P-1550

Web Search Using Genetic Algorithm

By

R.Yuvaraj

Reg.No:- 71203405023

Of

Kumaraguru College of Technology
Coimbatore-641006.

A Project Report
Submitted to the
FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements
For the award of the Degree
Of

MASTER OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

June, 2005

Bonafide Certificate

Certified that this project entitled “**Web Search Using Genetic Algorithm**” is the bonafide work on **Mr.R.Yuvaraj** who carried out the research under my supervision.Certfied further, that to the best of my knowledge the work reported hearing does not form part of any other project report or dissertion on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]
Project Guide

[illegible]
Head of the Department

The candidate with University registration number **71203405023** was examined by us in the project viva voce exam held on 22/6/'05

R. Dinesh
Internal Examiner

[illegible]
External Examiner
22/6

Acknowledgement

I express my sincere thanks to **Dr.K.K.Padmanabhan**, Principal, Kumaraguru College of Technology, Coimbatore, for providing all facilities to carry out this project.

I record my heartfelt gratitude to our beloved professor **Dr.S.Thangaswamy**, Head, Department of Computer science and engineering, Kumaraguru College of Technology, Coimbatore, for rendering timely help for the successful completion of this project.

I greatly indebted to my guide **Mrs.D.Chandrakala**, Senior Lecturer, for her inspirational guidance and invaluable suggestions in every respect for the completion of the project work.

Iam very thankful to **Mr.R.Dinesh**, Assistant professor, Department of Computer science and engineering and **Mrs.L.S.Jayashree**, Senior Lecturer, Department of Computer science and engineering, for their timely help for the completion of the project.

Iam extremely thankful to my parents and friends for their continuous support without which the successful completion of this project would not have been possible.

ABSTRACT

To establish an Intelligent Agent Systems that uses Genetic Algorithm and uses an novel technique to learn user profiles from user's search histories, to collect and to improve retrieval effectiveness.

Web search engines typically respond to user keyword queries by retrieving relevant URLs from their own databases. They range from general, like AltaVista, *to very specific, like Web seek. Search features as well as database* size vary, and no one Search engine provides the best index for all subjects. Users have therefore had to learn different features and interfaces for every search engine.

Meta search engines offer a partial solution to this problem. Instead of maintaining a local database, they use the indexes of other search engines and give users a single interface to and from the search results. Meta search is a valuable Web search tool, but it still leaves some problems unsolved. For instance, it does not help users form good search strategies, nor does it offer ways to handle an overload of results.

In recent years, intelligent software agents have emerged from artificial intelligence (AI) research as a tool for addressing this problem. Here a prototype agent system is deployed for collecting information from the Internet and filtering it according to a profile of user interests. These systems are called *intelligent assistants*. It includes a genetic algorithm, which enables it to learn the user's interests and to adapt as user interests change over time. The learning process is driven by user feedback to the agent's filtered selections.

In this project, an intelligent assistant system is developed, its learning *agent, and the genetic algorithm. We conclude with results from two preliminary* experiments that tested the accuracy and adaptability of the learning agent.

கருத்துச் சுருக்கம்

மரபு வழிமுறைத் திட்டத்தைப் பயன்படுத்தும் திறன் முகவர் அமையத்தை நிறுவுதலும், பயனாளரின் படிமங்களை அவரது தேடல் வரலாற்றிலிருந்து தொகுத்து அடைவு நேரத்தை மேம்படுத்தவல்ல புதிய தொழில் நுட்ப அணுகு முறையயை பயன்படுத்தியுள்ளோம்

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 HOW SEARCH ENGINE WORKS

Search engine is the popular term for an information retrieval (IR) system. While researchers and developers take a broader view of IR systems, consumers think of them more in terms of what they want the systems to do — namely search the Web, or an intranet, or a database. Actually consumers would really prefer a finding engine, rather than a search engine.

Search engines match queries against an index that they create. The index consists of the words in each document, plus pointers to their locations within the documents. This is called an inverted file. A search engine or IR system comprises four essential modules:

1. A document processor
2. A query processor
3. A search and matching function
4. A ranking capability

While users focus on "search," the search and matching function is only one of the four modules. Each of these four modules may cause the expected or unexpected results that consumers get when they use a search engine.

1.1.1 Document Processor

The document processor prepares, processes, and inputs the documents, pages, or sites that users search against. The document processor performs some or all of the following steps:

1. Normalizes the document stream to a predefined format.
2. Breaks the document stream into desired retrievable units.
3. Isolates and metatags sub-document pieces.

4. Identifies potential indexable elements in documents.
5. Deletes stop words.
6. Stems terms.
7. Extracts index entries.
8. Computes weights.
9. Creates and updates the main inverted file against which the search engine searches in order to match queries to documents.

Steps 1-3: Preprocessing

While essential and potentially important in affecting the outcome of a search, these first three steps simply standardize the multiple formats encountered when deriving documents from various providers or handling various Web sites. The steps serve to merge all the data into a single consistent data structure that all the downstream processes can handle. The need for a well-formed, consistent format is of relative importance in direct proportion to the sophistication of later steps of document processing. Step two is important because the pointers stored in the inverted file will enable a system to retrieve various sized units — site, page, document, section, paragraph, or sentence.

Step 4: Identify elements to index

Identifying potential index able elements in documents dramatically affects the nature and quality of the document representation that the engine will search against. In designing the system, we must define the word "term." Is it the alpha-numeric characters between blank spaces or punctuation? If so, what about non-compositional phrases (phrases in which the separate words do not convey the meaning of the phrase, like "skunk works" or "hot dog"), multi-word proper names, or inter-word symbols such as hyphens or apostrophes that can denote the difference between "small business men" versus small-business men." Each search engine depends on a set of rules that its document processor must execute to determine what action is to be taken by the "tokenizer," i.e. the software used to define a term suitable for indexing.

Step 5: Deleting stop words

This step helps save system resources by eliminating from further processing, as well as potential matching, those terms that have little value in finding useful documents in response to a customer's query. This step used to matter much more than it does now when memory has become so much cheaper and systems so much faster, but since stop words may comprise up to 40 percent of text words in a document, it still has some significance. A stop word list typically consists of those word classes known to convey little substantive meaning, such as articles (*a, the*), conjunctions (*and, but*), interjections (*oh, but*), prepositions (*in, over*), pronouns (*he, it*), and forms of the "to be" verb (*is, are*). To delete stop words, an algorithm compares index term candidates in the documents against a stop word list and eliminates certain terms from inclusion in the index for searching.

Step 6: Term Stemming

Stemming removes word suffixes, perhaps recursively in layer after layer of processing. The process has two goals. In terms of efficiency, stemming reduces the number of unique words in the index, which in turn reduces the storage space required for the index and speeds up the search process. In terms of effectiveness, stemming improves recall by reducing all forms of the word to a base or stemmed form. For example, if a user asks for *analyze*, they may also want documents which contain *analysis*, *analyzing*, *analyzer*, *analyzes*, and *analyzed*. Therefore, the document processor stems document terms to *analy-* so that documents which include various forms of *analy-* will have equal likelihood of being retrieved; this would not occur if the engine only indexed variant forms separately and required the user to enter all. Of course, stemming does have a downside. It may negatively affect precision in that all forms of a stem will match, when, in fact, a successful query for the user would have come from matching only the word form actually used in the query.

Systems may implement either a strong stemming algorithm or a weak stemming algorithm. A strong stemming algorithm will strip off both inflectional suffixes (*-s, -es, -ed*) and derivational suffixes (*-able, -aciousness, -ability*), while a weak stemming algorithm will strip off only the inflectional suffixes (*-s, -es, -ed*).

Step 7: Extract index entries

Having completed steps 1 through 6, the document processor extracts the remaining entries from the original document.

Step 8: Term weight assignment

Weights are assigned to terms in the index file. The simplest of search engines just assign a binary weight: 1 for presence and 0 for absence. The more sophisticated the search engine, the more complex the weighting scheme. Measuring the frequency of occurrence of a term in the document creates more sophisticated weighting, with length-normalization of frequencies still more sophisticated. Extensive experience in information retrieval research over many years has clearly demonstrated that the optimal weighting comes from use of "tf/idf." This algorithm measures the frequency of occurrence of each term within a document. Then it compares that frequency against the frequency of occurrence in the entire database.

Not all terms are good "discriminators" — that is, all terms do not single out one document from another very well. A simple example would be the word "the." This word appears in too many documents to help distinguish one from another. A less obvious example would be the word "antibiotic." In a sports database when we compare each document to the database as a whole, the term "antibiotic" would probably be a good discriminator among documents, and therefore would be assigned a high weight. Conversely, in a database devoted to health or medicine, "antibiotic" would probably be a poor discriminator, since it occurs very often. The TF/IDF weighting scheme assigns higher weights to those terms that really distinguish one document from the others.

Step 9: Create index

The index or inverted file is the internal data structure that stores the index information and that will be searched for each query. Inverted files range from a simple listing of every alpha-numeric sequence in a set of documents/pages being indexed along with the overall identifying numbers of the documents in which the sequence occurs, to a more linguistically complex list of entries, the tf/idf weights, and pointers to where inside each document the term occurs. The more complete the information in the index, the better the search results.

1.1.2 Query Processor

Query processing has seven possible steps, though a system can cut these steps short and proceed to match the query to the inverted file at any of a number of places during the processing. Document processing shares many steps with query processing. More steps and more documents make the process more expensive for processing in terms of computational resources and responsiveness. However, the longer the wait for results, the higher the quality of results. Thus, search system designers must choose what is most important to their users — time or quality. Publicly available search engines usually choose time over very high quality, having too many documents to search against.

