INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Université d'Ottawa · University of Ottawa

Information Retrieval Over the World Wide Web

by

© Iyad Zayour

Thesis submitted to the School of Graduate Studies and Research
of the University of Ottawa
in partial fulfillment of the requirements for the
Master's degree in Computer Science
Under the auspices of the
Ottawa-Carleton Institute for Computer Science

Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada
January, 1997

Abstract

The introduction of the World Wide Web and its increasingly growing popularity have created an immense repository of information. The amount of information, its diversity, and its rapid change caused a major problem -- how to use this information effectively and efficiently. Available tools for information retrieval on the Web suffer from many shortcomings in satisfying the information needs for users in a practical way.

In this thesis, we try to address some of these problems and to come up with an efficient solution for large portion of these information needs. In particular, we try to create a solution for users interested in locating facts on popular topics -- topics which are expected to be queried frequently. Our solution is the "Web Information Collector" System (WIC). The WIC is a Web based system which helps in gathering information from the Web, it permits creating and customizing a local document base to store the gathered information, then it provides novel facilities to query, retrieve and navigate this information. The details of this system will be described in the thesis.

Acknowledgments

I would like to thank my supervisor Dr. Douglas Skuce for his support, understanding, and guidance during my work.

Thanks for all other people who helped me, especially Wratko Hlavina for his ideas, Keith White for his troubleshooting, Jamie for his corrections, Judy for her technincal help, and her plants for making the lab an enjoyable place to be.

Table of Contents

Table of Figures

1. Chapter 1 : Overview

1.1 Introduction

A major task confronting information science concerns how to get the information you want in a practically useful way. In general, we can divide information, from a computer processing perspective, into structured (stored in database and managed by a Data Base Management System) or unstructured information. The latter is mainly stored in text documents. Text is the primary way that human knowledge is stored [Harmon 92].

The theme of this research is about the management of textual information with special emphasis on the World Wide Web. This kind of work is usually called Information Retrieval (IR). By definition, information retrieval is the subfield of computer science that deals with the automated storage and retrieval of documents [Baeza-Yates & Frakes 92].

1.2 The classical information retrieval models

1.2.1 A brief history

Information retrieval research has been around for about 40 years [Salton & McGill 83]. In the early sixties, much theoretical work was done on information retrieval, and many techniques, systems and models have been proposed. In the mid-seventies a number of mathematically sophisticated information retrieval models were defined such as the "vector space" and "probabilistic" models for retrieval. Some significant experimental methodology for information retrieval also emerged at this stage. One of them was the "relevance feedback" technique which was an effective way of enhancing retrieval. These developments, both theoretical and experimental continued into the eighties. However, at this stage, there was a lack of theoretically consolidated models proposed and the trend was to integrate different models and evaluation methods into hybrid systems to achieve better performance. In the nineties, there was more tendency toward AI technology such

as knowledge-based and machine learning approaches, intermediary expert systems, and natural language processing techniques (NLP). In recent years, Internet-based information retrieval has become a focus [Zeng 96].

1.2.2 A general procedure for information retrieval

A general information retrieval procedure can be described by the following steps :

1. Gathering: a document base -- the collection of textual documents-- has to be gathered first. This document base is the pool of information which needs to be managed and used effectively in order to satisfy the information requirement.

2. Indexing: This may be carried out manually or automatically. Manual indexing -- the process of having humans selecting the terms to be indexed -- is an extremely time consuming task, and usually only to be conducted by experienced and expensive experts. Automated indexing involves scanning the text and constructing an associated data structure (the index) aimed at making computer access of information more efficient.

3. Querying: Before a user conducts a search, he or she has some more or less well defined purpose for seeking information in the collection. The query is the description of the information need, and is usually expressed in forms which the system can understand such as Boolean expressions, key word phrases, or even natural language sentences.

4. Retrieval and evaluation: The match between the query and the index terms or keywords leads to the selection of possibly relevant retrieved texts. These are evaluated or used, and either the user will leave the information retrieval system, or the evaluation leads to some modification of the query and/or the information need. The process of query modification through user evaluation is known as "relevance feedback" in information retrieval. It involves an iterative search process where search queries are progressively redefined based on feedback from the document base.

1.2.3 Recall and precision

Traditionally, the retrieval effectiveness is measured by a pair of terms known as "recall" and "precision" [Zeng 96]. Recall is defined as the number of relevant documents actually retrieved from the document base divided by the total number of relevant documents (retrieved and not retrieved) in the document base. Precision is defined as the number of relevant and retrieved documents divided by the total of retrieved documents.

In principle, a search should achieve high recall by retrieving almost everything that is relevant while at the same time maintaining high precision by rejecting most of the irrelevant documents. In practice, however, it is known that the recall and precision tend to vary inversely, and that it is difficult to retrieve everything that is wanted while also rejecting everything that is unwanted.

1.2.4 Major information retrieval models

Throughout the history of information retrieval, two models have proved to be of most practical value. The first is the Boolean retrieval model which is the standard model for current large-scale, commercial information retrieval systems [Salton & McGill 83]. The second is the vector space model which was developed by Salton. This model was implemented in the SMART system and has attracted a lot of attention in the information retrieval community [Daoud 93].

1. Boolean Retrieval model: the earliest retrieval model, based on the concept of an exact match (as opposed to probabilistic inexact match) of a query specification with the index of text documents. The term "Boolean" is used because the query specifications are expressed as words or phrases, combined using the standard operators of Boolean logic e.g.("term1" *and* "term2" *or* "term3"). The system parses the query, retrieves documents which match the query terms and then performs logical operations corresponding to the logical operator of the query.

