Vrije Universiteit Brussels

Faculty of Science

Department of Computer Science

Academic Year 2001-2002

Enhancing Web Information Search Using Meta-Searching and Web Automatons

Elijah Kipkosgei Tenai

Thesis presented as partial fulfillment of the requirements to obtain the degree of Master of Science in Computer Science

Promotor:

Prof. Dr. Olga De Troyer

Abstract

The Web has emerged as a crucial information propagation medium in the information age. As a consequence, availability of information on the Web has exploded. Equipping consumers with better Web search tools is therefore becoming a matter of priority for Web researchers.

Standard Web search engines have remained popular and, indeed, are very essential tools for Web users to search for useful information. However, users often are faced with the limitations of these search engines. These limitations include presence of obsolete data in their databases, limited coverage of the WWW, vulnerability to keyword spamming and inaccurate ranking of search results.

This thesis is founded on an attempt to demonstrate alternative approach to Web searching, which can improve upon traditional approaches to Web searching. A tool was developed, which combines meta-searching and use of Web automatons to process user queries and produce higher quality results than individual regular search engines.

Acknowledgements

My sincere gratitude goes to my thesis Promotor Prof. Dr. Olga De Troyer for her guidance and invaluable advice that helped shape this thesis work.

My study at VUB was on VLIR Scholarship. I am most grateful to Prof. Dr. G. Eisendrath, Prof. K. Mavuti and Dr. T. M. Waema for the worthwhile opportunity.

I am privileged to have had friends who were there for me. Thanks John, Ibrahim, Jane, Carol and Miora for helping me not take things too seriously. On the same note, I also wish to express my genuine gratitude to staff at EDUCO for the numerous acts of kindness they accorded me.

Special ‘thank you’ must go to my fiancée, Betty, who has had to put up with a lot with incredible endurance during my period far away from home pursuing this study endeavor. Thank you for your love, patience, support and the countless reasons we know.

Most importantly, I thank almighty God for the awesome gift of life and for keeping my heart from trusting in vanity. In humble reverence, I owe my whole being to Him.

Contents

1. Introduction

The modern age has been dubbed the "information age" following the crucial role that exchange of information and ideas has taken shape worldwide. General affordability of IT tools and widespread accessibility of the Internet and WWW has led to what has been hailed the global information boom. Availability of information has exploded. However, this overwhelming availability of information has not necessarily translated to widespread use. In particular, users on the WWW often find themselves swamped with information but with few tools to help gather and manage relevant information for their consumption. It is thus becoming equally crucial that as the Internet and WWW evolves and grows by the minute [Powel & Wicke, 1995], so should the routes and tools to negotiate passage to their information resources.

Web search engines have remained the prime information discovery mechanism on the Web. According to GVU 8th survey [GVU, 1999], 84 % of WWW users use search engines to find information on the Web. Out of the spirit of competition, frequent enhancements are made by owners of search engines to appropriately index available Web information resources in order to improve communication by providing efficient access to the growing amount of information. However, there are still limitations to the current search engines, improvements may help to maximize the benefits of the Web. Most search engines, for example, lack in comprehensiveness and the timeliness of Web information coverage. In many available search engines, users have to contend with the deficiency in efficiency, overall coverage and precision in information searches.

Thus, as the worth of the Web, as the predominant information propagation medium, continues to rise, equipping Web information consumers with better Web information search tools continue to merit greater significance in Web research.

The motivation for this thesis is pivoted on this idea. The thesis attempts to explore ways of enhancing Web information search by improving accuracy and effectiveness of Web search through use of meta-searching and Web automatons to address the limitations that users who rely on single standard engines face. These limitations include:

- ✓ Inadequate coverage of the WWW information resources. Combining search results from multiple search engines (meta-searching) can be employed to solve this problem.

- ✓ Presence of obsolete content in some engines' indexes resulting in outdated outcomes on search results, for example in situations where indexed sites exhibit highly ephemeral characteristics and the Web search engines are slow to update their indexes. Automatons could be used to quickly check and eliminate such results for the user.

- ✓ Vulnerability of the search engines to keyword spamming by Website developers that cause deceptive increase in the ranks of their pages on search results. Web automatons could be used to compute genuine ranks for Web pages found on search results by downloading the pages and locating the query-terms and, where necessary, re-rank the outcomes of the search for the user.

- ✓ Occasional inaccessibility of certain search engines due to engine or network problems. A tool that automates meta-searching will overcome this problem by returning results from other search engines that are available.

1.1 Purpose

This thesis demonstrates the application of Web automatons and meta-searching to help Web users deal with limitations of search engines and the long and tedious process of searching for information on the Web.

1.2 Research Methodology

A Meta-search tool, which includes a multiengine search facility, Web robots and relevancy checking module, was developed.

The tool has an intuitive user-interface. Among other functions, the interface allows a user to select and simultaneously submit queries to multiple search-engines and to retrieve and browse through the aggregated or a particular search engine's search results. The interface includes a simple Web browser to enable a user to visualize the results of the search and to follow hyper-links. The interface is equipped with appropriate navigational tools to enable a user to perform most of the rudimentary Web browsing tasks.

The tool uses Web robots to automatically retrieve each Web page referenced by the returned Web search results, for purpose of relevancy checking, and to detect and eliminate obsolete hits from search results.

The relevancy-checking module applies a uniform scoring measure, based on degree of relevance to the search query, to all pages referenced in the search results returned by the multiple search engines. The tool downloads and analyzes the actual pages assigning each a score. The scores are then used as the basis for ranking the aggregated search results, with the most relevant results ranked first.

1.3 Overview

This thesis document is divided to seven chapters, organized as follows. The current chapter, chapter one, presented a statement of the problem that is addressed in the rest of this thesis, describing the motivation, the purpose, and the research methodology used.

The next two chapters will provide review of background information where fundamental ideas for the implementation of the thesis project were drawn up. Chapter two gives an overview of the basic operations, the techniques used, and the challenges faced by Web search engines.   Chapter three discusses extensively on Web automatons, commonly called Web Robots, Spiders or Crawlers. Specifically, the chapter defines and describes what Web Robots are, how they work, the concerns and ethics of their operations, a breakdown of the possible application areas they can be utilized for, among other issues.

Chapter four presents a description of the developed Meta-search tool. The underlying architecture of the tool is described in detail, followed by a discussion of the implementation issues and operational features of all the significant components of the tool.

Chapter five elaborates on the experiments performed with developed Meta-search tool and the results obtained.

Chapter six contains the concluding remarks. It covers the summary and highlights of the contributions of the thesis, and some suggestions for future work.

Finally the references, Websites and bibliographic materials, consulted in the course of the development of this thesis work are listed in chapter seven.

2. Web Search Engines: Overview of Operation and Challenges

2.1. Web Search Engines

The World Wide Web is currently the most advanced information system deployed on the Internet, and embraces within its data model most information in previous networked information systems [Berners-Lee]. This has resulted in a highly distributed information retrieval system consisting of varied documents, software, sounds, images, and other file system data located on different computers dispersed around the Internet. The World Wide Web's popularity has been growing enormously and has made colossal amounts of information available for consumption by any person, anywhere in the World.

Concomitant with the immense growth of the Web is the difficulty in locating relevant information; given the sheer amount of information, locating information of interest can be a chaotic and messy process, without the aid of appropriate navigation tools, Pinkerton in [Pinkerton, 1994] calls it " the resource discovery problem".

Web search engines have remained the most popular way to locate information on the Web. A myriad of search engines exist on the Web [Eagan & Laura, 1996]. These engines employ a variety of tools and techniques for discovering and retrieving information on the Web for users. These techniques are geared to best harness the proliferation of information on the Web.

In a nutshell, the Web search engines are expected to do two things: Systematically gather and index data from the Web, and other Internet sources, and, thereafter, provide access to a databases with these results through a search facility.

To gather and index data from the Internet, search engines use automated programs, called robots, which traverse many Web pages in the Web, by recursively retrieving linked pages, and scanning through them for relevant information. As a robot gathers the information from the pages, it also updates the search engine database with the information. Robots are also known as Spiders, Crawlers, Wanderers and Worms (See chapter 3).

Many search engines also offer catalogs of topics for those who prefer to browse. The catalogs are created manually and give a hierarchical ordering of Web pages [Weiss, 1996] clustered by relevance to the concepts they contain. Yahoo[4] is a popular example of a search engine that implements catalogs extensively.

Some Web search engines do not search the Web themselves to build databases. Instead, they allow searches to be sent to several search engines all at once then merge the results on one page, a technique called Meta-searching (See 2.4). Search engines employing the technique are normally referred to as Meta-search engines. Some examples of Meta-search engines on the Web include Metacrawler[5], Dogpile [6] and Mamma [7].

The search facility is ordinarily accessed via a search Form on a Web page found at the search engine Website. Typically, a user types a query on the Form and submits to the search engine. The search engine searches through its database for information which best match the query and returns a list of links (normally termed hits) to the matching resources. The list of matches is usually ranked according to how relevant they are deemed to be to the search query.

A number of parameters determine the success of search engines. These include: the speed of searching, the availability of search options and features, size, content and currency of the database, the relevancy of search results, how they are presented, and the overall ease of use. But generally speaking, a good search engine should provide results that are easy to understand and with enough information to make a decision about the usefulness of the results.

2.2. Information Retrieval Techniques on Web Search Engines

The challenge for Web search engines is not in the volume of hits they can produce. Search engines can generally produce a lot of hits. The important issue is, can a user get what he or she wants?

What features prominently in evaluation of the quality of search engines is their effectiveness in producing relevant results. That is, how best a search engine can retrieve information relevant to user queries, while minimizing the amount of irrelevant information.

In their paper, Christos and Douglas [Christos & Oard] survey the techniques currently available to meet this challenge. They have divided the techniques into two classes: The more traditional methods, which include full text scanning, signature files, inversion, and clustering; and the more recent methods, which attempt to improve query precision and capture more semantic information about each document in order to achieve better performance. These methods include concept-based searching, query expansion, artificial intelligence, natural language processing, latent semantic indexing, probabilistic logic, query by example, and the use of neural networks.

The algorithms implementing the traditional approach methods rely on the occurrence of a given set of keywords to identify the possibly relevant documents. The beauty with these methods is their simplicity. Keyword matching is one of the simplest methods of determining level of relevancy of documents to a query. The idea is that if a query is described by certain keywords, then a document containing those keywords has high chance of being relevant to the query. The obvious drawback to these methods is that only a small amount of the information associated with a document is used as a basis for making the decision on whether a document is relevant or not. Christos and Oard [Christos & Oard] observe, however, that despite this inherent limitation, these methods often achieve acceptable precision because the full text of a document often contain a significant amount of redundancy.

The other significant inadequacy with the traditional approach methods is that they do not attempt to make actual “understanding” of the input query. As a result, the techniques may, for instance, produce spurious results that are out of a user’s interests when a user types ambiguous keywords on the query (say for example “Bug”). Further, many relevant documents may be left out when generating search results, especially in domains where the same concept can be described by many different words (e.g. information retrieval, resource discovery, searching) [Kass et al, 1992]. Only checking for patterns of keywords is therefore not enough to model appropriate relevancy-checking mechanisms. Semantic and contextual information is also required.

Modeling a user’s interests by semantic and contextual analysis of his her query demand information retrieval systems with relatively high level of intelligence. Advances in artificial intelligence research, in particular natural language understanding, is providing the necessary technology that can help in the design of such “intelligent” information retrieval systems. Such systems determine the relevance of a text based on an analysis of the relationship of the content or semantics of the text to a user’s query by use of methods that incorporate use of sophisticated term weighting, automatic thesauri, natural language processing, relevance feedback, and advanced pattern matching. Such is the trend of techniques employed by modern search engines.

The search capabilities and features vary between search engines [Notess]. A comprehensive Web site dedicated to search engine comparisons at http://www.searchenginewatch.com/, maintained by Danny Sullivan, has a summary of characteristics of major search engines.

Many search engines also use Boolean logic (such as ‘AND’ and ‘OR’ operators) to assist users in more effectively stating what they want to find by making the query more precise

2.3. Factors Used in Determining Page Relevancy

A user typically presents a query, which may include several keywords to search engine, and expect the search engines to produce matches to the query sorted and ranked, so that the most relevant ones come first.

Search engines rank results according to a set of criteria in their algorithms. Exactly how a particular search engine's relevancy checking algorithm work is a closely kept trade secret [Sullivan, 2001]. However, major search engines generally consider some of the following factors:

✓ Keyword Frequency

A search engine will analyze how often (frequent) keywords appear in a Web page. Frequency is a count on the number of times a keyword appears on a page or in an area on the page. Those with a higher frequency are often deemed more relevant than other Web pages [Sullivan, 2001].

✓ Keyword Density

Keyword density, also known as keyword weight, is the percentage or concentration of keywords in page in relation to all other words on the page. The calculation varies, depending on whether the keyword is a single word or a multi-word phrase. The higher the weight the better, but only to some extend. If the weight becomes too high some search engines may suspect "keyword spamming" and probably penalize.

✓ Keyword Prominence

Keyword Prominence is a measure of how close to the start of a page area a keyword appears. Typically, a keyword that appears closer to the top of the page or area will be considered more relevant.

✓ **Keyword Proximity**

Keyword proximity means how close keywords are to each other. In general the closer the search keywords are to each other in a document the more relevant a document will be considered to be.

✓ **Site Popularity**

Some search engines rank sites in part based on the site’s link popularity [Sullivan, 2001]. This ranking measurement considers a site’s "value" based on the number of other Web sites whose owners felt the site was sufficiently important to link to. If a lot of other Web sites link to a site, chances are the site is relatively important and thus should be ranked higher.

2.3 The Challenge of Information Discovery on the Web: The Invisible Web

Information discovery is the process of finding and retrieving information resources that are relevant to the user of a system. The “find” process involves locating the resources and presenting these to the user as a possible (and partial) solution of the find [RDU, 1995]. Discovering documents on the Web simply means learning their identities on the vast Web information space. Identities of documents of the Web currently take the form of Uniform Resource Locators (URL).

Information discovery is the business of Web search tools -Web search engines. In a perfect case, these tools are expected to “discover” and provide users access to any resource on the Web. But the sheer abundance, high rate of increase and variety of documents on the Internet make this universal coverage of Web information not only impractical, but a daunting task to be accomplished by any single engine. In their paper [Lawrence & Gilles, 1999] the authors remark, “the major Web search engines index only a fraction of the total number of documents on the Web. No engines indexes more than about one third of the ‘publicly indexable Web’”. They attribute this situation to “a limitation on the engines’ network bandwidth, disk storage, computational power, or a combination of these.”

A study conducted by search company BrightPlanet in [Bergman, 2001] estimates that the inaccessible part of the Web is about 500 times larger than what search engines already provide access to. It can, therefore, be construed a gigantic amount of information on the Web in reality remain to be or cannot be discovered by an individual or a search engine. This (undiscovered information) constitute part of the Web that’s normally termed as the “Invisible Web”.

In [Bergman, 2001] two groups of Web content are identified. The "surface" Web, that consists of a group of static, publicly available Web pages, and which everybody knows as the “Web”. The other is the group he calls the "deep" Web, which consists of specialized Web-accessible databases, and dynamic Web sites that only produce results dynamically in response to a direct request.

Public information on the deep Web is said to be currently 400 to 550 times larger than the commonly defined World Wide Web [Bergman, 2001].