The steps in query processing are as follows (with the option to stop processing and start matching indicated as "Matcher"):

- Tokenize query terms.
  Recognize query terms vs. special operators.

  ————————> Matcher

- Delete stop words.
- Stem words.
- Create query representation.
  ————————> Matcher
- Expand query terms.
- Compute weights.
  ————————> Matcher

Step 1: Tokenizing

As soon as a user inputs a query, the search engine — whether a keyword-based system or a full natural language processing (NLP) system — must tokenize the query stream, i.e., break it down into understandable segments. Usually a token is defined as an alpha-numeric string that occurs between white space and/or punctuation.

Step 2: Parsing

Since users may employ special operators in their query, including Boolean, adjacency, or proximity operators, the system needs to parse the query first into query terms and operators. These operators may occur in the form of reserved punctuation (e.g., quotation marks) or reserved terms in specialized format (e.g., AND, OR). In the case of an NLP system, the query processor will recognize the operators implicitly in the language used no matter how the operators might be expressed (e.g., prepositions, conjunctions, ordering).

At this point, a search engine may take the list of query terms and search them against the inverted file. In fact, this is the point at which the majority of publicly available search engines perform the search.

Steps 3 and **4: Stop list and stemming**

Some search engines will go further and stop-list and stem the query, similar to the processes described above in the Document Processor section. The stop list might also contain words from commonly occurring querying phrases, such as, "I'd like information about." However, since most publicly available search engines encourage very short queries, as evidenced in the size of query window provided, the engines may drop these two steps.

Step 5: Creating the query

How each particular search engine creates a query representation depends on how the system does its matching. If a statistically based matcher is used, then the query must match the statistical representations of the documents in the system. Good statistical queries should contain many synonyms and other terms in order to create a full representation. If a Boolean matcher is utilized, then the system must create logical sets of the terms connected by AND, OR, or NOT.

An NLP system will recognize single terms, phrases, and Named Entities. If it uses any Boolean logic, it will also recognize the logical operators from Step 2 and create a representation containing logical sets of the terms to be AND, OR, or NOT.

At this point, a search engine may take the query representation and perform the search against the inverted file. More advanced search engines may take two further steps.

Step 6: Query expansion

Since users of search engines usually include only a single statement of their information needs in a query, it becomes highly probable that the information they need may be expressed using synonyms, rather than the exact query terms, in the documents which the search engine searches against. Therefore, more sophisticated systems may expand the query into all possible synonymous terms and perhaps even broader and narrower terms.

This process approaches what search intermediaries did for end users in the earlier days of commercial search systems. Back then, intermediaries might have used the same controlled vocabulary or thesaurus used by the indexers who assigned subject descriptors to documents. Today, resources such as WordNet are generally available, or specialized expansion facilities may take the initial query and enlarge it by adding associated vocabulary.

Step 7: Query term weighting (assuming more than one query term).

The final step in query processing involves computing weights for the terms in the query. Sometimes the user controls this step by indicating either how much to weight each term or simply which term or concept in the query matters most and *must* appear in each retrieved document to ensure relevance.

Leaving the weighting up to the user is not common, because research has shown that users are not particularly good at determining the relative importance of terms in their queries. They can't make this determination for several reasons. First, they don't know what else exists in the database, and document terms are weighted by being compared to the database as a whole. Second, most users seek information about an unfamiliar subject, so they may not know the correct terminology.

Few search engines implement system-based query weighting, but some do an implicit weighting by treating the first term(s) in a query as having higher significance. The engines use this information to provide a list of documents/pages to the user.

After this final step, the expanded, weighted query is searched against the inverted file of documents.

1.1.3 Search and Matching Function

How systems carry out their search and matching functions differs according to which theoretical model of information retrieval underlies the system's design philosophy. Since making the distinctions between these models goes far beyond the goals of this article, we will only make some broad generalizations in the following description of the search and matching function.

Searching the inverted file for documents meeting the query requirements, referred to simply as "matching," is typically a standard binary search, no matter whether the search ends after the first two, five, or all seven steps of query processing. While the computational processing required for simple, unweighted, non-Boolean query matching is far simpler than when the model is an NLP-based query within a weighted, Boolean model, it also follows that the simpler the document representation, the query representation, and the matching algorithm, the less relevant the results, except for very simple queries, such as one-word, non-ambiguous queries seeking the most generally known information.

Having determined which subset of documents or pages matches the query requirements to some degree, a similarity score is computed between the query and each document/page based on the scoring algorithm used by the system. Scoring algorithms rankings are based on the presence/absence of query term(s), term frequency, tf/idf, Boolean logic fulfillment, or query term weights. Some search engines use scoring algorithms not based on document contents, but rather, on relations among documents or past retrieval history of documents/pages.

After computing the similarity of each document in the subset of documents, the system presents an ordered list to the user. The sophistication of the ordering of the documents again depend on the model the system uses, as well as the richness of the document and query weighting mechanisms. For example, search engines that only require the presence of any alpha-numeric string from the query occurring anywhere, in any order, in a document would produce a very different ranking than one by a search engine that performed linguistically correct phrasing for both document and query representation and that utilized the proven tf/idf weighting scheme.

However the search engine determines rank, the ranked results list goes to the user, who can then simply clicks and follow the system's internal pointers to the selected document/page.

More sophisticated systems will go even further at this stage and allow the user to provide some relevance feedback or to modify their query based on the results they have seen. If either of these are available, the system will then adjust its query representation to reflect this value-added feedback and re-run the search with the improved query to produce either a new set of documents or a simple re-ranking of documents from the initial search.

1.2 What Document Features Make a Good Match to a Query

We have discussed how search engines work, but what features of a query make for good matches? Let's look at the key features and consider some pros and cons of their utility in helping to retrieve a good representation of documents/pages.

• **Term frequency**:

How frequently a query term appears in a document is one of the most obvious ways of determining a document's relevance to a query. While most often true, several situations can undermine this premise. First, many words have multiple meanings — they are polysemous. Think of words like "pool" or "fire." Many of the non-relevant documents presented to users result from matching the right word, but with the wrong meaning.

Also, in a collection of documents in a particular domain, such as education, common query terms such as "education" or "teaching" are so common and occur so frequently that an engine's ability to distinguish the relevant from the non-relevant in a collection declines sharply. Search engines that don't use a tf/idf weighting algorithm do not appropriately down-weight the overly frequent terms, nor are higher weights assigned to appropriate *distinguishing (and less frequently-occurring) terms, e.g., "early-childhood."*

• **Location of terms:**

Many search engines give preference to words found in the title or lead paragraph or in the metadata of a document. Some studies show that the location — in which a term occurs in a document or on a page — indicates its significance to the document. Terms occurring in the title of a document or page that match a query term are therefore frequently weighted more heavily than terms occurring in the body of the document. Similarly, query terms occurring in section headings or the first paragraph of a document may be more likely to be relevant.

• **Link analysis:**

Web-based search engines have introduced one dramatically different feature for weighting and ranking pages. Link analysis works somewhat like bibliographic citation practices, such as those used by Science Citation Index. Link analysis is based on how well-connected each page is, as defined by Hubs and Authorities, where Hub documents link to large numbers of other pages (out-links), and Authority documents are those referred to by many other pages, or have a high number of "in-links" .

• **Popularity:**

Google and several other search engines add popularity to link analysis to help determine the relevance or value of pages. Popularity utilizes data on the frequency with which a page is chosen by all users as a means of predicting relevance. While popularity is a good indicator at times, it assumes that the underlying information need remains the same.

• **Date of Publication:**

Some search engines assume that the more recent the information is, the more likely that it will be useful or relevant to the user. The engines therefore present results beginning with the most recent to the less current.

• **Length** :

While length does not necessarily predict relevance, it is a factor when used to compute the relative merit of similar pages. So, in a choice between two documents both containing the same query terms, the document that contains a proportionately higher occurrence of the term relative to the length of the document is assumed more likely to be relevant.

• **Proximity of query terms** :

When the terms in a query occur near to each other within a document, it is more likely that the document is relevant to the query than if the terms occur at greater distance. While some search engines do not recognize phrases per se in queries, some search engines clearly rank documents in results higher if the query terms occur adjacent to one another or in closer proximity, as compared to documents in which the terms occur at a distance.

• **Proper nouns :**

Sometimes have higher weights, since so many searches are performed on people, places, or things. While this may be useful, if the search engine assumes that you are searching for a name instead of the same word as a normal everyday term, then the search results may be peculiarly skewed.

CHAPTER 2

JUSTIFICATION OF THE PROJECT

All traditional search engines have some draw backs, they are.

1. Overload of results.
2. Partial solution to a problem.(User queries)
3. Database size varies.

   In order to eliminate this, an intelligent agent systems using Genetic Algorithm is used.

The Webnaut learning agent uses AI techniques to accomplish the difficult task of learning a user's interests. At the same time, the learning agent collects new documents of interest from the Web and recommends them to the user. While surfing the Web, a Webnaut user can bookmark a document as "Very Interesting." The words in the document are then processed by the text information extractor (described below) and used to *direct the agent's search for similar Web documents.* By *searching for* documents with relevant words combined with logical operators, the learning agent can create complex representative structures of the user's long-term interests.

a) Vector-space model:

Webnaut uses the vector space model, borrowed from the field of information retrieval, to represent information included in collected documents.1 According to this model; all words included in a document collection are extracted to create the dictionary vector **D**, whose every element is a word *di*. A vector **W** can thus represent a document from the collection, where every element of *wi* is the weight of the word *di* for that particular document.