A major problem with this model is that it does not allow for any form of relevance ranking of the retrieved document set, thus all retrieved documents are presumed to be equally good, or equally poor for the user. Another problem is that the standard Boolean retrieval methodology does not provide any direct control over the size of the output: some query statements may provide no output at all, whereas other statements provide an unmanageably large number of retrieved documents.

2. "vector space" model: This model treats documents and queries as vectors in a multidimensional space, the dimensions of which are the words used to represent the documents. Queries and documents are compared by comparing the vectors using similarity measures, and the retrieved documents can be presented to the user in decreasing order of the query-document similarity [Daoud 93]. The assumption is that the more similar a vector representing a document is to a query vector, the more likely that the document is relevant to that query.

In the latter model, an important refinement is that the terms of a query, or document representation, can be weighted, to take account of their importance. These weights are computed on the basis of the statistical distributions of the terms in the database, and in the document.

1.3 Motivation

Recently, due to the introduction of the World Wide Web, interest in information retrieval has grown. More and more people are using the Web as a major source of information. Also the amount of available information on the Web is doubling in size every four months according to some estimates [Venditto 96]. This leads to higher demands on efficient and effective ways to locate and use this information.

Getting the required information from the Web in a feasible way seems to be a major problem confronting the efficient exploitation of the Web resources. The immense

amount of the information available on the Web, its rapid growth, and its open and uncontrollable nature are all factors which emphasis this problem.

Currently, there are two dominant approaches to alleviate this problem. First, we have resource lists which are a kind of pre-compiled index (arranged by topics) pointing to useful resources. This can take a simple form like a file of bookmarks or a more advanced form like a hierarchical structure supplemented by searching capabilities called virtual libraries e.g. "Yahoo[1]" (discussed in Chapter 4) . This approach is far from being a comprehensive solution as it depends on manual efforts to compile resources. Thus, it is very difficult to keep up with dynamic nature of the Web as there are continuously new resources which are posted and others are eliminated.

The second approach is that of the Web search engine e.g. Alta Vista or OpenText. Search engines are the most popular method to locate information on the Web. In general, search engines are similar to the classical information retrieval systems as they use Boolean queries and return whole documents as the result but may incorporate some additional features e.g. ranking. Although they deliver a more comprehensive solution than virtual libraries, search engines suffer from major shortcomings inherited from the classical model; in particular, they suffer from the low precision problem. This problem is especially amplified on the Web compared to the classical model where its queried information (the document base) belongs to a controllable domain and is manageable in size. A typical result of a Web search engine would contain hundreds or even thousands of all sorts of documents from different domains which happen to contain terms satisfying the query. The only available way to locate the required information is to manually go through each document. This approach is a very time consuming process especially due to the network delays in navigation inherited from the Web nature.

---

[1] Yahoo is accessible at : http://www.yahoo.com

These shortcomings of search engines lead to a serious under-usage of the available information on the Web. The current retrieval process caused a large portion of the users to exclude the Web as an alternative for some kinds of information requests.

Some users might be interested in a small piece of information like a definition of a concept. In fact, t is likely that most information system users are interested in facts rather than in documents that must be studied before the needed information can be extracted [Salton & McGill 83]. Currently, such users even after they locate relevant documents, they have to spend considerable time skimming the whole documenst in order to locate the needed information which is a time consuming process.

It is obvious that the available approaches for the Web information exploitation are inadequate to constitute a practical solution for the effective use of the Web information resources because they are either very time consuming or non comprehensive. In particular, these approaches are totally unfeasible for the situation where users are interested in facts rather than in complete documents. A new approach is necessary to address these problems.

1.4 Contribution

This thesis will present a step towards alleviating the problem of efficient exploitation of information on the Web. We focused on the task of locating facts in a practical way within a large amount of text. Although applied in conjunction with the Web, most of the introduced techniques could also be applied on different models such as intranet or local on-line documentation.

We have designed and implemented a system called the Web Information Collector (WIC) which is intended to overcome some of the shortcomings of available Web information tools. The system is based on four key ideas:

1. To build a local document base collected from the Web in order to be a repository of relevant information. The repository will support further customized retrieval.

2. To refine the granularity of retrieval from a document (as in the classical model) to the paragraph level. This is expected to significantly increase precision and thus minimize the human time and effort necessary to locate relevant information [Croft 92]. We have also included features to facilitate the result navigation such as bolding matches, showing surrounding of paragraphs and the source documents of the paragraphs.

3. To develop a paragraph ranking algorithm which approximates the relevance of the returned paragraphs and sorts them accordingly. We believe that the synergy of these ideas would make the task of using information of the Web more efficient in many situations.

4. To embed and integrate our system (the WIC) into another general knowledge management system called IKARUS (discussed in Chapter 6). It will be possible to invoke the WIC from within IKARUS and its results would be used to add information to the IKARUS knowledge base. Also, IKARUS should help enhancing the WIC performance by supplementing it with different facilities.

We began this work by surveying the literature to determine the most efficient implementation for the local retrieval part of the system. We have done some analysis for each information retrieval technique from the data structure and algorithm perspective. Also, we have surveyed and analyzed the prominent available information retrieval tools for the Web.

1.5 Who could use the system?

The Web Information Collector is intended to be used by any person who wants to locate information and facts from the Web (or originated from the Web). However, we think

that it will be most useful for those who have a repeated need to locate facts in a particular domain. This is because the more frequently the locally saved data is queried (and returns satisfactory results) the more practical becomes using the system. A good example is the software engineer who needs facts about a programming language or operating system he or she is using.

1.6 Organization

Chapter 2 is a survey of the major information retrieval techniques. The survey concentrates on efficiency considerations. We will describe the data structure and algorithm for each surveyed technique accompanied with some analysis and evaluation. The analysis will be focused on what is crucial for the implementation of our system.

