Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition

Article · September 2002

Source: CiteSeer

4 authors, including:

Hamed Motahary
Uppsala University
1 PUBLICATION **28** CITATIONS

Donald Perlis
University of Maryland, College Park
145 PUBLICATIONS **1,768** CITATIONS

Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition

A. Abdollahzadeh Barfourosh[1]
H. R. Motahary Nezhad
Intelligent Systems Lab.,
Computer Engineering Dept.,
Amir Kabir University of Technology,
Tehran, Iran
ahmad@ce.aku.ac.ir

M. L. Anderson
D. Perlis
Institute for Advanced Computer Studies, and
Department of Computer Science,
University of Maryland,
College Park, MD, USA
{mikeoda@cs.umd.edu, perlis@cs.umd.edu}

Abstract

As more information becomes available on the World Wide Web (there are currently over 4 billion pages covering most areas of human endeavor), it becomes more difficult to provide effective search tools for information access. Today, people access web information through two main kinds of search interfaces: *Browsers* (clicking and following hyperlinks) and *Query Engines* (queries in the form of a set of keywords showing the topic of interest). The first process is tentative and time consuming and the second may not satisfy the user because of many inaccurate and irrelevant results. Better support is needed for expressing one's information need and returning high quality search results by web search tools. There appears to be a need for systems that do reasoning under uncertainty and are flexible enough to recover from the contradictions, inconsistencies, and irregularities that such reasoning involves.

Active Logic is a formalism that has been developed with real-world applications and their challenges in mind. Motivating its design is the thought that one of the factors that supports the flexibility of human reasoning is that it takes place step-wise, in time. Active Logic is one of a family of inference engines (step-logics) that explicitly reason in time, and incorporate a history of their reasoning as they run. This characteristic makes Active Logic systems more flexible than traditional AI systems and therefore more suitable for commonsense, real-world reasoning.

In this report we mainly will survey recent advances in machine learning and crawling problems related to the web. We will review the continuum of supervised to semi-supervised to unsupervised learning problems, highlight the specific challenges which distinguish information retrieval in the hypertext domain and will summarize the key areas of recent and ongoing research. We will concentrate on topic-specific search engines, focused crawling, and finally will propose an Information Integration Environment, based on the Active Logic framework.

Keywords: Web Information Retrieval, Web Crawling, Focused Crawling, Machine Learning, Active Logic

[1] Dr. Barfourosh is currently visiting
Active Logic Group,
Computer Science Department,
University of Maryland,
College Park, MD, 20742

Dr. Barfourosh would like to thank Dr. Marie-Cristine Rousset(Laboratoire de Recherche en Informatique, Artificial Intelligence and Inference Systems Group, Universite Paris-Sud, Orsay) for helpful discussions of the challenges of Information Retrieval and Integration.

Contents

Figures

Title **Page**

Tables

Title **Page**

1 Introduction

The web has had very rapid growth in number of pages, number of hosts and number of domain names (e.g. Cisco.com) registered worldwide [1]. There are more than 4 billion web pages and it is estimated that the number of web pages will exceed 16.5 billion by 2003 [2]. Almost 3 million pages or 59 Giga bytes of text are added daily, and the average life span of a web page is about 44 days [3]. To keep up with the changes to web content, one would need to download about same amount of bytes of information per day, which would mean you would need a connection capable of downloading 10 mega bytes of text per second [4]. The number of internet hosts increases exponentially. Figure 1 shows the growth of the internet hosts. The largest search engines have done an impressive job in extending their reach, though web growth itself has exceeded the crawling ability of search engines [5, 6]. Even the largest popular search engines, such as Alta Vista[2] and HotBot[3] index less than 18% of the accessible web as of February 1999 [6], down from 35% in late 1997 [7]. Figure 2 shows the percentage of the web coverage by the end of 2001. Today Google is probably biggest search engine, and has gathered more than 2 billion pages and covers only about 40 percent of the publicly available web pages.

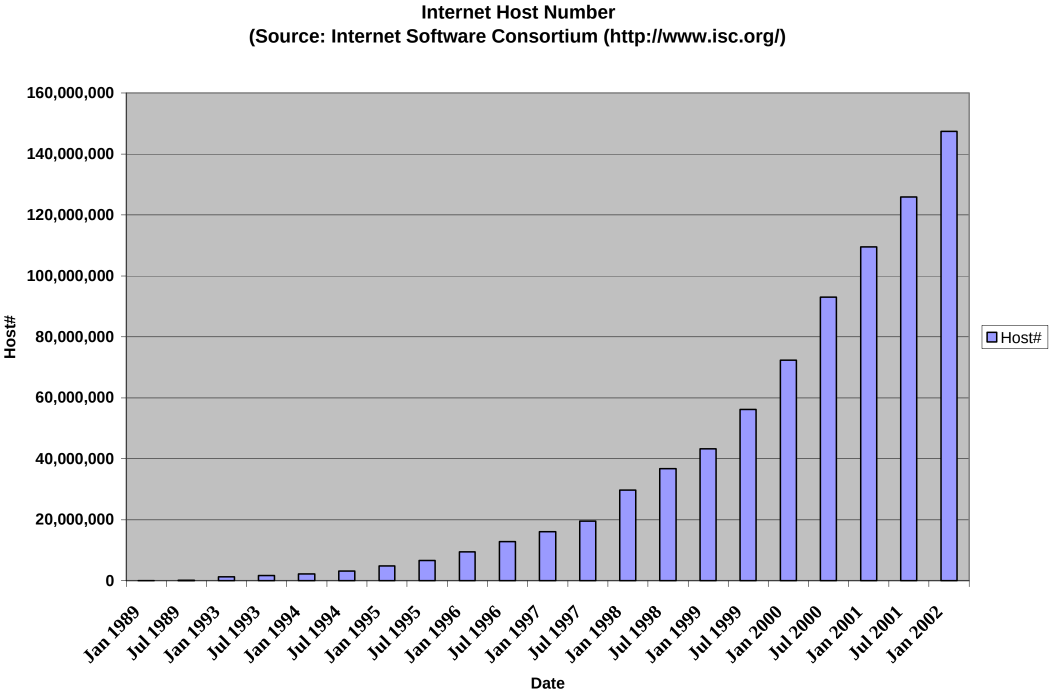

Figure 1. Internet host numbers since 1989 up to 2002 (Data from Internet Software Consortium[4]).

The number of queries that search engines must handle has grown incredibly, too. In March and April 1994, the World Wide Web Worm received an average of about 1500 queries per day. In November 1997, AltaVista claimed it handled roughly 20 million queries per day. With the increasing number of users on the web, and

[2] http://www.altavista.com
[3] http://www.hotbot.com
[4] http://www.isc.org/

automated systems that query search engines, it is likely that top search engines will handle hundreds of millions of queries per day [8].

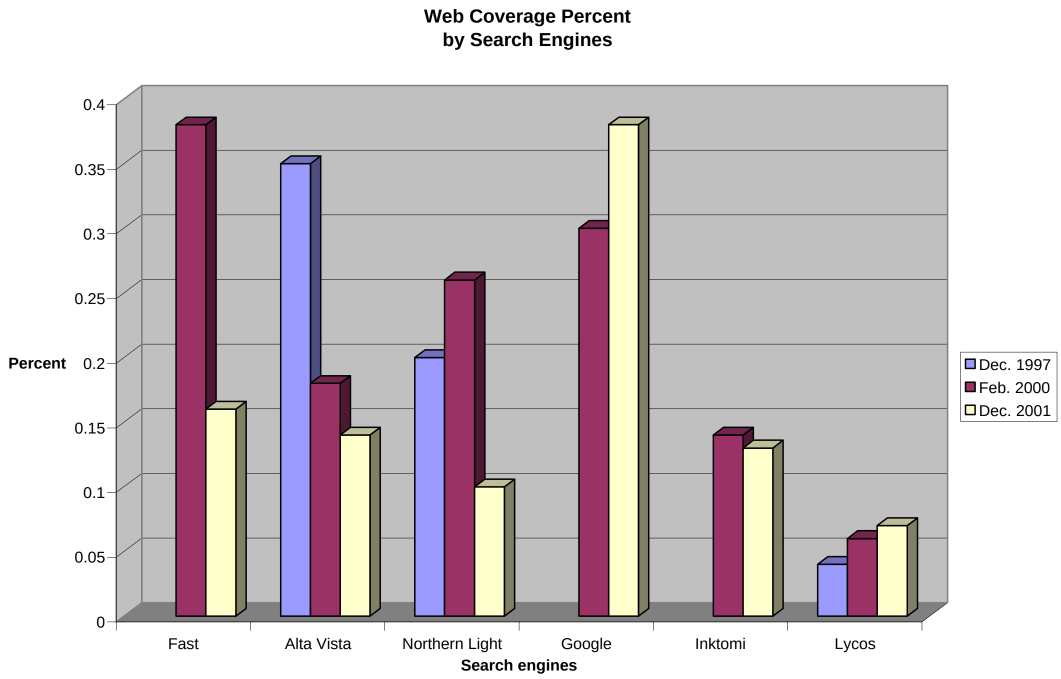

Figure 2. Web coverage percentage by popular search engines (Data from *The Search Engine Report*[5], Dec. 18, 2001).

Web search engines create and maintain an index of words within documents they find on the web. They return to a user a ranked list of relevant documents as search results. Few of these results may be valuable to a user [9]. Several ranking methods have been proposed to improve the ranking of resulting documents [10]. For this reason, it may be helpful to use some user information context in returning and ranking results.

Search engines are listed among the top accessed sites [11] and most people use them to find interesting information on the web [12]. As the web continues to grow, major general-purpose search engines have been faced with serious problems. They are unable to index all the documents on the web, because of the rapid growth in the amount of data and the number of documents that are publicly available. Their results may be out-of-date, and they do not index documents with authentication requirements or the information behind search forms. As more people share their information with others, the need for better search services to locate the interesting information is becoming increasingly important [10].

One way to deal with huge amount of web content is to build topic-specific search engines, each of them focusing on one or a limited number of topics, such that they crawl the related hyperlinks and avoid traversing the irrelevant part of the web.

[5] http://searchenginewatch.com/sereport/index.html

[13,14]. This approach needs web crawling mechanisms to be improved so that a crawler can better distinguish among links to achieve high performance.

> "Several different measures have been proposed to quantitatively measure the performance of classical information retrieval systems, most of which can be straightforwardly extended to evaluate web search engines. However, web users may have a tendency to favor some performance issues more strongly than traditional users of information retrieval systems. A basic model from traditional retrieval systems recognizes a three-way trade-off between the speed of information retrieval, precision and recall. In the context of information retrieval, precision is defined as the ratio of relevant documents to the number of retrieved documents and recall is defined as the proportion of relevant documents that are retrieved [15].
> Most web users who utilize search engines are interested in precision as precision of the results displayed in the first page of the list of retrieved documents [15]. Since measuring the recall rate for each web search engine query is not a trivial work, some researches recognize the recall in form of finding the most information rich pages, called authority pages and hub pages [16], i.e., pages that have links to many authority pages are also recognized as being very valuable." (See Table 1).

Classical Information System	**web Search Engines**
Speed	Interactive response times
Precision	Precision of the results displayed in the first page
Recall	Finding the most information rich pages, called authority & Hub pages

Table 1. The comparison of the measures of performance in classical information systems and web search engines (Data from [15]).

Two methods of finding interesting information on the web are *querying* and *browsing* [17]. Querying is keyword-based search. The search engines post the user query to their index of keywords and return a ranked list of documents. Browsing is navigation through a hierarchy-like set of links toward the target topic. In every step, the user selects the links that (s)he guesses will lead to relevant documents. Querying is appropriate when user has a clear goal. Browsing is suitable when user cannot express his goal explicitly by a Boolean combination of the keywords. Browsing also is suitable for situations where the goal is general information on a topic.

People have difficulty with the typical query mechanisms of search engines. Search engine queries tend to be the same as those used by the first search engines. They accept the user query as a set of keywords in the form of a Boolean expression. Unfortunately the keyword-matching method usually returns too many low quality results [8, 14, 18]. Furthermore, the formulation of some complex queries in the form of a Boolean expression is difficult or impossible. People may find natural language a better choice to convey their information needs, since simple keywords may not be able to convey the complex search semantics that a user wishes to express [18].

In following sections, we review several basic concepts in the area of web information search. In particular we will consider the machine learning approaches in domain-specific search engines and focused crawling methods. In section 2 we describe the types of search engines, and our goal is to answer the question: "How does a search engine work?" Section 3 reviews different web content and structure analysis. Section 4 describes machine-learning approaches used in information retrieval. Section 5 mainly describes the topic-specific search engine and focused crawler architecture. Section 6 introduces the Information Integration Environment based on Active Logic. Our conclusions are presented in section 7.

2 Search Engines

A "search engine" is a resource that provides the ability to search information on the Internet [19]. Search engines come in three major flavors:

- Web crawlers
- Web portals
- Meta-Search engines

Each of them has its strengths and weaknesses. In the following subsections we review each in detail. Since different search engines provide different services and features, comparison among them is an important matter for users. Several parameters that can be used to compare search engines [20] that are listed in tables 2 and 3 and 4. Table 2 shows the parameters from "searching features" perspective.