Information-retrieval systems typically represent the user's question in this way so that a query vector can be created. The cosine angle between the query and the weight vectors of every document in the collection can be used as a similarity measure to recover documents that match the user's request.

In the Webnaut prototype implementation, all keywords that hold the greater weight from all the text documents that the user provides as examples are merged in a file called Dictionary. The representation of the Dictionary is an $N \times 3$ matrix, where N is the number of keywords. The first column of the matrix includes the keywords, the second column includes the total number of documents that contain the keywords, and the last column includes the sum of the keywords' frequencies in all the texts that appear.

The $N \times 3$ matrix allows the comparison of different keyword weighting schemes. The Dictionary's keywords are sorted according to their weight, which is given by the following formula:

$$wi = (tf_i / tf_{max}) / \sqrt{} (tf_i / tf_{max})^2$$

Where *tfi* is the frequency of the keyword *i* in all texts in which it appears; *tf*max is the maximum keyword frequency of all keywords in the Dictionary; and *N* is the number of keywords in the Dictionary. The number of keywords stored in the dictionary is a user-defined parameter; in our implementation it was limited to 1,000 words to avoid a memory overload.

The inverse document-frequency weighting schema *tfidf* has been implemented but was not used in this prototype, since the Dictionary is considered a virtual document that includes only those keywords of interest to the user. The *tfidf* is quite popular in similar applications, but a collection of documents is necessary for it to be applied. The user may change the system's configuration and select to use the *tfidf* weighting schema, but to do so he or she must create a collection of documents; that is, a sufficient number of examples must be provided before the learning agent can begin the process of searching for new information.

In the approach described here, keyword weights do not depend on the number of documents in which the word appears, and the learning agent can help the user from the very first example. On the other hand, the weights depend solely on keyword frequency, which makes it easy to decrease or increase them according to user feedback.

The Webnaut learning agent also presents all new information collected by the genetic algorithm (GA) agent. The user's positive or negative feedback on GA agent recommendations is used to modify the keyword weights, included in the Dictionary.

b) Text information extractor:

This component, which consists of a lexical analyzer, a stop word removal algorithm, and a stemming algorithm, analyzes HTML pages. The lexical analyzer tokenizes the input HTML page in three steps: It selects all hyperlinks in the document, removes the HTML tags, and finally removes any script language commands.

The stop word algorithm removes all high-frequency words such as "for" and "the," all common words such as days and months, and all numbers. (Common words are included in a Thesaurus file, which can be enriched by the user.) Porter's algorithm2 is applied to the remaining text, stripping it of inflectional and derivational word endings. This stemming algorithm reduces the morphologic variants of a word to a single index term (yielding fewer words in the dictionary) and increases recall by grouping similar concepts under single terms, hereby increasing quality of predictions.

2.1 Genetic Algorithm Agent

The GA agent collects and evaluates new HTML pages from the Web, using information included in examples provided by the user. Pages that score high are served to the user by the learning agent.

2.2 GA Agent Implementation

The GA agent implements evolutionary techniques to a genetic algorithm. A primary genetic algorithm (PGA) creates a population of sets of unique keywords selected at random from the Dictionary. A secondary genetic algorithm (SGA) then creates a population of sets of logic operators for each of the PGA's members. The MetaSearch agent serves the queries, as described earlier.

A user's Dictionary, for example, may contain the following words: intelligent, agents, genetic, algorithms, source, code, reinforcement, and learning.

The user may be interested in intelligent agents and genetic algorithms but not reinforcement learning. In this case, the keywords can be combined with logic operators to describe the user's interests as follows: intelligent AND agents AND genetic AND algorithms NOT reinforcement NOT learning.

A user-defined number of members, each with a vector of random unique keywords from the Dictionary, is created during the initialization of the PGA population. During evaluation, each keyword set is combined with the SGA population members, or random sets of logic operators. Each set of logic operators and keywords creates a number of different queries that are then served by the MetaSearch agent, which collects and provides appropriate URLs.

$$\boldsymbol{Sim(D_D,D_S)} = \sum \mathbf{W_{DK},W_{SK}} \;/\sqrt{\sum \mathbf{W_{DK}}^2 \sum \mathbf{W_{SK}}^2}$$

The "fitness" of each member in the SGA population is the average of the similarity between the MetaSearch agent's results, which correspond to the specific query and the Dictionary. The similarity function between two documents *DS* and *DD*, represented in a space of *N* keywords by their weighted keyword vectors *WDk* and *WSk* is a predefined number of the fittest members from all queries are stored in a vector to provide the basis for an evaluation using the total population of logic operators. The average value of the similarities of the documents stored in this vector is also the fitness for each member in the PGA population. After the population evaluation, genetic operators are applied to form a new population of logic operands.

2.3 Genetic operators:

The main genetic operator is the crossover, which operates on two individuals. The crossover operator takes two SGA members, randomly splits them at a point, and then recombines them to produce two new members.

As the inversion operator, mutation operates on a single individual, switching a selected SGA member's operator to another operator at random. If the operator is OR, for example, it will be changed randomly to AND or NOT from a selected point on.

After the recombination operators are applied, the fittest members are selected to form a new population, and the same loop will continue running for a predefined number of generations. The same recombination operators, not including inversion, will be applied slightly differently to the new sets of keywords. In the PGA, the crossover operator takes two sets of keywords, randomly splits them, and then recombines them to produce two new sets of key keywords. Keywords from the two sets are exchanged if and only if the same words don't appear in the new sets more than once.

Two different algorithms are used for mutation in the PGA. The first algorithm selects two members of the population and exchanges their words at a selected point if and only if the same words don't appear in either set more than once. The second algorithm selects one member of the population and exchanges the keyword at a selected point with a new keyword from the Dictionary. Finally, the fittest members of the PGA population are selected and the same loop continues for a number of generations.

2.4 Primary Genetic Algorithm

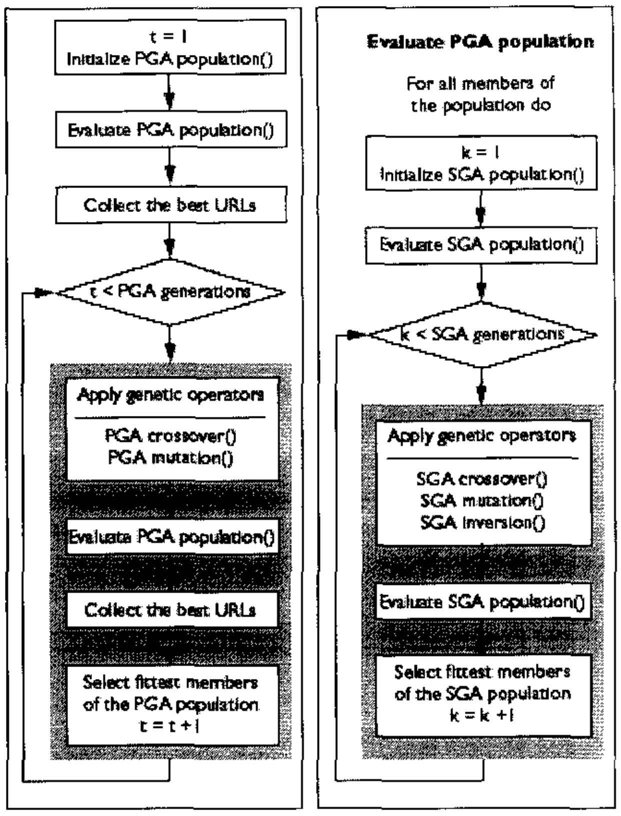

t = 1
Initialize PGA population()
Evaluate PGA population()
Collect the best URLs
t < PGA generations
Apply genetic operators
PGA crossover()
PGA mutation()
Evaluate PGA population()
Collect the best URLs
Select fittest members of the PGA population
t = t + 1
Evaluate PGA population
For all members of the population do
k = 1
Initialize SGA population()
Evaluate SGA population()
k < SGA generations
Apply genetic operators
SGA crossover()
SGA mutation()
SGA inversion()
Evaluate SGA population()
Select fittest members of the SGA population
k = k + 1

Fig-2.1

CHAPTER 3

WEB SEARCHING AND INFORMATION RETRIEVAL

3.1 Search-Engine Architectures

We can distinguish three architectures for Web searching: traditional (or centralized), metasearch, and distributed search. Search engines can also be part of the more general architectures such as search services or portals.

3.1.1 Centralized Architecture

The goal of general-purpose search engines is to index a sizeable portion of the Web, independently of topic and domain. Each such engine consists of several components, as Figure shows.

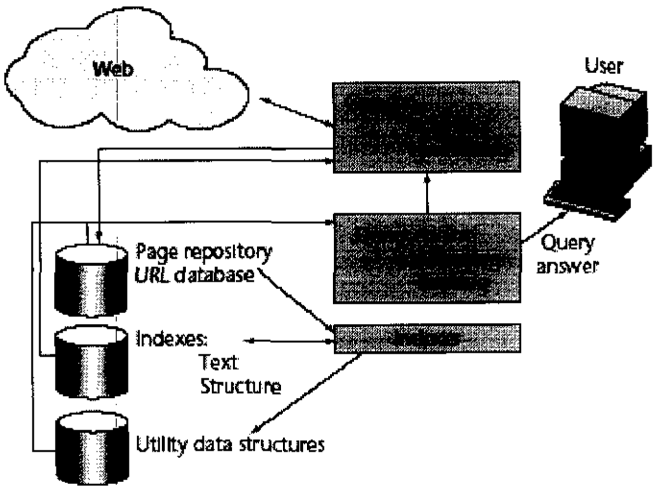


Fig-3.1

A *crawler* (also called a *spider* or *robot*) is a program controlled by a crawl control module that "browses" the Web. It collects documents by recursively fetching links from a set of start pages; the retrieved pages or their parts are then compressed and stored in a *page repository*. URLs and their links, which form a Web graph, are transferred to the crawler control module, which decides the movement in this graph. Obviously, off-site links are of interest. To save space, documents' identifiers (docIDs) represent pages in the index and other data structures; the crawler uses a database of URLs for this purpose.