In Chapter 3 we will introduce the partial indexing technique which is what we have chosen for our system. This chapter includes a description of partial indexing, how it works and some empirical efficiency analysis.

Chapter 4 introduces the basic concepts and a brief history of the World Wide Web. Next, it discusses the Web as an information model, a hybrid between classical information retrieval and hypertext model. Research ideas and available tools related to information management on the Web will be surveyed.

Chapter 5 will describe the Web Information Collector system. First the system will be described from a functional perspective addition to some implementation details. Following, experiments carried out on the WIC will be described and results would be analyzed.

In chapter 6 there will be an evaluation of the WIC and for its development We will describe how much it achieved its intended goals, its shortcomings and a comparison with Web search engines. Finally, there will be some suggestions for enhancements to be done in the future.

2. Chapter 2 : Information retrieval techniques

This chapter surveys and analyzes the major techniques for information retrieval from the algorithm and data structure perspective.

2.1 Motivation

Most of the existing literature is driven by the classical information retrieval assumptions, some of which are not valid for our approach. The first assumption is that the document base is static or rarely updated, therefore it is acceptable for the indexing time to be significantly long. This is not true for our approach as we are building an information retrieval system that would be continuously updated from the Web. Note that, in the literature, emphasis is done on searching time and space requirements while the index construction time is poorly studied.

The second invalid classical assumption concerns the expected size of the document base. We design our system as to be open to the Web, making it less necessary to store huge amounts of information. The stored information would be focused, only what is expected to be used frequently would be stored. At any time, the user can dynamically gather more data to satisfy his or her information needs. Hence, the local collection is expected to be smaller in size than in classical information retrieval.

The third assumption in question concerns the number of terms per query. We expect to be dealing with a relatively large number of terms per query as we envision that this work will be a part of larger and more sophisticated system which involves query expansion--adding synonyms and conceptually related terms (see Chapter 6 on IKARUS). Query expansion yields a significant number of terms, enough to affect the performance of search and retrieval [Rabbati & Sizka 84]. So we want to look for an alternative which handles this aspect in an optimal way.

Finally, we wanted to make sure that our system would be able to handle properly querying for compound terms -- a term made of more than one word but denoting a single concept, e.g. "operating system". We believe that the compound terms are very important for enhancing precision in information retrieval.

To get all our distinct requirements in the most efficient way, we had to examine the major available information retrieval techniques. In this chapter we will shed a light on the basic concepts, advantages and disadvantages of each technique emphasizing the aspects which are considered critical for an efficient implementation of our system.

2.2 Information retrieval techniques

In information retrieval, text accessing methods can be divided in general into two categories: Full text scanning and index based access. Full text scanning is the trivial way to locate occurrence of strings in the document base. It is based on sequentially comparing the strings of terms with the corresponding strings in text.

In general, full text scanning has a poor response time -- O(n) where n is the number of text characters--, therefore it is too slow to be considered an acceptable alternative for information retrieval. Yet, there are still situations where full text scanning is used such as when applied on special purpose hardware or when used in cooperation with index based method that restricts the scope of searching [Ciaccia & Zezula 93].

On the other hand, virtually all commercial (non-experimental) information retrieval systems are based on an index file [Salton & McGill 83]. The classical information retrieval system consists of two data structures: a database of unstructured text documents, often concatenated in one long text file (we use the term document base) and the index file.

The indexing technique is what normally determines the efficiency of a particular technique. When we talk about information retrieval techniques from the data structure and algorithms perspective, we will be actually talking about the index data structure and its associated searching and construction algorithms.

2.2.1 Introduction to indexing

Generally defined, an index is a mechanism for efficiently locating a given term in a text [Bell 94]. In the context of computer science, an index is a data structure built by preprocessing the text. This data structure, acting as a surrogate for the text, should allow more rapid term location than in full text scanning.

Indexes are mainly classified as either lexicographic indices -- based on dictionaries of terms -- or indices based on hashing [Harmon 92]. Other attributes which categorize indexes are the granularity of the pointers (sparse versus full or dense) and the coverage of the index relative to the lexicon -- a list of all terms that appear in the document base.

The granularity of the index is defined as the resolution to which term locations are recorded within each document. Depending on granularity we have a sparse index or a dense index. An increase in granularity means a corresponding increase in storage requirement e.g. an up-to- word granularity index will need more than twice the storage required for a document granularity [Bell 94]. Because coarse granularity does not return the exact position of a term within the text, text scanning is usually used to scan the chunk of text pointed to by the index in order to determine the exact location of the term occurrence.

An index which contains each term occurring in the text is called a full text index. Usually, even the full text index has what is called a stop word list-- a list of very frequently occurring words such as "the" and "of" in English language. Words belonging to the stop lists would not be indexed. In non full text index, one approach is to build a dictionary of terms where only those terms found in the dictionary will be indexed.

2.2.2 Inverted file

This is the most popular type of index. The inverted file is a kind of lexicographic index [Harmon 92]. An inverted file contains for each term in the lexicon an inverted file entry that stores a list of pointers to all occurrences of that term in the main text of the document base [Bell p79].

In inverted files, the pointers are usually either of low granularity -- point to the document which contain the term -- or of high granularity -- point to the occurrence of term within a line of a document. Figure 1 shows a data structure for an information retrieval system based on inverted files. The main index file contains the indexed terms, the number of occurrences of the term (hits) and a pointer to a posting file location. The posting file stores the pointers to the term occurrences in the document file.

