These "stop words" are excluded as a way to save storage space or to speed searches.

- **Meta Description & Meta Keywords**

  This shows which search engines support the meta description and meta keywords tags. This does NOT mean that using these tags gives pages a ranking boost. That is covered in a separate section, below.

- **ALT Text / Comments**

  This shows which search engines index ALT text associated with images or text in comment tags.

- **Stemming**

  Some search engines will search for variations of a word based on its stem. For example, entering "swim" might also find "swims" and maybe "swimming," depending on the search engine.

12.3 **Ranking**

Most search engines use the location and frequency of keywords in a web page as the basis of ranking. The exact mechanism is slightly different for each engine.

In addition to location and frequency, some engines may give a page a relevancy boost based on other factors. These usually can help a little, but they don't guarantee a boost to the top. Some major factors are listed below.
<table>
<thead>
<tr>
<th>Ranking</th>
<th>Yes</th>
<th>No</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Tags Boost Ranking</td>
<td>None</td>
<td>AltaVista, Excite,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google, Lycos</td>
<td></td>
</tr>
<tr>
<td>Reviewed Status</td>
<td>None</td>
<td>AltaVista, Excite,</td>
<td></td>
</tr>
<tr>
<td>Boosts Ranking</td>
<td></td>
<td>Google, Lycos</td>
<td></td>
</tr>
<tr>
<td>Link Popularity</td>
<td>AltaVista, Excite, Google</td>
<td>Lycos</td>
<td>Very important at Google</td>
</tr>
<tr>
<td>Boosts Ranking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct Hit Boosts</td>
<td>None</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>Ranking</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3: Comparison of Ranking Features**

- **Meta Tags Boost Ranking**

  Some search engines that support the meta description and keywords tag will also give pages an extra boost if search terms appear in these areas. Not all search engines that support the tags also give a ranking boost.

- **Reviewed Status Boosts Rankings**

  Some search engines also review sites or list them in an associated directory. They may also give a boost to sites that have been listed in this way.

- **Link Popularity Boosts Rankings**
As described above, all search engines can determine the popularity of a page by analyzing how many links there are to it from other pages. Some engines give pages with lots of links, or links from important web sites, a relevancy boost.

- **Direct Hit Boosts Rankings**

Direct Hit is a system that measures what users click on from search results in order to refine relevancy rankings. This shows which search engines use this as a factor.

12.4 **Spam**

All major search engines penalize sites that attempt to "spam" the engines in order to improve their position. One common technique is "stacking" or "stuffing" words on a page. This is where a word is repeated many times in a row. If the search engines spot a spamming technique, they may downgrade a page's ranking or exclude it from listings altogether. The items below cover design elements that could cause a spam penalty.

<table>
<thead>
<tr>
<th>Spam</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Refresh</td>
<td>AltaVista, Lycos</td>
<td>Excite, Google</td>
</tr>
<tr>
<td>Invisible Text</td>
<td>Others</td>
<td>Excite, Google</td>
</tr>
<tr>
<td>Tiny Text</td>
<td>AltaVista, Lycos</td>
<td>Excite, Google</td>
</tr>
</tbody>
</table>

*Table 4: Comparison of Spam Features*

- **Meta Refresh**

Some site owners create target pages that automatically take visitors to different pages within a web site. The meta refresh tag is one typical way of doing this. Some search engines will refuse to index a page with a high meta refresh rate. “Go” will not index
pages with any redirection, whatsoever. Google doesn't worry much about meta refresh
tags or the items below because its link popularity ranking system defeats spam attempts.

- **Invisible Text**

  This is the technique of placing text on a page in the same color as the background,
  making it invisible to human viewers. Many search engines either refuse to index this text
  or will not index any page containing invisible text.

- **Tiny Text**

  This is the technique of placing text on a page in a small font size. Pages that are
  predominantly heavy in tiny text may be dismissed as spam. Or, the tiny text may not be
  indexed. As a general guideline, try to avoid pages where the font size is predominantly
  smaller than normal.

### 12.5 Search Engine Features For Searchers

The search engine features chart below is designed primarily for users of search engines.
It summarizes key search commands and search assistance features.