The *indexer* processes the pages collected by the crawler. It first decides which pages to index—for example, it might discard duplicate documents. Then, it builds various auxiliary data structures. Most search engines build some variant of an inverted index data structure for words (*text index*) and links (*structure index*). The inverted index contains for each word a sorted list of couples (such as docID and position in the document).

The *query engine* processes user queries—and returns matching answers—in an order determined by a ranking algorithm. The algorithm produces a numerical score expressing an importance of the answer with respect to the query. Its capabilities and features depend on additional data structures (called *utility* data structures) such as lists of related pages, similarity indexes, and so forth. The numerical score is usually a combination of query-independent and query-dependent criteria.

The former judge the document regardless of the actual query; typical examples include its length and vocabulary, publication data (such as the site to which it belongs, the date of the last change to it, and so on), and various connectivity based data such as the number of links pointing to a page (called *in-degree*). Query-dependent criteria include a cosine measure for similarity in the vector space model (which is well known from traditional IR techniques) and all connectivity-based techniques. All defined measures can contribute to the resulted measure.

3.1.2 Metasearch Architecture

One way to provide access to the information in the hidden Web's text databases is through *Meta searchers*, which can be used to query multiple databases simultaneously. A Meta searcher performs three main tasks. After receiving a query, it finds the best databases to evaluate the query (*database selection*), translates the query in a suitable form for each database (*query translation*), and then retrieves and merges the results from the different databases (*result merging*) and returns them to the user.

A Meta searcher's database selection component is crucial in terms of both query processing efficiency and effectiveness. Database selection algorithms are traditionally based on pre-collected statistics that characterize each database's contents. These statistics, often called *content summaries*, usually include at least the *document frequencies* of the words that appear in the database. To obtain a database's content summary, a meta searcher relies on the database to supply the summary (for example, by using Semantic Webtags).

Unfortunately, many Web-accessible databases are completely autonomous and don't report any detailed metadata about their contents that would facilitate meta searching. With such databases, only manually generated descriptions of the contents are usable, so this approach is not scalable to the thousands of text databases available on the Web today. Moreover, we wouldn't get the good quality, fine-grained content summaries required by database selection algorithms. Some researchers recently presented a technique to automate content summary extraction from searchable text databases: it seems that the deeper recesses of the Web aren't really hidden. By systematically retrieving small sample contents, we can model information sources.

3.1.3 Distributed Search Architecture

Whatever successful global ranking algorithms for centralized search engines are, two potential problems occur: high computational costs and potentially poor rankings. Additional semantic problems are related to the exclusive use of global context and the instability of ranking algorithm .Distributed heterogeneous search environments are an emerging phenomenon in Web search. Although the original Internet was designed to be a peer-to-peer (P2P) system, Web search engines have yet to make full use of this potential. Most major Web search engines are currently based on cluster architectures.

Earlier attempts to distribute processes suffered many problems—for example, Web servers got requests from different search-engine crawlers that increased the servers' load. Most of the objects the crawlers retrieved were useless and subsequently discarded; compounding this, there was no *coordination among the crawlers.* A *new completely distributed* and decentralized P2P crawler called Apoidea is both self-managing and uses the resource's geographical proximity to its peers for a better and faster crawl.

Another recent work explores the possibility of using document rankings in searches. By partitioning and combining the rankings, the decentralized crawler manages to compute document rankings of large scale web data sets in a localized fashion. The most general approach is a *federation* of independently controlled metasearchers along with many specialized search engines. These engines provide focused search services in a specific domain.

3.2 Page Importance and Its Use in Retrieval

In general, we must measure a page's importance in order to rank it. Three approaches help with this process: *link*, *content* (*similarity*), and *anchor*. In terms of IR, these measures reflect a *model* of Web documents. The best-known link-based technique used on the Web today is a variant of the PageRank algorithm implemented in the Google search engine.

It tries to infer a Web page's importance from just the topological structure of a directed graph associated with the Web. A page's rank depends on the ranks of all the pages pointing to it, with each rank divided by the number of out-links those pages have. In the simplest variant, the PageRank of a page *k*, *Pr* (*k*) is a nonnegative real number given by

$$Pr(k) = \Sigma(h,k)\ Pr(h)/o(h),\ k = 1, 2, \ldots, n,$$

where *Pr*(*h*) is the PageRank of page *h*, *o*(*h*) is the out-degree of page *h*, and *the sum is extended to all Web pages h pointing to page k* (*n is the number of* pages on the Web). If a page *h* has more out links to the same page *k*, all these out-link count as one. According to this definition, then, *Pr* (*h*) depends not only on the number of pages pointing to it, but also on their importance.

This definition raises some problems—something like a *rank sink* can occur (a group of pages pointing to each other could have some links going to the group but no links going out).

Another interesting technique—Kleinberg's algorithm, also called HITS (Hypertext Induced Topic Search)—is used at query time and processed on a small subset of relevant documents, but not all of them. It computes two scores per document. *Authoritative pages* relevant to the initial query have large in-degree: they are all the authorities on a common topic, and a considerable overlap in the sets of pages point to them. The algorithm then finds *hub pages*, which have links to multiple relevant authoritative pages: if a page were a good authority, many hubs would point to it. These ideas are not new. Some were exploited decades ago in bibliographic citation analysis and later in the field of hypertext systems. In the content-based approach, we compute the similarity score between a page and a predefined topic in a way similar to the vector model. Topic vector *q* is constructed from a sample of pages, and each Web page has its own vector *p*. The similarity score Sim (*p*, *q*) is defined by the cosine similarity measure.

Anchor text is the visible hyperlinked text on the Web page. In the anchor-based approach, page quality can be judged by pattern matching between the query vector and the URL's anchor text, the text around the anchor text (the anchor window), and the URL's string value. Approaches used in isolation suffer various drawbacks. The usual content-based approach ignores links and is susceptible to spam, and the link-based approach is not adequate for pages with low in-degree. Due to the Web's dynamism, this problem appears most frequently when we attempt to discover new pages that have not been cited sufficiently. The approach relying on text near anchors seems to be the most useful for Web similarity-search tasks. Similar to vector models, it must involve additional considerations concerning term weighting and anchor window width. With small anchor windows, for example, many documents that should be considered similar are in fact orthogonal (they don't have common words).

Obviously, all previously defined measures can contribute to the end page measure result for page ranking.

3.3 Issues and Challenges in Web Search Engines

Search engine problems are connected with each component of the *engine's architecture and each process it performs—search engines can't* update indexes at the same speed at which the Web evolves, for example. Another problem is the quality of the search results. We've already looked at their lack of stability, heterogeneity, high linking, and duplication (near 30 percent). On the other hand, because the hidden Web's contents' quality is estimated to be 1,000 to 2,000 times greater than that of the surface Web, search result quality can be expected to be higher in this case. One of the core modules of each search engine is its crawler. Several issues arise when search engines crawl through Web pages.

• *What pages should the crawler download?* Page importance metrics can help, such as interest-driven metrics (often used in focused crawlers), popularity-driven metrics (found in combination with algorithms such as PageRank), and location-driven metrics (based on URL).

• *How should the search engine refresh pages, and how often should it do so?*

Most search engines update on a monthly basis, which means the Web graph structure obtained, is always incomplete, and the global ranking computation is less accurate. In a *uniform refresh*, the crawler revisits all pages with the same frequency, regardless of how often they change. In a *proportional refresh*, the crawler revisits pages with a frequency proportional to the page's change rate (for example, if it changes more often, it visits more often).

• *How do we minimize the load on visited Web sites?*

Collecting pages consumes resources (disks, CPU cycles, and so on), so the crawler should minimize its impact on these resources. Most web users cite load time as the Web's single biggest problem.

• *How should the search engine parallelize the crawling process?*

Suppose a search engine uses several crawlers at the same time (in parallel).

How can we make sure they aren't duplicating their work?

A recent research study highlighted several problems concerning the quality of page ranking.

• *Spam*. To achieve a better ranking, some Web authors deliberately try to manipulate their placement in the ranking order. The resulting pages are forms of spam. In text spam, erroneous or unrelated keywords are repeated in the document. Link spam is a collection of links that point to every other page on the site. *Cloaking* offers entirely different content to a crawler than to other users.

• *Content quality*. There are many examples of Web pages containing contradictory information, which means the document's accuracy and reliability are not automatically guaranteed. If we calculate page importance from the anchor text, for example, we would want at least this text to be of high quality (meaning accurate and reliable).

• *Quality evaluation*. Direct feedback from users is not reliable because such user environment capabilities are usually not at our disposal. So, search engines often collect implicit user feedback from log data. New metrics for ranking improvement, such as the number of clicks, are under development.