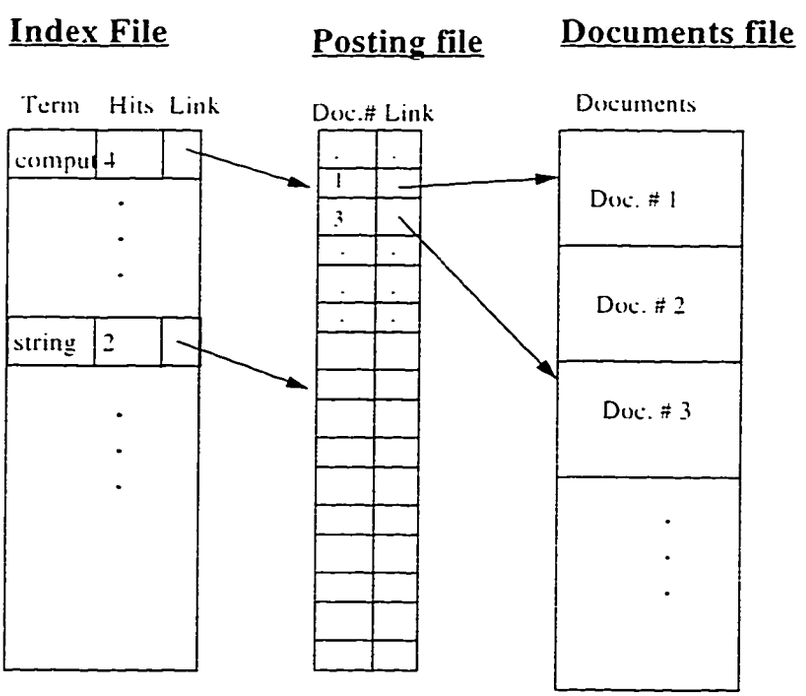


Figure 1: An inverted file model.

In general, an inverted file has two kinds of implementation, as a sorted array or as a B-tree. The sorted array is the simplest and most popular implementation of the inverted file. The list of keywords is stored in a sorted array. Each array entry also contains a pointer to the list of occurrences of the term. This array is searched usually using binary search, although some systems adopt the search for secondary storage considerations. The main disadvantage of this approach is that updating the array is very expensive. On the other hand, it is easy to implement and reasonably fast, the search time is O(log n).

The other implementation structure for an inverted file is as a B-tree. A B-tree is a multiway balanced tree optimized for secondary storage. The advantage of B-trees over arrays is that it is more efficient for frequently updated data and it provides faster search than the array especially for secondary storage. However, it requires more space than the array implementation and is much more complex to implement.

2.2.2.1 Advantages & Disadvantage

The big advantage of inverted files is that they are very fast to search -- they provide a logarithmic search time.

However, they suffer from many shortcomings:

They need a storage overhead varying from 30% to 100% depending on the data structure, the use of stop words and the granularity.

Index construction -- the creation of the inverted files-- is very time consuming. The indexing process involves:

a) Lexical analysis which is the process of extracting individuals words or token.

b) Sorting of extracted words which is very costly on secondary storage.

c) Term duplication removal.

Inverted files are not suitable for document bases which are frequently modified. Most of the time --especially in the array implementation-- a full index regeneration is needed when the document base is modified.

An inverted file does not provide any optimization for the case of a query containing many terms. Documents containing each individual query term are retrieved separately into distinct lists. These lists would be next merged (union) then and/or intersected corresponding to the logical operators of the query.

2.2.3 PAT structure

The PAT is a new lexicographic index for text [Gonnet 92]. It is a data structure that allows very efficient searching after preprocessing the text. Several advanced search problems such as approximate matching and regular expression could be implemented on this data structure very efficiently.

PAT views the text as one long string, instead of a collection of keywords in record. Each position in the text corresponds to a semi-infinite string-- the string that starts at that positions and extends to the end of the text.

2.2.3.1 PAT tree

The first obvious implementation of a PAT structure is as a Patricia tree, and this implementation is called PAT tree. A Patricia tree is a digital binary trie with the additional constraint that single-descendant nodes are eliminated. Tries, in general, are recursive tree structures that use the digital decomposition of strings to represent a set of strings to direct the searching. The root of the trie uses the first character, the children of the root use the second char, and so on [Baeza-Yates 92].

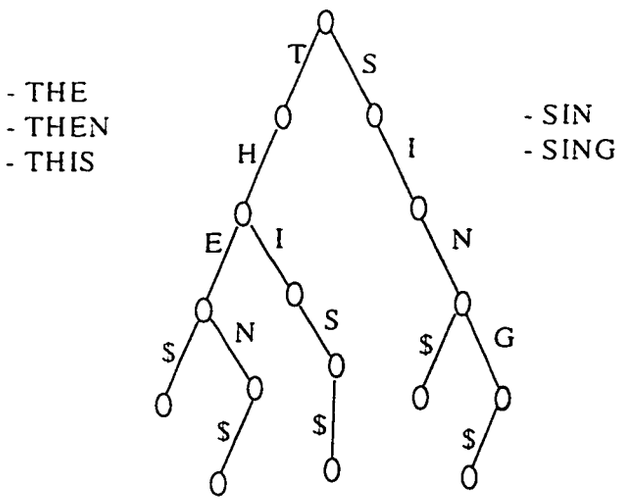


Figure 2: a trie data structure storing words

A PAT tree is a Patricia tree constructed over all the possible sistrings (semi strings) of a text. In Patricia trees the individual bits of the keys are used to decide on the branching. In addition, Patricia trees contain 2 types of nodes: internal and external. For a text of size n (char) there are n external nodes and n-1 internal nodes. Internal nodes contain an indication of which bit of the query is to be used for branching; while external nodes contains a pointer to the keyword. In PAT these pointers are integer displacements corresponding to sistrings.