Evaluation Parameter	Description
Default search	How does the engine put the keywords together, for example '*AND'* between the words (inclusive) , or '*OR'* between them (alternative).
Keyword/Concept default	Concept searching occurs when the engine not only searches for the exact character string, but also for word forms, and even synonyms and other words that statistically appear with the typed word.
Exclusion possibility	Ability to exclude web pages (results) including special terms, search engines represent it by putting a minus or '*NOT'* in front of excluded term
Truncation	Possibility of finding various form of a word by adding a truncation symbol (such as '*') on the end of the word
Search restrictors	Ability to search for terms or values contained only in certain portions of a page, rather than anywhere in the entire page or within special kind of pages (sound, image, video) or in special site domains (.com, .edu)
Date searching restrictors	Try to place a date restriction in search query. Date restrictions can be useful to locate newly created or recently updated web pages, weeding out older results.
Phrase searching	Ability of using quotation marks around some terms or a kind of Boolean connector such as *ADJ* between the terms for phrase searching
Nesting	Support the use of parentheses to nest various parts of a search query, for example *(apple or blueberry) ADJ pie* that means *apple pie" OR "blueberry pie"*
Multi-level search	Ability to first casting a wide net, then narrowing by searching only within that set of results
Case sensitive	If the search engine is case sensitive or no?
Language restrictor	Ability to search the web pages in various languages such as English, German, …
Natural language support	Can it handle queries in natural language?

Table 2. Evaluation parameters of the search engines from search language perspective.

Table 3 shows the parameters of search engine coverage, database and manner of search. Table 4 shows the parameters from returned result perspective.

Evaluation Parameter	Description
Content size	How big is its database, i.e. how many web pages are indexed in its database
Search parts	If they search full text of web page or a specific parts of it such as keywords, titles, headings, links of web pages, …
Various kind of web resources indexing	Indexing the document from other internet sources such as usenet, peoples, email texts, …
Focused topic	Whether the search engine focuses on a specific topic or document type or it is a general-purpose one
Web crawling strategy	The manner that search engine traverses the web link's graph, for example breadth-first-search, according to priority queue and some parameters, such as Hub and authority score of page, Page-ranking, …

Table 3. Evaluation parameters of search engines from the engine's perspective.

Evaluation Parameter	Description
web pages Ranking methods	Different parameters used to specify the rank of web pages in returned result list, such as site popularity, …
Various display option	If various options are available to rank the returned result, such as by date, by site, …
Suggested search	Suggestions for further searching based on the initial search are provided or no. These suggestions can be simple, such as synonyms or alternative search terms, or may be more sophisticated, such as suggestions for searching in different, specialized databases.
Similar searches	If someone locates a web page that is highly relevant to his research issue, It might be interested in finding more pages that are very similar, is it available?
Translated results	Possibility of offering a tool to translate a given result page from one language to another.

Table 4. Evaluation parameters of search engines from the perspective of returned results.

2.1 Web Crawlers

Web crawlers, also known as "robots," "spiders," "worms," "walkers," and "wanderers," are almost as old as the web itself [21]. The first crawler, "Matthew Gray's Wanderer", was written in the spring of 1993, roughly coinciding with the first release of NCSA Mosaic [22]. A web crawler seeks the Internet looking for pages to index. In general it starts with a set of predefined web addresses and downloads them. For each page, it extracts its URLs in order to follow them later in a specified manner, for example breath-first-search. Then it indexes all of its word and phrases and maybe the relative position of the words to each other. Later, a user can search this index for the presence of a particular word, phrase or even combination of some words in a web document. Usually, web crawlers store the complementary information for each page, such as time of download and update, different ranks that are computed off-line, header and title, etc.

The "crawler" concept stands for the fact that it extracts all URLs within a downloaded page to be followed later. Figure 3 shows the workflow of a typical web crawler. A web crawler starts with a single or a set of default pages and it continues (theoretically) until it has downloaded every web page on the Internet. It is assumed that it can traverse all of the web graph links from the start page set. However, it is clear that this theory is not correct in practice, in at least two aspects. Firstly, it cannot reach all of the web documents from a single point in its graph. There is not a path from any given page to every other page on the Internet. Secondly, a search engine cannot cover all of Internet pages because of its power and time limitation in gathering all of the web documents. New pages are added daily with more speed than the web crawler gathers web pages, and some web pages will be updated long before the web crawler could crawl them again.

Web crawlers are typically automatic and the keywords are stored in indexes, each of them associated with the documents they were found within. Human maintenance has not generally had an important role in web crawler indexing and querying. Web crawlers are best for finding *specific* information, but not *common* information [21]. If an interesting search topic is very general, e.g. "computers," a web crawler will return thousands of results that contain the word "computer". In this list, finding an appropriate page is difficult. For this kind of information, a web *portal* is a better fit. In next subsection, we will review web portals.

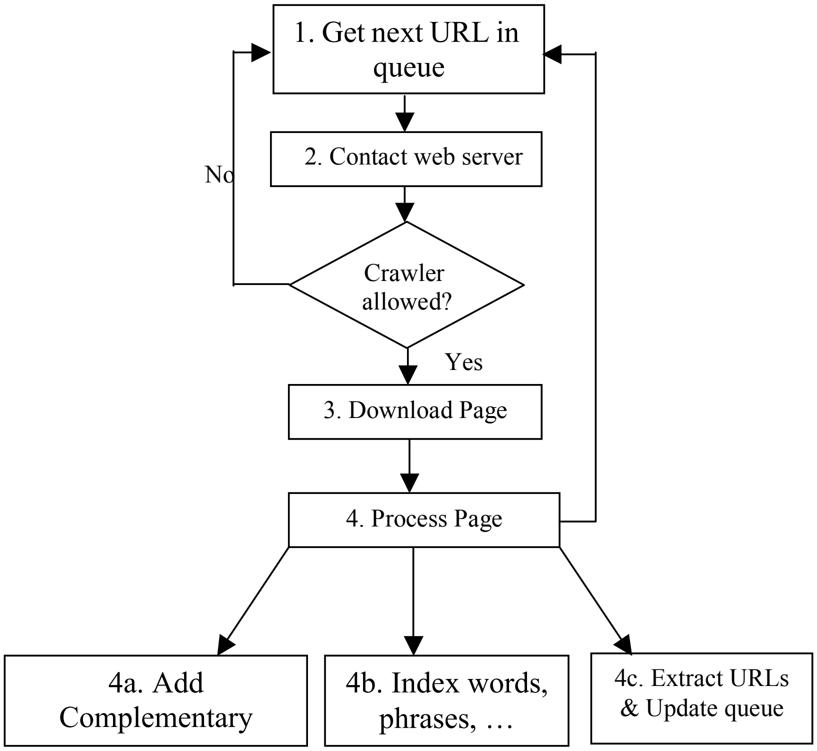

Figure 3. Workflow of a typical web crawler (from [19])

There are several hundred commercial web crawlers. **AltaVista**[6], **Excite**[7], **Google**[8]**,** **HotBot**[9], **Lycos**[10] and **Northern Light**[11] are some of the most popular web crawlers.

2.2 Portals

A web portal is generally considered to be a site that organizes information by topic [19]. It is also called "web directory". In web crawlers, users can define the search criteria and search the web crawler's indexed database for those criteria. A portal, in contrast, organizes the sites by topic to help in navigating and finding what the user is looking for. Most portals are human maintained. The large portals work hard to try to catch all of the intuitive categorizations of a given topic, but this does not always work. Portals also allow one to search their archives much like a web crawler, but search is generally possible only on the summaries and titles of sites in the portal, not on their contents.

Portal providers generally construct portals according to the web pages that companies, institutes and individuals send to them; in addition, their employees search and browse the web to find interesting web pages on various topics. Figure 4 shows the workflow of a topical portal. Some portals, after acquiring the web pages, associate a number among 0 to 4 to each web page according to "Usability, Freshness, and Understandability". Furthermore, from each page a summary will be extracted, from 2-3 lines to a full paragraph. Then they use different parameters to construct a hierarchy of topics. Acquired web pages will be placed in lower levels of this hierarchy according to their relevancy. This hierarchy will be maintained by humans to update and add more relevant documents. The top level of this hierarchy has a limited number of general topics, varying between 14-26 topics. The number of subtopics in lower levels depends on the topic and available web pages about it. There is some research that aims to automate classification and construction of these hierarchies, and some of these projects result in an acceptable level of precision [23, 39, 41, 42].

Portals are very efficient for finding common information, but they are unable to organize everything, so specific information is not nearly as easy to find [19]. This is one of the first rules to know when deciding between a crawler and a portal in order to find information. Some of the popular portals are Yahoo![12], Looksmart[13], Open Directory Project[14], IBM's Patent Database[15]. These portals differ according to the

[6] http://www.altavista.com
[7] http://www.Excite.com
[8] http://www.google.com
[9] http://www.hotbot.com
[10] http://www.lycos.com
[11] http://www.northernlight.com
[12] http://www.yahoo.com
[13] http://www.looksmart.com
[14] http://www.dmoz.org
[15] http://www.delphion.com

amount of information they keep about every web pages and information items like summaries, titles, URLs, etc.

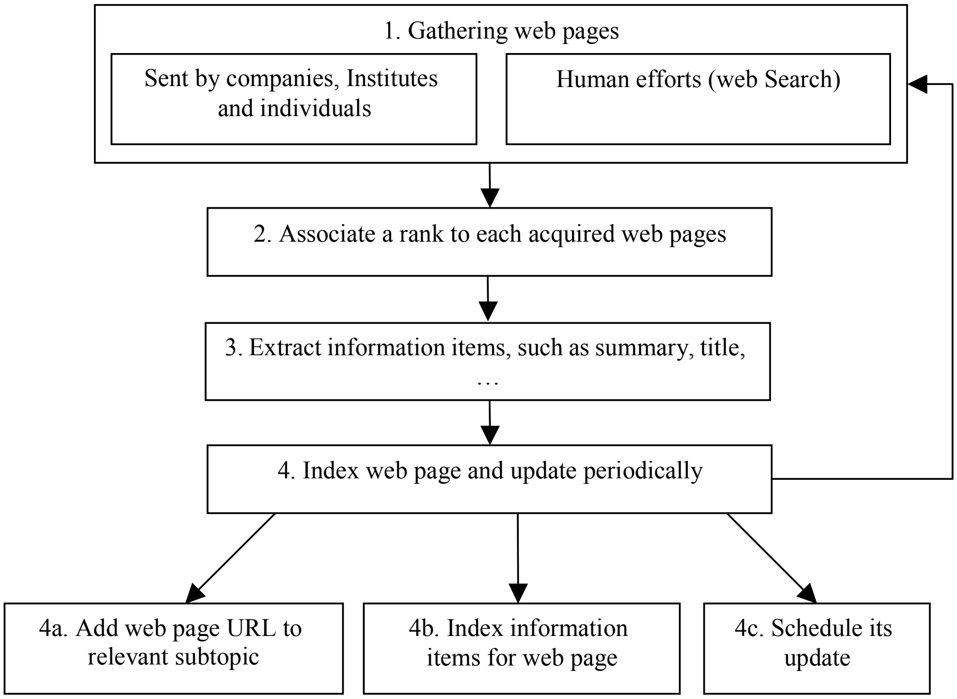
1. Gathering web pages
Sent by companies, Institutes and individuals
Human efforts (web Search)
2. Associate a rank to each acquired web pages
3. Extract information items, such as summary, title, …
4. Index web page and update periodically
4a. Add web page URL to relevant subtopic
4b. Index information items for web page
4c. Schedule its update

Figure 4. Workflow of a topical portal

2.3 Meta-Search Engines

Meta-search engines (or meta-crawlers) are sites that take queries (keywords or even natural language queries), send them to a large number of search engines and return the results to user. Meta-search engines use three methods to search the web:

- Direct list of search engines
- Sequential searches
- Concurrent search

2.3.1 Direct list of search engines

This kind of search engine sends the user query directly to a list of search engines and acquires their results for that query, as if the user directly posed his query in each of them in isolation. The benefit of this kind of search engine is that they save the user's time. This approach may also cover some search engines that the user has never tried. A number of meta-search engines rank the returned results of search engines using parameters such as search engine popularity, query terms, etc.

2.3.2. Sequential search

In this kind of search engine a user can select some search engines from a list and send the user query to these selected search engines. Usually the results will be shown just as they are returned from the search engines. These meta-search engines wait to receive all of the results and then display the result page, so it is as slow as the slowest selected search engine.

2.3.3 Concurrent search

This kind of meta-search engine is similar to sequential search method, but it does not wait to receive the whole result from all search engines. As it receives the first search engine results, it displays them, and new received results will be added gradually. This approach decreases the time before the user sees the first results from the search engine.

There are two tips about meta-search engines, first, a meta-search engine can only take inputs that are supported by all search engines that it uses, or it must convert the user's query into a standard form supported by every search engine. Thus, the lowest common denominator of those sites' features will determine what user can enter. In addition, the transformed query may not satisfy the user intention. Since meta-search engines do not allow for input of many search variables, their best use is to find hits on obscure items or to see if something is on the web. Second, what is the point of convenience? The real convenience is finding the best result quickly, not getting the largest number of bad results. The primary motivation of using a meta-search engine is that, since the web is huge and the most search engines in isolation cover a small fraction of web and have low recall and precision in their search results, it is better to use the results of several search engines, combine and re-rank them using a good algorithm, and return this result to the user. This scheme may increase recall and precision. However, while meta-search engines represent cumulative search results over other search engines, they still do not cover the entire web. Figure 5 shows the workflow of a topical meta-search engine. Some of the best-known meta-search engines are Dogpile[16], Mamma[17], Metacrawler[18], and Askjeeves[19].