Bergman notes that most of the traditional search engines cannot "see" or retrieve content in the “deep” Web. The reason is because traditional search engine robots cannot probe beneath the surface, i.e. cannot retrieve those pages that do not exist until they are created dynamically as the result of a specific search. This group of Web content is therefore not widely known by "average" surfers despite the immense amount of information it holds.

In [Bailey, et al] the authors have aptly referred to the “inaccessible” Web information as the “Dark Matter” of the Web, in analogy to the dark matter of the Universe, as used in astronomy. They’ve classified them as follows:

✓ Rejected “Dark Matter”

The “dark matter” that arises because a page is rejected for some reason and thus did not load [Bailey, et al]. Dynamically generated Web content, i.e. “deep” content, falls in this category. Robots lack the ability to search a database directly, thus cannot load the dynamically generated pages, causing them to be rejected.

"Surface" content can also be rejected. A good example is where a robot trying to index some site chooses to exclude itself from parts of a site for policy or convention reasons, such as when robots.txt file at the site specify a resource should not be indexed.

✓ Restricted "Dark Matter'

These are linked material, but which are only restricted to observers with access permissions - require usernames and passwords. Without the permissions, indexing robots cannot access the information.

✓ Undiscovered "Dark Matter"

These are pages that remain undiscovered, maybe because necessary links, which would otherwise locate the material, are never found.

✓ Removed "Dark Matter"

The Dark matter in this category arise when information, available at some point in time, is removed or is no longer available. There are two subcategories:

- **Ephemeral "Dark Matter"**
  This category of dark matter arises when a URL, pointing to a site, is retained but the information, which was once available, is replaced by new information.

- **Dead "Dark Matter"**
  These arise when both the URL and the Page, that once contained information, are removed completely from public access on the Web.

2.4 Meta-Searching

Meta-searching describes the process of searching by querying multiple search engines in parallel. Tools implementing meta-searching provide a unified query interface to the multiple search engines (see fig 2.1). The users get an illusion that they are searching on a single information source. Meta-search engines do not necessarily maintain their own databases but search those of other engines in order to provide the information necessary to fulfill user queries. MetaCrawler[5], one popular meta-search engine, for instance, searches the databases of each of the following engines: About, Ask Jeeves, Fast, Excite, FindWhat, LookSmart, OpenDirectory, Overture and Sprinks.

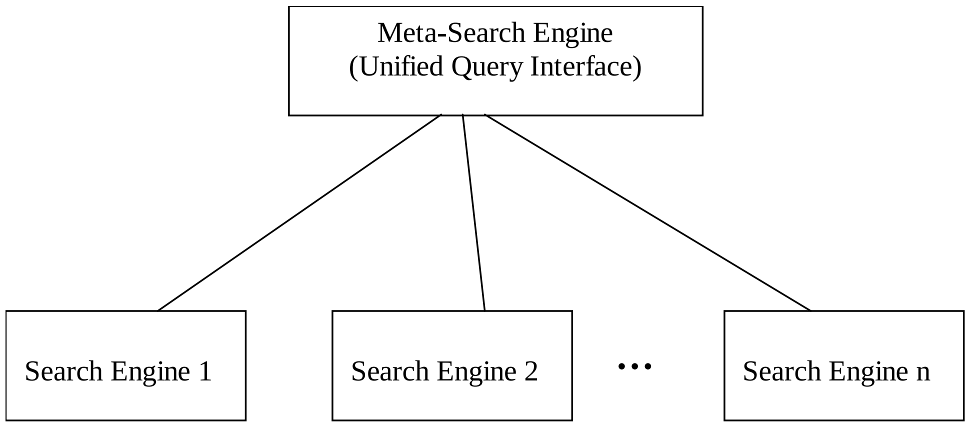


Fig. 2.1: A Simple (2 level) Meta-Search Engine Architecture

A key basis for the support of Meta-searching is the potential of this technique to address the problem of information discovery. As already outlined (see 2.3), standard search engines are generally limited in their coverage of the Web. The study done in [Lawrence & Gilles, 1999] made an interesting observation, that despite significant growth in the databases of the major search engines, at least as of the time of the study, the level of overlap of Web coverage between major search engines was low. They found out that by meta-searching using six engines in the study, they could achieve Web coverage of approximately 3.5 times as much over an average engine and twice the coverage of the largest engine.

Expanding the Web coverage, through Meta-search engines, help increase the chance that a user will find what he or she wants from the Web, particularly if the information he or she is searching for is obscure or belong to highly focused and sporadic domain.

Meta-searching could also be combined with other features such as mechanisms to eliminate duplicate hits on the collated results, mechanisms to verify information's existence, mechanisms to perform relevancy checking, or use of supplementary query syntax supplied by the user, in order to return pre-processed results that are most relevant and useful to the user.

3. Web Robots: The Automatons of the World Wide Web

In the context of the World Wide Web, Robots, which are also often called Spiders, Wanderer, Crawlers, Gatherers and so on, are programs that automatically traverse the Web information space, by recursively retrieving linked pages, and reporting their findings. They automate routine online tasks for users and make the vast information resources of the Web, which would otherwise be unmanageable by humans, more accessible.

They are considered robots rather than programs because they are somewhat autonomous: They often make decisions about what pages to retrieve without prompting from their users. They are also social in that they can react to their environment and communicate with both humans and non-humans. Possibly it is for these reasons that the evocative term “agent” is also used when referring to them; they are often called “Web agents”. [Nwana, 1996] has a different opinion however. He asserts: “Spiders are strictly speaking not agents because, even though they explore, autonomously, the topology of the Web; generally, they neither learn nor collaborate with other Spiders, yet.”

Robots are, nevertheless, the most successful class of automatons on the Web, providing extremely valuable service to Internet users. Without them, it would be almost impossible to build operational Web indexes and keep them up to date. To paraphrase [Eichmann]:

“The Web is highly dynamic, and will remain so for the foreseeable future. Robots are too valuable a user resource given the chaotic nature of the Web for them not to be employed, even if their usage is forced into simulation of user behavior in order to avoid detection.”

Many of the so-called “Internet Starting Points[1]” use robots to scour the Web looking for new information [Ressler, 2000] and turn the diverse information sites on the Web into usable information resources.

3.1 How Do Web Robots Work?

To begin with, it is essential to point out that the names Spiders, Wanderer, Worms or Wanderers are potential source of confusion on the way Web robots operate. The names give the false impression that the robot itself moves. Current Web robots do not move however. In reality, they are implemented as a single software system that retrieves information from remote sites using standard Web protocols (HTTP requests) [Koster, 1995]. Given a Uniform Resource Locator (URL) address for a Web document, a robot first establishes a connection to the remote server, issues an HTTP[2] GET[3] command to retrieve the document, and after that process the retrieved document at the host machine.

“A Web robot works like a browser with an autopilot.” [Prosise, 1996]

A Web robot works by starting with a queue of known URLs to pages waiting to be processed. Initially it could be a singleton, also termed as the ‘seed’ or the root.

Taking a URL from the queue, the robot, just like a Web browser, downloads the referenced page. The robot then scans the page for links to other pages and adds them to the queue of URLs waiting to be processed. It then picks the next URL from the queue (in some order) then retrieve the referenced page and repeat the process over again - as many times as necessary. Depending on how the robot is programmed, the robot could be made to follow links to other sites or follow only links within the same server host or domain.

[1] Also called Jumpstations: essentially search engines or Web directories
[2] HyperText Transfer Protocol – Protocol used in exchanging documents on the Web
[3] HTTP command for requesting a document from a Web Server

As the robot downloads each page it accordingly executes the functions the robot is intended to do, such as identifying "dead" links, gathering some statistics regarding Web usage, building index for some search engine, etc.

Once it's turned loose, a robot that employs the described simple recursive navigation algorithm can cover vast distances in cyberspace, and, because the Web changes daily, the robot's path can change daily, too. In a conceptual sense, the robot "crawls" the Web like a spider. All it needs is a place to start. [Prosise, 1996]

3.2 Web Robot Crawling Techniques

Web robots can be designed to automatically traverse the hypertext hierarchy using two approaches: Breadth-First or Depth-First.

3.2.1 Breadth-First Crawling Approach

The idea in this approach is to retrieve all the pages around the starting point first before following links further away from the start, as illustrated in fig 3.1 below:

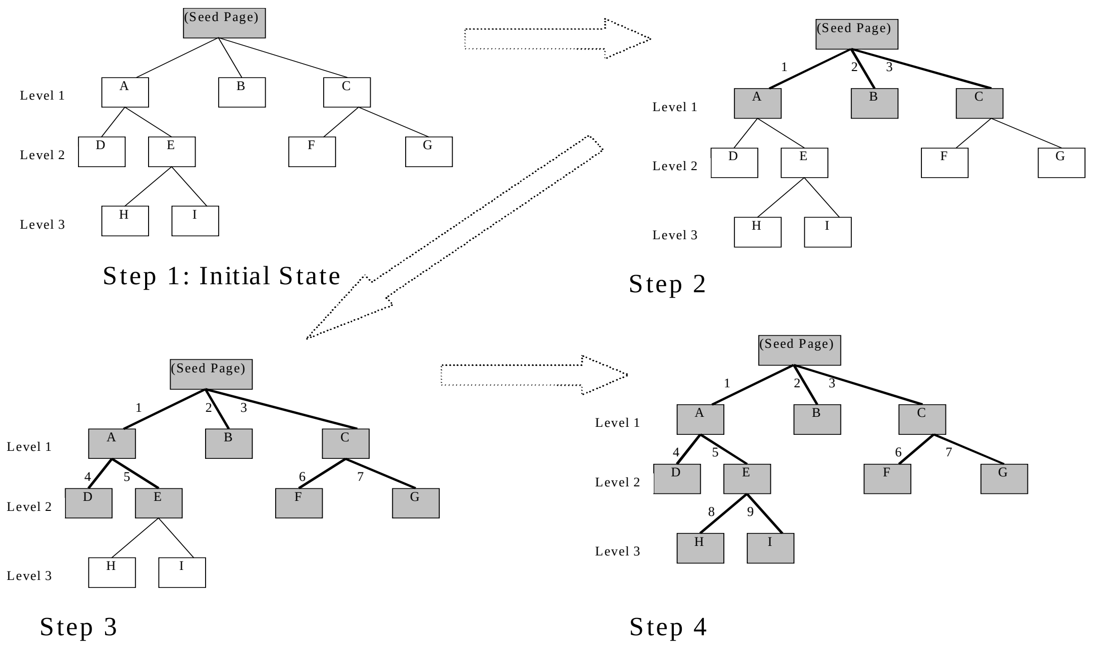


Fig 3.1: Illustration of Breadth-First Crawling

Fig 3.1 provides a graphical illustration of Breadth-First crawling. As usual, the robot starts with a seed page, which forms the starting point of the crawl process (step 1). All the pages referenced by the links extracted from this page, represented by level 1, will be visited by the robot first (step 2), followed by those they link (level 2) as illustrated in step 3, and finally by those further down the hypertext hierarchy (level 3), as shown in step 4. The numbers against the edges of the tree structure illustrate the order in which the succession of visits is made.

Traversing the Web graph in breadth-first search order is a good crawling strategy. It tends to discover high-quality pages early on in the crawl. [Najork & Wiener, 2001]

If the links are pointing to several hosts, this approach can distribute the load quickly, so that no one server must respond to requests continually.

It's also easier for robot programmers to implement parallel processing using this approach. For example, using concurrent threads in Java (simultaneous program paths), it is possible to process the several pages on same level in parallel. For this reason, this approach is the most common way robots are programmed to follow links.

3.2.2 Depth-First Crawling Approach

The other approach, Depth-First crawling, employs a narrow, but deep, way of traversing the hypertext structure. This is in contrast to the wide and shallow traversal in Breadth-First approach. Starting from the seed page, the robot picks the first link on the page and follows it, then the first link on the second page, and so on until it cannot go deeper, returning recursively. The crawl process is illustrated by transitions in fig 3.2 below:

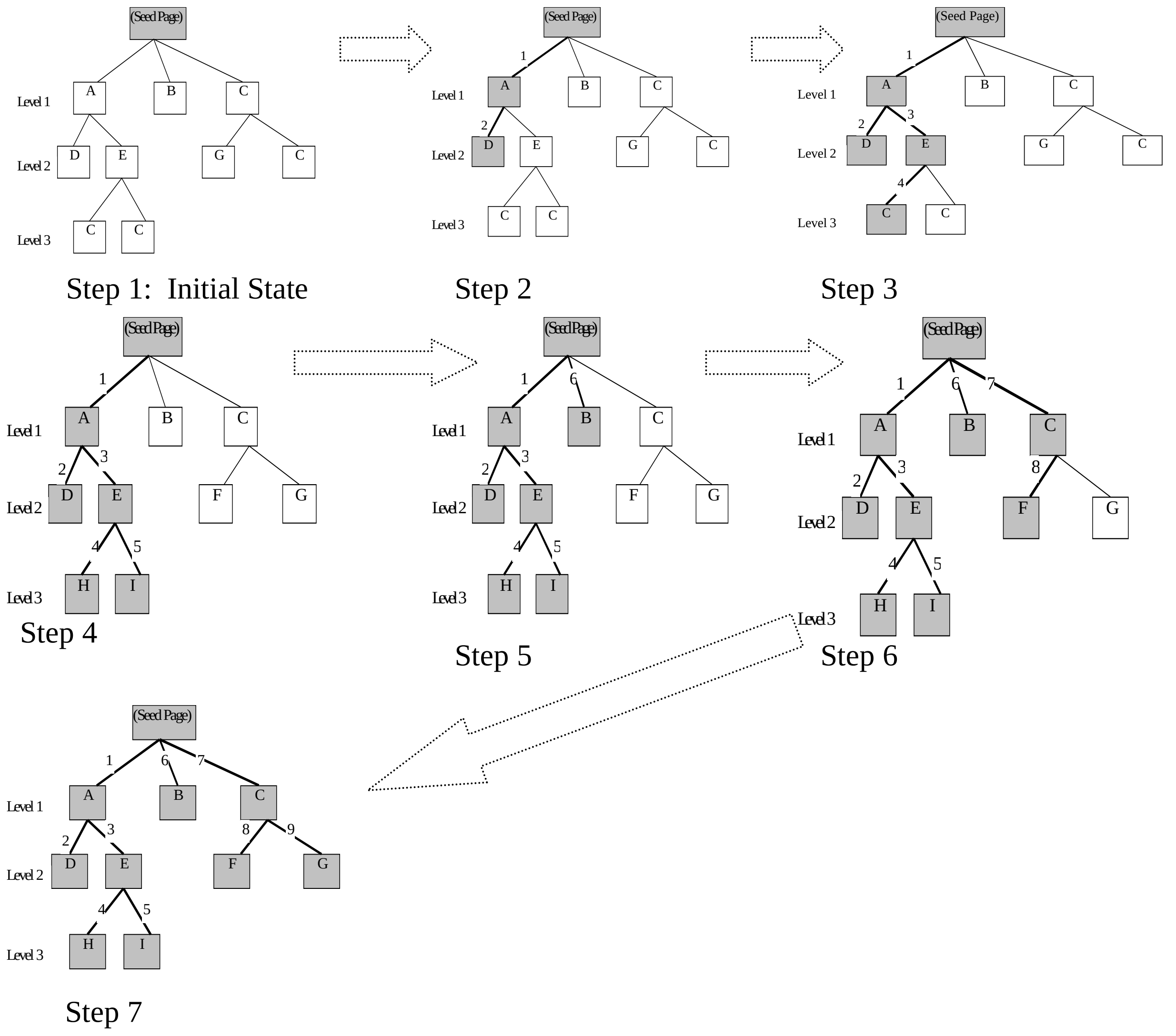


Fig 3.2: Illustration of Depth-First Crawling

Fig 3.2 presents a graphical illustration of Depth-First crawling. Starting from a seed page, which forms the starting point of the crawl process, the robot goes as far down each link path as possible before trying the next link path

One problem with Depth-First approach is the high possibility of a badly written robot getting “lost” pursuing a single thread, especially if the maximum depth is not defined - the robot may expend too much time on one single site at the expense of other Websites. It is also difficult to implement parallel visits to pages.