2.2.3.2 PAT array

The PAT array, also called "structure suffix array", was discovered in 1990 by Manber & Myers. It is an efficient implementation of a PAT structure. A PAT array is much more efficient in term of construction time and in terms of storage requirements than the tree implementation. However, PAT arrays perform much worse in some kinds of special search such as the regular expression search which works very elegantly on the PAT tree.

For this search, for example, the time increases when applied on an array by O(log n) versus the time when using a tree.

2.2.3.3 Advantages & Disadvantages

PAT trees are very fast for search; the search time is proportional to the length of the keywords. On the other hand, PAT trees are very expensive in terms of storage space. It takes 18n bytes to index n locations. It is possible, though, to ameliorate this result through several techniques such as bucketing of external nodes and limiting the indexing points e.g. indexing only the beginning of words instead of each character. In terms of construction time, indexing is generally expensive, but modifications for the document base are handled efficiently.

The big plus of PAT structures is their use in other kind of searches that are either difficult or very inefficient over inverted files. These searches include searching for phrases (especially those containing frequently occurring words), multiple terms query, prefix search, range search, longest repetition search, regular expression search, most significant search, approximate string searching [Gonnet 92].

2.2.4 Bitmap

A Bitmap is a very simple indexing structure [Bell 94]. For every term in the lexicon, a bit vector is stored where each bit corresponds to a document. A bit is set to one if the term appears anywhere in that document, and zero otherwise.

Bitmaps are particularly efficient for answering Boolean queries--faster query processing than inverted files. To resolve this kind of query, the bit vectors for the terms are simply combined using the appropriate Boolean operations.

Bitmaps are fast, easy to use and implement, but extravagant in storage: for a text of N documents and n distinct words, a Bitmap occupies Nn bits. Due to their storage requirements Bitmaps are rarely used nowadays unless in specialized applications.

2.2.5 Signature files

Signature files are based on the idea of an inexact filter, they provide a quick test which discards many of the non-qualifying items. The qualifying items will definitely pass the test, some additional items may pass it accidentally. In this sense, a signature file is a probabilistic method for indexing text.

In a signature file index, each document has an associated signature or descriptor (the index), in which every indexed term is used to generate several hash values, and the bits of the signature corresponding to those hash values are set to one [Bell 94].

More precisely, each text document is divided into logical blocks-- pieces of text that contain a constant number of distinct, uncommon words. Each word yields a "word signature" which is a bit pattern of a fixed size. The word signatures are OR'ed together to form the block signature. Block signatures are concatenated to form the document signature. Searching for a term is handled by creating the signature of the term and examining each block signature for similar signature pattern [Faloutsos 92].

The documents are stored sequentially in the "text file". Their "signatures" (the hash-coded bit patterns) are stored in the "signature file". When a query is issued, the signature file is scanned and many non qualifying documents are discarded. The rest are either checked in order to eliminate the accidentally qualified items (called "false drops") or they are returned to the user as they are.

2.2.5.1 Advantages & Disadvantages

The signature-based methods are much faster than full text scanning (1 or 2 orders of magnitude faster depending on the individual method). Compared to inversion, they require a modest space overhead (typically 10 -15%). Moreover, they can handle insertions more easily than inverted files, because they need "append-only" operations - no reorganization or rewriting of any portion of the signatures.

On the other hand, signature files may be slow for large document bases. Particularly, because their search time is linear -- the signature file is linearly scanned to locate a term [Falotsous 92]. In addition, false drops cause unnecessary access to the text which is a slow process because it involves random disk access. Typically, false drops are not frequent but they do occur.

Searching for compound terms is done quite efficiently because signatures preserve the notion of proximity between successive words. Multiple terms queries are also efficient especially in cases where terms are related by conjunction. The terms of a query can be OR'ed into one bit pattern and compared with signature by a single operation. On match, individual terms are compared to minimize false drops.

2.2.6 Hash tables

Hashing is an information retrieval strategy for providing efficient access to information based on keys [Wartik 92]. In general, to hash is to define and implement a mapping from a domain of keys to a domain of locations. The domain of keys can be any data type. The domain of locations is usually the m integers between 0 and m-1 which corresponds to locations in the hash table-- an array of m locations called buckets.

Whenever hashing is used two parameters must be determined. First, the hash function (the mapping function between the key and the entry of the hash table) has to be chosen with respect to certain considerations like the type of data to be hashed. The second

parameter is to decide upon is the collision resolution strategy -- what to do when collisions occur.

In the hashing process, collisions occur when two or more keys map to the same location. Whenever a collision occurs, some extra computation is necessary to further determine a unique location for the key. Collisions therefore degrade performance. Performance of hashing is measured relative to two facets: number of collisions and amount of unused storage. Optimization of one occurs at the expense of other.

To handle collisions, two strategies are used. First, there is chained hashing, so named because each bucket stores a linked list (a chain) of key information. If many keys are mapped to the same bucket they will be placed in the list associated with the bucket. At retrieval time, the list is scanned for the key. Second, we have open addressing which is most suitable when the likelihood that the number of keys exceeds the number of buckets is low. When a key is mapped to an empty bucket the key information will be stored in the bucket. If the bucket is not empty, another bucket (within the table) will be selected. This process, called probing, repeats until an empty bucket is found.

Finally, note that in cases where the number of keys is not large, it may be possible to have a hash function which does not yield collisions. Such a function is called a "perfect hash function" [Daoud 93].

2.2.6.1 Advantages & Disadvantages

Under many conditions, hashing is effective both in time and space. Information can usually be accessed in constant time (although the worst-case performance can be quite poor).