[16] http://www.dogpile.com
[17] http://www.mamma.com
[18] http://www.metacrawler.com
[19] http://www.askjeeves.com

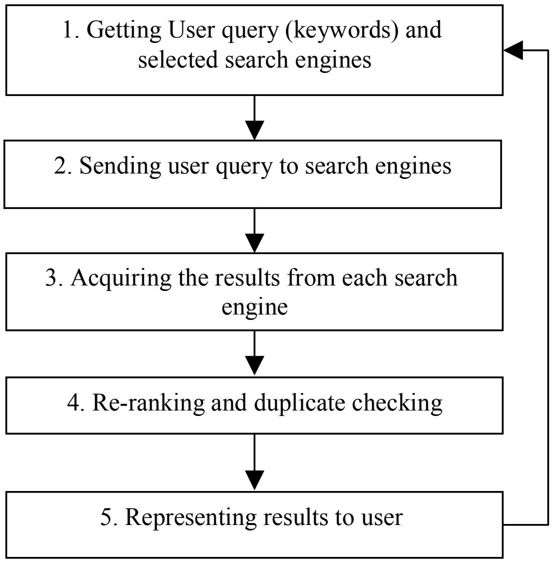
1. Getting User query (keywords) and selected search engines
2. Sending user query to search engines
3. Acquiring the results from each search engine
4. Re-ranking and duplicate checking
5. Representing results to user

Figure 5. Workflow of a topical meta-search engine

3 Web Analysis

The web is similar to a graph, in that links are like edges and web pages are like nodes. Several approaches have been proposed to overcome the current limitations of web crawlers. Some approaches use web structure (relation between web links) to guide web crawlers in finding their path through the web, and some approaches use web content (text within each page) to perform the same thing. As we shall see in the next subsections, a combination of these two aspects of web search will improve the functionality of web crawling strategies further.

3.1 Web Content Analysis

The basic use of the content of a web page in search engines is in the form of exact query matching in a document index database. Our intention here is to shortly review different document similarity measures. We consider two kinds of similarities [26]:

- *Document-query similarity* is important for identifying similarity of user query to documents.
- *Document-document similarity* is important for finding similar documents in the document pool of a search engine. Some search engines represent this feature.

3.1.1 Document-query similarity

"Suppose that we have a term vector $T(t_1, t_2, \ldots, t_n)$ represent all of the terms in our corpus. Furthermore $D(t_i, t_j, \ldots, t_k)$ represents presence of the correspondent terms in T in document D and $Q(t_a, t_b, \ldots, t_m)$ represent the presence of the correspondent terms in T in query Q. There are several methods for weighting the terms in document and query. There are some local weighting functions such as binary weighting that weights are either 0 or 1 shows presence of terms, or Term-frequency weighting, Log-Entropy weighting. There are also some global weighting functions that are simple and inexpensive to implement such as Normal, TFIDF, IDF, and

Entropy [26]. One of the simplest formulations of a query-document similarity value is

$$Similarity(Q,D) = \sum_{k=1}^{t} w_{q_k} w_{d_k}$$

Where Wq_k is weight of term k in query and Wq_k is weight of term k in documents. Note that when binary term weights are used, this similarity measures the number of terms that appear in both Q and D."

Adapted from [26]

3.1.2 Document-document similarity

Measuring the similarity of two documents is important in search engines, because when a user finds a web page that is on-topic, he may desire to find other relevant documents. There are different parameters which determine the relatedness of two documents: some are related according to their content, and others by their in and out hyperlinks. In this section we consider only document content. Table 5 shows the different models of document-to-document similarity measuring.

Model	Description	Methods
String distance (Edit distance)	Considers the distance as amount of difference between strings	Levenshtein distance, LikeIt
Statistics of words	Considers frequency of words in documents to judge on similarity	TFIDF, LSI
Document components or structure	Considers structure or components of documents, for example references, abstract, title, keywords and … in research papers	Citation analysis, ParaSite

Table 5. Three models of measuring document-to-document similarity [Data from 26, 30]

3.1.2.1 String distance model

In Levenshtein distance [27], the difference between two strings is the number of insertions, deletions, or substitutions of letters required to transform one string into another. In LikeIt [28, 29] a string distance is based on an algorithm that tries to build an optimal weighted matching of the letters and multi-graphs (groups of letters). LikeIt tries to match sub-strings in a larger string.

3.1.2.2 Statistics of words model

TFIDF [30] (*term frequency × inverse document frequency*) is based on word frequencies in documents. In fact it is suitable for sets of documents, especially as part of a large number of documents. The common (stop) words such as "the", "a", etc., are ignored for computational efficiency. Sometimes stems of word are considered instead of complete words. A stemming approach by Porter [31] tries to return the same stem from several forms of same word (e.g. "learning", "learn", "learned" all become "learn"). In this approach, for each word in document, the weight of the word is calculated based on word frequency in a given document, the number of documents that include the word, the highest word frequency in a document, and the number of all documents in the document pool. Then the distance between two documents is calculated by dot product of the two word vectors for those documents.

Latent Semantic Indexing is an approach that uses a vector space model of a document to measure the similarity between two documents. Vectors in the vector model space are the weight of the words in document.

> "Latent Semantic Indexing (LSI) is a variation of the vector space model of information retrieval that uses techniques of singular value decomposition (SVD) to reduce the dimensionality of the vector space."
>
> Soumen Chakrabarti [7]

This method considers the co-occurrence of words in a document, so it can derive a relationship between words and inherent concepts. The similarity of two documents is calculated by the cosine of the reduced vector space of two documents. Since this approach uses conceptual matching rather than exact word matching, researches [32] show that LSI provides better results than standard TFIDF, with fewer training sets of documents per category.

3.1.2.3 Document components or structure model

We can use the knowledge about document components or structure to judge the similarity between two documents. This approach is well suited for situations in which documents are of a specific type, or have special components or structure. In the case of research papers, for instance, they have similar structure and components, such as title, abstract, keywords, references. Also, common citations can be used as a parameter [33]. The ParaSite system [34] uses the nearness of links to referenced web pages in the HTML structure of a referencing web page as an indicator of relatedness of the referenced pages.

CiteSeer [35] is a computer science research paper finder that uses several methods for document similarity measurement. CiteSeer uses the LikeIt string distance to measure the edit distance between the headers of document. It uses common authors, institutions, or words in the title of documents to reduce the LikeIt distance between headers. CiteSeer also uses common citations to make an estimate of document similarity. This measure, "Common Citation × Inverse Document Frequency" (CCIDF) is analogous to word-oriented TFIDF word weights. CiteSeer combines different methods of document similarity to result in a final similarity distance measure that is hopefully more accurate than any single method alone.

3.2 Analysis Link Structure of Web

> "Every web page has some number of out-links and in-links (See Figure 6). We can never know whether we have found all the in-links of a particular page, but if we have downloaded it, we know its entire out-links at that time."
>
> Page and Brin [8].

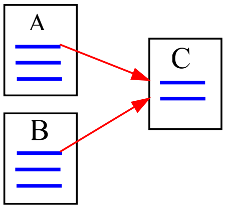

Figure 6. A and B are in-links of C (from [8]).

Some approaches use the link structure of the web to find the importance of the web pages or to determine their relatedness to a particular topic. In the next subsections we present two approaches, the “Page Rank” method and the “Hub and Authority” concept, which use the link structure of the web for this purpose.

3.2.1 Ranking

Web pages differ from each other in the number of in-links that they have. For example, the popular sites like “Yahoo!” and “Netscape” have many in-links but a typical homepage of a university student may have few in-links. The number of references (citations) to a thing is evidence of its importance; many Nobel Prizes are assigned according to this fact. Considering this, we can say that highly linked pages are more “important” than pages with few in-links [36]. L. Page and S. Brin proposed the Page Rank algorithm in [8, 36, 37] that calculates the importance of web pages using the link structure of the web.

> “It is somehow different and is more sophisticated than simply counting the number of in-links of a web page. The reason is that there are many cases where simple citation counting does not correspond to our common-sense notion of importance. For example, if a web page has a link from the Yahoo home page, it may be just one link, but it is a very important one. This page should be ranked higher than other pages with more links but only from obscure places. Page Rank is an attempt to see how good an approximation to “importance” can be obtained from just the link structure of the web.”

Page and Brin [8].

3.2.1.1 Page Rank Algorithm

Page Rank makes use of the link graph of the web. The Page Rank algorithm is defined as follows:

> “We assume page A has pages T1...Tn which point to it (i.e., are citations). The parameter d is a damping factor, which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C (A) is defined as the number of links going out of page A. The Page Rank of a page A is given as follows:
>
> $$PR(A) = (1-d) + d\,(PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))$$
>
> Note that the Page Ranks form a probability distribution over web pages, so the sum of all web pages’ Page Ranks will be one.”

Page and Brin [8]

Note that the rank of a page is divided evenly among its out-links to contribute to the ranks of the pages they point to. The equation is recursive, but starting with any set of ranks and iterating the computation until it converges may compute it.

Page Rank can be calculated using a simple iterative algorithm, and corresponds to the principal eigen vector of the normalized link matrix of the web. Page Rank algorithm needs a few hours to calculate the rank of millions of pages [8].

3.2.1.2 Page Rank Algorithm application

This algorithm is used as the base of the web crawling algorithm in the Google search engine. Google makes use of both link structure and anchor text [8]. Google orders the URL’s according to several parameters. Then the URLs with highest ranks will be crawled first [37]. It uses Page Rank as a parameter for measuring page importance among crawled pages to rank the result for user queries [8].

3.2.2 Hub and Authority pages

As we described in section 3.2.1, the importance of a pages can be extracted from the link structure of web. In this approach two kinds of pages are identified from web page links: pages that are very important and authorities in a special topic, and pages that have great number of links to authority pages. Kleinberg, when he was as visiting researcher in IBM's Almaden Research Lab, proposed an algorithm to identify these pages on the web. In the next subsection we review this algorithm, called HITS.

3.2.2.1 HITS Algorithm

For a given query, HITS will find good sources of content (defined as authorities) and good sources of links (defined as hubs) [16]. Authorities have large in-degree. Hub pages are pages that "pull together" authorities on a given topic, and allow us to throw out unrelated pages of large in-degree (Those pages are simply universally popular like Yahoo!). (See Figure 7)

> "Hubs and authorities exhibit what could be called a mutually reinforcing relationship: a good hub is a page that points to many good authorities; a good authority is a page that is pointed to by many good hubs".
>
> Kleinberg [16]

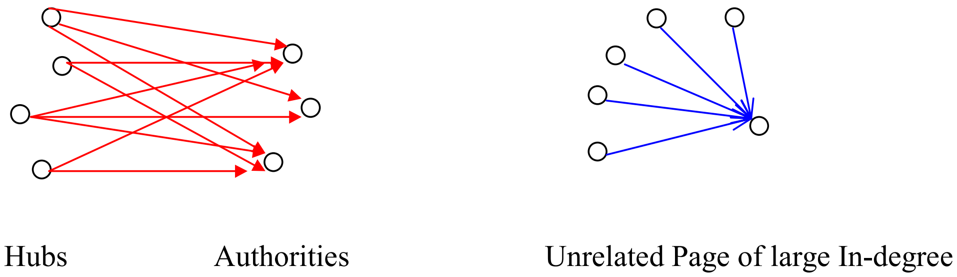

Figure 7. A densely linked set of Hubs and Authorities (from [16])

HITS associates a non-negative authority weight $x^{<p>}$ and a non-negative hub weight $y^{<p>}$(Figure 8). The weights of each type are normalized so that their squares sum to 1.

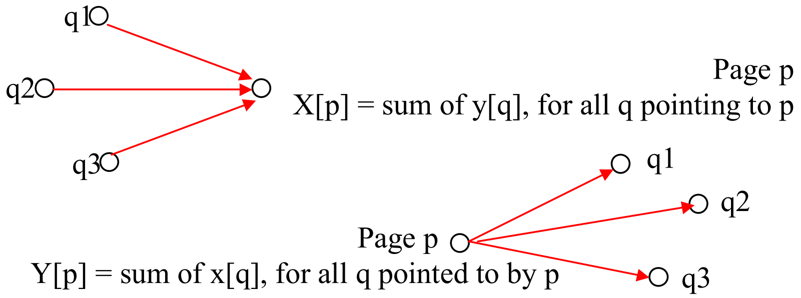
q1
q2
q3
Page p
X[p] = sum of y[q], for all q pointing to p
q1
q2
Page p
q3
Y[p] = sum of x[q], for all q pointed to by p

Figure 8. The basic operations of HITS (from [16])

> "Numerically the mutually reinforcing relationship can be expressed as follows: if p points to many pages with large x-values, then it should receive a large y-value; and

if p is pointed to by many pages with large y-values, then it should receive a large x-value. Given weights $x^{<p>}$,$y^{<p>}$, then the x-weights is as follows:"

$$X^{<p>} \leftarrow \sum_{q:(q,p)\in E} y^{<q>}$$

and y-value is as follows:

$$Y^{<p>} \leftarrow \sum_{q:(p,q)\in E} x^{<q>}$$

Kleinberg [16].