By and large, Depth-First approach is somewhat unsophisticated. A Breadth-First search generally provides a better description of a Website than a Depth-First approach, especially if not all the pages are to be crawled.

3.3 Other Issues with Web Crawling

3.3.1 Focused vs. Generic Crawling

The explosive dimensions of the World Wide Web pose extraordinary scaling challenges that for some information seekers, general-purpose crawlers and search engines sometimes proof inadequate. “The sheer diversity of content at a generic search site snares all but the craftiest queries in irrelevant results” [Chakrabarti].

A section of Web users demand deep, high-quality information on specific subjects. Focused crawlers, rather than ordinary generic crawlers, may be most useful in this case.

Focused crawlers are programmed to selectively seek out pages that are relevant to a pre-specified set of topics. The topics are presented to the crawler in form of example documents. Focused crawlers use a hypertext classifier that can learn from the given examples in order to identify Web pages and sites that are relevant to the topics from the Web.

Unlike a generic crawler that collects and indexes all accessible Web documents in order to be able to answer all possible ad-hoc queries, a focused crawler analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the Web. This leads to significant savings in hardware and network resources, and helps keep the crawl more up-to-date [Chakrabarti].

3.3.2 Spidering Depth

Another issue with robots is how deep they can be allowed to crawl into a site. As illustrated by Fig. 3.1 and Fig. 3.2, this refers to the number of levels down the depth of the hypertext hierarchy the robot is programmed to go, starting from a seed page at level

0. Obviously the deeper it is, the longer the spidering process a robot may be forced to take.

Appropriate spidering depth is determined from the characteristics of the sites to be visited. Some sites have the most important information at the top-level pages and the pages deep in the site less relevant, in which case the robot can visit only the top few levels of the Websites and collect all necessary information. In other sites, the top-level pages may contain mainly lists of links, and the detailed content deeper in. As a result the robot would be required to navigate deeper the hierarchy to obtain the necessary information.

3.3.3 Server Load

It's not hard to imagine how robots can place a strain on a Web server's resources [Prosise, 1996]. When a Web robot visits a site, it may use multiple connections to request a large number of pages in a short space of time - as if a hundred or more extra users are logged in all at once. This may lead to a significant degradation in the performance of the Web server, such as slowing down of the server, denial of access to other Web users or even server crash.

[Koster, 1993] has raised reasonable concerns about this potential hazard of Web robots. He insists that robot designers should exercise reasonable restraint, especially on the scope and frequency of execution of page requests, in order to minimize potential Web load on the Web servers. A good robot should be careful not to tie system resources by requesting pages from the server at top speed. This is particularly important when the robot is limited to a single host and the server is busy with user interactions. [Koster, 1993] suggests that wherever possible, "the robot should employ clever techniques to rotate queries between different servers in a round-robin fashion in order to distribute load and minimize impact on one particular server".

3.4 Robots Ethics

As already noted, robots traversing the Web have potentially detrimental impacts if not checked. Besides placing demands on network, a Web robot also places extra demand on servers. The strain placed on the network and hosts can be aggravated to even worse levels by badly written robots. A group of Internet users have recognized this problem and have attempted to evolve a code of ethics and guidelines for developing responsible Web robots. A more widely cited of these guidelines is Martijn Koster's “Guidelines for Robots Writers” [Koster, 1993].

Martijn Koster's Guidelines for Robots Writers [Koster, 1993] were meant to address the increased load on Web servers by robots. It served as a basis for discussion amongst robot authors and resource providers to create the first community pressure on robot authors to act ethically [Eichmann]. The "Guidelines" together with unofficial standard known as Standard for Robot Exclusion (SRE), which originated from the WWW Robots Mailing List, are supposed to have been the first wide-ranging attempt at Web robot ethics.

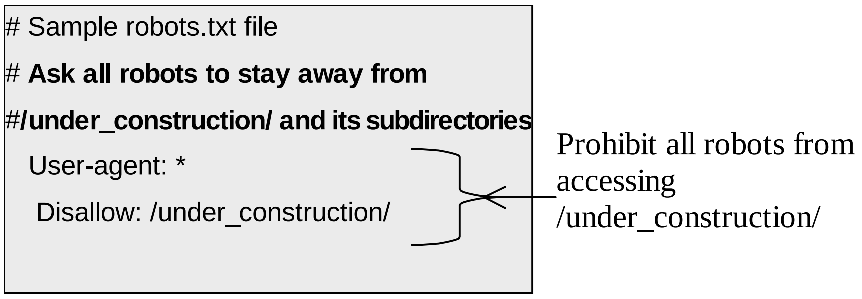
3.4.1 The Standard for Robot Exclusion (SRE)

Standard for Robot Exclusion defines a protocol that permits Web masters to limit what robots can do on their sites. Using the protocol, people can exclude robots from designated areas of their Web sites.

The standard states that before a robot can access a server, it must first look for a robots.txt file on the server. This file contains explicit directives that tell which Robots are excluded from visiting certain URLs at the site, or the complete site altogether.

The robots.txt file should be HTTP-accessible and must be located in the root directory of the Web server, for example http://www.mywebsite.com/robots.txt.

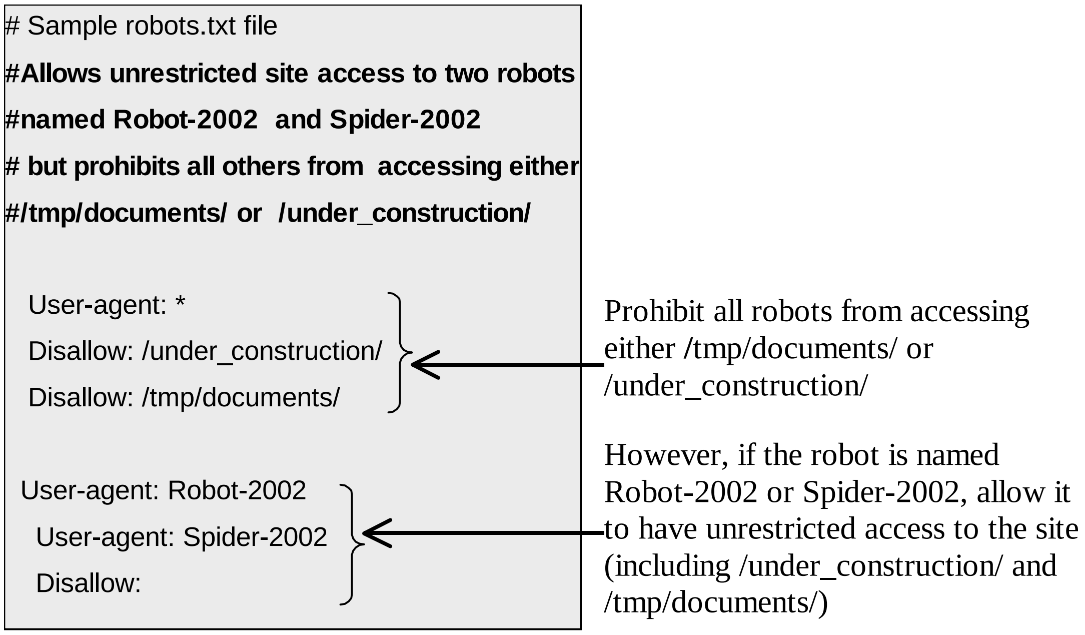
The syntax of robots.txt is relatively simple. Each section includes the name of the user-agent (robot), or * for all user-agents, and a disallow field which describes the paths the robot should not follow. Comments can be included, by preceding each line with #.
Fig 3.3 and fig 3.4 are examples of robots.txt files.



```
# Sample robots.txt file
# Ask all robots to stay away from
#/under_construction/ and its subdirectories
  User-agent: *
   Disallow: /under_construction/
```

Prohibit all robots from accessing /under_construction/

Fig 3.3: Sample Robots.txt file. Note ‘*’ means “all” and ‘#’ precedes comments.



```
# Sample robots.txt file
#Allows unrestricted site access to two robots
#named Robot-2002  and Spider-2002
# but prohibits all others from  accessing either
#/tmp/documents/ or  /under_construction/

  User-agent: *
  Disallow: /under_construction/
  Disallow: /tmp/documents/

 User-agent: Robot-2002
   User-agent: Spider-2002
   Disallow:
```

Prohibit all robots from accessing either /tmp/documents/ or /under_construction/

However, if the robot is named Robot-2002 or Spider-2002, allow it to have unrestricted access to the site (including /under_construction/ and /tmp/documents/)

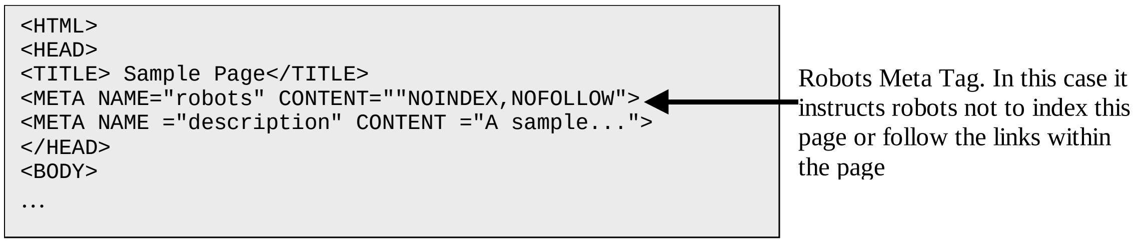
Fig 3.4. Sample robots.txt file

It should be noted, however, that conformity to the directives in robots.txt is voluntary on the part of the robot. The standard is not enforced [Prosise, 1996]. Some robots are not SRE-Compliant and can ignore the directives on the file. For the SRE standard to work, a robot must be programmed to look for a robots.txt file that would tell it exactly what it should and should not do on a particular site. A robot that obeys the directives in the robots.txt file is considered SRE-Compliant.

3.4.2 Robots Meta Tag

Web page developers can also specify on individual Web pages whether they can be indexed by robots, or that the links on the page should be followed by robots or not. This is useful where a page author does not have appropriate privileges to change the robots.txt file.

A robots Meta tag, placed in the HTML <HEAD> section of a page, can specify either or both of these actions, see fig 3.5.



```
<HTML>
<HEAD>
<TITLE> Sample Page</TITLE>
<META NAME="robots" CONTENT=""NOINDEX,NOFOLLOW">
<META NAME ="description" CONTENT ="A sample...">
</HEAD>
<BODY>
…
```

Robots Meta Tag. In this case it instructs robots not to index this page or follow the links within the page

Fig. 3.5: Illustration of Robots Meta Tag

The table below summarizes the syntax for the Robots Meta Tag content[4]:

Meta Tag Syntax	Notes
<meta name="ROBOTS" content="NOINDEX">	Do not index, but follow links
<meta name="ROBOTS" content="NOFOLLOW">	Index, but do not follow links
<meta name="ROBOTS" content="NOINDEX,NOFOLLOW">	Do not index or follow links
<meta name="ROBOTS" content="INDEX,FOLLOW">	Index and follow Links. This is the default behavior

Note: Currently only few robots support this tag[5].

[4] Adapted from" HTML Author's Guide" - http://www.robotstxt.org/wc/meta-user.html
[5] "HTML Author's Guide" - http://www.robotstxt.org/wc/meta-user.html

3.4.3 Summary of Guidelines

Below are some other general conventions for writing Web robots, excerpted from [Koster, 1993]

✓ Accountability***:*** A robot should identify itself and the owner to allow server maintainers set the robot apart from human users using interactive browsers and contact the owner in case of problems. HTTP protocol headers have fields like "USER-AGENT" and "From", for example, which can be used to present the name of the robot and contacts of the author respectively to the Web server.

✓ Robots should be careful not to hog resources:
To minimize impact on the Web servers, robot authors should try to keep to the following:

- Make sure the robot runs slowly. This is will be particularly important if the robot will be making multiple and successive visits to a single host
- Make sure the robot asks only for what it can handle - both in format and in scale. For example, if a link points to a ".ps", ".zip", ".gif" etc, and the robot only handles text, there is no point asking for such a document.
- Ensure the robot does not loop or repeat (Web is a graph, not a tree). The robot should keep track of all the places it has visited to check that it is not looping
- If known, the robot should be set to run at opportune times. On some systems there are preferred times of access, when the machine is only lightly loaded, the robot should be set to work at such times.

3.5 Uses of Robots

Robots can be used to perform a number of useful tasks [Koster, 1995]. These include Web indexing, mirroring and link maintenance. Robots can also be used gather data, for statistical analysis of the Web.

✓ Resource Discovery

This is perhaps the most widely applied use of Web robots. Web has too much material, and it's too dynamic [Koster, 1993]. It’s thus only natural that humans have had to delegate the work of information gathering on the Web to robots.

A good example is the way most search engines use robots to build their databases. Starting with few links, given to it as “seed”, such a robot starts to crawl the Web. When it encounters new pages and links, it gathers the summarizing information (such as titles or headings) from the page or retrieves certain occurrence of words in whole page. It then stores this information, organized in some predetermined form, in a database, which in turn can be made available for queries through the search engine.

The robot could also be made to automatically, at regular intervals, crawl the web to update the database with new information and delete old information when detected.

With this, all a Web user needs to do is simply initiate a query on the database. Even if the database doesn't contain the exact item he wants to retrieve, it is likely to contain references to related pages, which in turn may help reference the target item [Koster, 1995].

✓ Mirroring

Another application of robots is mirroring. Mirroring is essentially copying a collection of files associated with a Website and making them available at another Website - probably across continents.

Using mirroring, it is possible to avoid unnecessary network load between remote sites and, by creating local copies of the original site, substantially decrease access times. Mirroring could also be used to create redundancies to cope with situations where the original host is temporarily unavailable or overloaded.   But these mirrors must be maintained up-to-date, a task which can be tedious and cumbersome if done manually. Web robots come handy automatons to this process.

A robot can accomplish mirroring by starting at the home page of a target server and recursively traversing through its entire links to determine the files that need to be copied then retrieving copies of them and storing at a local machine - perhaps after appropriately updating the links the files reference.

✓ Maintenance

One difficulty in maintaining hypertext structures is that hyperlinks to other pages may become "broken”, when HTML documents they reference are moved or removed. Robots that perform automatic validation of links can be used to locate and notify Web authors of these broken links.

“Robots can perform lots of tedious housekeeping for us “ [Annunziato].

✓ Statistical Analysis

Robots have also been used to gather statistical data for quantitative measurements and analysis of various issues relating to the Web. An example is the NontriSpider robot used at Kasetsart University, Thailand, in March 2000 to generate statistical analysis of the status of Thai Web based information. Full details are available at http://anreg.cpe.ku.ac.th/links/thaiwebstat/index.html. The robot gathered data that were used to investigate certain properties of Web documents, such as distribution of file sizes, number and type of embedded images, top ten big domains, language usage, etc.