Hashing is a very fast way to search (ideally search time can be constant). Space usage is not optimal but at least acceptable in most situation. Indexing time for hashing (creating the hash table)is acceptable. It involves lexical analysis and hashing. Updating the hash table does not need any more effort than for indexing it the first time. Terms can be added

or removed without affecting the existing ones because hashing is normally not order preserving.

Yet, hashing is not widely used in information retrieval because in information retrieval the expected number of keys is very large, which increases collisions. Also, hashing is data dependent i.e. extra effort should be exerted on creating or finding the hashing function which would be appropriate to a particular kind of data. Relevant factors which affect the nature of the hash function include some knowledge about the domain information regarding the number of keys that will be stored, and stability of data. Lack of such knowledge can lead to large performance fluctuations [Faloutsos 92].

2.2.7 Summary and comparison

From the above discussion, it is evident that there is no technique which is the best for all situations. Each technique favors some aspects of performance at the expense of other aspects.

Inverted files are suitable for a static (or rarely updated) document base which tends to be large in space requirements. Choosing inverted files is best when storage requirement is a second priority after the search time which is in this case satisfactory.

Signature files, on the other hand, favor storage space over searching time. They have a small index size, an efficient indexing time (linear) but a slow response time (also linear). They are ideal where the document base is frequently incremented (appending new documents). Their ability to handle multiple query terms and their ability to preserve the notion of proximity between words is a significant plus.

PAT structures are very promising and the exploitation of their full potential is still an open question [Gonnet 92]. PAT encompasses features such as approximate matching, regular expression and compound term search which was before exclusive to full text

scanning algorithms. Also, PAT handles incremental indexing and multiple query terms at an acceptable efficiency level.

Unfortunately PAT trees are extravagant in space (18 char for each indexing position) and consequently very bad in indexing time. The array implementation, on the other hand, compromises the advanced feature of the tree and its efficiency in favor of less storage requirement. The PAT array is a very promising data structure but still in its infancy, algorithms for the array are still evolving.

Hash tables are the most efficient alternative for locating terms rapidly, presuming that certain condition which minimize collisions can be satisfied. Minimizing collisions is a major and problematic concern. A good hash table performance is expected only when the number of indexed terms is not large and the type of data (the text) is homogenous as collisions depends on these two factors.

	Building	Search	Index Update	Space
Inverted Array	Slow	Fast	Costly	30-100 %[2]
Inverted B-tree	Slow	Very fast	Acceptable	> array
PAT tree	Very slow	Very fast	Acceptable	1800 %[3]
PAT array	Acceptable	Fast	Costly	400 %[4]
Bitmap	Slow	Very fast	Costly	Extravagant
Hash table	Fast	Very fast	Very good	Acceptable
Signature	Fast	Acceptable	Good[5]	10-15%

Table 1: Performance comparison of information retrieval techniques

[2] The percentage is relative to the size of the text to be indexed.

[3] Assuming all characters are indexed.

[4] Assuming all characters are indexed.

[5] The most efficient update is when appending new documents

3. Chapter 3 : The Choice: Partial Indexing

In this chapter we will describe the Partial Indexing technique, our choice for WIC. This chapter also includes a description of some empirical analysis on this kind of indexing.

3.1 Efficiency analysis question

In the previous chapter we described in general the indexing techniques and their suitability for our requirements. From the analysis in Chapter 2, no technique clearly appears have enough advantages to be an ideal candidate for our system. In order to have a more accurate assessment, we need to have a better insight on what affects performance form an efficiency perspective.

3.1.1 Theoretical complexity

We are used to estimate the performance of algorithms by finding their asymptotic growth rate or what is called the order function. Although this is a good indication of how in general, the algorithms would behave, in practice this is not enough. The major shortcoming of the asymptotic analysis is that it is too vague. It only tells to what class of performance a certain algorithm belongs. The order function notation hides, among other things, the coefficient of the function or the constant which might be of major significance. For example, in our case most algorithms are approximately of the same class of performance; still practically they are significantly different. Furthermore, the order notation approximates only the CPU processing time ignoring other vital performance parameters related to hardware such as the I/O time.

3.1.2 Empirical analysis

Contrasting theory, we have empirical analysis or analysis by experimentation. Empirical analysis should give an accurate assessment because they return exact numbers. However, this is only true when the theoretical foundations are well understood. That is, experiments are generally useful when they can be used to prove (or disprove) theoretical hypothesis or when one can

perform generalization on the results. This is why there should be a good understanding of all given parameters affecting the results before the experiment is carried out in order for conclusions to be reliable. In particular, we think that it is very important to have a good understanding of the final contribution of the different hardware components in the performance in general.

3.2 Hardware parameters in efficiency analysis

Throughout the history of computer hardware, there has been a non-parallel evolution of different hardware components. Practically, performance is heavily dependent on hardware. As one element of hardware gets faster and cheaper, we may pay less attention to optimizing the performance aspects related to this particular element and rather rely on its abundance to achieve the required efficiency.

The three main hardware elements of computing systems which affect performance are: the CPU speed, the main memory size, and the secondary memory speed (disk). Performance is the result of the combination of these three elements. Over the past 30 years the gap of performance and price continue to expand between these elements – some elements improve faster than the others. It is estimated that the speed of a processor doubles each 1.5 year while the speed of I/O (related to disks) increases much slower [Stalling 95]. Although never considered as a bottleneck, still main memory also has a slow improvement in respect to access speed . Hence the gap in term of price and speed between CPU and main memory on one hand and the disk on the other hand continues to expand. The result is that disk access has become the real bottleneck in computing systems

Nowadays, with the current hardware configurations, the size of the gap between a memory access and a disk access is so significant that it changes the traditional view of the performance analysis. In one instance, a group of researchers working on an indexing algorithm for the PAT structure at the university of Waterloo have shown that a O(n2) algorithm requiring fewer disk access has scored a better performance than another O(nlogn) algorithm requiring more disk access even when applied on large n -- a collection of 500 MB of text [Gonnet 92]. This can be

easily justified by the fact that a good computer can perform about 30 disk access per second: compare that to 10 nanoseconds to access a register and 50 nanoseconds to access the main memory [Stalling 95].