Bharat and Henzinger [38] point out that HITS did not work well in all cases due to the following three reasons:

- *Mutually Reinforcing Relationships Between Hosts:* Sometimes a set of documents on one host point to a single document on a second host. This drives up the hub scores of the documents on the first host and the authority score of the document on the second host. The reverse case, where there is one document on a first host pointing to multiple documents on a second host, creates the same problem. Since our assumption is that a single author or organization authored the set of documents on each host, these situations give undue weight to the opinion of one "author".
- *Automatically Generated Links.* Web documents generated by tools (e.g. web authoring tools, database conversion tools) often have links that were inserted by the tool.
- *Non-relevant Nodes:* They show that the neighborhood graph often contains documents not relevant to the query topic. If these nodes are well connected, topic drift problem arises: the most-highly ranked authorities and hubs tend not to be about the original topic. For example, when running the algorithm on the query "mango fruit" the computation drifted to the general topic "fruit".

Bharat and Henzinger proposed an approach to solve the topic drift problem [38]. They used the web page's content in addition to its graph structure. For each node in the link graph, they consider its relevancy to the query topic by calculating a relevancy weight to topic, W[n]. They use W[n] * H[n] instead of H[n] in computing the authority score of nodes, considering following notation, we have:

A[n]: The authority score of node, H[n]: The hub score of a node,

W_{Auth}: Authority weight of node, W_{Hub}: Hub weight of node

$$H[n] = \sum_{(n',n)\in N} A[n'] \times W_{Hub}(n',n)$$

$$A[n] = \sum_{(n',n)\in N} H[n'] \times W_{Auth}(n',n)$$

Both of Page-Rank and HITS algorithms use the links structure of web to find importance of web pages, Table 6 shows a comparison between these approaches.

HITS	Page Rank
Can distinguish between pages with high number of in-links but not related to topic and related to given query	Blindly calculates the importance of a page according to its in-links and out-links regardless of given query
HITS is suitable for topic driven page importance measuring	Page Rank is suitable for measuring overall ranking of sites and pages and their importance from the perspective of people citation regardless of topic, estimating the popular or highly cited sites
Refined HITS [38] considers the web page content in addition to link structure	Page Rank uses just link structure of the web

Table 6. A comparison between HITS and Page Rank algorithms

3.2.2.2 HITS Algorithm Application

For the first use, it was implemented in the Clever[20] search engine from IBM. Then several enhancements to this approach were proposed by researchers at the IBM Almaden center [40]. Finally an improved version of it was used in the *Focused Crawler* [13, 14] from same research institute.

4 Machine Learning Approaches in Information Retrieval

Learning is needed to improve the functionality of systems [18]. Different algorithms and methods for machine learning are used. These categories of algorithms are known as machine-learning algorithms. Search engines, like other computer systems, have used machine-learning approaches to improve their functionalities in various aspects. In this section we review machine-learning approaches that are used in web information retrieval systems. Definitions, algorithms and applications of each method are briefly discussed. Some example search engines that use machine learning approaches are described in section 5.

In machine learning, we have three main categories of approaches. The difference between these approaches is in how the learner agent learns. Table 8 shows these approaches and their brief description.

Approach Name	Description
Supervised Learning	The learner agent learns from a collection of labeled training data, in other words, in the training phase the agent knows the correct answer for each input state. In the test phase the agent must guess the label for un-labeled cases.
Semi-supervised learning	The learner agent learns from a small collection of labeled training data. In this category of problem a small collection of training data are available and we can not provide for the system enough labeled training data like real world problems.
Un-supervised learning	The learner agent gives a set of un-labeled data and it performs the expected work according to some specific measure such as similarity. No training data will be given to agent.

Table 8. Main three categories of machine learning approaches (Data from [7])

[20] http://www.almaden.ibm.com/cs/k53/clever.html

4.1 Supervised learning

Supervised learning – also called classification – is based on learning from a training data set. Each data item in the training set has a label or a class; in other words the learner is told the correct answer for each item. This class of algorithms learns from the training set and tries to guess the label for an input item in the test phase.

Classification has several applications in the hypertext and semi-structured data domains. Classification is important in order to guess the relevance of a web document to the crawl topic. In this case, in the training phase the agent builds a model from a set of pre-classified documents (acquiring some parameters in a model). Another important application of classification is in constructing directories in portals. Web searchers may find directories easier to use when finding some general information. Automatic construction and maintenance of such portals is one of the leading research areas in applying classification in web IR systems [23, 39, 40].

4.1.1 Naïve Bayes classifier

In this model of learning [7, 44] it is assumed that text documents are generated from a parametric model. This model estimates the model parameters from training data. For each new document, using the estimated parameters and Bayes rule, the classifier calculates the probability of generation of document by each class. The classification is selecting the class with highest probability.

The classifier parameterizes each class with word frequency and document frequency. "Naïve" stands for this assumption that each word occurs in a document independent of all of other words in document and also independent of its occurrence location. Using the following notations, [23] formalizes this model as follows:

w_t : word *t* in document , c_j : Class *j* in training data, V : Vocabulary,
$P(c_j)$: frequency of document in class *j* in compare with all of other classes,
$P(w_t|c_j)$: Probability of occurring of word w_t in each document of class c_j,
d_i: document *i* with a set of word, $P(c_j|d_i)$: Probability that document *i* is generated by class *j*, w_{dik} : k^{th} word in i^{th} document.

Using Bayes rule and Naïve assumption we could write:

$$P(c_j|d_i) \propto P(c_j)\, P(d_i|c_j)$$
$$\propto P(c_j)\, \Pi_{(k=1\,..\,|di|)} P(w_{dik}|c_j) \quad (1)$$

The $P(w_t|c_j)$ and $P(c_j)$ parameters are learned from the training data set. To estimate probability of word w_t in class c_j, $P(w_t|c_j)$, The frequency of word w_t is enumerated in each class c_j is enumerated. If $N(w_t,d_i)$ is the number of occurrence of word w_t in document d_i then we could write:

$$P(w_t \mid c_j) = \frac{1 + \sum_{d_i \in D} N(w_t, d_i) P(c_j \mid d_i)}{|V| + \sum_{s=1}^{|V|} N(w_s, d_i) P(c_j \mid d_i)} \qquad (2)$$

Where $0 < P(c_j|d_i) < 1$. The frequency parameter of each class, $P(c_j)$ is calculated as:

$$P(c_j) = \frac{1 + \sum_{d_i \in D} P(c_j \mid d_i)}{|C| + |D|} \qquad (3)$$

Where |C| is number of classes and |D| is number of documents. Experimental results show that if the number of training documents is large, then the accuracy of classification is good [45].

There are various classification approaches. Support Vector Machine (SVM) is a classification method that has been used for multi-class classification of hypertext documents in comparison with well-known Naïve Bayes classifiers. Experimental results show that it has much lower error score than Naïve Bayes in this context [46].

4.1.2 Learning Relations

Learning relations between documents is an approach to extend the classification of documents, described in [7].

> "Consider the problem of classification of web pages of a computer science department web site in "Faculty", "Student", "Project", "Course" classes. Simply, the following relation between these classes can be found:
>
> teaches(faculty, course),
> advises(faculty, student),
> enrolled(student, course),
>
> Learning such relations may improve the functionality of classification and it enables us to answer more complex queries such as "courses that are taken by students" or even "list of faculty that supervise more than 5 students". Learning the relation between pages can be done by exploiting hyperlinks in a web site. Word statistic of neighborhood documents in additional to graph structure of web pages can be used to augment learning relations."
>
> Soumen Chakrabarti [7].

4.2 Semi-supervised learning

Semi-supervised learning is a goal-directed activity, which can be precisely evaluated, whereas unsupervised learning is open to interpretation [47]. On the other hand, supervised learning needs a large training data set, which must be obtained through human effort [47]. In real life, most often one has a relatively small collection of labeled training data, but a larger pool of unlabeled data. In the web context our training data is a small set of labeled documents. The label is document class, and our goal is to guess the label of an un-seen document. In this category we review learning from labeled and unlabeled documents in section 4.2.1. In some semi-supervised approaches, a learner agent learns from interaction with a dynamic environment. In these environments, providing a set of training data for the agent is very difficult or even impossible, because of the dynamics inherent in the environment and correspondingly huge number of states and actions. In these environments the learner agent is never told the correct action in a state, instead it is told how good or how bad its action was. One requirement of this model is a measure of the goodness of action that the agent takes in a state. In section 4.2.2 we will review reinforcement learning from this category of learning methods.

4.2.1 Learning from labeled and unlabeled documents

Expectation Maximization (EM) [47] is an approach that uses a set of labeled and unlabeled documents to learn a multi-class classification problem. This algorithm first trains a Naïve Bayes classifier using only labeled documents and builds a model of data. EM is a class of iterative algorithms for maximum likelihood or maximum *a posteriori* parameter estimation in problems with incomplete data [47]. For a model of training data with some missing values, EM iteratively uses the current model to estimate the missing values, and then uses the missing value estimates to improve the model. EM is an iterative two-step process, E-step and M-step. The E-step calculates probabilistically-weighted class labels, $P(c_j|d_i)$, for each document using the classifier and equation (1). The M-step estimates new classifier parameters using all the documents and equations (2) and (3). The iteration of E-step and M-step continues until (near-) convergence. Results are mostly favorable---compared to naïve Bayes alone, error is reduced by a third in the best cases, but care needs to be taken in modeling classes as mixtures of term distributions.

4.2.2 Reinforcement learning

The term "reinforcement learning" refers to a framework for learning optimal decision making from rewards or punishment [48]. It differs from supervised learning in that the learner is never told the correct action for a particular state, but is simply told how good or bad the selected action was, expressed in the form of a scalar "reward".

Using following notations, [23] formalizes a task in reinforcement learning as follows:

S: set of states, s: a state in S,

A: set of available actions, a: an action in A,

T: $S \times A \rightarrow S$, a state-action transition function (mapping state/action pairs to the resulting state),

R: $S \times A \rightarrow R$, a reward function (mapping state/action pairs to a scalar reward),

The learner agent interacts with a dynamic environment. In each time step, the agent selects an action in a given state, receives a reward as a result of the taken action, and transitions to a new state. The goal of the agent is to learn a mapping from states to actions called a *policy*, π: $S \rightarrow A$ that maximizes reward over time. The "reward over time" is considered as a discounted sum of rewards into an infinite future. The discount factor is γ, which $0 \leq \gamma < 1$. Using this factor, rewards received sooner will be more valuable than rewards received later. The value of each state, following policy π, is defined as:

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r_t$$

Where r_t is the reward received in time step t after starting in state s. The optimal policy, written π^* is the one that maximizes the value, V^{π} (s), for all states s. To learn the optimal policy its value function, V*, and its more specific correlate function (Q-function) is learned. Let Q* (s,a) be the value of selecting action a from state s, and thereafter following the optimal policy. This is expressed as:

$$Q^*(s,a) = R(s,a) + \gamma V^*(T(s,a)).$$

Then the optimal policy is selecting from each state the action with the highest expected future reward: $\pi^*(s) = \arg\max_a Q^*(s,a)$.

The Cora search engine is a domain-specific web crawler that uses reinforcement learning to learn to find computer science research papers from web sites of computer science departments.

4.2.3 Case Based Reasoning

Case Based Reasoning (CBR) is an approach to developing knowledge-based systems that are able to use past experiences to solve current similar problems [49]. In fact, it is a type of reasoning based on the reuse of past experiences called cases [50]. Cases are descriptions of experienced situations by learner agent. The agent acquires these experiences from interaction with its environment. In CBR two kinds of cases are stored in a case database, *successful cases* and *failure cases*. Successful cases should be reused as a basis to solve new problems that present a certain similarity [50]. Failure cases can prevent the agent from repeating similar actions that lead to unsuccessful results. Case Base Reasoning has been used in a variety of applications. As some examples we can name the use of CBR in document-retrieval systems, product-selection tasks in E-commerce environments, help-desk applications, diagnosis tasks in medicine and technical equipment, and system configuration and planning and design tasks [49]. Case Based BDI-Agent [43] is a domain-specific search engine that searches for the homepages of researchers with a particular interest. It uses CBR as its learning component to learn from past search results and reuse past cases to answer to similar queries in the future. It also uses a domain-specific knowledge base to enhance its search results.

4.3 Unsupervised learning

An unsupervised learner agent is neither told the correct action in each state, nor is an evaluation of action taken in each state. In other words, there is not any training phase. In the hypertext context, the learner is given a set of hypertext documents, and then is expected to discover a hierarchy according to the documents' similarities, and organize the documents along that hierarchy. This approach is also called clustering. A good clustering method will collect similar documents together in or near the leaves of a hierarchy, and dissimilar nodes will be joined near the root of the hierarchy. Clustering is used to enhance search, browsing, and visualization. In the next subsections we will review some clustering methods.

Some of the classical clustering methods like k-means and hierarchical agglomerative clustering are applied to hyperlink documents [7]. Usually documents are represented in vector space model; each word of document is an element of its vector in a model such as TFIDF. The similarity of two documents is calculated as the cosine of the angle between their corresponding vectors, or the distance between the vectors, provided their lengths are normalized.