4. The Developed Meta-Search Tool

4.1 Introduction

This chapter discusses a Meta-search tool developed for the purpose of this thesis and implemented based on the ideas described in the preceding chapters. First, the architecture of the tool is described in detail; next a discussion of some implementation issues is given. In chapter 5, some experiments carried out with the tool are described. These experiments allow us to study the performance and relative effectiveness of some search engines used by the tool.

4.2 The Architecture

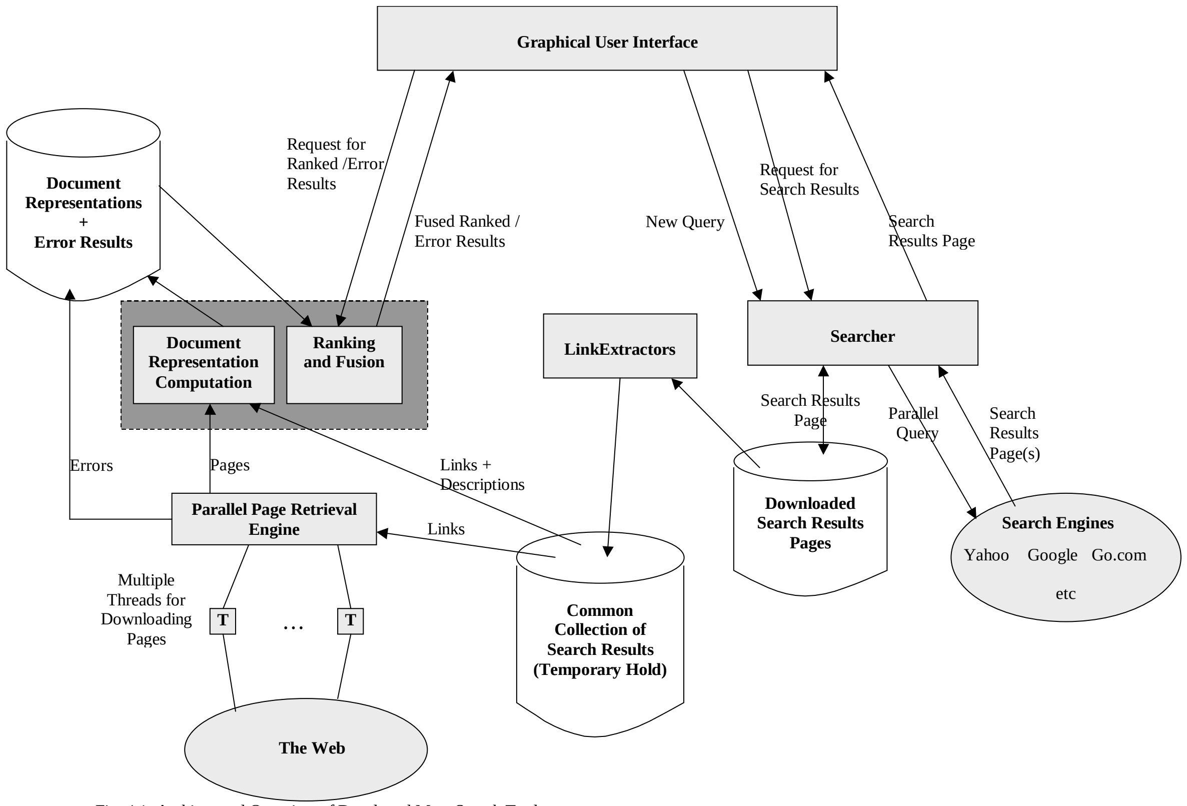

Graphical User Interface
Request for Ranked /Error Results
Fused Ranked / Error Results
New Query
Request for Search Results
Search Results Page
Document Representations + Error Results
Document Representation Computation
Ranking and Fusion
LinkExtractors
Searcher
Search Results Page
Parallel Query
Search Results Page(s)
Errors
Pages
Links + Descriptions
Parallel Page Retrieval Engine
Links
Downloaded Search Results Pages
Search Engines
Yahoo Google Go.com etc
Multiple Threads for Downloading Pages
T
…
T
Common Collection of Search Results (Temporary Hold)
The Web

Fig. 4.1: Architectural Overview of Developed Meta Search Tool

The tool is essentially composed of five components, which encapsulate fairly distinct functions. These are: The Graphical User Interface, the Searcher, the Link Extractors, Document Representation, Fusion and Ranking component and the Parallel Page

Retrieval Engine. Each of these components will be discussed more extensively in the sub-sections that follow.

The way the tool operates, in brief, is as follows:

1. From the *Graphical User Interface* a user types a query, then chooses the search engines to be queried before submitting the query to the engines

2. The query and the names of the selected engines are first passed to the *Searcher* component

3. The *Searcher* component formulates query strings that match the query syntax for each of the selected search engines before dispatching, in parallel, the appropriate search request to the separate engines.

4. The *Searcher* component then retrieves the returned search results from the engines and store in a virtual database of *Downloaded Search Results Pages*. The database is actually implemented using Hash Map container in Java.

5. From there, the *LinkExtractors* are invoked to pre-process the *Downloaded Search Results Pages*. There is a *LinkExtractor* for each search engine, which encapsulates specific algorithm used to analyze pages returned by the search engine. Each *LinkExtractor* parses through the search results page belonging to its search engine extracting, from each search result it encounters, the hyperlink, the anchor text, the snippet describing the result, among other information, and storing in the (virtual) database of *Common Collection of Search Results*. As they do so, they ensure no duplicate matches, for instance where they are returned by the more than one Web search engine, are stored in the common database.

6. The *Parallel Page Retrieval Engine* utilizes the *Common Collection of Search Results*. Implemented as a robot, the *Parallel Page Retrieval Engine*

automatically retrieves all the pages referenced in the *Common Collection of Search Results,* passing each of them to the *Document Representation Computation* Component. The *Document Representation Computation* component generates necessary keyword vectors representing each document (the term document here refers to WWW page) it receives and stores the vectors, together with the URL and other details of the document as found in *Common Collection of Search Results*, in *Document Representations Database*. If a document cannot be retrieved, probably because the link is broken or malformed, its URL and description is stored in the database of *links with errors*. Upon examination of a link, its associated entry in the *Common Collection of Search Results* is deleted.

7. From the Graphical User Interface a user can choose to browse either the individual search engine results, the combined and ranked search results or descriptions of all hits that encountered errors during retrieval.

8. Individual search engine results can be viewed by retrieving the respective engine's search results page from the *Downloaded Search Results Pages* database.

9. When a user requests for combined and ranked search results, the *Ranking and Fusion* component first runs through the *Document Representations* database computing the scores of WWW pages represented in the database using their keyword vectors. It then sorts the results in the database by their scores before generating a HTML formatted document of the results, which is returned for display.

10. If a user requests for error results (those hits that had errors during attempts to retrieve them for document representation computation), the *Ranking and Fusion* component simply generates a HTML document fusing the entries in the database of *Error Results*, which is then presented to the user.

4.3 Framework for Document Representation, Fusion and Ranking

The purpose of a meta-search system is to provide a unified access to results from multiple search engines. As already discussed a meta-search engine does not maintain its own index of documents, rather it sends a query to multiple back-end search engines and then merges their results. There are two obvious challenges that arise in meta-searching: first is the variation in the search engine ranking mechanisms, search engines differ significantly in their ranking schemes. Second issue is to do with the fact that the underlying search engines may return considerably varied set of search results in response to the same search query. How do we predict which of the returned results are relevant or not, and how do we merge these results to obtain unbiased ordering of the search results? This is a problem also termed the Fusion problem in [Callan et al, 1995].

In [Callan et al, 1995], Callan compares four typical rank fusion methods, namely: interleaving, raw score, normalized score and weighted score. He suggests the following: That if only the document rankings are available, the results from each collection can be interleaved (interleaving fusion). If the relevant scores of documents are available and the scores from different collections are comparable, multiple results can be merged based on documents' scores directly (raw score fusion). If the scores are incomparable, one can normalize them by standardizing statistics such as IDF (inverse document frequency) for the set of collections being searched in some cases (normalized). The other alternative to both simple interleaving and normalized score is merging based on weighted scores. In this case documents are ranked against the product of documents' scores and weights of collections (weighted score fusion).

In the case of the of the implemented Meta-search tool, the search results from the multiple search engines were treated as a single collection. The approach taken in the implementation is to apply a uniform scoring measure to all downloaded documents, which is based on degree of relevance to the search query. The tool downloads and analyzes the actual pages returned by the search engines, assigning each a relevance score based on weights of search terms used in the query. The relevance scores are used as the basis for establishing an ordering scheme for the retrieved search results, where those results appearing at the top of this ordering are considered more likely to be relevant to the user query.

The following subsections cover the approach used in developing relevance checking mechanisms and ranking criterion.

4.3.1 The Vector Space Model

The framework for document representation and relevance ranking used in the Meta-search tool is based on the standard information retrieval model, the Vector Space Model.

Vector Space Model consider that each document in a collection can be described by a set of representative keywords extracted from the document, called index terms T_j. Classic Vector Space Model is composed of a t-dimensional Vectorial space and standard linear algebra operations on the vectors [Baeza-Yates & Ribeiro-Neto, 99]. Each document D_i in a collection is represented a by a t-dimensional vector:

$$D_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,t})$$

Where $w_{i,j}$ represents the weight of the j^{th} index term. A term weight represents the relative importance of the index term in the document. The vectors can be visualized as documents representations on a V-dimensional Euclidean space, where V is the vocabulary size (number of terms used to represent the documents). Fig. 4.2 below, for instance, illustrates a three-dimensional document space (where three index terms, T_1, T_2 and T_3, are used)

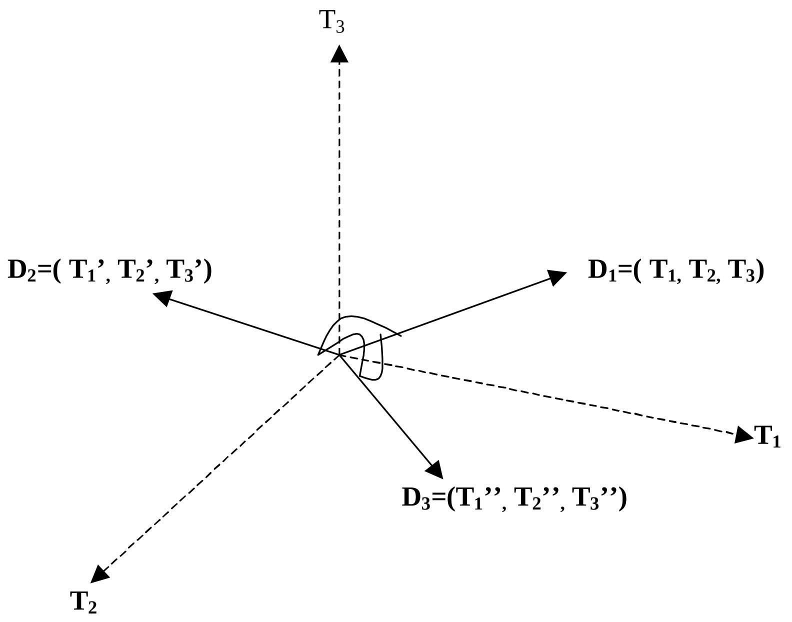


Fig 4.2: Illustration of a Vector Representation of a Document space

Given the index term vectors, d_i and d_j, of two documents D_i and D_j, Salton et al [Salton, et.al], the inventors of the model, propose that it is possible to compute similarity coefficient between the two documents, *sim*(D_i, D_j), which reflects the degree of similarity in the corresponding terms and term weights. Examples of such a similarity measure is the Cosine of the angle between these two vectors, given by:

$$sim(D_i, D_j) = \frac{\overrightarrow{d_i} \bullet \overrightarrow{d_j}}{|\overrightarrow{d_i}| \times |\overrightarrow{d_j}|}$$
$$= \frac{\sum_{k=1}^{t} w_{k,i} \times w_{k,j}}{\sqrt{\sum_{k=1}^{t} w_{k,i}^2} \times \sqrt{\sum_{k=1}^{t} w_{k,j}^2}}$$

Eqn. 4.1: Cosine Measure Equation

To apply the vector model to document retrieval, the documents and the user-queries can both be represented as vectors, accomplished by assigning weights to the index terms in the query and in the documents in the collection. These term weights can then be used to compute the degree of similarity between each document stored in the collection and the user query then ranking them by this measure. Ranking based on the degree of similarity to the user query allows the vector model to take into consideration documents that match the query terms only partially [Baeza-Yates & Ribeiro-Neto, 99]. How the index term weights are obtained is elaborated in section 4.3.2.

4.3.2 The TF-IDF Method

A common method for obtaining the weights of a term in a document is the so-called TF-IDF method. In this method, the weight of a term is determined by two factors:

- The Term Frequency (TF), which is quantified by the measure of the raw frequency of a term inside a document. This is a measure of how well a term describes the document's contents [Baeza-Yates & Ribeiro-Neto, 99].
- Inverse document Frequency (IDF). This is a measure of dissimilarity among the documents in the collection, quantified by measuring the inverse of the frequency of a term among the documents in a collection.

Precisely, using TF-IDF method [Lee et al, 1996], the weight of a term, j, in a document, i, is given by the following equation:

$$w_{i,j} = tf_{i,j} \times idf_{i,j} = tf_{i,j} \times \log \frac{N}{n_j}$$

Eqn. 4.2: General TF-IDF Equation

Where tf_{ij} is the Term Frequency of the term, j, in document i; N is the total number of documents in the document collection and n_j is the number of documents in the collection that contain term j.

The Inverse Document Frequency (IDF) is a factor that enhances the terms that appear in fewer documents, while downgrading the terms that appear in many documents. Terms that appear in many documents in the collection are not very useful for distinguishing a relevant document from a non-relevant one. Thus, the idea is to diminish the importance of such terms and highlight the less common terms in the document. To illustrate, suppose the document collections has 100 documents (so we have N=100). Then the IDF values for different examples of n_j would be:

n_j	5	30	90
IDF	1.3010	0.5229	0.0458

Notice that a less frequently used term in the document collection, in our case one that appears in only 5 documents, is given higher weight than a term that is used frequently, for instance one that appears in 90 documents.

Several variations of the above equation exist [Baeza-Yates & Ribeiro-Neto, 99]. One useful modification is the use of normalized frequency for TF, instead of simply using the raw frequency, given by:

$$tf_{i,j} = \frac{freq_{i,j}}{freq_{\max,j}}$$

Eqn. 4.3: Normalized Frequency

Where $freq_{i,j}$ is the raw frequency of term j in document i and $freq_{max,j}$ is the maximum frequency of term j in the document collection.

A variation of Eqn. 4.2 suggested by Salton and Buckley [Baeza-Yates & Ribeiro-Neto, 99], is

$$w_{i,j} = \left(0.5 + \frac{0.5\,freq_{i,j}}{freq_{\max,j}}\right) \times \log \frac{N}{n_j}$$

Eqn. 4.4 TD-IDF Formula

4.3.3 Extracting Document Representations

Unlike the conventional Information retrieval systems, where the documents in the collection are stored in a static collection while queries are submitted to the system, the developed Meta-search tool operates by first downloading before comparing the downloaded documents to the query in context in order to determine the relevance of the documents. Since all we are concerned with is to determine the relevance of a Web page to the user query, it is not necessary to store the entire Web page, instead the most crucial thing is to gather only the essential statistics out of the page that will be useful in scoring its relevance.