*Note:* This chart covers AltaVista, Excite, Google, Infoseek, Lycos, WebCrawler and
Yahoo. This chart is as of May 24, 1999.
### 12.5.1 Search Engine Math Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>How</th>
<th>Supported By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include Term</td>
<td>+</td>
<td>All</td>
</tr>
<tr>
<td>Exclude Term</td>
<td>-</td>
<td>All</td>
</tr>
<tr>
<td>Phrase</td>
<td>“ “</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Note: Semiautomatic at AltaVista, Google)</td>
</tr>
<tr>
<td>Match Any Term</td>
<td>Auto</td>
<td>AltaVista, Excite, Infoseek, WebCrawler, Yahoo</td>
</tr>
<tr>
<td></td>
<td>Menu</td>
<td>Lycos</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>Google (impossible)</td>
</tr>
<tr>
<td>Match All Terms</td>
<td>Auto</td>
<td>Google, Lycos</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>Can be done at all using + symbol or menu options</td>
</tr>
</tbody>
</table>

Table 5: Comparison of Search Engine Math Commands
### 12.5.2 Power Searching Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>How</th>
<th>Supported By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title Search</td>
<td>title:</td>
<td>AltaVista, Infoseek</td>
</tr>
<tr>
<td></td>
<td>other</td>
<td>some above via menus,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lycos (via menu), Yahoo (t:)</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>Excite, Google, WebCrawler</td>
</tr>
<tr>
<td>Site Search</td>
<td>domain:</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>AltaVista (host:), Infoseek (site:), Lycos (via menu)</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>Excite, Google, WebCrawler, Yahoo</td>
</tr>
<tr>
<td>URL Search</td>
<td>URL:</td>
<td>AltaVista, Infoseek</td>
</tr>
<tr>
<td></td>
<td>other</td>
<td>Lycos (via menu), Yahoo (u:)</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>Excite, Google, WebCrawler</td>
</tr>
<tr>
<td>Link Search</td>
<td>Link:</td>
<td>AltaVista, Infoseek, Google</td>
</tr>
<tr>
<td></td>
<td>linkdomain:</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>Excite, Google, WebCrawler, Yahoo (N/A)</td>
</tr>
<tr>
<td>Wildcard</td>
<td>*</td>
<td>AltaVista, Yahoo</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>Excite, Google, Infoseek, Lycos, WebCrawler</td>
</tr>
</tbody>
</table>

Table 6: Comparison of Power Searching Commands
12.5.3 Search Assistance Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Yes</th>
<th>No</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Searches</td>
<td>AltaVista, Excite, Infoseek</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>Clustering</td>
<td>Infoseek</td>
<td>Others</td>
<td>Excite and Google have some clustering features</td>
</tr>
<tr>
<td>Find Similar</td>
<td>Excite, Infoseek</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>Stemming</td>
<td>Infoseek, Lycos</td>
<td>AltaVista, Excite</td>
<td></td>
</tr>
<tr>
<td>Date Range</td>
<td>AltaVista, Yahoo</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>Search Within</td>
<td>Infoseek, Lycos</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>AVista, Infoseek</td>
<td>Excite, Lycos</td>
<td></td>
</tr>
<tr>
<td>Direct Hit</td>
<td>Lycos</td>
<td>Others</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Comparison of Search Assistance Features
### 12.5.4 Display Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Yes</th>
<th>No</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort By Date</td>
<td>Infoseek</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>Date Displayed?</td>
<td>AltaVista, Infoseek</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>Increase Number Of Results?</td>
<td>Excite, Google, Infoseek, Lycos, WebCrawler</td>
<td>AltaVista</td>
<td>N/A for Yahoo</td>
</tr>
</tbody>
</table>