• *Web conventions*. Web pages are subject to certain conventions such as anchor text descriptiveness, fixed semantics for some link types, metatags for HTML metadata presentation, and so on. Search engines can use such conventions to improve search results.

• *HTML mark-up*. Web pages in HTML contain limited semantic information hidden in HTML mark-up. The research community is still working on streamlined approaches for extracting this information (an introductory approach appears elsewhere12).

Most search engines perform their tasks by using important keywords, but the user might not always know these keywords. Moreover, the user might want to submit a query with additional constraints such as searching a specific Web page or finding the pages within a Web graph structure.

CHAPTER 4

LITERATURE SURVEY

4.1"Agents that Reduce Work and Information Overload "by P.Maes.

This article focuses on a novel approach to building interface agents. It presents results from several prototype agents that have been built using this approach, including agents that provide personalized assistance with meeting scheduling, email handling, electronic news filtering and selection of environment.

4.2 "Effectively Finding Relevant Web Pages from Linkage Information "

By Jingyu Hou and Yanchun Zhang, Member, IEEE Computer Society

This paper presents two hyperlink analysis-based algorithms to find relevant pages for a given Web page (URL). The first algorithm comes from the extended co citation analysis of the Web pages. It is intuitive and easy to implement. The second one takes advantage of linear algebra theories to reveal deeper relationships among the Web pages and to identify relevant pages more precisely and effectively.

The experimental results show the feasibility and effectiveness of the algorithms. These algorithms could be used for various Web applications, such as enhancing Web search. The ideas and techniques in this work would be helpful to other Web-related researches.

4.3 "Domain-Specific Web Search with Keyword Spices "

By Satoshi Oyama, Takashi Kokubo, and Toru Ishida, Fellow, IEEE

Domain-specific Web search engines are effective tools for reducing the difficulty experienced when acquiring information from the Web. Existing methods for building domain-specific Web search engines require human expertise or specific facilities.

However, we can build a domain-specific search engine simply by adding domain-specific keywords, called "keyword spices," to the user's input query and forwarding it to a general-purpose Web search engine. Keyword spices can be effectively discovered from Web documents using machine learning technologies. This paper will describe domain-specific Web search engines that use keyword spices for locating recipes, restaurants, and used cars.

4.4"WEB SEARCHING AND INFORMATION RETRIEVAL "

By JAROSLAV POKORN ´Y,Charless University

The first Web information services were based on traditional information retrieval algorithms, which were originally developed for smaller, more coherent collections than the Web. Due to the Web's continued growth, today's Web searches require new techniques—exploiting or extending linkages among Web page.

CHAPTER 5

PROPOSED LINE OF ATTACK

In a Dynamic Document Search every word in the document is parsed (read) and matched with the search words. Results are displayed based on the matches found.

Reading every word of the article matching it with the search word over thousands or even lakhs of documents is very difficult task. Also by default, PHP is configured to run maximum 30 seconds.

5.1 Building Database:

The database consists of three tables.

- Content Table,
- Keyword Table,
- Link Table.

5.1.1 Content Table

Content table holds article's title, and abstract. Keyword table holds keyword. *Keyword field is indexed. Link table holds keyword id, content id.*

The SQL Statement for creating these three tables are shown below.

Content Table:

```
CREATE TABLE content (
contid mediumint(9) NOT NULL auto_increment, title text,
abstract longtext,  PRIMARY KEY (contid) ) TYPE=MyISAM;
```

5.1.2 Keyword Table

```
CREATE TABLE keytable (
 keyid mediumint NOT NULL auto_increment,
keyword varchar(100) default NULL,
PRIMARY KEY (keyid),
KEY keyword (keyword) ) TYPE=MyISAM;
```

5.1.3 Link Table

```
CREATE TABLE link (
keyid mediumint NOT NULL,
contid mediumint NOT NULL)
TYPE=MyISAM
```

5.2 Preparing Database

An input interface with HTML form is created to enter title and document. After filling and hitting enter, the title and the abstract is stored in the content table. The generated new content id is stored in a variable temporarily. In the next step and 'Upload Engine' that parses each word in the abstract and process the whole text. It removes common words like is, was, and, if, so, else, then etc. Then stores each word in wordmap array. See that every word has only one entry in the wordmap array.

For every word in the wordmap array, keyword table is parsed and *math is found. If there is a match, the generated key id, and content id* generated id earlier is stored in the link table. Else, the new keyword is inserted in the keyword table and with the generated keyword table and content id the link table is updated.

Searching keyword table for every word is a long process. This also reduces the efficiency of the program. To implement this all the keywords in the keyword table is stored in an associative array $allWords. An associative array is one, which works on **B-Tree** algorithm and very useful to perform searches. Here is the function.

5.2.1Common Words

$COMMON_WORDS is an associative array that stores an array of words, which are commonly used in English Language. These words have to be removed while parsing the file.

$COMMON_WORDS=array ("a"=>1, "as"=>1);

You can add as many common words as you like. See source code for full list of common words.

5.2.2 Functions:

5.2.2.1 Extract Words Function:

This function filters words by allowing only alphabetic characters. To implement this, a technique called STATE MACHINE is used to filters the characters.

Alphabetic characters are taken as STATE1 and other characters (Numeric and Special Characters) as STATE0. Initially the machine will be in the STATE0. While parsing letters, it encounters alphabetic characters, the machine switches to STATE1 else it will remain in the same state. As a result we get a word with only alphabetic characters. As a result we get a list of words stored in an array returned to the called function.

5.2.2.2 Filter Common and Duplicate Words Function

This function is called after ExtractWords () function. This parses filtered words removes common words like 'a',' is', 'was',' and'.... Other words are taken as valid words, remove duplicate among them and then stored in an associative array $wordMap and this array is returned to the called function.

5.3 Process Form function

This is the core part of the upload program. After finishing filtering, removing common words and duplicate words, this function is called. First this function inserts the title and abstract in the content table. The newly generated content id stored in $contentId. Then it updates keyword and link table.

For every word in the $wordMap array, if the word is already exists in keyword table, it inserts the key id, content id in to link table. Conversely, if the word is not found, it inserts the new word in keyword table, the generated new key id is stored in $keyId. Then it updates link table by inserting key id content id in link table.

5.4 Search Engine:

PHP script is written that makes it possible to query the database through a HTML form. This will work as any other search engine: the user enters a word in a textbox, hits enter, and the interface presents a result page with links to the pages which contains the word that is searched for.

For example, the results are displayed the order in which the pages are presented is selected by the number of search words appeared in each document.

Declare an associative array $CommonWords that contains common words like 'is', 'in', 'was' etc.

First convert all the search words in to lower case.

```
$search_keywords=strtolower(trim($keywords));
```

Next, we have to perform an explode operation on search words that will store each search word in an array. The code is shown here.

```
$arrWords = explode (" ", $search_keywords);
```

Next, remove duplicate words in $arrWords.

```
$arrWords = array_unique($arrWords);
```

In a search operation, first we have to remove the common words like 'is', 'in', 'was' ... This refines our search criteria. To implement this we store common words in an associative array $CommonWords.

Next, remove common words in the search words. Search words are stored in $searchWords and common words are stored in $junkWords. Here is the code.

```
<?php
$searchWords=array();
$junkWords=array();
foreach($arrWords as $word)

if(!$CommonWords[$word]){
$searchWords[]=$word;
}else{
$junkWords[]=$word;
}
?>
```

We can display results in two ways.

Type 1: Display the document if all the search words present in the document.

Type 2: Display the document if any one of the search words is present.

If you want to perform the Type 1 operation, include the following code snippet in to your program.

```
$noofSearchWords=count ($searchWords);
```

$noofSearchWords stores the number of search words. Later after searching search words in key word table we get results. There we can perform logical AND operation that will display our desired results. If $noofSearchWords is equal to number of records, the next part of the program gets executed. Else "NO SEARCH RESULT FOUND" is displayed.