In simple terms this is how the tool works: First, the submitted user query is split into distinct words, called *index terms,* which are then recorded in a shared structure called *term dictionary,* where all parallel processes can have access to a single copy. Then, Referring back to Fig 4.1, as the *parallel page retrieval engine* progressively returns each downloaded page, the *document representation computation* module performs a text analysis of the page, generating a *term vector* containing the raw occurrence frequency of the respective terms in the document, which is then stored as the Document's representation. Later, these frequencies will be used to calculate the normalized frequency (TF) based on equation 4.3.

In formal terminology, suppose $K= \{k_1, k_2, \ldots, k_t\}$ is a set of t index terms that were obtained from the search query and stored in the *term dictionary*. Then, with every downloaded and successfully parsed document D_j, there will be an associated index term vector d_j, represented by

$$d_j = (f_{1,j}, f_{2,j}, \ldots f_{t,,j})$$

Where $f_{i,j}$ is the raw occurrence frequency of term k_i in document D_j.

When counting the term occurrence in a document, the tool uses the following criteria:

1. Any term enclosed within the Meta-Tags of the HTML documents is counted only once. It ‘s well known that web developers tend to use the Meta-Tags section for keyword spamming[6].
2. All HTML tags or text included as part of attributes to the tags, such as a URL within hyperlink anchor tags, or image file names within image Tags, are skipped.
3. All embedded Scripts, JavaScript, ASP, PerlScript, VBScript, PHP or Applets are excluded.
4. All terms, occurring within other text of the HTML file, are counted. Case insensitive sub-string pattern matching is used to locate the terms.

The following data structures are used in document representation: a *Term Dictionary HashMap*, *Term-Vector Array, Term-Vectors HashMap,* and *Document Description Hash Map*.

The *Term Dictionary HashMap* maps each index term, T_i, obtained from the search query to a unique term-ID. The HashMap uses the terms as Keys to the Hash Map. Each data entry for the HashMap is an array of three integers. The first integer represents the number, n_i, of downloaded and analyzed documents which contained the term, the second is the maximum frequency occurrence, $Freq_{max,i}$, of the term in the document collection and the third integer is the unique ID for the term (the Term-ID).

The *Term-Vector array* implements the vector, d_j, of the raw occurrence frequencies of index terms. It is generated for every document, D_j, analyzed. The size of the array is equivalent to the total number of index terms stored in the *Term Dictionary HashMap*. The array uses the unique term IDs, associated with each of the terms in the *Term Dictionary HashMap*, to mark the storage index of the computed raw occurrence frequency for the terms in the array. Fig 4.3 below illustrates the relationship between

[6] Excessive, repeated use of keywords or hidden text purposely inserted into a Web page to promote retrieval

the two data structures, i.e. the *Term Dictionary HashMap* and Document *Term-Vector array(s).*

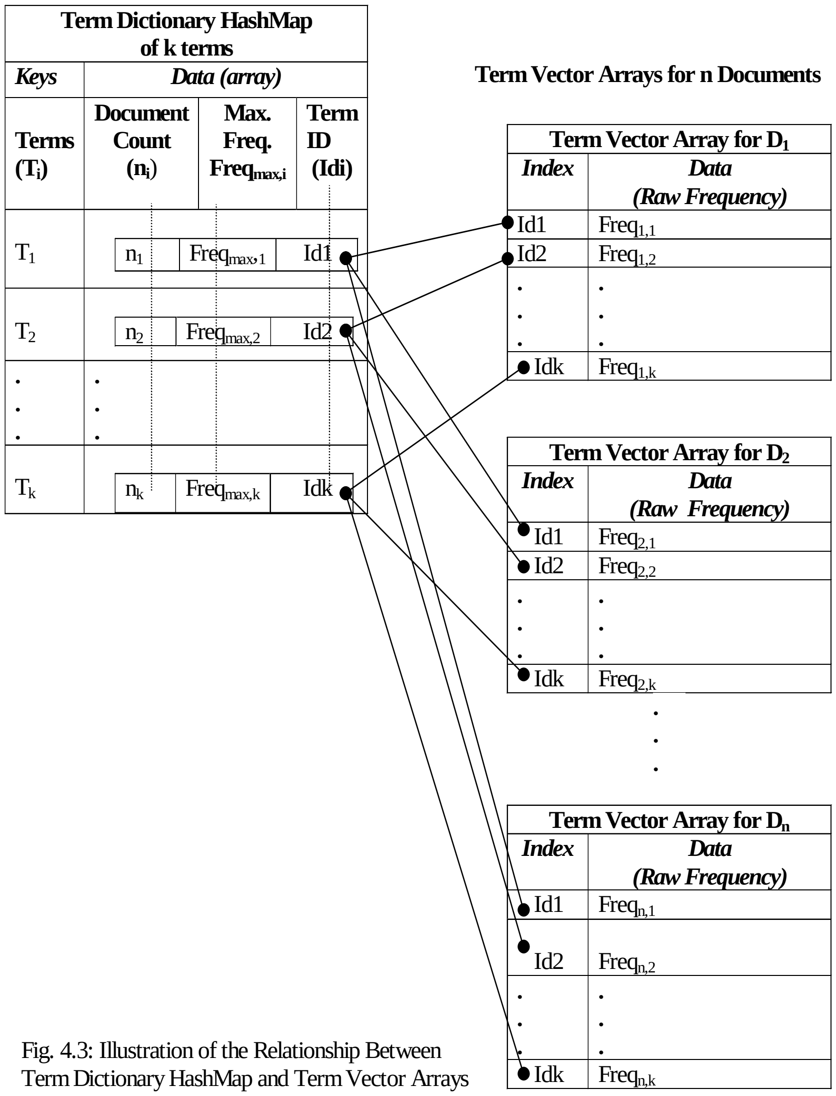

Term Dictionary HashMap of k terms
Keys
Data (array)
Terms (T_i)
Document Count (n_i)
Max. Freq. $Freq_{max,i}$
Term ID (Idi)
T_1 n_1 $Freq_{max,1}$ Id1
T_2 n_2 $Freq_{max,2}$ Id2
T_k n_k $Freq_{max,k}$ Idk
Term Vector Arrays for n Documents
Term Vector Array for D_1
Index
Data (Raw Frequency)
Id1 $Freq_{1,1}$
Id2 $Freq_{1,2}$
Idk $Freq_{1,k}$
Term Vector Array for D_2
Id1 $Freq_{2,1}$
Id2 $Freq_{2,2}$
Idk $Freq_{2,k}$
Term Vector Array for D_n
Id1 $Freq_{n,1}$
Id2 $Freq_{n,2}$
Idk $Freq_{n,k}$

Fig. 4.3: Illustration of the Relationship Between Term Dictionary HashMap and Term Vector Arrays

Term-Vectors HashMap on the other hand is an accumulator for all the *term-vector arrays* of analyzed documents. The URL of the document is used as the Key to the HashMap.

The *Document Description HashMap* is used to hold arrays that contain the individual descriptions of the analyzed documents as extracted from the returned search engine results. Each of the arrays has the following data about a search result: the URL, the anchor text displayed by the search engine when returning the result, the snippet describing the result, the name of the search engine(s) that returned the result and the rank of the result on the search engine(s). In addition, the array also reserves storage for the document's overall score, which is initially set to zero, but will eventually be set to the actual document score after document score evaluation. The URL of the document is used as the key to the HashMap.

The *Term-Vectors HashMap* and *Document Description Hash Map* maintain a one-to-one mapping by using similar Keys. This is particularly essential in order to identify the particular entry to be updated in the *Document Description Hash Map* during document score computation (see 4.3.4).

4.3.4 Score Computation, Fusion and Ranking

The Meta-search Tool employs an algorithm based on TF-IDF and Cosine method to assign relevance scores to documents.

The relevance score indicates the level of similarity between a user presented query and the documents that were retrieved and analyzed by the Meta-search tool as a result of the query, represented by their document vectors.

To start with, some background is necessary. The full Vector Space Model, given for computing relevance score, $R_{i,q}$, of a document D_i with respect to a query q is given by:

$$R_{i,q} = \frac{\sum_{term_j \in q} \left(0.5 + 0.5 \frac{freq_{i,j}}{freq_{\max,j}}\right) IDF_j}{\sqrt{\sum_{term_j \in D_i} \left(0.5 + 0.5 \frac{freq_{i,j}}{freq_{\max,j}}\right)^2 IDF_j^{\,2}}}$$

Where: $IDF_j = \log \frac{N}{n_j}$

Notice from Eqn. 4.3 that $\frac{freq_{i,j}}{freq_{\max,j}}$ can be represented simply as

$tf_{i,j}$

Eqn. 4.5: Vector Space Model For Relevance Score

Eqn. 4.5 is actually the same equation we have in Eqn. 4.1, but, in this case, the weights for the query vector have been omitted in the equation. The reason is simple. Since the same query is compared with all the documents, the query vector introduces the same factor to all the calculations for scores on all documents involved and, thus, does not really affect ranking of the documents.

The denominator provides a normalization factor in the space of the documents. The purpose of normalization is so as to shed the effect of documents lengths on the document scores, enabling short documents to have equal chances to be ranked high as longer documents.

Calculation of the Normalization factor is, however, very expensive [Lee et al, 1996]. As may be noticed in the equation, it requires access to every term in the document (D_i), not just the terms specified in the query. In their paper, Lee et al, propose that to simplify the computation, the normalization factor can be dropped. So we now have:

$$R_{i,q} = \sum_{term_j \in q} \left(0.5 + 0.5 \frac{freq_{i,j}}{freq_{\max_i}} \right) IDF_j$$

Eqn. 4.6: Simplified Vector Model Equation

In their paper, Yumomo and Lee [Yumomo & Lee], discuss that, in fact, empirical results show that this simplified method works better in the WWW environment than the full vector model with Normalization. They attribute this to an observation by Salton and Buckley [Salton & Buckley, 1987] that Normalization typically does not work well for short documents – documents of few sentences. Most pages on the Web are typically short; some simply represent a topic by a having a hypertext anchor on a page.

Eqn 4.6 was, therefore, adopted for the relevance score calculation in the Meta-search tool. Some slight variation in the calculation of IDF was introduced however. The equation for IDF was modified to:

$$IDF = 1 + \log \frac{N}{n_i}$$

Eqn. 4.7: Modified IDF equation

The addition of the constant value, 1, does not, in general, introduce a difference with ranking of the documents (ordering of documents) based on relevance score computation in Eqn 4.6. The IDF in 4.7 still diminishes the importance of terms that appear in many documents. The only difference comes when a user presents a query that has only one term, k_i, and the term happens to appears in all the documents. In such a case IDF calculation in Eqn 4.6 returns a zero, forcing all the documents in the collections to be assigned a relevance score of zero, though the documents may have varied TF[7] values. Deciding which documents should be ranked first, at that point, becomes a problem.

[7] Term Frequency – see Eqn. 4.3 and Eqn. 4.5

However, for ranking purpose, it is reasonable that in such a case we should simply rank the documents by their TF values. This is what the relevance score calculation based on Eqn 4.7 for IDF allows us to achieve.

The Ranking and Fusion component of the Meta-search tool (see Fig. 4.1) performs the document score computation, ranking and fusion, as facilitated by the following outline of the algorithm:

```
Obtain size, N, of Term-Vectors HashMap - the total number of represented documents
For every document, $D_i$, represented by entry in Term-Vectors Hashmap Do
    Initialize Score($D_i$)
    Retrieve the Term Vector Array, d, for $D_i$
    For every query Term, T, in Term Dictionary HashMap Do
        Obtain number of documents, n, that contained the term from Term Dictionary HashMap
        Calculate $IDF = 1 + Log (N/n)$
        Obtain max. occurrence frequency, $Freq_{Max, T}$, of the term from Term Dictionary HashMap
        Obtain raw occurrence frequency, $Freq_{i,T}$, for the term from d
        Calculate normalized frequency $TF_{i,T} = Freq_{i,T} / Freq_{Max,T}$
        $Score(D_i) = Score(D_i) +( (0.5 + 0.5\ TF_{i,T}) \times IDF)$
    End
    Update the score for $D_i$ in Document Description HashMap with Score($D_i$)
End
Rank (Sort) the entries in Document Description HashMap by the scores
Generate a HTML formatted string, fusing all entries in Document Description HashMap
Return the HTML string for display
```

4.4 The Graphical User Interface

The role of the user interface is to provide abstraction mechanisms for all actions that have to be initiated by the user in order, for example, to submit queries and browse through the search results.

The Meta-search tool was written as a Java application; hence its user interface was implemented using the Java Object-Oriented GUI framework. The user interface allows the user to type in a query, select search engines to query, browse through search results, follow links, and, if desired, open a page or a link in a default Web Browser Window such as Netscape or MSIE, among other features. Figure 4.4 below shows a screen shot of the user interface.

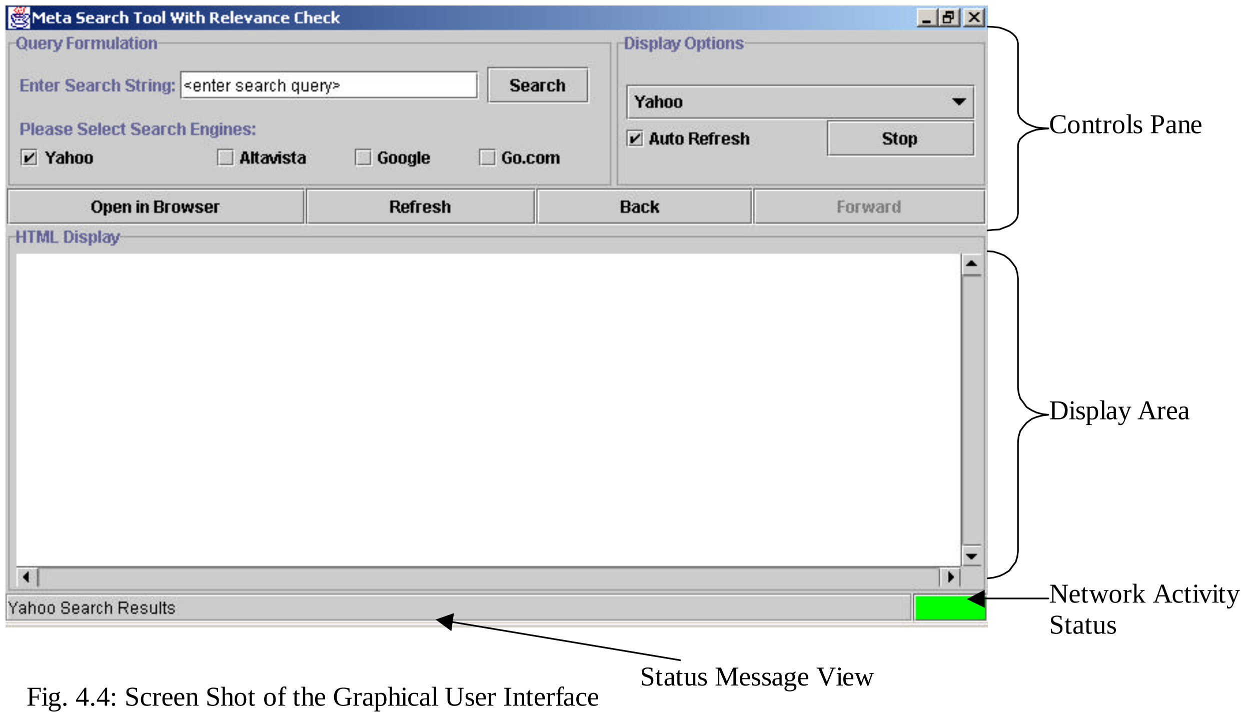


Fig. 4.4: Screen Shot of the Graphical User Interface

4.4.1 Query Submission

At the moment the tool can search on four search engines, namely Yahoo, Altavista, Google and Go.com[8]. The tool can, however, be expanded in future to include more search engines easily. Specifically, the only significant effort will be in developing and integrating appropriate Linkextractors (see 4.6) for the new search engines.