Table 8: Comparison of Display Features
12.5.5 Boolean Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>How</th>
<th>Supported By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>OR</td>
<td>All but...</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>Google, Infoseek, Yahoo</td>
</tr>
<tr>
<td>And</td>
<td>AND</td>
<td>All but...</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>Google, Infoseek, Yahoo</td>
</tr>
<tr>
<td>Not</td>
<td>NOT</td>
<td>All but...</td>
</tr>
<tr>
<td>AND NOT</td>
<td></td>
<td>AltaVista, Excite</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>Google, Infoseek, Yahoo</td>
</tr>
<tr>
<td>Nesting</td>
<td>( )</td>
<td>All but...</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>Google, Infoseek, Yahoo</td>
</tr>
<tr>
<td>Near</td>
<td>NEAR</td>
<td>AltaVista (10 words), Lycos (25 words), WebCrawler (2 words)</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>Others</td>
</tr>
</tbody>
</table>

Notes
At AltaVista, Boolean only works on advanced search page
At Excite-powered services, Boolean commands must be in UPPERCASE

Table 9: Comparison of Boolean Commands

12.6 Search Engine Sizes

The charts below show the size of each search engine's index. The larger the index, the more likely the search engine will be a comprehensive record of the web. That's especially useful for those looking for obscure material.

12.6.1 Current Size Comparison
Sizes are as reported by each search engine and as of September 1, 1999.
NOTE: Google estimates its size to be between 70 to 100 million pages, so 85 million is taken as an average. The service also says that because it can actually list pages it has never visited, its reach may extend to 300 million pages.
12.6.2 Search Engine Sizes Over Time

![Search Engine Sizes](chart.png)

When AltaVista appeared in December 1995, it used an index much larger than any other search engines at that time. Thus, competition forced most of them to increase their sizes in early 1996. Notice that from September 1996 until September 1997, none of the search engines increased size significantly, despite the fact that the web continued to grow.

12.6.3 Coverage of the Web

In July 1999, a study published in Nature estimated that there were 800 million indexable pages on the web. It also surveyed the size of search engines as of February 1999. The chart below shows how the study's estimates varied from the reported sizes of search engines in February 1999.
If the bars are close together, it suggests that the reported sizes given by the search engine are indeed correct.

If the "Reported" bar is higher than the "Estimated" bar, it suggests that the reported size for that search engine is inflated. It could also indicate that the research was not correct for that search engine.

Where the "Estimated" bar is higher than the "Reported" bar is higher, it suggests that the research may not have been correct for that search engine, or that the search engine itself underrepresented its index.

The chart below uses the Nature estimate of 800 million web pages as a basis for showing the percentage of the web covered by each search engine, using that estimate versus each search engine's current size, as of Sept. 1, 1999:
Over the years the search engines have evolved into very sophisticated Internet search tools. During early years, a search engine was judged by the percentage of the web indexed by it. But now the emphasis is shifting toward being the best index, rather than the largest index. That usually means being large, because most of the search engines want to have a complete account of the Web. But that also means having fresh information, something difficult to do when millions and millions of Web pages are to be checked up. None of the search engines index everything on the Web. No search engine can claim to have a perfect record of everything out there. Improving the relevancy of search means being a smarter search engine, not just bigger. The amount of information that can be quickly and efficiently searched for using search engines has been increasing, but what has been decreasing is the amount that can be searched versus what could be potentially searched for.

**Figure 8: Percentage of Web Indexed**

13 Conclusion

Over the years the search engines have evolved into very sophisticated Internet search tools. During early years, a search engine was judged by the percentage of the web indexed by it. But now the emphasis is shifting toward being the best index, rather than the largest index. That usually means being large, because most of the search engines want to have a complete account of the Web. But that also means having fresh information, something difficult to do when millions and millions of Web pages are to be checked up. None of the search engines index everything on the Web. No search engine can claim to have a perfect record of everything out there. Improving the relevancy of search means being a smarter search engine, not just bigger. The amount of information that can be quickly and efficiently searched for using search engines has been increasing, but what has been decreasing is the amount that can be searched versus what could be potentially searched for.
References


11. How does Lycos work? [http://ntrg.cs.tcd.ie/4ba2.95/Indexing/lycos.html#q1](http://ntrg.cs.tcd.ie/4ba2.95/Indexing/lycos.html#q1)

12. WebCrawler Search Features – [http://is099.tsc.k12.in.us/training/SEARCH/WEBCRAWL/AboutSea.htm](http://is099.tsc.k12.in.us/training/SEARCH/WEBCRAWL/AboutSea.htm)