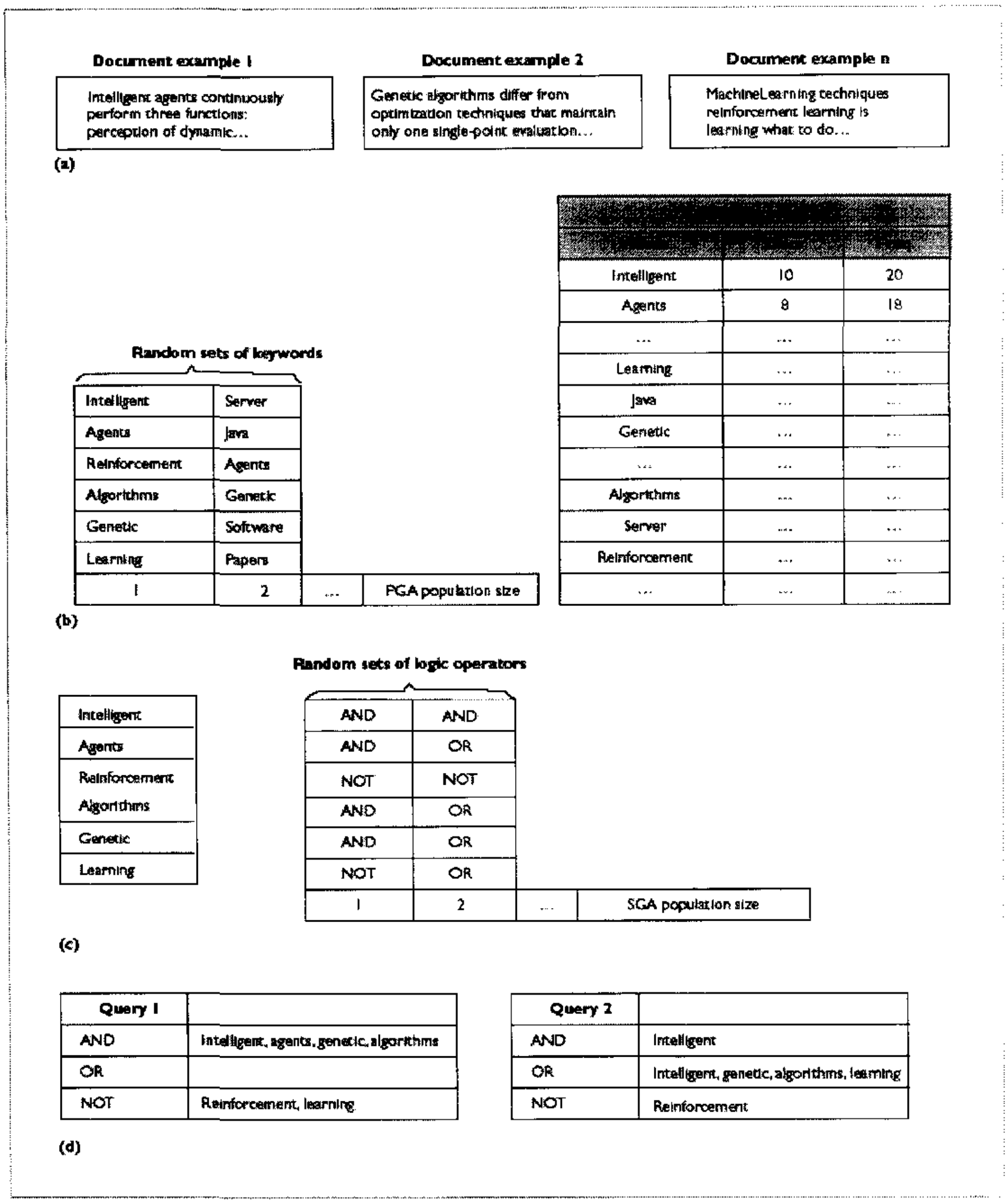
In the next step, we have to search for words in $searchWords array in the keyword table. The following code snippet will return you a list of keyids that matched query. As discussed earlier, if you need to perform Type 1 operation, you have check whether the number of search words and number of records in query. If they are equal, you can proceed to the next step else display search result not found.

The HTML page to get input from user is given below.

```
<html>
<head>
<title>Search Engine</title>
<style type="text/css">

body{ font-size:20; font-weight:bold; font-stretch:semi-expand; font-
family:MSserif; color:#0066CC;
background-color:#EEEEE4;
align:center; background-color:white }
h4{ background-color:#0066CC; color:#FFFFFF; font-family:verdana; }
h3{ color:#0066CC; }
th{ background-color:#6996ED; color:#FFFFFF; font-family:Arial; }
a{text-decoration:none;}
</style>
</head>
<body>
<?php
if($submit)
{
if(!$keywords){
$errmsg="Sorry, Please fill in search field";
form($errmsg);
}else{
//Start Timer
$start = getmicrotime();

//PERFORM SEARCH OPERATION AND DISPLAY RESULT
}else {
//end Timer
$end = getmicrotime();

//TOTAL TIME TAKEN TO DO SEARCH OPERATION
$time_taken=(float)($end-$start);
$time_taken=number_format($time_taken,2,'.','');

echo "<p>Your Query Executed in $time_taken Seconds";

$errmsg="<p>No Search result found for '$keywords'";
echo $errmsg;
echo "<br /><a href=\"#\" onclick=\"history.back()\">Back</a>";
}//endof isset ref
}//end of if key word exists
```

```
} else { //display the form
form($keyword);
} //END OF FORM DISPLAY ?>
</body>
</html>


<?php
function form($errmsg)
{ ?>
<h4 align="center">Search Engine</h4>
<b><?php echo $errmsg; ?></b>
<form method=POST action=<?php echo $PHP_SELF ?>>
Enter keywords to search on:
<input type="text" name="keywords" maxlength="100" />
<input type="submit" name="submit" value="Search" />
</form>
</body>
</html>
<?php
}

function getmicrotime()
{
list($usec,$sec)=explode(" ",microtime());
return ((float)$usec+(float)$sec);
}
?>
```

Function getmicrotime() returns time in microseconds. This function is called during start and end of the search process.



Document example 1

Intelligent agents continuously perform three functions: perception of dynamic...

Document example 2

Genetic algorithms differ from optimization techniques that maintain only one single-point evaluation...

Document example n

MachineLearning techniques reinforcement learning is learning what to do...

(a)

[illegible]	[illegible]	[illegible]
Intelligent	10	20
Agents	8	18
...	...	...
Learning	...	...
Java	...	...
Genetic	...	...
...	...	...
Algorithms	...	...
Server	...	...
Reinforcement	...	...
...	...	...

Random sets of keywords

Intelligent	Server		
Agents	Java		
Reinforcement	Agents		
Algorithms	Genetic		
Genetic	Software		
Learning	Papers		
1	2	...	PGA population size

(b)

Intelligent
Agents
Reinforcement Algorithms
Genetic
Learning

Random sets of logic operators

AND	AND		
AND	OR		
NOT	NOT		
AND	OR		
AND	OR		
NOT	OR		
1	2	...	SGA population size

(c)

Query 1	
AND	Intelligent, agents, genetic, algorithms
OR	
NOT	Reinforcement, learning

Query 2	
AND	Intelligent
OR	Intelligent, genetic, algorithms, learning
NOT	Reinforcement

(d)

Fig-5.1
(Document Processing)

Webnaut combines keywords from the primary genetic algorithm and logic operators from the secondary genetic algorithm to create a number of different queries that are served by the MetaSearch agent to provide the user with appropriate URLs.

(a) Dictionary keywords from the document examples;

(b) Initialization of the primary genetic algorithm;

(c) Initialization of the secondary genetic algorithm;

(d) Queries created by combining member from both populations.

CHAPTER 6

CONCLUSION

Using the genetic algorithm for searching the web documents is an efficient method to improve the retrieval effectiveness. And the search also adapts to the user's interest using the user profiles and their search histories.

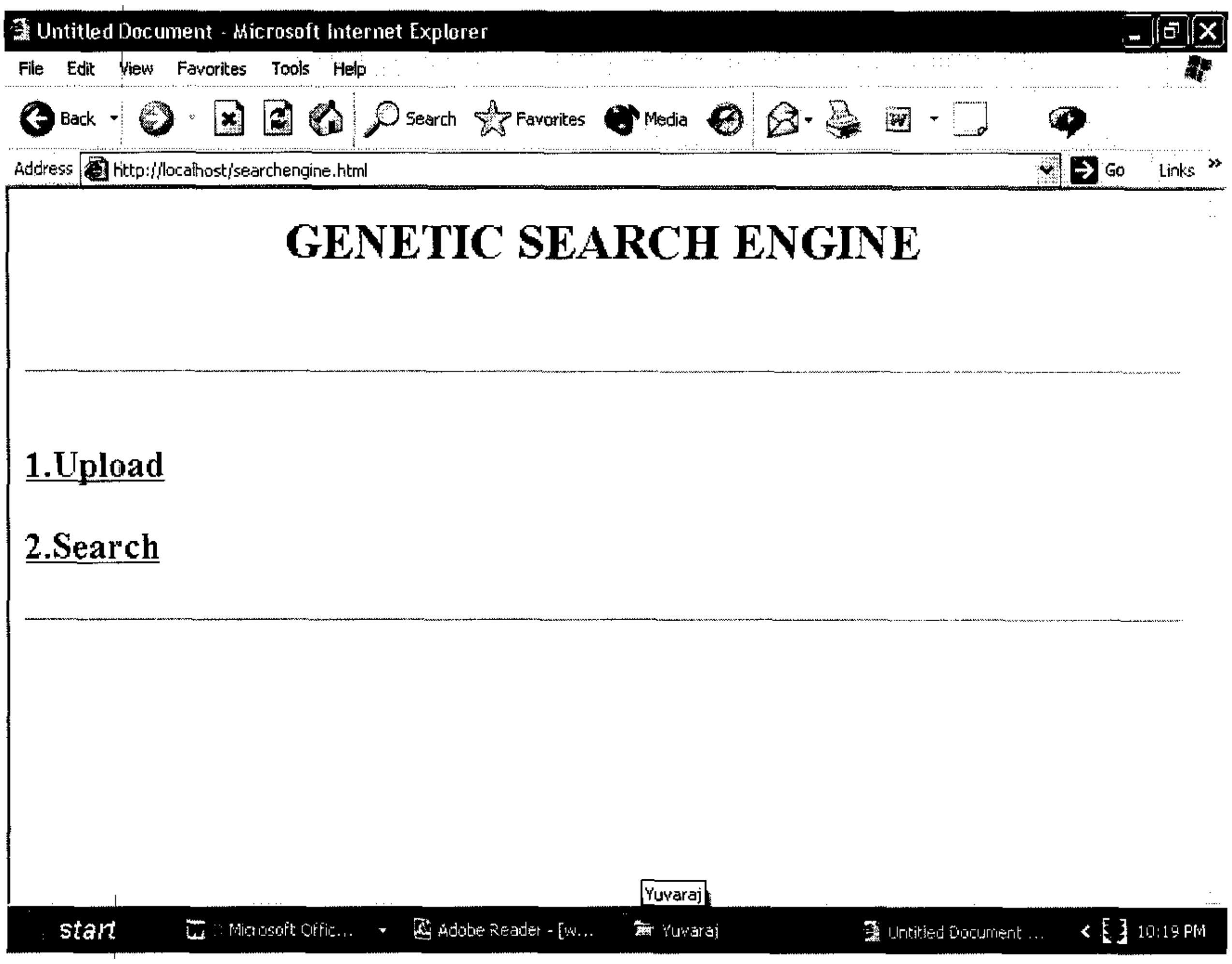
There occur some problems while performing search, which is ubiquitous, so a real time application which eliminates the problem using genetic algorithm should be deployed. Because the agent based filtering system based upon the AI and neural networks had produced good results. Research in agent-based information filtering systems for the Internet or large databases has underscored the need for better feature-extraction algorithms. Such algorithms are especially necessary when information is being culled from an unstructured page with many repeated words, irrelevant information, and nonstandard definitions.