[8] Please Note: Go.com (www.go.com) bought and for sometime used "Infoseek" as technology powering the Go Network's search functionality. Later they abandoned it in favor of Overture. As of the writing of this thesis Go.com's searches were powered by Overture.

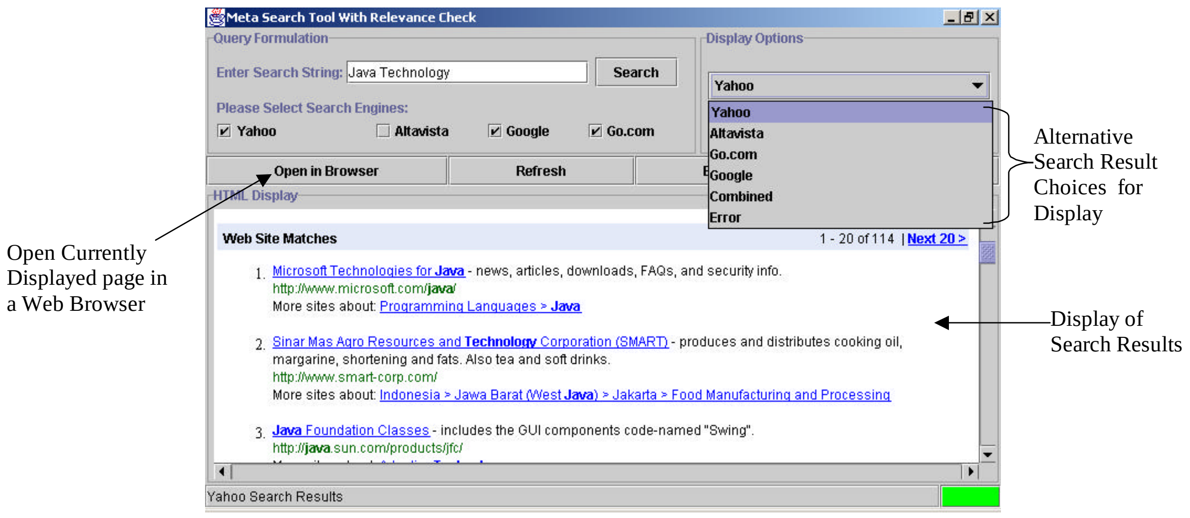
To a submit a query to one or more search engines, all a user needs to do is to select the search engine(s) on the control pane by checking on the Checkboxes against the search engine names. The next thing is to type the query on the slot labeled “Enter Search String” and click on the ‘Search Button’, or simply hit the ‘Return’ or ‘Enter’ key, to submit the query. On submission of the search query, the Meta-search tool dynamically translates the query into a format that matches the query syntax normally used by each of the selected search engines before, simultaneously, dispatching the search request to the engines.

The tool can accept single or multi-keyword search string (using space characters as delimiters). As yet, it does not support Boolean syntax.

4.4.2 Displaying and Browsing Search Results

The Meta-search tool allows a user to display and browse through Web pages that were returned by the individual search engines, the combined and ranked search results or those search results that were found with errors (i.e. hits which experienced errors during page retrieval for document representation calculation). Any of the alternatives can be selected from a drop down list implemented on the GUI, see fig 4.5.

If a user chooses to display ‘Combined’ or ‘Error’ results while the Meta-search tool is still in the course of processing the search results, then, if the ‘Auto Refresh’ Check-Box on the GUI is set on (Checked), the tool will keep updating the display area automatically with the combined and re-ranked results (or the Error Results, depending on which selection is active) as it progressively downloads and analyzes each page. Thus, a user does not have to wait until after all the pages have been downloaded and analyzed in order to view the ‘Combined’ (or ‘Error’) results. A user is free to set ‘on’ or ‘off’ the ‘Auto Refresh’ feature at any moment. Setting it ‘off’ simply stops the automatic update of the display for the ‘Combined’ or ‘Error’ results while the parallel page retrieval engine (see 4.5) is active, it does not stop or suspend the operations of the parallel page retrieval engine. As soon as the parallel page retrieval engine is through, the tool will display all ‘Combined’ or ‘Error’ results (whichever is selected).


Meta Search Tool With Relevance Check
Query Formulation
Search
Please Select Search Engines:
Display Options
Yahoo
Altavista
Go.com
Google
Combined
Error
Open in Browser
Refresh
HTML Display
Open Currently Displayed page in a Web Browser
Alternative Search Result Choices for Display
Display of Search Results

Enter Search String: Java Technology

[✓] Yahoo [ ] Altavista [✓] Google [✓] Go.com

Display Options: Yahoo

Web Site Matches 1 - 20 of 114 | Next 20 >

1. Microsoft Technologies for **Java** - news, articles, downloads, FAQs, and security info.
   http://www.microsoft.com/**java**/
   More sites about: Programming Languages > **Java**

2. Sinar Mas Agro Resources and **Technology** Corporation (SMART) - produces and distributes cooking oil, margarine, shortening and fats. Also tea and soft drinks.
   http://www.smart-corp.com/
   More sites about: Indonesia > Jawa Barat (West **Java**) > Jakarta > Food Manufacturing and Processing

3. **Java** Foundation Classes - includes the GUI components code-named "Swing".
   http://**java**.sun.com/products/jfc/

Yahoo Search Results

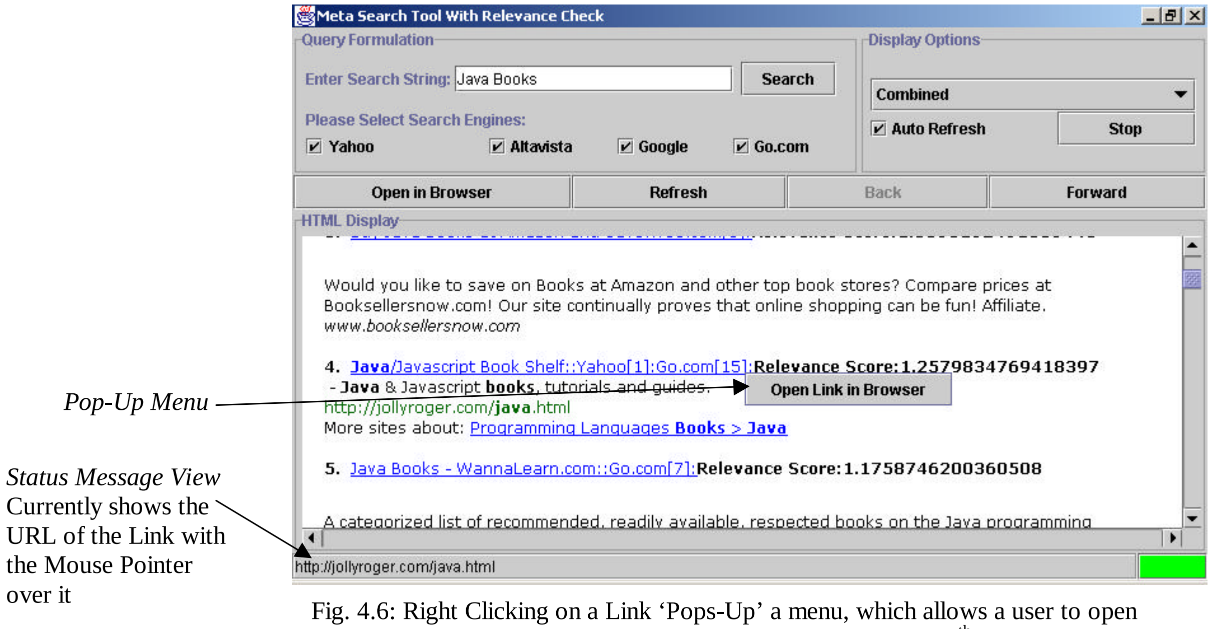
Fig. 4.5: GUI Display of Search Results

The combined and ranked results are listed, on the display area, ordered descending by the values of their relevance scores. Each result item consists of the rank, the title (anchor to the link), the search engine(s) that returned the result (see illustration in Fig. 4.6), the rank of the result on the search engine(s), the relevance score of the page it links to, and a short description of the result.

A user can follow any link by clicking on it. If the link is broken, or the page cannot be retrieved for some reason, the tool launches an error alert message to the user. Whenever the Mouse Pointer moves over any hyperlink found on the display area, the tool automatically displays its underlying URL on the Status Message View (see fig. 4.6).

The display area is implemented using a Java JEditorPane Swing Component. It is a complete HTML 3.2 renderer that can handle most HTML tags including Frames, Forms, and hyperlinks. It also supports plain text and RTF. However, unlike most conventional Browsers, JEditorPane does not, as yet, support a number of other Web based technologies such as JavaScript, Flash, XML and QuickTime. It cannot also launch application programs to handle non-HTML documents such as PDF or PostScript files. For this reason an option, on the GUI, was included to enable users to launch a displayed page, or follow any link, using a Web Browser.

A page, which is being displayed on the display area of the GUI, can be opened in a Web Browser by simply clicking a button labeled 'Open in Browser' on the GUI. To open some particular link in a Web Browser, a user need first Right-Click on the link, which fires a Pop-up Menu, then choose 'Open Link in Browser' from the menu (see fig. 4.6). Either of the actions causes the Tool to instruct the Windows Operating System to launch the default Web Browser application, such as Netscape or Internet Explorer, to fetch and display referenced page.



Enter Search String: Java Books

Please Select Search Engines: ☑ Yahoo ☑ Altavista ☑ Google ☑ Go.com

☑ Auto Refresh

Would you like to save on Books at Amazon and other top book stores? Compare prices at Booksellersnow.com! Our site continually proves that online shopping can be fun! Affiliate.
www.booksellersnow.com

4. Java/Javascript Book Shelf::Yahoo[1]:Go.com[15]:Relevance Score:1.2579834769418397
- Java & Javascript books, tutorials and guides.
http://jollyroger.com/java.html
More sites about: Programming Languages Books > Java

5. Java Books - WannaLearn.com::Go.com[7]:Relevance Score:1.1758746200360508

A categorized list of recommended, readily available, respected books on the Java programming

http://jollyroger.com/java.html

Fig. 4.6: Right Clicking on a Link 'Pops-Up' a menu, which allows a user to open the Link in a Web Browser. Notice the specified link is ranked 4th and was returned by Yahoo and Go.com. In Yahoo it is ranked 1st, whereas in Go.com 15th.

4.5 The Parallel Page Retrieval Engine

The *Parallel Page Retrieval Engine* is a key component of the Meta-search tool; its main purpose is to autonomously retrieve all the pages referenced by the returned search results for document representation computations.

It is implemented as a simple Web robot, but it does incorporate features such as multithreaded page retrieval, load balancing, and delaying of page requests to avoid overloading the host machine and remote websites.

As noticed in Fig 4.1, the *Parallel Page Retrieval Engine* obtains the links to be retrieved from the *Common Collection of Links* generated by *LinkExtractors*. This collection is first randomized to ensure that the links returned by same search engine do not follow each other consecutively. This is to avoid a potentially biased situation where all links from a particular search engine are processed first before any link from the other engines can be processed. The *Parallel Page Retrieval Engine* treats this collection as a queue.

The retrieval engine starts with a single robot process, which iteratively spawns downloading threads for every link found in the *Common Collection of Links* (see illustration in Fig. 4.7). In each such iteration, a new downloading thread is launched, which in turn takes a link from the collection, downloads the page to which it points to, then pass the page to the *Document Representation Component* for processing. If the page download action times out then the retrieval engine simply adds the link, and its description, to the collection of *Error results*. This process continues until the queue (*Common Collection of Links*) is empty.

The iterative algorithm can, however, lead to an exponential explosion in the number of threads, a situation that could easily lead to program crash (or host overloading). To prevent this, a mechanism to enforce the maximum number of threads that can be allowed to run at any particular instant was implemented.

The mechanism is based on a ticketing model proposed by Vanhelsuwe [Vanhelsuwe]. In the system, a pool of tracked “tickets” is maintained. Before spawning a new thread, a ticket must be obtained first, which gives the thread a running permission. Once a thread is through with its work, it must release its ticket back to the ticket pool before terminating, which in turn is recycled for use in future threads. If all tickets are in use, then the spawning of new threads is forced to block, until one of the running threads releases its ticket. The maximum number of threads that can run is, thus, limited to the maximum number of tickets that can be issued.

The initial pre-processing done by *LinkExtractors* ensures that the *Common Collection of Links* is free of duplicate entries. Thus, the *Parallel Page Retrieval Engine* can retrieve any referenced page only once. However, it is possible that a number of links may be pointing to different pages, but in actual sense the pages are hosted on the same Web server. For this reason, a brief random delay period was introduced to randomize scheduling of threads waiting to obtain tickets to prevent rapid successive requests for Web pages, a condition that can easily lead to overloading of target Web server.

The Engine simply downloads pages; it does not attempt to follow links contained in the pages. Therefore the typical risk, known with Web robots, of getting stuck in endless loops, or revisiting same pages over and over again, does not arise.

The tool allows a user to stop the *Parallel Page Retrieval* Engine from spawning new threads during a search process. Clicking the 'Stop' button on the GUI does this. It must be noted, however, that 'Stopping' does not 'interrupt' or 'kill' already running threads; this was on purpose in order to avoid arbitrary behavior that could potentially leave the data structures used by the system in an inconsistent state. Already running threads are left to run until they eventually die (terminate) on their own. Clicking the 'Stop' button effectively toggles off (halts) the spawning of new downloading threads for the current search. When a new search in initiated, for instance by clicking on the search button, the start of new download threads for the new search is, by default, toggled back on again.

One other notable feature on the GUI is the visual aid for the user concerning the activity of the *Parallel Page Retrieval engine*. As the Downloading Threads progressively retrieve pages the Status Message View on the GUI is set with cue messages indicating what is going on, such as "Page download in progress", "Relevancy checking going on", "Relevancy checking completed" etc. For similar purpose the Network Activity Status (see Fig. 4.4) is set Red whenever a page download is initiated and back to Green when the download is complete.