4.3.1 K-means clustering

> "K-means is one of the basic techniques of clustering. The number of "k" in k-means clustering shows the desired number of clusters of document. Initially, the agent selects k number of seed documents arbitrarily. Each of seed document delegates a cluster. Thereafter each document is assigned to most similar cluster (seed document). The coordinate of the seed in the vector space is recomputed to be the centroid of all the documents assigned to that seed. This process is repeated until the seed coordinates stabilize. The process of document assigning continues until no document remained."
>
> Soumen Chakrabarti [7].

4.3.2 Agglomerative clustering

> "In agglomerative clustering that is a bottom-up clustering approach, documents are continually merged into super-documents or groups until only one group is left; the merge sequence generates a hierarchy based on similarity. We formalize the self-similarity of a group Γ as:
>
> $$s(\Gamma) = \frac{1}{|\Gamma|(|\Gamma|-1)} \sum_{d_1, d_2 \in \Gamma, d_1 \neq d_2} s(d_1, d_2)$$
>
> Where $s(d_1,d_2)$ is the similarity between documents d_1 and d_2 , often defined as the cosine of the angle between the vectors corresponding to these documents in TFIDF vector space. Many other measures of similarity have been used. The algorithm initially places each document into a group by itself."
>
> Soumen Chakrabarti [7].

5. Domain-Specific Search Engines

As the web continues to grow exponentially, the idea of crawling the entire web on a regular basis becomes less and less feasible [51]. General-purpose search engines have certain limitations that we pointed to previously. We can summarize the fundamental limitations as:

- *Querying Mechanism*: They accept the user query in the form of a set of *keywords*, and complex user needs may not be easily formulated in a Boolean combination of keywords. However, some search engines can get other implicit context information or even accept the user query in form of natural language.
- *Keyword Exact Matching*: They rank the results shown to the user according to the exact keyword matching method.
- *Low web Coverage Rate*: They cover only a portion of the publicly available documents in web. To see web coverage rate, see section 1. The highest coverage rate search engine hardly covers 40 percent of web documents.
- *Long result list with low relevancy to user query*: The results of search engines may not satisfy user query; and even when the relevant documents are be in the search engine database, and they may not be high enough in the result list for the user to see them.

Due to these basic problems and also because of the need to include information on specific domain, *domain-specific search engines* are proposed. A domain-specific search engine is defined as: "an information access system that allows access to all the information on the web that is relevant to a particular domain" [51]. We classify the search engines as shown in figure 9.

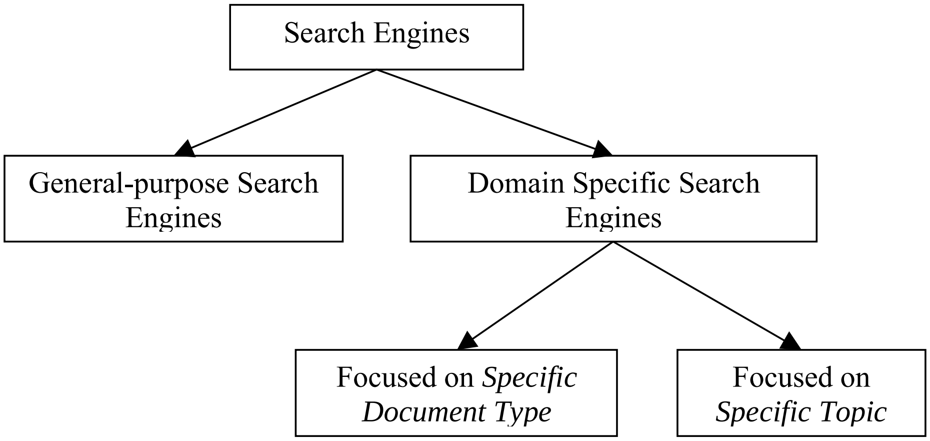
Search Engines
General-purpose Search Engines
Domain Specific Search Engines
Focused on *Specific Document Type*
Focused on *Specific Topic*

Figure 9. Classification of search engine systems

As shown is figure 9, we can classify search engines as general-purpose search engines and domain-specific search engines. *"Domain"* in domain-specific search engines can be specialized with two existing kinds of search engines, search engines which focus on *specific document type* such as resumes, homepages, movies, etc. And search engines focus on *specific topic* like computer science, climbing, sport, etc. Table 9 shows some example search from each category.

Search Engine Category	**Examples**
General-purpose search engines	AltaVista, Excite, HotBot, Lycos, Northern Light
Specific-document type search engine	Ahoy, Cora[*], Case-Based BDI Agent
Specific topic search engines	IBM Focused Crawler, Context Focused Crawler, Cora*

Table 9. Examples of different type of search engines

As we reviewed in section 2, a search engine or a web crawler has several parts: a *crawler* that traverses the web graph and downloads web documents; an *indexer* that processes and indexes the downloaded documents; and a *query manager* that handles the user query and returns relevant documents indexed in the database to the user. In this report we focus on the crawling mechanism of search engines. We focus on *focused crawling* (e.g., [37, 13, 24, 52]) that is a new crawling approach for domain-specific search engines.

While a general-purpose crawler visits the links of web in a breadth-first manner, a focused crawler reorders the links in the queue as to their predicted likelihood to lead to pages that are relevant to a particular topic [13]. Focused crawlers, like domain-specific search engines, can be divided into two categories: focused on specific *topic* or on a specific *document type*. If the goal of crawling is crawling pages related to "Agent-Based Systems", that is the task of topic crawlers. But, if the goal is to fetch all course pages in a university or over the world, then this is the task of focused crawling on document type. All course pages are not on some related topic

[*] Since Cora finds computer science research papers, it can be categorized in document specific (research paper finder) and also specific topic (computer science literature) search engine

like “Agent-Based Systems”, but they are of the same type. This kind of crawler would also crawl on any specific types like resume, homepage, call-for-paper, FAQ, product, movie, papers, etc. There are some search engines in this category. Ahoy[21][53] is a homepage search service based on a crawler specially tuned to locate homepages.

Focused crawling is a new approach to topic-specific search engines, introduced by Soumen Chakrabarti et al. [13]. They describe their crawler as follows:

> “A focused crawler seeks, acquires, indexes, and maintains pages on a specific set of topics that represent a relatively narrow segment of the web. Thus, a distributed team of focused crawlers, each specializing in one or a few topics, can manage the entire content of the web. Rather than collecting and indexing all accessible web documents 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. Focused crawlers selectively seek out pages that are relevant to a pre-defined set of topics. These pages will result in a personalized web within the World Wide Web. Topics are specified to the console of the focus system using exemplary documents and pages (instead of keywords). Such a way of functioning results in significant savings in hardware and network resources, and yet achieves respectable coverage at a rapid rate, simply because there is relatively little to do. Each focused crawler is far more nimble in detecting changes to pages within its focus than a crawler that crawl the entire web.
>
> The ideal focused crawler retrieves the maximal set of relevant pages while simultaneously traversing the minimal number of irrelevant documents on the web. Focused crawlers therefore offer a potential solution to the currency problem by allowing for standard exhaustive crawls to be supplemented by focused crawls for categories where content changes quickly.”
>
> Chakrabarti et al. [13].

5.1 Topic-Specific Focused Crawlers

In this report, we concentrate on topic-specific focused crawlers and in the rest of this report “focused crawler” will refer to this kind. We can recognize three basic works in focused crawling. As stated earlier, Soumen Chakrabarti et al. [13, 14] presents the focused crawler as a new approach to topic-specific web resource discovery. This work was also done largely at IBM’s Almaden research center. Cora [23, 24, 25] is a computer science research paper search engine that uses reinforcement learning to guide its focused crawler. Rennie and McCallum at JustResearch and Carnegie Mellon University originally initiated this work. Another approach to focused crawling uses the context graph [52] to build a model of context within which topically relevant pages occur on the web. In the next subsections we survey new approaches to focused crawling, then we review the three basic focused crawlers named above, and finally we appraise these three approaches in the last subsection.

5.2 Recent researches on Focused Crawling

One of the first focused crawlers was the Fish system [54]. Fish is a client-based real-time information retrieval system. It crawls the hypertext documents on the web and uses “depth-first search” method to follow links within web documents. This search engine adopts the “Fish School” assumption and assumes that relevant documents to a topic should be near each other in link structure. Fish follows more

[21] http://www.cs.washington.edu/research/ahoy

links from pages that are relevant to the topic of search. It identifies relevant documents based on keywords and regular expressions.

Another approach to focused crawling [37] uses the "importance of web pages" parameter to reorder links in its queue. Various heuristic measures for identifying page importance are tried, such as similarity to a driving query, number of pages pointing to this page (*backlinks*), page-rank and location (in a hierarchy). We described the page-rank approach in the earlier section on web structure analysis.

Focused crawling as a new approach to topic-specific resource discovery is introduced in [13]. The topic under focus is represented as a set of example documents instead of a set of keywords. This system has three main components, a classifier, distiller and crawler. The classifier makes relevance judgments on pages to decide on link expansion and the distiller determines centrality of pages to determine visit priorities. The latter is based on connectivity analysis and uses a variation of the HITS algorithm [16]. Soumen Chakrabarti proposed an improved version of this approach in [55, 56]. They added another classifier to this system, called *Apprentice*. The task of the new classifier is assigning priority to unvisited URLs while the old classifier provides templates for the new classifier. Apprentice uses HTML tags and the DOM[22] (Document Object Model) tree, as well as user feedback to learn classification.

A site mapping application which uses focused crawling is described in [57]. The crawler starts in a certain point and traverses the web graph toward relevant documents. This approach uses a vector space model to compute the similarity of documents to the focus topic. Next-level pages inherit a discounted score of this original relevancy. Anchor text and URL text is also used to judge the priority of links to follow within a relevant page. In fact, this approach is an improvement on the Fish system, described above.

Cora [23, 24, 25] is a computer science search engine that uses reinforcement learning to guide a focused crawler. This crawler has two phases, training and testing. In the training phase it learns a mapping from text in a neighborhood of URLs to a scalar value. This value of a URL is some discounted reward that it receives by following that URL. Rewards are on-topic documents that are accessible by following that URL, one or more links away. In the test or work phase, it assigns a scalar value to each unvisited URL from its neighborhood text. This value is the estimated sum of rewards that are obtainable from that URL. The crawler picks the URLs from a priority queue in which URLs are ordered according to their estimated sum of future discounted rewards. One strength of this approach is that it considers the future reward of links in their crawling priority, so the likelihood of crawling a link within an off-topic document that may lead to reasonable amount of on-topic documents is high. We will describe this approach in more detail in the next section.

Another kind of focused crawler considers the *hidden* aspects of the web [58]. The goal of this crawler is to access the huge amount of hidden information behind web search forms. They propose a task-specific human-assisted approach in a system called HiWE (Hidden Web Explorer). They model forms as a set of (element, domain) pairs and try to determine suitable input values based on labels from form layout. The crawler seems to perform better on larger forms (more descriptive labels and finite domains).

The use of a context graph to guide a focused crawler is described in [52]. Researchers of this approach state that assigning proper credit to a link during the

[22] http://www.w3.org/DOM/

crawl is the important problem of focused crawling. For example, some off-topic pages may lead to on-topic pages, some levels deep from the start page. To address this problem a context-focused crawler was proposed, which uses general-purpose search engines to find the back-links of pages, and uses these to construct a context graph for each page. Then this set is used to train a set of classifiers to assign documents to several classes according to their expected distance from on-topic documents. Graphs and classifiers are constructed for each seed document up to particular layer, showing the expected distance to target pages. A Naïve Bayes classifier is used for each layer. The links with lower distance to target documents are tried first. We review this kind of focused crawler in the next sections.

Web Topic Management System (WTMS) is another approach to focused crawling that is described in [59]. They proposed a crawler that only downloads pages very near (parent, child, and sibling) to relevant documents. Documents in this system are represented in a vector model space and their relevancy to the focused topic is computed by techniques in this model. This crawler also follows URLs that include keywords of focused topic. It sets a threshold for each branch of the web and stops following URLs of this branch if the relevance score of the area falls below the threshold.

An evaluation of focused crawlers is reported in [60]. Initially a set of classifiers for 100 topics was built to be used in the evaluation of the crawled documents. The researchers in this approach believe that a good focused crawler should remain in the vicinity of the topic. They evaluated the crawlers based on ability to remain on-topic during the crawling session. Three different strategies of focused crawling are evaluated:

- *Best-first search*: Uses a priority queue and orders the links according to similarity between focus topic and page where they were found.
- *Page-rank*: Orders the page according to their page-rank score. This method is described in web structure analysis section (3.2.1).
- *Infospider*: Uses a back-propagation neural network and learns text around the links.

The results of this research show that Best-First-search out-performs the others; Infospider is in second place and finally page-rank has the lowest efficiency among them. They conclude that the page-rank method is too general to be very useful in topic-specific search.

A focused crawler that tries to learn the link structure of the web is detailed in [61]. This approach tries to find some features in a page that make it more likely that its links lead to on-topic pages. For example, page content, URL structure of page, link structure of web or a combination of them can be considered as features. The researchers of this approach claim that learning the link structure of the web is a more general framework than assuming a predefined structure for it. They learn link structure with a statistical model. The result of this research shows that the combined features model is more efficient than using each of them in isolation. This approach is also robust against different starting points on web.