This clearly marks the emergence of a threshold in performance: does the working context of an algorithm fit or does not fit in memory?. In other words, this single parameter may decide the acceptability of a certain algorithm. This implies that in order to assess any algorithm, we should look first at this parameter which may be enough to qualify or disqualify the algorithm. Having an algorithm that requires minimal disk access means that it is very likely to fall into an acceptable range of performance. Otherwise, a more rigorous analysis is needed where a better performance largely depends on tuning the disk access load.

3.3 The Choice: Partial Indexing

From the above discussion we can conclude that due to the hardware limitations, we need an indexing mechanism which requires minimum disk referencing. This observation is exactly what inspired Udi Manber to introduce the "Partial Indexing" technique and implement it in the Glimpse system [Manber 94]. We have chosen Glimpse (and consequently the Partial Indexing method) to be the local search engine for our system. It satisfies most of our requirements particularly for its incremental indexing capabilities (adding file or removing it without total reindexing) and its elegant handling of multiple query terms. Next we will discuss the Partial Indexing techniques and how it works in general.

3.3.1 Overview

The idea behind Partial Indexing is that, since inverted file indexing is a slow process particularly because the index doesn't fit totally in memory, then why not build a smaller index (typically 2-5 % of text) which would fit. Searching will not be as fast as in standard inverted file but it will stay acceptable. The drawback in the speed of search would be compensated by the efficiency of many other operations which would be available on the small index.

Partial Indexing is a hybrid between full text indexing and sequential search (full text scanning). A small index is used to limit the text to be sequentially searched. Manber argues that under the assumption that nowadays a typical computer would have at least 16 Mb of RAM and that a Glimpse partial index only takes 2-4 % of the size of text, it is expected that the performance for both indexing and searching to be "acceptable" on text of size of order of 500 Mb of text. Since a 2 % of 500 Mb would fit in 16 MB of RAM along with its indexing context i.e. programs and other temporary data structure [Manber 94]. Acceptability in this sense reflects the actual response time for the user. If this time is tolerable then we can say that the search time is acceptable. Finally, note that the 500 Mb limits on the size of the text is suitable for our requirements because it is far beyond our expectation of the size of our local document base and nowadays most workstations have at least 32 Mb of main memory.

Also, another strong advantage of Partial Indexing is that by having the index fitting in memory, this overcomes one of the major shortcomings of inverted files: namely their bad performance in accommodating a dynamic document base. In Partial Indexing it is very feasible to add or delete documents from the index, because the index can be updated instead of being completely regenerated in the case when the index does not fit in memory. From our experience, we found that adding or deleting a file to the index is a very fast operation relative to the regeneration of the index. In general, any operation on the index is "acceptable" as long as all work can be done in memory.

3.3.2 How Partial Indexing works

The main idea behind reaching such a small size for the index in Partial Indexing is to use a very coarse granularity. In Partial Indexing, the text is divided into 256 blocks where a block may span multiple files. The reduction in size comes from, first, the size of pointer in the index file. Because we have only 256 different blocks, a block can be pointed to by a single byte. This is opposed to 4 byte pointers used in standard inverted files. Second, reducing granularity to a block minimizes the number of pointers stored in the index because for a given term we need to store only one pointer per text block regardless of how often it appears in the block, and even when it

appears in different files within the block. This will impose an upper bound on the number of pointers (256) an indexed term can have.

3.3.3 Searching in Partial Indexing

After the index is constructed, when the user submits a query, the query terms are looked up in the small index. The block numbers for which a match occurred are copied to a list of candidate blocks. If the query contains a logical operator then corresponding logical operation will take place. These operations might cause a reduction in the number of block pointers in case of intersection and union (duplicate removed). Next, the remaining candidate blocks would be sequentially searched using the "agrep" function (the sequential searching utility of Glimpse) and records containing the match would be retrieved to the user. This is why the searching method based on Partial Indexing is called a "two level search". In the first level, the index is searched in order to eliminate blocks which certainly do not qualify. In the second level, the actual full text scanning for the blocks nominated by the first level would take place .

3.3.4 Factors affecting the search time in Partial Indexing

Obviously the major factor which determines the searching time is the size of text to be sequentially searched . The index searching time is minimal because typically the index itself is very small, sorted and fits completely in memory. Searching the text mainly involves sequential disk access for reading the data. Some extra random disk accesses are needed because the text to be searched may not be contiguous (physically stored in non sequential disk blocks). Thus we can safely say that the performance is mainly determined by the number of candidate blocks to be sequentially searched. Following are the major factors which affect this number:

1. How common the query terms are:

If a query term is a common term then it is likely to appear in most of the blocks. This means that more blocks need to be sequentially searched. The less common the terms are the

more efficient is the search. Consequently, we get the best performance when a query term does not exist in the index because in such a case the text does not need to be consulted.

2. The number of OR'd query terms:

When there is a number of OR'd terms (terms related by the OR logical operator), all blocks containing at least one term must be searched. Therefore, an OR operator increases the number of candidate blocks and consequently the performance degrades because this means more text must be scanned.

3. The number of AND'd query terms:

On the contrary to logical OR, the AND logical operator in a query decreases the number of candidate blocks. This is because AND imposes more conditions on a block to qualify as a candidate for sequential text search (a block has to contain more than one term at the same time) thus AND improves performance.