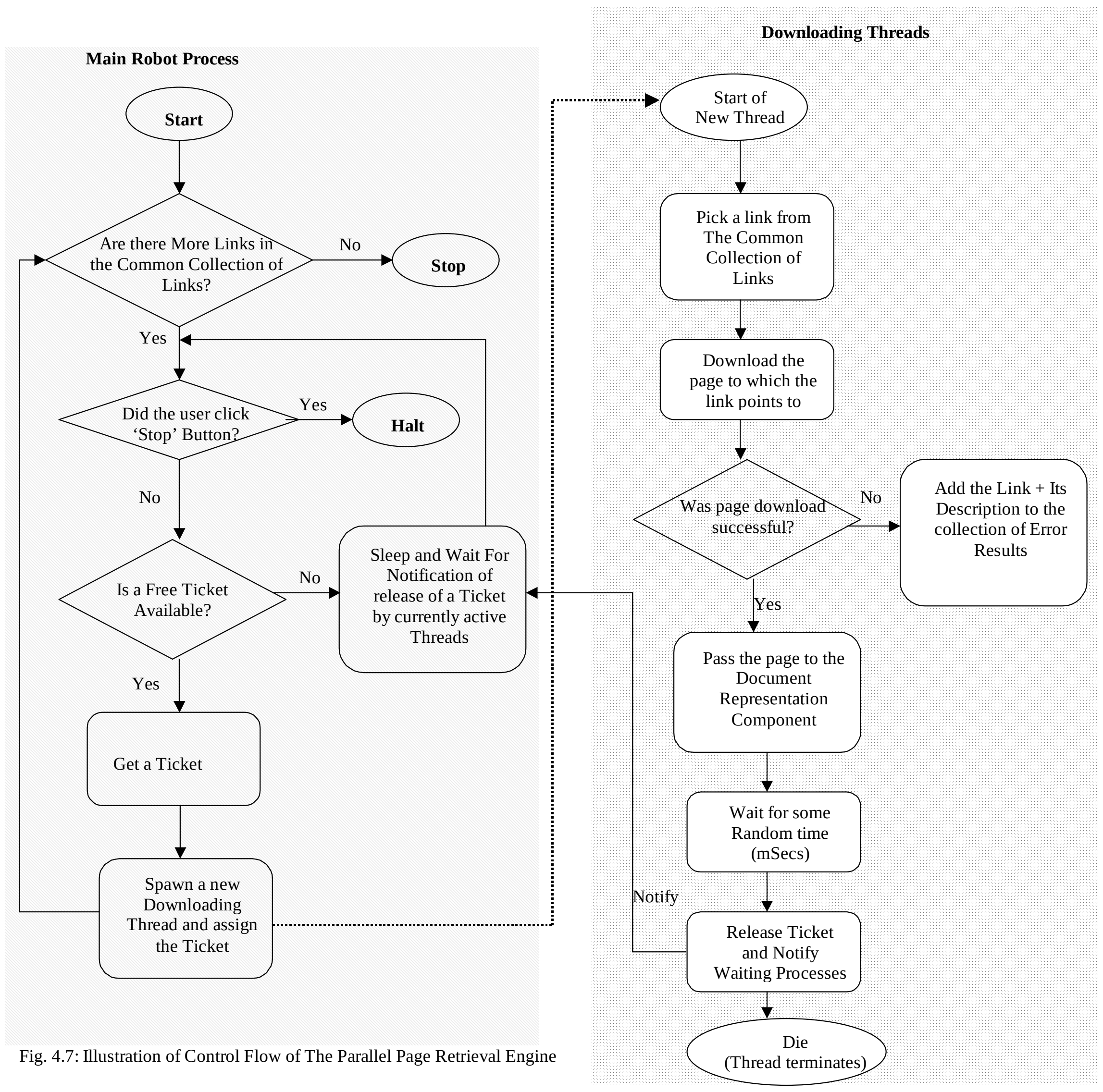

Main Robot Process
Start
Are there More Links in the Common Collection of Links?
No
Stop
Yes
Did the user click 'Stop' Button?
Yes
Halt
No
Is a Free Ticket Available?
No
Sleep and Wait For Notification of release of a Ticket by currently active Threads
Yes
Get a Ticket
Spawn a new Downloading Thread and assign the Ticket
Downloading Threads
Start of New Thread
Pick a link from The Common Collection of Links
Download the page to which the link points to
Was page download successful?
No
Add the Link + Its Description to the collection of Error Results
Yes
Pass the page to the Document Representation Component
Wait for some Random time (mSecs)
Notify
Release Ticket and Notify Waiting Processes
Die (Thread terminates)

Fig. 4.7: Illustration of Control Flow of The Parallel Page Retrieval Engine

4.6 The LinkExtractors

LinkExtractors encapsulate Web Search engine specific rules for interpreting returned search results pages. The Meta-search tool contains a *LinkExtractor* component for each of the search engines it employs, at present for Yahoo, Altavista, Google and Go.com. The rules help in deciding which parts of the page to process (body of results) and which to ignore. They also define exactly how to extract specific data of interest. The Meta-search Tool uses the *LinkExtractors* to extract links, titles and the snippets describing the respective search result hits, from the content of results pages. Basic HTML parsing and pattern matching techniques, supported by Java, are utilized to achieve this.

The complexity of a *Linkextractor* depends on the page design used by the search engine. It was noticed that specific rules engineered for one Web search engine could not be easily transferred to the other engines, because of a great deal of differences in the way the results from the search engines are formatted. Thus each *LinkExtractor*, to a large extend, required development from scratch. Moreover, if a search engine changes its page design, then its *Linkextractor* module must be modified to be consistent with the changes in page format.

With reference to Fig. 4.1, the *Linkextractors,* for the search engines selected during a query, are invoked as soon as the *Searcher* component is through with retrieving the search results from all the engines. The *Linkextractors* cooperate to build the virtual database of *Common Collection of Links*, ensuring that no entry in the database is a duplicate of another. Each entry consists of the extracted URL, the title (anchor text), the description used by the search engine for the result, the search engine that returned the result and the rank of the result on the search engine. If a *Linkextractor* encounters a URL that is already stored in the database, for example in the case where multiple search engines return a particular link, it simply modifies the relevant entry in the database to include the name of its search engine and the rank of the link according to the engine.

4.7 Comparison to Existing Meta-Search Engines

From the description of the tool, some key differences of the tool from the existing Meta-search engines may be appropriate at this juncture.

Like all the existing popular meta-search engines, such as MetaCrawler[5] and Mamma[7], the tool collates the returns of search results from the multiple search engines it queries, removes duplicates, ranks them and returns the results to the user. A significant difference however is the ability of the developed tool to download all individual pages referenced by the search results in order to determine the relevance of results in the context of the query terms used. The mentioned meta-search engines rely on the page summaries returned by the search engines they query. It is obvious that the relevance of a page to a query becomes certain only after actually downloading it and extracting the context of the query terms within the page. The feature at the same time ensures that the returned hits point to existing pages and, thus, are not obsolete results from a database that a particular back-end search engine failed to update.

Apart from allowing a user to view and browse through combined search results from multiple search engines it queried, the tool also provides interested users with the means to browse through the search results of the individual engines involved in the query, unlike most meta-search engines which only lists the combined results. The tool also allows the user to view the list of all the hits found with errors, i.e. results found with broken links. Perhaps this feature could be most useful for those wishing to sample the freshness of data in the databases of the search engines they use.

It must be noted however that the intention for the development of the tool was not to discredit, or replace, existing regular search engines or meta-search engines, which actually provide valuable services and indeed attract a variety of users for diverse number of reasons, but rather was a research idea aimed at investigating, and demonstrating where possible, how Web searching can be improved.

5. Experiments With the Developed Meta-Search Tool

Some experiments were informally carried out to study the performance and relative effectiveness of the search engines used by the tool.

5.1 Relevance and Quality of Returned Results

To begin with, determining the relevance or quality of the search results is not easy. Relevance is a measure of pertinence to the initiated query; quality is a measure of value to the user. First difficulty arises from the unavailability of standard test data. Second problem is that quality or relevance of returned results is a highly subjective issue that depends on the user's information need and may vary from user to user. The Meta-search tool simply lists all search results in descending order of their scores, in hope that the most highly scored results (top ranked) would be the most relevant and of value to the user.

While we admit that more robust testing techniques are necessary in order to gain complete insights of search engines, which could also involve an understanding of the aspect of human role in formulating the queries and evaluating results, we believe it is also possible to gain some general insights into search engine performance, with respect to relevance measure used in the design of the developed Meta-search tool, by using some sample data.

A simple evaluation experiment was carried out. In the experiment a set of nineteen simple queries[9] were submitted to the search engines (Yahoo, Google, Altavista and Go.com) on the Meta-search tool. The table in Fig 5.1 shows the collected statistics. The collected statistics were then used to compare tendency of the search engines to return relevant results, the percentage of broken links per search engine and the degree of overlap in search results returned by the engines.

[9] Buenos Aires, Reykjavik, London, Warsaw, Moscow, Tokyo, Sydney, Mumbai, Washington, Nairobi, India, Africa, America, Europe, Australia, New Zealand, Asia, Russia, Scandinavia

Search Engine	Total No. of Hits	Broken Links	Ranked Results		Duplicate Links (Returned By Other Engines)			
			Upper Quartile	Lower Quartile	Yahoo	Google	Go.com	Altavista
Yahoo	380	7	78	112	0	68	10	31
Go.com	285	17	74	100	10	10	0	8
Google	190	5	61	34	68	8	10	43
Altavista	190	7	56	37	31	43	8	2

Fig. 5.1: Experimental Results

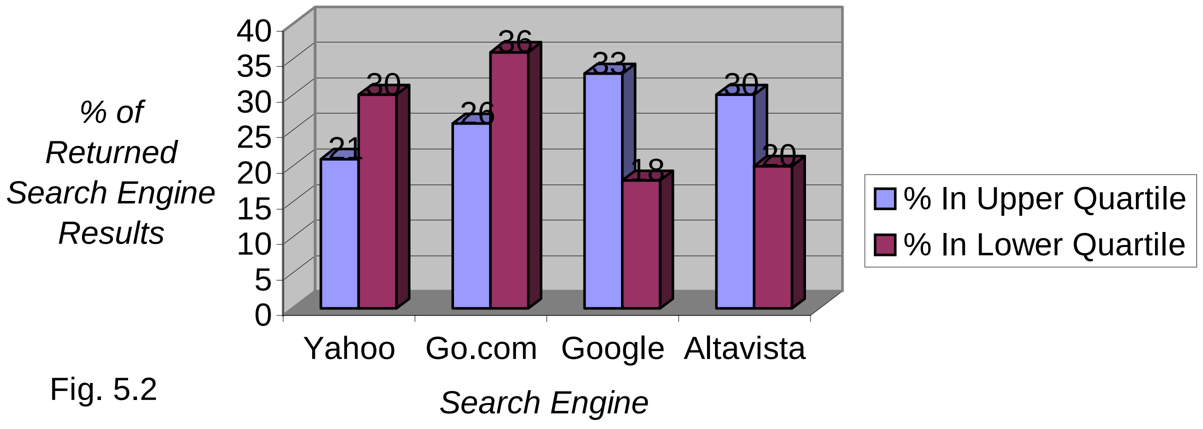


Fig. 5.2

Fig 5.2 graphically illustrates the observations made in Fig. 5.1. The bars represent the percentages of the individual search engine results that were found at the upper quartile and at the lower quartile sections in the overall order of combined and ranked search results. From the figure, it can be seen that the ranking algorithm used by the tool has significantly high tendency to rank a higher proportion of Google search engine results as more relevant in general (in upper quartile) than the results from Altavista, Yahoo and Go.com. The opposite case is seen with Go.com. Fairly high proportion of the search results returned by Go.com is ranked in the lower quartile. Yahoo on the other hand has the least difference between the percentages of relevant hits (those in upper quartile) and those deemed less relevant (those in lower quartile).

Not surprisingly, the relatively good performance for Google seem to concur with that has been observed in other studies. According to Searchenginewatch [Sulivan, 2002] of May 13, 2002, in which they cite a report where major search engines were ranked by cumulative time spend by users searching for information, they declare, “Google is leading by far over other search engines” – thus making it the most popular search engine. It's not hard to imagine why people would keep going back to a particular search engine, most certainly it is because they always get what they looking for (relevant results). From the same cited metrics, Yahoo is ranked second, Overture (the search engine used by Go.com) ninth. AltaVista is not mentioned though.

According to Google[Google], the page ranking technology they use, PageRank^{TM}, goes far beyond the number of times a term appears on a page. Apart from examining certain aspects in the occurrence of search terms on the page's content, they also heavily rely on the uniquely democratic nature of the Web [Google] where the count of links (they call them “Votes”) from other Websites pointing to the page are used as an indicator of the page's value. The system is said to have worked so well in producing highly relevant and high quality search results that Google has gained widespread praise.

Search results from Go.com are listings provided by Overture [8]. Overture use what they term Pay-For-Performance™ search. In this strategy, the rank positions of Websites that appear on search listings are greatly influenced by the type of online paid placement package (see [Overture]) that the Website owners are willing to invest in.

The other two search engines, Yahoo and AltaVista, do not give particulars of the page ranking and relevancy checking technology they use. An appropriate comparison cannot therefore be made.

5.2 Quality of Returned Hits

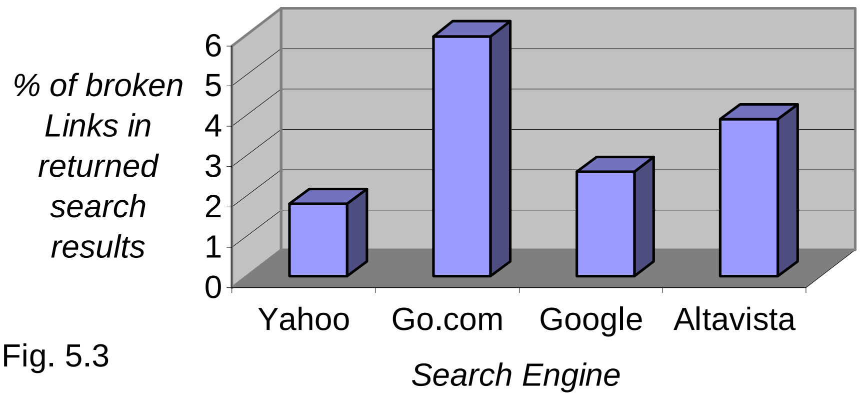


Fig. 5.3

The histogram in Fig 5.3 is based on the observations made in Fig. 5.1. It illustrates the percentage of broken links (bad hits) in the search results obtained from the four search engines. By comparison to the other search engines, a fairly large number of broken links appeared in the results returned by Go.com. Yahoo appears to have taken top quality honors.

A possible explanation may be that the engine Go.com uses refreshes data in its database less often than the other three search engines – Yahoo, Google and Altavista.

5.3 Overlap of Results

The chart in Fig. 5.4 below illustrates the percentage of the returned search results that were found to overlap (i.e. hits returned by at least two search engines) vis-à-vis those found to be unique (hits returned by single search engine), as observed from the experimental data in Fig. 5.1.