Here the searching is performed in an efficient manner and the searching time is compared with the other search engines. There is some improvement in the search engine when genetic algorithm is implemented.

CHAPTER 7

RESULT

7.1 Home page


Untitled Document - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/searchengine.html
Go Links
start Microsoft Offic... Adobe Reader - [w... Yuvaraj Untitled Document ... 10:19 PM

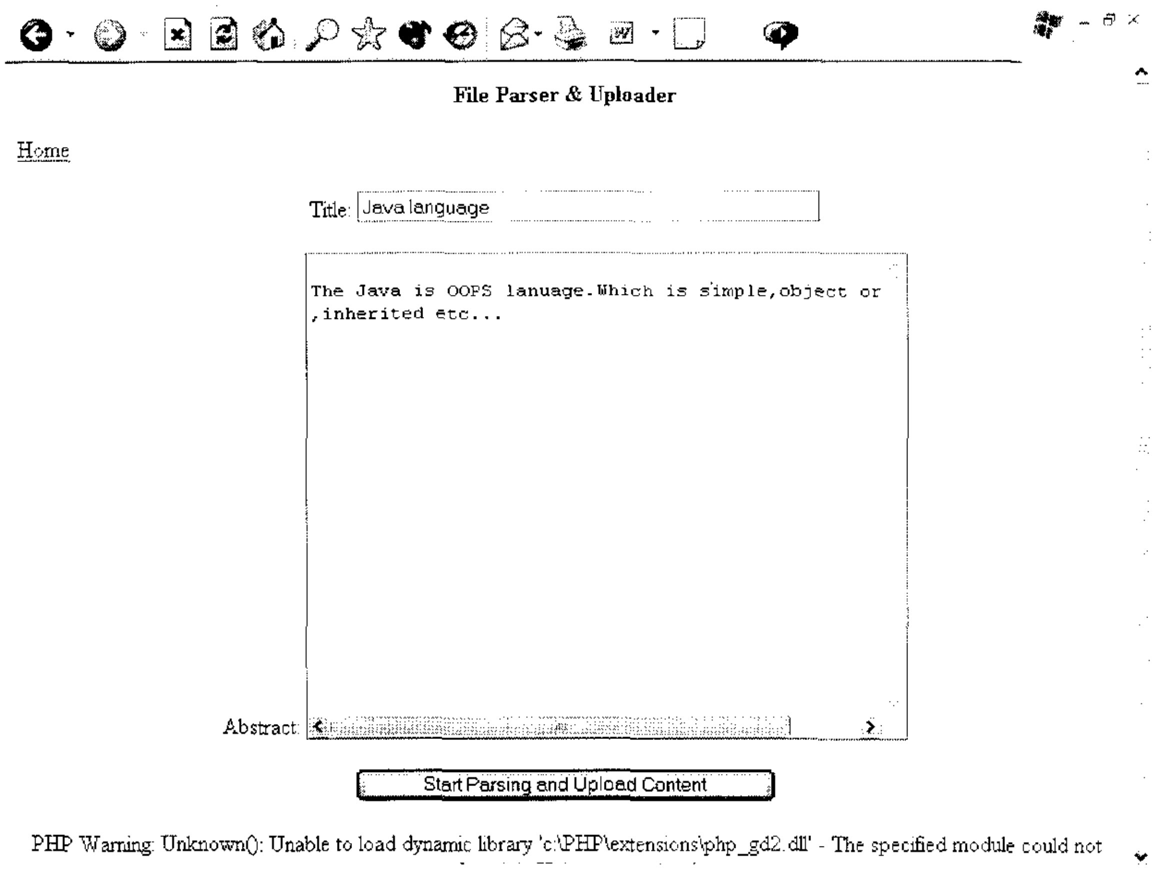
GENETIC SEARCH ENGINE

1.Upload

2.Search

Fig-7.1

7.2 Parser and upload window



Title: Java language

```
The Java is OOPS lanuage.Which is simple,object or
,inherited etc...
```

Abstract:

PHP Warning: Unknown(): Unable to load dynamic library 'c:\PHP\extensions\php_gd2.dll' - The specified module could not

Fig-7.2

7.3 Search Window

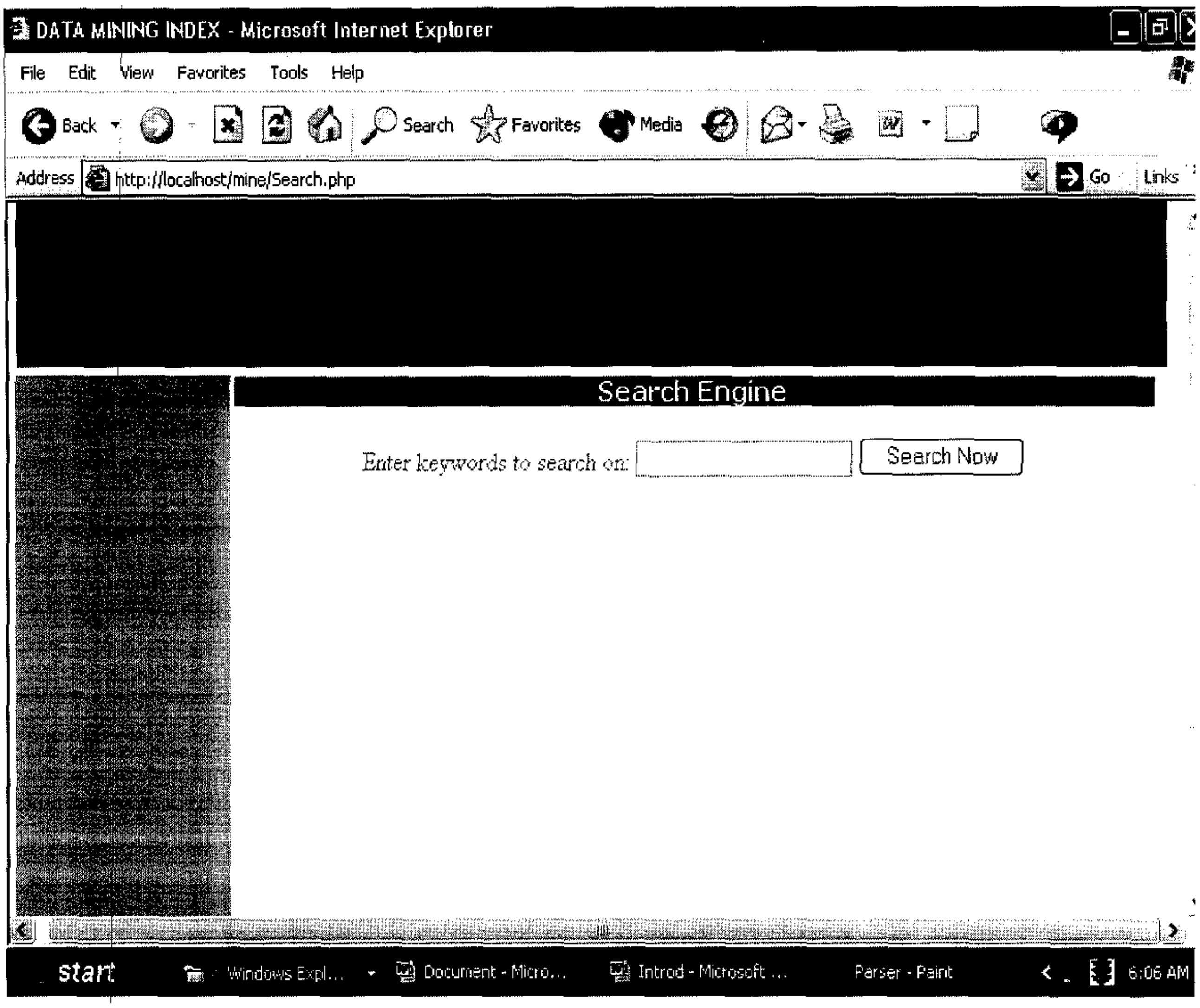
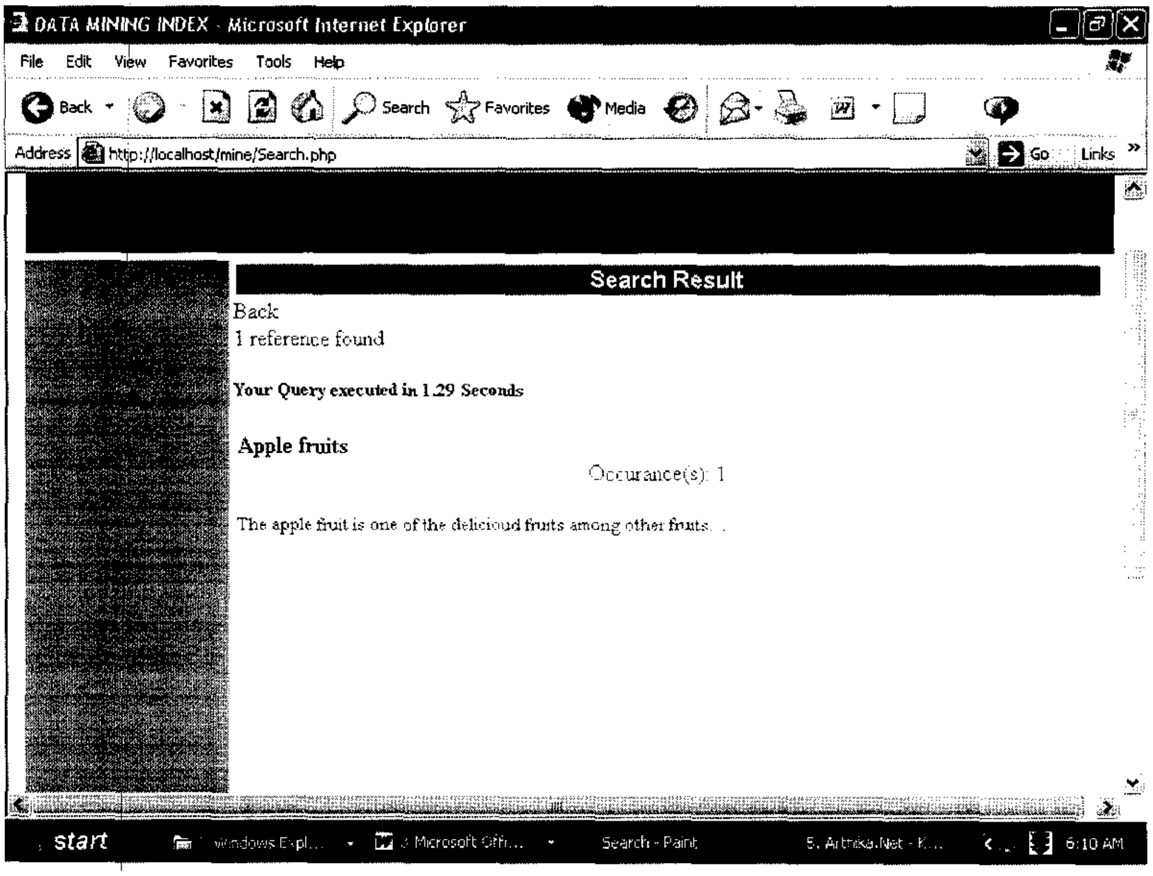