5.3 IBM Focused Crawler

Focused crawling [13, 14] as a new approach to topic specific resource discovery was proposed by Soumen Chakrabati and some of his colleagues when he was at IBM's Almaden center. The focus topic in this system is represented by a set of example pages that is provided by a user to the system. In the system described there is a user-browsable topic taxonomy where the user can mark some of the documents as *good* and select them as the focus topic.

The system has three main components: A *classifier* that makes judgments on the relevancy of crawled documents, and decides on following the links within pages. The classifier is an extended version of the Naïve Bayes classifier. The second component is a *distiller* that evaluates the centrality of each page to determine crawling priority of links within it. The distiller uses the bibliometric concepts of *hub* and *authority* pages as an approximate social judgment of web page quality. For each page it calculates the hub and authority scores of each web page with an extended version of the HITS algorithm [16]. It tries to crawl the links within pages with the highest hub score first, in hopes of finding new authorities first. The third component of the system is a dynamic *crawler* that crawls the web according to a re-orderable priority queue.

The system works in two phases: training and testing. In the training phase, the classifier is trained with some labeled data relevant to the focus topic. The training data set is acquired from existing taxonomy-like search engines (portal) such as Yahoo! and Open Directory Project. Figure 10 shows the architecture of this system.

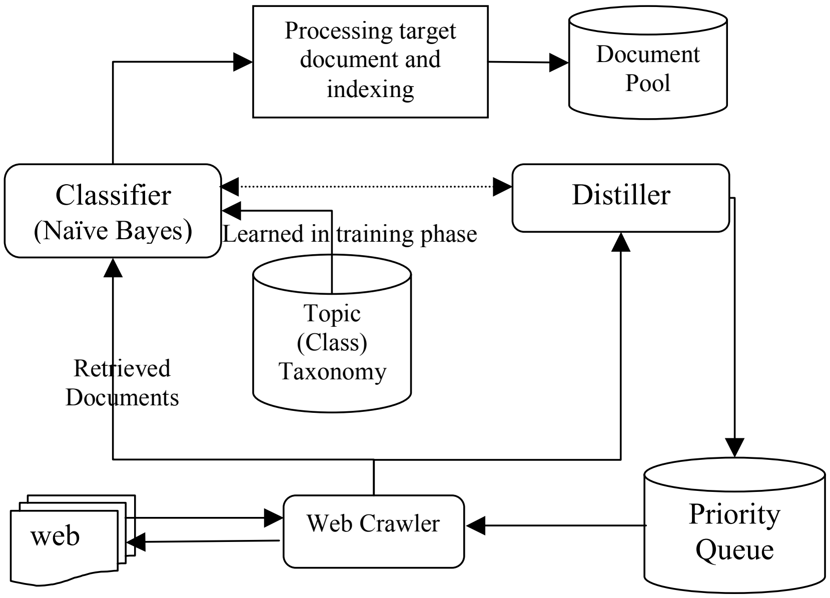

Figure 10. Architecture of IBM focused crawler, showing how classifier, distiller and crawler collaborate with each other.

5.4 Cora Domain Specific Search Engine

The Cora search engine [48, 49, 50] automatically spiders, classifies and extracts computer science research papers from the web. The papers in CORA are organized into a taxonomy with 75 leaves, and various fields such as author and title are extracted from each paper. Additionally, bibliographic information is extracted from each paper, allowing bibliometric analysis to be performed. The creation and maintenance of Cora relies heavily on artificial intelligence and machine learning techniques. The tasks can be broken down into four components: spidering (crawling), extraction, reference matching and classification.

In this report we concentrate on the crawling methodology of Cora. We can divide the spidering task into two phases: Training and Testing. Figures 11 through 15 show the workflow of the Cora spider in these two phases.

> "One strength of reinforcement learning is that it provides a formalism for measuring the utility of actions that give no immediate benefit, but give benefit in the future. Reinforcement learning agents presents this delayed benefit by learning a mapping from each available action to a scalar value (Q-Value) indicating the sum of future discounted rewards expected from executing that action. The "discount" makes later rewards less valuable than sooner rewards, thus encouraging efficiency.
> In the spidering task, the on-topic documents are immediate rewards. The actions are following a particular hyperlink. The state is the bit vector indicating which on-topic documents remain to be consumed, and which actions have been discovered. The state does not include the current "position" (last page visited) of the agent since a crawler can jump to any known URL next. The number of available actions is large and dynamic, in that it depends on which documents the spider has visited so far. The key feature of topic-specific spidering that make reinforcement learning the proper framework for defining the optimal solution is: (1) performance is to be measured in terms of rewards over time, and (2) the environment presents situations with delayed reward."
>
> Rennie and McCallum [24]

The training phase involves two tasks: (1) assigning appropriate Q values to each hyperlink in the training set, and (2) learning a mapping from text to Q values using the training data. Target documents or rewards are computer science research papers on the web pages of computer science departments. As it is shown in Figure 11, assigning a Q-value to each URL is done according to a sum of the discounted number of rewards that are obtainable from a URL in the future. A more immediate strategy does not consider the future rewards in Q-value computation. Mapping from text to Q value is done by casting this regression problem as classification [62], discretizing hyperlink Q values and training a naive Bayes classifier on the corresponding neighborhood text. Figure 12 shows the pseudo code of mapping text to Q-value in the training phase. Figure 13 shows the pseudo code of mapping from neighborhood text of URLs to Q-Value in training data.

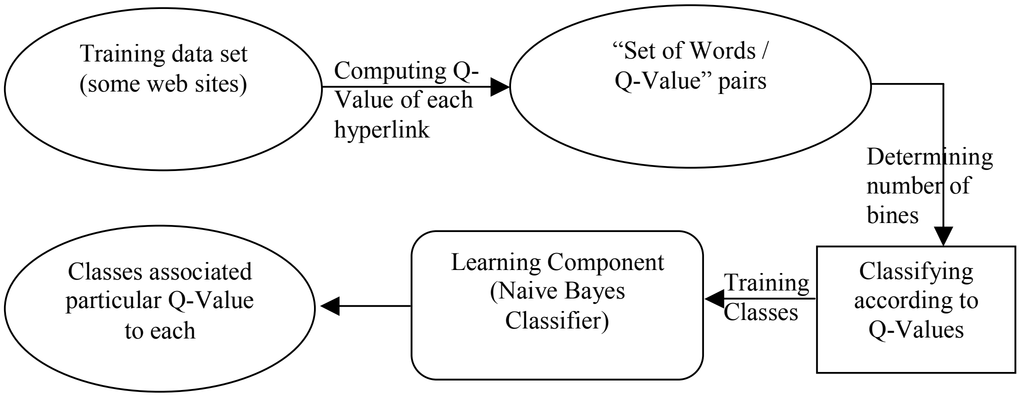
Training data set (some web sites)
Computing Q-Value of each hyperlink
"Set of Words / Q-Value" pairs
Determining number of bines
Classifying according to Q-Values
Training Classes
Learning Component (Naive Bayes Classifier)
Classes associated particular Q-Value to each

Figure 11. Workflow of training phase of Cora spider

```
For each Hyperlink do
  If crawling strategy is *Immediate Reward*,
     If Hyperlink directly leads to a target document, Q-Value = 1
       else Q-Value = 0.
  If crawling strategy is *Future Reward*,
     Q-Value = $\Sigma_t \gamma^t R_t$ , $0 \leq \gamma < 1$          (Sum of future discounted rewards)
     R = Number of rewards in step t from Hyperlink,
```

Figure 12. Pseudo code of assigning Q-Value to each URL in training data

```
For each Hyperlink/Q-Value pair do
   Place the neighborhood text of the Hyperlink into the bin corresponding to
   its Q-Value

Train Naïve Bays classifier with texts in bins as training data, every bin has a
specific Q-Value range
```

Figure 13. Learning a mapping from neighborhood text of URLs to Q-Value

In the testing (working) phase, the spider crawls the web and tries to follow the links with higher estimated reward first. Figure 14 shows the workflow of the working phase of the system. The crawler starts from the homepages of computer science departments and looks for research papers as target documents. Figure 15 shows the pseudo code of the system in working phase.

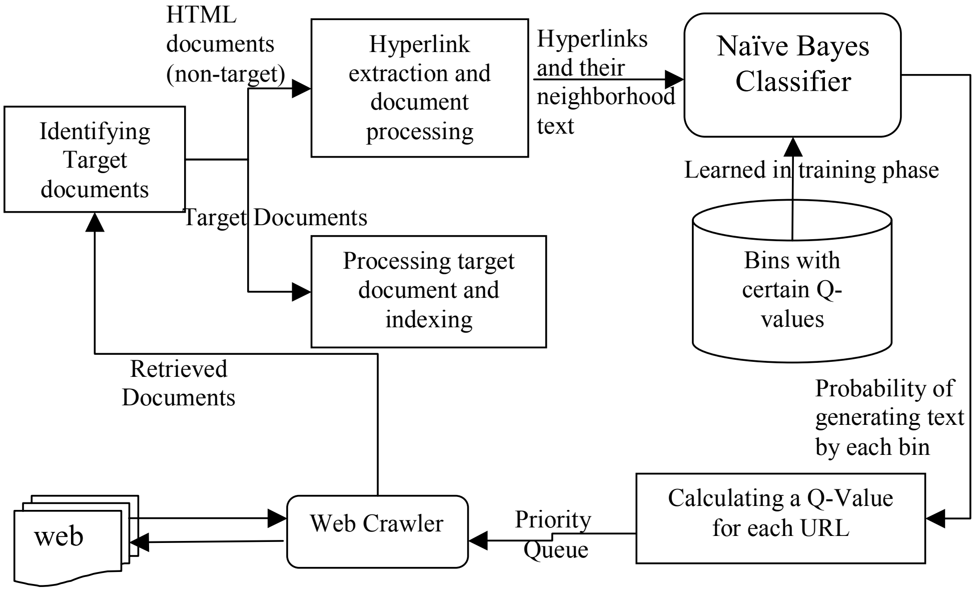
HTML documents (non-target)
Hyperlink extraction and document processing
Hyperlinks and their neighborhood text
Naïve Bayes Classifier
Identifying Target documents
Learned in training phase
Target Documents
Processing target document and indexing
Bins with certain Q-values
Retrieved Documents
Probability of generating text by each bin
web
Web Crawler
Priority Queue
Calculating a Q-Value for each URL

Figure 14. Workflow of Cora spider in working phase

```
Download the document of the Hyperlink with highest Q-Value,
    If the document is a target document:
        Store it in document pool to be processed and indexed later.
    Else:
              - Extract the page's URLs and send its anchor and its
                 neighborhood text to Naïve Bays classifier
              - Having the probability of generating the URL's text and its
                 neighborhood text with each class from Naïve Bays
classifier,
                 calculate Q-Value of URL according to:
                         $Q\text{-}Value\ (Hyperlink) = P_{url}(b_i) * Q(b_i)$, where
                     $P(b_i)$ = Probability of generating the URL and its associated text
                      with bin i, produced by Naïve Bays classifier.
                     $Q(b_i)$ = Associated Q-Value of bin i.
              - Insert (URL, Q-Value) pair in priority queue to be crawled
                 according to its Q-Value
```

Figure 15. Pseudo code of Cora spider working phase

Experiments show that this directed spider is three times more efficient than a spider based on breadth-first search, and also more efficient than other smart spiders that do not explicitly model future reward [24].

5.5 Context Focused Crawler

Context focused crawler [52] is an approach to focused crawling that tries to build a context graph for each web page (graph which shows the pages with link to that page in web link structure) and guess the distance of page to the target pages, then crawl the web pages with near distance to target pages first. The researchers of this approach believe that:

> "The major problem in focused crawling is performing appropriate credit assignment to different documents along a crawl path, such that short-term gains are not pursued at the expense of less-obvious crawl paths that ultimately yield larger sets of valuable pages. . .To address this problem we present a focused crawling algorithm that builds a model for the context within which topically relevant pages occur on the web. The context model can capture typical link hierarchies within which valuable pages occur, as well as model content on documents that frequently co-occur with relevant pages. This algorithm further leverages the existing capability of large search engines to provide partial reverse crawling capabilities. Algorithm shows significant performance improvements in crawling efficiency over standard focused crawling."
>
> Diligenti et al. [52]

We can recognize the work process of this focused crawler in two phases: training and testing. In the training phase a user should provide some seed documents relevant to topic. For each seed document all of its backlinks are gathered using a general-purpose search engine up to a certain level. Each gathered page is assigned to a specific layer according to its distance to seed (target) page. So each layer includes documents that are within a specific distance to target documents. Seed documents lay in layer 0. Then a set of Naïve Bayes classifiers is trained with documents in each layer. Every classifier learns documents in a specific layer.

In the testing phase, a new downloaded page is classified using the classifier and, according to its estimated distance to target documents, is assigned to a queue corresponding to each layer. The crawler crawls the URLs from queues with smaller numbers before others. Figure 16 shows the architecture of this crawler and Figure 17 and 18 show pseudo code for its training and testing phase, respectively.