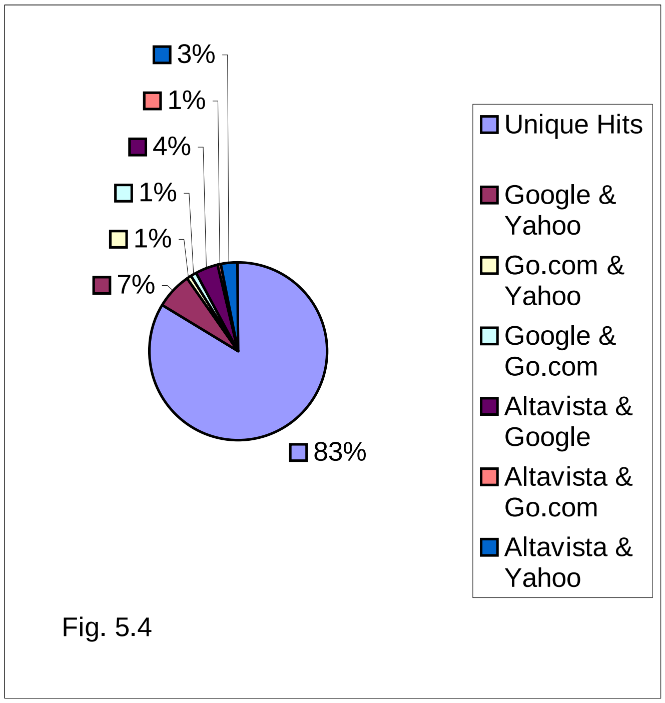


Fig. 5.4

The obtained search results show remarkably little overlap in the hits returned by the different search engines, less than 17% of the search results were found to overlap. This confirms an observation earlier made by Lawrence & Gilles [Lawrence & Gilles, 1999]. If this can be a simple pointer, then it certainly shows the level of improvement one can expect to get by employing several search engines rather than just one.

A notable observation from Fig. 5.4 is the significantly high overlap between search results of Google and those from other search engines, Yahoo and AltaVista, of approximately 7% and 4% respectively. One explanation to this could be a statement at SearchEnginewatch Website: “Yahoo supplements its results with those from Google. If a search fails to find a match within Yahoo's own listings, then matches from Google are displayed. Google matches also appear after all Yahoo matches have first been shown” [SearchEngineWatch, 2002]. The other reasonable explanation can be based on the figures of Search Engine Size Comparisons [SearchEngineWatch, 2001], which show that Google offers the largest collection of Web pages, hence largest coverage of the Web, than any other search engine currently. Thus, the likelihood of overlap in area of Web coverage between Google and other search engines is much higher.

Search results for Go.com on the other hand shows a consistently low overlap with those from other search engines, 1% in all cases. By comparison to the rest, this suggests that Go.com is actually more useful when meta-searching, since it has a high tendency to produce unique results that are not generated by the other search engines.

The low overlap with individual search engines represents an opportunity for meta-search engines, and definitely for the Meta-search tool developed for purpose of this thesis. The meta-search engines can recruit multiple search engines to carry out queries then return the best hits from the different search engines, thereby improving performance over what any individual search engine can achieve.

Generally, there are two possible reasons why the search engines tend to have relatively low overlap between their results. First, it could be because the different search engines employ different means of matching queries to relevant documents, thus returning different sets of search results in response to the same query. The other reason could be because the search engines have different indexing coverage of the World Wide Web, which yield highly disjointed results. The later reason is given much weight by [Lawrence & Gilles, 1999].

6. Conclusions and Directions for Future Work

6.1. Summary and Contribution

This thesis is founded on an attempt to explore alternative techniques to improve upon traditional approaches to Web searching. It was observed that relying on single standard engines has limitations. These limitations were pointed out; they include presence of obsolete data in their databases, limited coverage of the WWW and vulnerability to keyword spamming.

For the purpose of the thesis study, a Meta-search tool was developed. The tool includes, among other features, facilities to enable submission of queries to multiple search engines, combine the various search results, download and analyze each document referenced in search results, eliminate broken links (obsolete links) from the search results, and to show the results ranked by measure of relevance to the user query in context. All aimed at improving efficiency and accuracy of Web searching.

The dominant hypothesis was that by combining the results of various search engines it would be possible to obtain search results that represent a broader portion of the WWW, than if just one search engine were to be used.

It is by retrieving and separately analyzing all the pages referenced by search results that the tool is able to compute the genuine relevance of the results to a query. This also provides a means for non-existent pages (broken links) to be detected and discarded from presentation of results to the user.

The developed Meta-search tool employs Web automatons to perform parallel retrieval of pages for the analysis, in real time. Indeed, the tool demonstrates that on the spot automatic download and analysis of WWW pages, in order to determine match to a query, is possible and can be done within reasonable time.

Experiments with the tool (in chapter 5) confirm that the hypotheses that formed the basis for the thesis were correct and, also, that desired solutions were attained through the tool. These are as follows:

- ✓ That searching over multiple engines improves performance over what any individual search engine can achieve. The experiments with the tool show that 83% of combined hits returned by the engines were unique. The tool ranked best hits from the different search engines higher.

- ✓ That relatively high percentage of obsolete data exists in the databases of some search engines. The tool shows some engines have as high as 6%, as seen in 5.2. The tool was actually able to detect and eliminate these obsolete hits (broken links) from the results.

- ✓ That the relevance of a page to a query becomes clear only after downloading it and finding the query terms. The presence of high percentage of obsolete data in some engines, as seen in 5.2, is a simple attester to this claim. Undeniably, if some search engines refresh data in their database less often, then supposing that the referenced pages are highly ephemeral, it is possible that a number of the returned links could be pointing to pages that no longer have data that were initially used to index them. The developed tool was capable of downloading and determining the relevance of pages and re-ranking them accordingly. Ranking of results, as seen in 5.1, shows a pattern that seem to be in harmony with that has been observed in other studies: search engines that are known to be good (popular) perform well.

The tool comprises an intuitive user interface. Among other features, the interface can allow a user to view and browse through combined search results. Unlike most meta-search engines, it also provides a user with the means to browse through the search results of the individual search engines involved in a query.

The tool is simply an archetype at present though. It is however hoped that this would lead to further refinements and progress towards the ideal: that is, a search tool that returns the most relevant and accurate results from a broad Web search, and that is also fast, efficient and easy to use.

6.2 Directions for Future Work

The demonstrations made in the thesis may be further improved in a number of areas.

One such area is in the ranking algorithm used. Vector model is resilient and has been proven by consensus to be either superior or almost as good as other known alternatives [Baeza-Yates & Ribeiro-Neto, 99] as a ranking strategy with general collections. It is also simple and fast. This motivated the choice of the model in the implemented ranking algorithm. However, in a Web environment apart from keywords counts, which the Vector model uses, other factors may also be useful in determining importance of a page, such as the relative popularity of the page (as seen in the case of Google's PageRank^{TM} [Google]). Simple keyword count may also be enhanced by incorporating provisions for concept based matching, such as use of automatic thesauri. These could be considered for integration in future enhancement of the tool.

The tool can download and successfully compute ranks for HTML documents, and to some extend XML documents, at present. The Web however has other document formats such as PDF, or the formats associated with Multimedia files (Images, Sounds, Video), which require further investigations on how their information content can be handled.

The developed Meta-search tool relies on the back-end search engines for the principal and vital service of searching for information on the Web. The underlying content are owned by the respective engines and may be protected by copyright and other intellectual property or proprietary rights, which stipulate the terms of use. Deployment of such a tool therefore requires an amiable agreement with the search engines.

7. References

[1] http://www.google.com

[2] http://www.go.com

[3] http://www.altavista.com

[4] http://www.yahoo.com

[5] http://www.metacrawler.com

[6] http://www.dogpile.com

[7] http://www.mamma.com/

[8] http://www.overture.com

1. [Annunziato]. Dr. Jose Annunziato. *A Review of Robots, Spiders and Worms*. Available online: http://www.cs.uml.edu/~jannunzi/umass/96/spring/web/midterm/paper2/paper2.html
2. [Baeza-Yates & Ribeiro-Neto, 99]. Baeza-Yates R. and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-wesley, Reading, MA, 1999.
3. [Bailey, et al] Peter Bailey et al. *Dark Matter on the Web*. Australian National University, Canberra ACT, Australia. Available online: http://www9.org/final-posters/poster30.html
4. [Bergman, 2001]. Michael K. Bergman. *The Deep Web: Surfacing Hidden Value*. BrightPlanet, September 2001. Available online: http://www.brightplanet.com/deepcontent/tutorials/DeepWeb/deepwebwhitepaper.pdf
5. [Berners-Lee]. Tim Berners-Lee. *WorldWideWeb – Summary*. Available online: http://www.w3.org/Summary.html
6. [Bharat & Andrei, 1997]. Krishna Bharat and Andrei Broder. *A technique for measuring the relative size and overlap of public Web search engines*. DIGITAL, Systems Research Center, Palo Alto, U.S.A. Available online: http://www7.scu.edu.au/programme/fullpapers/1937/com1937.htm#Brake97

7. [Callan et al, 1995]. J. Callan, Z. Lu and W. B. Croft. *Searching Distributed Collections With Inference Networks*. Proceedings of 18th ICRD in IR, 1995.
8. [Chakrabarti]. Soumen Chakrabarti. *Focused Crawling: The quest for topic specific portals*. Available Online: http://www.cs.berkeley.edu/~soumen/focus/
9. [Christos & Oard]. Christos Faloutsos and Douglas Oard. *A Survey of Information Retrieval and Filtering Methods*. University of Maryland, Maryland, USA. Available online: http://www.enee.umd.edu/medlab/filter/papers/survey.ps
10. [Durfee, et. al, 1989]. Durfee E. et al. *Trends in Cooperative Distributed Problem Solving*. IEEE Transactions on Knowledge and Data Engineering, 1989.
11. [Eagan & Laura, 1996]. Ann Eagan and Laura Bender. *Spiders and Worms and Crawlers, Oh My. Untangling the Web*. 1996 University of Arizona, Tucson, April 26. Available online: http://www.library.ucsb.edu/untangle/eagan.html.
12. [Eichmann]. David Eichmann. *Ethical Web Agents*. University of Houston. Available Online: http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Agents/eichmann.ethical/eichmann.html
13. [Etzioni & Weld, 1995]. Oren Etzioni & Daniel S. Weld. *Intelligent Agents On The Internet: Fact, Fiction, And Forecast.* University of Washington Seattle, 1995. Available online: http://www.cs.washington.edu/homes/weld/papers/ieee-expert.pdf
14. [Genesereth & Rosenschein, 1994]. Rosenschein, J. S. and Genesereth, M. R. *Deals among rational agents*. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85), page 91-99, Los Angeles, United States, 1994.
15. [Google]. *Google Technology, Page Rank Explained.* Available at: http://www.google.com/technology/index.html
16. [Green et al, 1997]. Shaw Green et al. *Software Agents: A review*. Trinity College, Dublin, 1997. Available online: http://citeseer.nj.nec.com/cache/papers/cs/368/http:zSzzSzwww.cs.tcd.iezSzresearch_groupszSzaigzSziagzSzpubreview.pdf/green97software.pdf

17. [GVU, 1999]. *GVU's 8th WWW User Survey*. GVU Center, Georgia Institute of Technology, Atlanta, 1999. Available online: http://www.gvu.gatech.edu/user_surveys/survey-1997-10/
18. [Kass et al, 1992]. Kass, Robert and Stadnyk, Irene. *Modeling Users' Interests in Information Filters*. Communications of the ACM, Dec. 1992, Vol. 35, No. 12.
19. [Koster, 1993]. Koster, M. *Guidelines for Robot Writers*. Nexor Corp. Available Online: http://www.robotstxt.org/wc/guidelines.html.
20. [Koster, 1995]. Koster, Martijn. *Robots in the Web: threat or treat*? 1995. Available online: http://www.robotstxt.org/wc/threat-or-treat.html
21. [Lawrence & Gilles, 1999]. Steve Lawrence and C. Lee Giles. *New 1999 study on the accessibility and distribution of information on the Web*. NEC Research Institute, 1998. Available online: http://www.neci.nec.com/homepages/lawrence/websize.html
22. [Lee et al, 1996]. Lee D. et al. *Effectiveness of Document Ranking and Relevance Feedback Techniques*. University of Science and Technology, Clear Water Bay, Hong Kong, January, 1996.
23. [Najork & Wiener, 2001]. Marc Najork and Janet L. Wiener. *Breadth-first search crawling yields high-quality pages*. Compaq Systems Research Center, Palo Alto, CA, USA. 2001. Available Online: http://www10.org/cdrom/papers/208/
24. [Notess]. Greg R. Notess. *Comparing Internet search engines*. Montana State University. Available online: http://www.csu.edu.au/special/online99/proceedings99/103a.htm
25. [Nwana, 1996]. Hyacinth S. Nwana. *Software Agents: An Overview*. Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996. Cambridge University Press, 1996. Available online: http://193.113.209.147/projects/agents/publish/papers/agentreview.htm
26. [Overture]. *Overture Premium Listings™*. http://www.overture.com/d/USm/about/advertisers/ays_reach.jhtml
27. [Pinkerton, 1994]. Brian Pinkerton. *Finding What People Want: Experiences with the WebCrawler*. Presented at the Second International WWW Conference in 1994. Available online:

http://www.thinkpink.com/bp/WebCrawler/WWW94.html

28. [Powel & Wicke, 1995] Bob Powel and K. Wicke. *Atlas to the World Wide Web*. Zift-Davis, Emerville, California, NewYork, 1995.
29. [Prosise, 1996]. Jeff Prosise. *Crawling the Web: A guide to robots, spiders, and other shadowy denizens of the Web.* PC Magazine, July 1996.
30. [RDU, 1995]. Resource Discovery Unit. *Resource Discovery--A Definition*. Resource Discovery Unit, DSTC Ltd, 1995. Available Online: http://www.dstc.edu.au/ RDU/RD-Defn/
31. [Ressler, 2000]. Sandy Ressler. *The Art of Electronic Publishing: The Internet and Beyond*. Prentice-Hall, Inc, Upper Saddle River, New Jersey, 2000. Chapter online: http://www.prenhall.com/electronic_publishing/html/chapter1/01_4.html
32. [Salton, et.al]. Salton G. et al. *A vector Space Model for Automatic Indexing*. Communication of the ACM, November 1975.
33. [Salton & Buckley, 1987]. Salton G., Buckley C. *Term Weighting Approaches in Automatic Text Retrieval*. Cornel University. New York. November 1987.
34. [Stone & Manuelo, 1997]. Peter Stone and Manuela Veloso. *Multiagent Systems: A Survey from a Machine Learning Perspective*. Carnegie Mellon University, Pittsburgh, USA. Available Online: http://www-2.cs.cmu.edu/afs/cs/usr/pstone/public/papers/97MAS-survey/revised-survey.html
35. [Sullivan, 2001]. Danny Sullivan. *Search Engine Submission Tips*. Available online: http://searchenginewatch.com/webmasters/
36. [Sullivan, 2002]. Daniel Sullivan]. *Google Tops In "Search Hours" Ratings*. The Search Engine Report, May 13, 2002. Available online: http://searchenginewatch.com/sereport/02/05-ratings.html
37. [SearchEngineWatch, 2001]. *Search Engine Sizes*. Dec. 18, 2001. Available online: http://www.searchenginewatch.com/reports/sizes.html
38. [SearchEngineWatch, 2002]. *The Major Search Engines*. Jan. 22, 2002. Available online: http://searchenginewatch.com/links/major.html.

39. [Vanhelsuwe]. Laurence Vanhelsuwe. Automating Web Explorations. http://www.javaworld.com/

40. [Weiss, 1996]. Scott Weiss. *Finding Stuff on the Web*. The Johns Hopkins University Baltimore, Maryland. Available online:
http://www.cs.jhu.edu/~weiss/websrch.html

41. [Yumomo & Lee]. Yuwomo B., D. Lee. *Search and Ranking Algorithms for Locating Resources on the World Wide Web*. Proceedings of ICDE, 1996.