DATA MINING INDEX - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/mine/Search.php
Go Links
Search Engine
Enter keywords to search on:
Search Now
start Windows Expl... Document - Micro... Introd - Microsoft ... Parser - Paint 6:06 AM

Fig-7.3

7.4 Result Window


DATA MINING INDEX - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/mine/Search.php
Go Links
start
6:10 AM

Search Result

Back

1 reference found

Your Query executed in 1.29 Seconds

Apple fruits

Occurance(s): 1

The apple fruit is one of the delicioud fruits among other fruits. .

Fig-7.4

7.5 PERFORMANCE REPORT

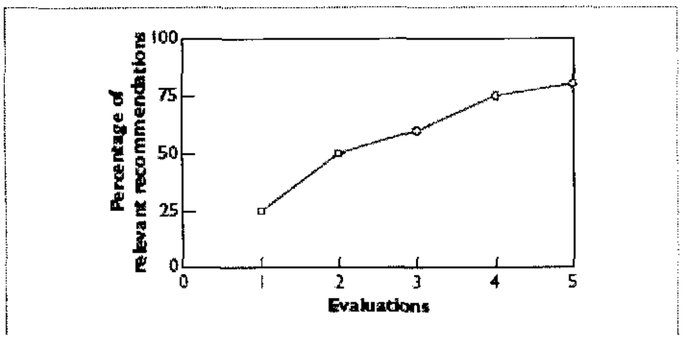


Fig-7.5

Percentage of relevant recommendations per evaluation. In this experiment, a set of 10 pages on Java programming was used for creating user profiles for a group of four users. At the end of the fifth evaluation, on average, 80 percent of the prototype's recommendations were relevant to the user's needs.

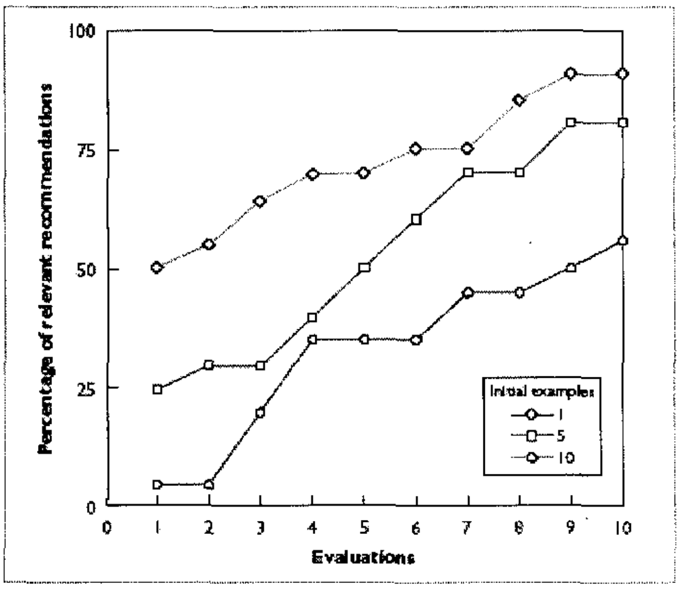


Fig-7.6

Relevant recommendations per evaluation for different sets of related initial examples. In this experiment, the percentage of relevant recommendations increased in proportion to the number of initial examples given and the number of evaluations performed.

CHAPTER 8

RELATED WORKS IN AGENT WEB BASED SERVICES

The artificial intelligence research community has promoted the use of intelligent software to collect information from large, complex, unstructured, and heterogeneous information spaces like the World Wide Web. During the past few years, the notion of software agents has become popular as a means to advise users of the discovery of useful information.1,2 Several agent systems have been proposed to help users with Web browsing3,4 and with filtering Internet news and e-mail messages,5,6 and shopping7 and lifestyle sites.

Many information-retrieval methods, machine-learning algorithms, and heuristics have been applied to the task of creating user profiles. One system similar to Webnaut is Syskill & Webert.9,10 Syskill & Webert recommends pages from a collection of preselected pages or URLs and trains a Bayesian classifier to create a probabilistic profile.The user uses a three-point scale to rate pages on a specific topic, such as biomedical engineering, collected from an index maintained by an expert or group of experts; Syskill & Webert then defines the user profile by analyzing the information on each page.The profile can be used in two ways: to suggest links that might be of interest to the user and to construct a Lycos query that would find such links. Bayesian classifier-based machine-learning techniques require many examples to be effective and are best suited in cases where the user's interests remain constant in a single domain for a long period of time. Furthermore, a Bayesian classifier cannot draw a decision boundary when the user has many interests that do not belong to a single domain.

Webnaut, by contrast, uses machine-learning techniques based on an evolutionary algorithm.The algorithm learns document features that can be combined with logic operators to form rich concepts. Webnaut also makes it easy to explore new areas of interests.

REFERENCES

1. P. Maes,"Agents That Reduce Work and Information Overload," *Comm.ACM*, vol. 37, no. 7, 1994, pp. 30-40.

2. O. Etzioni and D.Weld, "Intelligent Agents on the Internet: Fact, Fiction, and Forecast," *IEEE Expert*, vol. 10, no. 4, 1995, pp. 44-49.

3. R.Armstrong et al., "WebWatcher: A Learning Apprentice for the WWW," *Proc.AAAI Symp. Information Gathering from Heterogeneous, Distributed Environments*, AAAI Tech. Report SS-95-08, Menlo Park, 1995, pp. 6-12.

4. H. Lieberman, "Letitzia: An Agent That Assists Web Browsing," *Proc. 14th Int'l Joint Conf. on AI*, Montreal, Canada, 1995, pp. 924-929.

5. K. Lang,"NewsWeeder: Learning to Filter Netnews," *Proc.12th Int'l Machine Learning Conf.*, Lake Tahoe, Calif., 1995, pp. 331-339.

6. T.R. Payne et al., "Experience with Rule Induction and k-Nearest Neighbor Methods for Interface Agents That Learn," *IEEE Trans. Knowledge and Data Engineering*, vol. 9, no. 2, 1997, pp. 329-335.

7. R.B.Doorendos et al., "A Scalable Comparison-Shopping Agent for the WWW," *Proc. 1st Int'l Conf.Autonomous Agents*, AAAI, Menlo Park, Calif., 1997, pp. 39-48.

8. B. Krulwich, "LifeStyle Finder: Intelligent User Profiling," *AI Magazine*, vol. 18, no.2, 1997, pp. 37-45.

9. M. Pazzani and D. Billsus,"Learning and Revising User Profiles:The Identification of Interesting Web Sites," *Machine Learning*, vol. 27, no. 3, 1997, pp. 313-331.

10. M.Ackerman et al.,"Learning Probabilistic User Profiles:Applications for Finding Interesting Web Sites,Notifying Users of Relevant Changes to Web

Pages and Locating Grant Opportunities," *AI Magazine,* vol. 18,no.2, 1997,pp. 47-56.