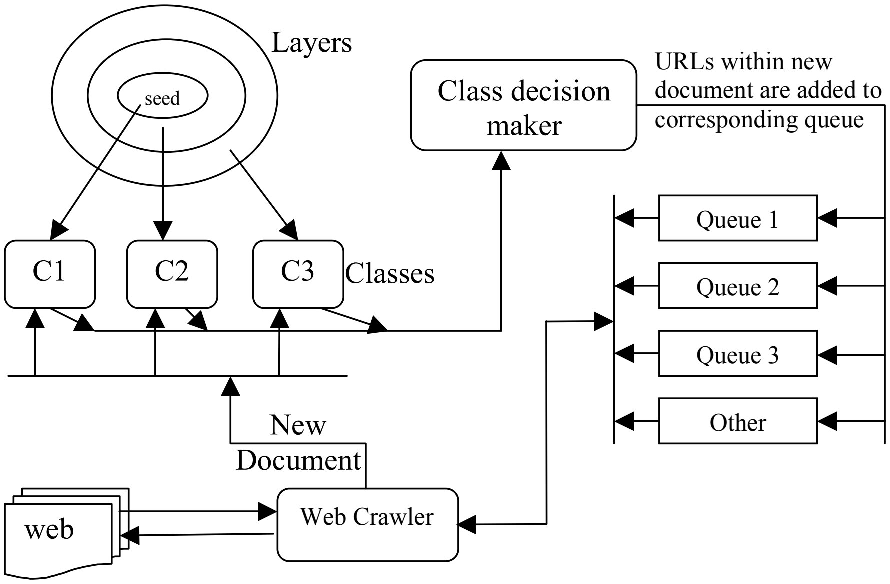

Figure 16. Architecture of *Context Focused Crawler* (From [52]).

```
For each document in seed pages do:
    -  Construct its context graph up to a certain level
    -  Assign each document to a particular layer according to its distance to seed
       document

Train a Naïve Bayes classifier for each layer using documents in that layer
```

Figure 17. Pseudo code of training phase of context focused crawler

```
Download documents of URLs in the smallest numbered queue ( If it is empty
jump to next small numbered queue)

For each new document do:
    -  Classify document with trained Naïve Bayes classifier to estimates its
       distance to target documents
    -  If it is a target document (classified as class 1) store it to be processed later
       Else
           o  Assign URLs within new document to a particular queue according
              to its estimated distance to target document
           o  Assign so far estimated documents to "Other" layer
```

Figure 18. Pseudo code of testing phase of context focused crawler

5.6 Appraisal of three basic Focused Crawlers

The purpose of this section is to appraise three basic approaches to focused crawling. These approaches are Cora, IBM Focused Crawler and Context Focused Crawler that we detailed in the previous sections. Table 10 shows the fundamental characteristics of these crawlers in comparison with each other.

	Type of Input	Training Data	Learns What?	Target Documents	Identifying target documents method
IBM Focused Crawler	Exemplary documents	Classified relevant documents	Identifying target documents	on-topic HTML files	Uses trained Naïve Bayes classifier
Cora	Some web site	Some web site	A mapping from neighborhood text of URLs to a scalar value (Q-Value)	Computer science research paper files (.ps, .ps.z, .ps.gz)	A hand-coded algorithm identifies with more than 95% accuracy
Context Focused Crawler	Seed documents	Pages in Context Graph of seed documents	Estimating distance of current page to target pages	on-topic HTML files	Uses trained Naïve Bayes classifier

Table 10. Characteristics of three basic approaches of focused crawling

Each of these approaches has some strengths and some major problems. In the next subsections, we highlight their abilities and disabilities in more detail.

5.6.1 IBM Focused Crawler

We summarize the problems of this focused crawler as:

- ***Fixed model of classifier***, IBM focused crawler uses a fixed model of relevancy class as a classifier to evaluate topical relevancy of documents. A more adaptive classifier uses documents that are marked as relevant by the classifier to update the classifier. However, ensuring flexibility in the classifier without simultaneously corrupting the classifier is difficult [52].
- ***Does not model future rewards***. One major problem faced by this focused crawler is that is does not learn that some sets of off-topic documents often lead reliably to highly relevant documents [52]. In other words it does not model the future reward of links. For example, a home page of a computer science department is not relevant to "Reinforcement Learning", but links from that home page eventually lead to the home page of the "Machine Learning Laboratory" or the home page of a researcher where it may find valuable target pages.
- ***Lack of comparison of results***. The results reported in papers of this approach are not compared with results of human-maintained portals like the Open Directory Project, so judgments on the quality of gathered pages in this approach is hard.
- ***Exemplary documents***. Representation of focus topic in the form of some high quality documents related to topic is sometimes hard for the user.

5.6.2 Cora Focused Crawler

We summarize the problems of this focused crawler as:

- ***Slow initialization***. The main purpose of using reinforcement learning in this crawler is to learn a set of off-topic documents that leads to on-topic documents in next levels. But in order to learn this set of documents, Cora needs to repeatedly crawl a substantial part of the target web sites during the learning phase [52].
- ***Specifying representative web sites***. "The web site or server on which the document appears is repeatedly crawled to learn how to construct optimized paths to the target documents. This approach places a burden on the user to specify representative web sites." [52].
- ***Difficulty with target pages in sites other than the start web site***. This approach only finds the target pages in the start web site and leaves the target pages in other web sites [52].
- ***Unable to find documents further than 4 levels from the start page.*** Reported results show that it is able to find target documents up to 3-4 hops beyond the current page. Since it uses words around URL to do a mapping to a scalar, the set of neighborhood words of URLs farther than 4 jumps is large and too general to learn.

5.6.3 Context Focused Crawler

We summarize the problems of this focused crawler as:

- ***Requirement for reverse links***. "The major limitation of this approach is the requirement for reverse links to exist at a known search engine for a reasonable fraction of the seed set documents." [52].
- ***Limited to 2-3 layers for efficient crawling***. It learns the documents with particular distance to target pages using a set of Naïve Bayes classifiers. Learning of documents further than 2 or 3 jumps is difficult because of the heterogeneity and variety of the web pages found at those distances from the target documents.

6 ALII: Information Integration Environment based on Active Logic framework.

Active Logic is a kind of "step logic," which was developed in [63] as formal mechanism for modeling the ongoing process of reasoning. Unlike traditional logical formalisms, a step-logic does not calculate a final set of conclusions which can be drawn from an initial set of facts, but rather monitors the ever-changing set of conclusions as time goes on. There are special persistence rules so that every theorem α present at time t implies itself at time t+1; likewise there are special rules so that if the knowledge base contains both a theorem α and its negation –α, these theorems and their consequences are "distrusted" so they are neither carried forward themselves nor used in further inference. An active logic, then, consists of a formal language (typically first-order) and inference rules, such that the application of a rule depends not only on what formulas have (or have not) been proven so far (this is also true of static logics) but also on what formulas are in the "current" belief set. In general the current beliefs are only a subset of all formulas proven so far: each is believed when first proven but some may subsequently have been rejected. Active Logics have the following characteristics: they are *situated in time, maintain a history, tolerate contradictions, and allow meta-reasoning to be done.*

Based on the above definition, active logics are a family of inference engines that incorporate a history of their own reasoning as they run. Thus at any time ***t***, an active logic has a record of its reasoning at all times prior to ***t***, and it also knows that the current time is ***t***. As it continues to reason from time ***t***, that reasoning is also recorded in the history, marked at time *t+1* as having occurred at time ***t***. Thus an active logic records the passage of time in discrete steps, and the "current" time slides forward as the system runs. It is convenient to regard its current inferences as occurring in a working memory that is then transferred to the history (or long-term memory) in the next time-step. Thus, an active logic has time-sensitive inference rules and consequently time-sensitive inferences. In active logics the current time is itself noted in the working memory—`Now (t)`—and this changes to `Now (t+1)` one step later. (A time-step should be thought of as very fast, perhaps 0.1 sec in correspondence with performance of elementary cognitive tasks by humans). Thus active logics "ground" *now* in terms of real time-passage during reasoning.

These characteristics make active logics suitable for use in various domains including time situated planning and execution [66]; reasoning about other agents [67]; reasoning about dialog [68, 69], including updating and using discourse context [70]; and autonomous agency [71].

There are many examples of active logics in various papers. We present here a couple of simple rules.

1) Time step update rule: `t: Now(t), then: t+1:Now(t+1)`

is a rule that says: if at the current step, Now has the value t, then, at the next step, let Now have the value (t + 1). This enables the active logic to keep track of step numbers and therefore of time. This is a basic rule and is included in all active logics.

2) Another example is the contradiction rule:

```
t:P, not(P), then: t+1:contra(P, not(P))
```

If at a step, we have both P and not(P) present in the database, at the next step, we add contra(P, not (P)) to the database to indicate the contradiction. There will be other rules that will cause the consequences of P and not(P) not to be derived in later steps, and rules that will attempt to resolve the contradiction and reinstate either P or not(P) to the database at a later time.

3) We can also have modus ponens: `t:P, P→Q, then conclude:t+1:Q.`

This says: if at time t, the database contains P and (P → Q), then in the next time step, conclude Q. Note that if the database contains P, (P → Q) and (Q → R), we do not get R immediately, but only after 2 steps. First, we use P and (P → Q) to obtain Q, then in the second step, we use this together with (Q → R) to derive R.

4) The inheritance rule keeps formulas in the database unless there is a contradiction:

```
    t:P,not_know(not(P)),\+ P = Now(t),then conclude:
t + 1: P.
```

not_know(P) is true iff P is not in the current database. Since the database is finite, this poses no computational problems. "\+ P = Now(t)" verifies that P is not of the form Now(t) and prevents time from being inherited. This rule also prevents the lemmas of a contradiction from being inherited.

5) Let the sentences initially present in the database be: Now (0), Bird (tweety), Bird (x) & not_know (not (fly (x))) → fly (x). With the above rules of inference, this is what the database looks like at consecutive steps:

At step 0: Now(0), Bird(tweety), Bird(x) & not_know(not(fly(x))) → fly(x)

At step 1: Now(1), Bird(tweety), Bird(x) & not_know(not(fly(x))) → fly(x), fly(tweety)

since “not(fly(tweety))” is not present in the database at step 0.

The database will not change thereafter.

6.1 Why Active Logic for Focused crawling?

As more information becomes available on the World Wide Web, it becomes more difficult to provide effective tools for accessing this information. Today, web users access the web through two dominant interfaces: clicking on hyperlinks (browsing) and searching via keyword queries (crawling).

Users have two main tools to help them locate relevant resources on the web: Catalogs and Search Engines. Human experts construct catalogs. They tend to be highly accurate but can be difficult to maintain as the web grows. To keep up with this growth search engines were designed to eliminate human effort in cataloging web sites. A search engine consists of a mechanism that “crawls” the web looking for new or changed pages, an indexing mechanism and a query interface. Users generally query against the system index, although many contemporary search engines now also use link analysis to some degree. However, link analysis only helps to identify the most popular pages, and popularity may or may not correlate to relevance for a particular query.

As we have seen there are numerous attempts to improve the search engine. One of the major efforts in this regard is in improving web Crawlers, also known as robots, spiders, worms, walkers, and wanderers, described in detail in sections 2 and 5.

Two critical factors for the design, implementation and maintenance of a Focused Crawler are: conceptual modeling of the domain, and reasoning support over the conceptual representation. Knowledge representation and reasoning techniques play an important role for both of these factors. By using an Active Logic based framework in Focused Crawler architecture, we will be able to create an engine to implement these roles well.

As is mentioned briefly in the last section, the focused crawler has three main components: a classifier, distiller, and a crawler. Presently focused crawlers tend to use probabilistic reasoning in their components. Detailed description of these components has been given in section 5.

A major problem faced by existing crawlers is that it is frequently difficult to learn that some sets of off-topic documents often lead reliably to highly relevant documents. This deficiency causes a problem in traversing the hierarchical page layouts that commonly occur on the web. Consider, for example, a researcher looking for papers on ‘Natural Language Processing’. A large number of these papers are found on the home page of researchers at computer science department at universities. When a home page finds the home page of a university, a good strategy would be to follow the path to computer science (CS) department, then to the researcher’s page, even though the university and CS department pages in general would have low relevancy scores. An adaptive focused crawler based on active logic could in principle learn this strategy by building a tree from query and expanding this tree while the system was in process.

ALII [72] utilizes a compact context representation and constructs a hierarchy model of query and web pages. The crawler based on active logic also utilizes the limited keyword crawling possible using general search engine indices efficiently focus-crawl the web.

There are three distinct stages to using active logic when performing a focused crawl session:

1. An initialization phase when a query present to a search engine. In this phase the initial tree will be constructed.
2. A crawling phase that extracts the pages from web sites. In this stage the associated trees are constructed for each of the seed pages.
3. A process phase that evaluates each page tree within the query tree. In this phase the query tree will improved from time (t) to time (t+1).

In implementing this idea for specific query, we will use above steps. The results will show how the query changes from time ‘*t*’ to ‘*t+n*’ and lead to the expectation of related responses from focused crawler. In this implementation we will first have to build hierarchy trees from the query and focused crawler responses at time ‘*t*’. By following the links on responses from focused crawler, the system will consequently improve both the query tree and the result tree. The trees’ improvement in time ‘t+1’ is based on active logic procedures combined with the focused crawler architecture.

The query tree processing considers the weight of each node representing in the result tree from the focused crawler result. The associated weight to each node is calculated based on node relevancy to focused topic or query [72]. Based on weight, the tree may be pruned in one step, instead of expanded. Such an optimized tree presents more adequate and related results to the query.

The following characteristics of Active Logic are considered in this phase.

Ignorance-assessment amounts a lookup at time “t” of what was known prior to t.

Contradictory information can (sometimes) be detected and used to curtail nonsensical inferences as well as to initiate repairs.

Defaults can be characterized in terms of lookups to see whether the result page is (directly) contrary to the default knowledge regarding the specified topic.

Reasoning can be kept current, i.e., inferences can be triggered to occur when they are needed. From these characteristics, an environment based on active logic will be situated in time, will maintain a history of its own reasoning, will tolerate contradictions, and will enable meta-reasoning.

All outputs of the process phase can be used as knowledge for input into the classifier component. The ALII [72] project aims at developing methods and tools for a Model-based, Semantic Integrating of information sources on web pages. Work in progress is centered on a demonstrator application for information services. ALII is a representation environment and logical reasoning tool with a formal foundation in Active Logic. ALII can seek (through the links), acquire, index and maintain pages that are relevant to a predefined set of topics and effectively build high quality collections of web documents.

7 Conclusion

Search engine technology has gone through several evolutions and finally reached the point where Artificial Intelligence can offer tremendous help. We have reviewed this evolution from the beginning up to now and surveyed several different techniques that have been developed to improve search engine functionality. In particular we highlighted some machine learning approaches to information retrieval on the web and concentrated on topic-specific search engines. Finally, we proposed an information integration environment based on active logic. Our approach uses current technology in a good manner to provide better results.

8 References

[1] NetNames Statistics , http://www.netnames.com , 12/28/1999

[2] John Gantz and Carol Glasheen, *The Global Market Forecast for Internet Usage and Commerce: Based on Internet Commerce Market Model™*, IDC (International Data Corporation) Report, Version 5, 1999. *http://www.idc.com*

[3] Internet Fact and State, http://optistreams.com/factsandstats15.htm

[4] The Censorware Project, http://www.censorware.org/web_size, Jan. 26, 1999

[5] S. Lawrence and C.L. Giles, *Searching the World Wide Web*, *Science* 80:98-100, 1998.

[6] S. Lawrence and C.L. Giles, *Accessibility of Information on the Web*, Nature 400:107-109, 1999.

[7] S. Chakrabarti, *Data mining for hypertext: A tutorial survey*, SIGKDD: SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM 1(2): 1-11, 2000.

[8] L. Page, S. Brin, *The anatomy of a large-scale hypertext web search engine*, Proceeding of the seventh International World Wide Web Conference, 1998.

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

[10] S. Lawrence, *Context in web Search*, IEEE Data Engineering Bulletin, Volume 23, Number 3, pp. 25-32, 2000

[11] Media Metrix, *Media Metrix announces top 25 digital media/web properties and sites for January 1999*, 1999.

[12] Graphics, Visualization, and Usability center, Georgia Institute of technology, *GVU's WWW User Surveys*, http://www.gvu.gatech.edu/user_surveys/

[13] S. Chakrabarti, M. van der Berg, and B. Dom, *Focused crawling: a new approach to topic-specific web resource discovery*, Proceeding of the 8th International World Wide Web Conference (WWW8), 1999.

[14] S. Chakrabarti, M. H. van den Berg, B. E. Dom, *Distributed Hypertext Resource Discovery Through Examples*, Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[15] M. Kobayashi and K. Takeda, *Information retrieval on the web*, IBM Research Report, RT0347, April 2000.

[16] J. Kleinberg, *Authoritative sources in a hyperlinked environment*. Proceeding of 9th ACM-SIAM Symposium on Discrete Algorithms, 1998. Extended version in Journal of the ACM 46, 1999.

[17] M. Jaczynski, B. Trousse, *Broadway: A Case-Based System for Cooperative Information Browsing on the World-Wide-web*, Collaboration between Human and Artificial Societies, pp. 264-283, 1999.

[18] Q. Yang, H. F. Wang, J. R. Wen, G. Zhang, Y. Lu, K. F. Lee and H. J. Zhang, *Toward a Next Generation Search engine*, Proceedings of the Sixth Pacific Rim Artifact Artificial Intelligence Conference, Melborne, Australia, 2000.

[19] LookOff E-book, *Engine Basics*, http://www.lookoff.com/tactics/engines.php3 , *Oct 24 2000.*

[20] D. Botluk, *Search Engines Comparison*, 2001, LLRX Company, available at: http://www.llrx.com/features/engine2001.htm

[21] The web Robots Pages. http://info.webcrawler.com/mak/projects/robots/robots.html

[22] Internet Growth and Statistics: Credits and Background. http//:www.mit.edu/people/mkgray/net/background.html

[23] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, *Automating the construction of internet portals with machine learning*, Information Retrieval Journal, 1999.

[24] J. Rennie and A. McCallum, *Using reinforcement learning to spider the web efficiently*, Proceeding of International Conference on Machine Learning (ICML), 1999.

[25] A. McCallum and K. Nigam, J.Rennie, and K. Seymore, *Building domain-specific search engines with machine learning techniques*, AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace, 1999.

[26] M. Kobayashi and K. Takeda, *Information Retrieval on the web: Selected Topics*, IBM Research, Tokyo Research Laboratory, IBM Japan, 1999.

[27] V. I. Levenshtein, *Binary codes capable of correcting spurious insertions and deletions of ones (original in Russian)*, Russian Problemy Peredachi Informatsii 1, pp. 12–25, 1965.

[28] P. Yianilos, *The LikeIt intelligent string comparison facility*, NEC Institute Tech Report 97-093, 1997.

[29] P. Yianilos, *Data structures and algorithms for nearest neighbor search in general ametric spaces*, In Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms, pp. 311–321, 1993.

[30] G. Salton, C. Yang, *On the specification of term values in automatic indexing*, Journal of Documentation 29, pp. 351–372, 1973.

[31] M.F. Porter, A*n algorithm for suffix stripping*, Program 14(3):130–137, 1980.

[32] R. Dolin, J. Pierre, M. Butler, and R. Avedon, *Practical evaluation of IR within automated classification system*, Eighth International Conference of Information and knowledge Management, 1999.

[33] G. Salton, *Automatic indexing using bibliographic citations*, Journal of Documentation 27, pp. 98–110, 1971.

[34] E. Spertus, *ParaSite: Mining structural information on the web*. Proceeding of The Sixth International World Wide web Conference, 1997.

[35] K.D. Bollacker, S. Lawrence, and C. Lee Giles, *CiteSeer: An Autonomous web Agent for Automatic Retrieval and Identification of Interesting Publications*, 2nd International ACM Conference on Autonomous Agents, pp. 116-123, 1998.

[36] S. Brin, R. Motvani, L. Page, T. Winograd, *What can you do with the web in your packet*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998.

[37] J. Cho, H. Garcia-Molina, L. Page, *Efficient Crawling through URL ordering*, Seventh International web Conference (WWW), 1998.

[38] K. Bharat and M. Henzinger, *Improved algorithms for topic distillation in a hyperlinked environment*, SIGIR Conference on Research and Development in Information Retrieval, vol. 21, ACM, 1998.

[39] S. Chakrabarti, B. Dom and P. Indyk, *Enhanced hypertext categorization using hyperlinks*, Proceedings of ACM SIGMOD, 1998.

[40] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins, *Mining the link structure of the World Wide web*, IEEE Computer, 1999.

[41] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghvan, *Using taxonomy, discrimination, and signatures to navigate in text databases*. VLDB, 1997.

[42] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghvan, *Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies*, VLDB Journal, 1998.

[43] C. Olivia, C. Change, C. F. Enguix, A.K. Ghose, *Case-Based BDI Agents: An Effective Approach for Intelligent Search on the web*, Proceeding AAAI-99, Spring Symposium on Intelligent Agents in Cyberspace Stanford University, USA, 1999.

[44] T. M. Mitchell, *Machine Learning*, New York: McGraw-Hill, 1997.

[45] D. D. Lewis, *Naive (Bayes) at forty: The independence assumption in information retrieval,* In ECML-98, 1998.

[46] J. Rennie, R. Rifkin, *Improving Multiclass Text Classification with the Support Vector Machine,* Massachusetts Institute of Technology, Artificial Intelligence Laboratory Publications, AIM-2001-026, 2001.

[47] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell, *Text classification from labeled and unlabelled documents using EM*, Machine Learning Journal, 1999.

[48] L. P. Kaelbling, M. L. Littman, and A. W. Moore, *Reinforcement learning: A Survey*, Journal of Artificial Intelligence Research, pp. 237-285, 1996.

[49] J.L. Kolonder, *Case-Based Reasoning*, Morgan Kauffman, Publisher Inc, 1993.

[50]] B. Bartsch-Spörl, M. Lenz, , A. Hübner, *Case-Based Reasoning – Survey and Future Directions*, Knowledge-Based Systems, Lecture Notes in Artificial Intelligence, Vol. 1570, Springer-Verlag, Berlin, Heidelberg pages 67-89 ,1999.

[51] W. Cohen, A. McCallum, D. Quass, *Learning to understand the web*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2000.

[52] M. Diligenti, F. Coetzee, S. Lawrence, C. Lee Giles, M. Gori. *Focused Crawling using Context Graph*s, 26th International Conference on Very Large Databases, VLDB 2000, pp. 527–534, 2000.

[53] J. Shakes, M. Langheinrich, O. Etzioni, D*ynamic reference sifting: a case study in the homepage domain*, Proceedings *of* the Sixth International World Wide web Conference, pp.189-200, 1997.

[54] P. De Bra, G. Houben, Y. Kornatzky and R. Post, *Information Retrieval in Distributed Hypertexts*, Proceedings of the 4th RIAO Conference, pp. 481- 491, 1994.

[55] S. Chakrabarti, R. Jaju, M. Joshi, K. Punera, *Analysing fine-grained hypertext features for enhanced crawling and topic distillation*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2002.

[56] S. Chakrabarti, K. Punera, M. Subramanyam, *Accelerated Focused Crawling through Online Relevance Feedback*, In 12th World Wide Web Conference, 2002.

[57] M. Hersovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim and S. Ur, *The Shark-Search Algorithm - An Application: Tailored web Site Mapping*, Proceedings of the Seventh International World Wide web Conference, 1998.

[58] S. Raghavan and H. Garcia-Molina, *Crawling the Hidden web*, *Stanford Digital Libraries Technical Report,* 2000.

[59] S. Mukherjea , *WTMS: A System for Collecting and Analyzing Topic-Specific web Information*, Proceedings of the 9th International World Wide web Conference, 2000.

[60] F. Menczer, G. Pant, P. Srinivasan and M. Ruiz, *Evaluating Topic-Driven web Crawlers*, In Proceedings of the 24th Annual International ACM/SIGIR Conference, 2001.

[61] C. Aggarwal, F. Al-Garawi, P. Yu, *Intelligent Crawling on the World Wide web with Arbitrary Predicates*, Proceedings of the 10th International World Wide web Conference, Hong Kong, 2001.

[62] L.Torgo, J. Gama, *Regression using classification algorithms*, Intelligent Data Analysis 1(4), 1997.

[63] J. Elgot-Drapkin. *Step-logic: Reasoning Situated in Time*. Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park, 1988.

[64] J. Elgot-Drapkin, S. Kraus, M. Miller, M. Nirkhe and D. Perlis, *A Unified Formal Approach to Episodic Reasoning*. Technical report, University of Maryland, 1988.

[65] J. Elgot-Drapkin and D. Perlis, *Reasoning Situated in Time {I}: Basic Concepts,* Journal of Experimental and Theoretical Artificial Intelligence, 2(1):75—98, 1990.

[66] K. Purang, D. Purushothaman, D. Traum, C. Andersen, D. Traum and D. Perlis. *Practical Reasoning and Plan Execution with Active Logic*. IJCAI'99 Workshop on Practical Reasoning and Rationality, 1999.

[67] , S. Kraus, D. Perlis, *Assessing others' knowledge and ignorance,* Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems, pp. 220-225, 1989.

[68] D. Perlis and K. Purang and C. Andersen, *Conversational Adequacy: Mistakes are the essence*, International Journal of Human Computer Studies, 553--575. 1998.

[69] D. Traum, C. Andersen, Y. Chong, D. Josyula, M. O'Donovan-Anderson, Y. Okamoto, K. Purang D. Perlis, *Representations of Dialogue State for Domain and Task Independent Meta-Dialogue*, Electronic Transactions on AI, forthcoming.

[70] J. Gurney, D. Perlis and K. Purang, *Interpreting Presuppositions Using Active Logic: From Context to Utterances*, Computational Intelligence. 1997.

[71] W. Chong, M. O'Donovan-Anderson, Y. Okamoto and D. Perlis, *Seven Days in the Life of a Robotic Agent*, GSFC/JPL Workshop on Radical Agent Concepts, NASA Goddard Space Flight Center, Greenbelt, MD, US, 2002.

[72] A. A. Barfourosh, H.R. Motahary Nezhad, M. Anderson, Don Perlis, *ALLI: An Information Integration System Based on Active Logic Framework*, Proceedings of the Third International Conference on Management Information Systems, Greece, pp.339-348, 2002.