4. Compound terms:

A compound term is treated as a set of AND'd terms plus an adjacency condition. That is, the block which contains all individual words of a compound term would qualify as a candidate at the first level search then this block would be verified for adjacency at the second level by sequential search. If there is no adjacency between individual words of the compound term, then the text of the block would not be returned.

3.4 Index searching in Partial Indexing

The second major variation in Partial Indexing, other than reducing the granularity, is the method by which the index file is searched. Manber has chosen to use a sequential search (full text scanning) for the index file itself instead of binary search, hash table, or B-tree search which are typical for the classical model. This would be justified, first, by the fact that the size of the index is expected to be small thus it is expected to fit totally in memory and with the current hardware performance i.e. it is assumed to be tolerable to sequentially search a few megabytes in memory.

Second, sequential search would make possible the use of available algorithms for advanced search problems such as those for approximate and regular expression search.

Finally, it is important to note another advantage for sequentially searching the index which has a particular importance for our system -- the case of multiple terms per query search. Under Partial Indexing, it is possible to use an algorithm for this problem which has its performance independent of the number of query terms. Note that in the standard inverted file search the number of terms significantly affects the performance [Rabbati & Zizka 84], because the search is carried out separately for each term and the results are combined afterwards.

3.5 Conclusion

The trade off between indexing time and searching time is critical for our approach. Given the huge difference in performance between disk and main memory, we always want the whole index context-- index data plus associated programs and overheads-- to fit in the main memory. For this purpose, the ability to adjust the size of index is crucial. In the case of Partial Indexing approach, Manber has taken a very conservative approach by minimizing the granularity to guarantee that the index and its context will virtually always fit in memory. By doing so, a whole new set of operations on index becomes affordable including the accommodation of index to dynamic document bases and optimization for the multiple terms per query problem.

In short, Partial Indexing satisfies most of our requirements and it also allows many additional valuable operations.

3.6 Experiments

In order to consolidate our analysis and validate our conclusions, we carried out an empirical analysis on Glimpse. We first experimented the indexing time, then the searching performance.

3.6.1 Terminology

Real time (also called wall clock time) is the time during which a user waits for the results of a program execution. System time is the pure CPU time spent on processing the search. By "caching" we mean that the data is stored in memory instead of disk. The operating system, after loading data the first time, caches this data (keeps it in memory) such that if referenced later it does not need to be loaded from disk again.

3.6.2 Experiments on indexing time

In order to measure the indexing time for Partial Indexing and to evaluate it versus the standard inverted file indexing we performed the experiments described below. Note that Glimpse supports standard indexing as an option. This option can be turned on by setting the granularity parameter to the byte level. Hereinafter, we will use the term "block level" for Partial Indexing and "byte level" for standard indexing.

Using the *time* UNIX command to measure the execution time, we ran the Glimpse index utility using two indexing options on collections of text of different size. First, we used the byte level indexing option which generates an index of granularity up to the byte location of an indexed term. This corresponds to the standard inverted file indexing. Then we used the block level indexing which corresponds to the Partial Indexing. Table 2 and Figure 3 depict the results. Note that these experiments were carried out on a Sparc-2 machine with 32 Mb of memory.

Text size	time for block level in seconds	time for byte level in seconds
1 mb	8.9	33.4
2 mb	14.1	48.8
5 mb	27.4	148.8
11 mb	62	376.2
23 mb	122.3	1020

Table 2 : Result of indexing in real time

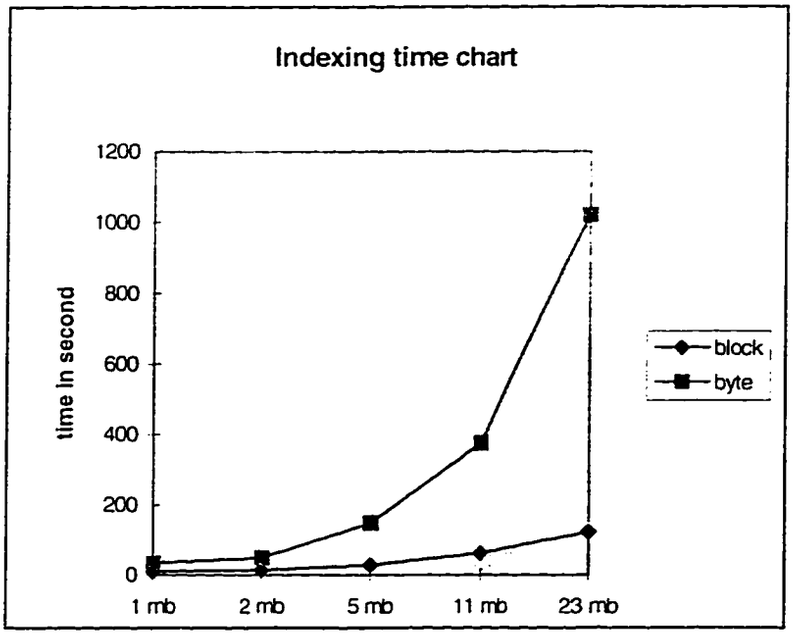


Figure 3 indexing real time for byte level vs. block level

It is evident from the above results that the byte level is more time consuming than that of block level indexing because it demands more storage (bigger index). However, the interesting observation is about the high incremental cost of the byte level approach. The next graph will depict this phenomenon.

Text size	time per mb for block level	time per mb for byte level
1 mb	8.90	33.40
2 mb	7.05	24.40
5 mb	5.48	29.76
11 mb	5.64	34.20
23 mb	5.32	44.35

Table 3: Per megabyte indexing in real time in seconds for byte level

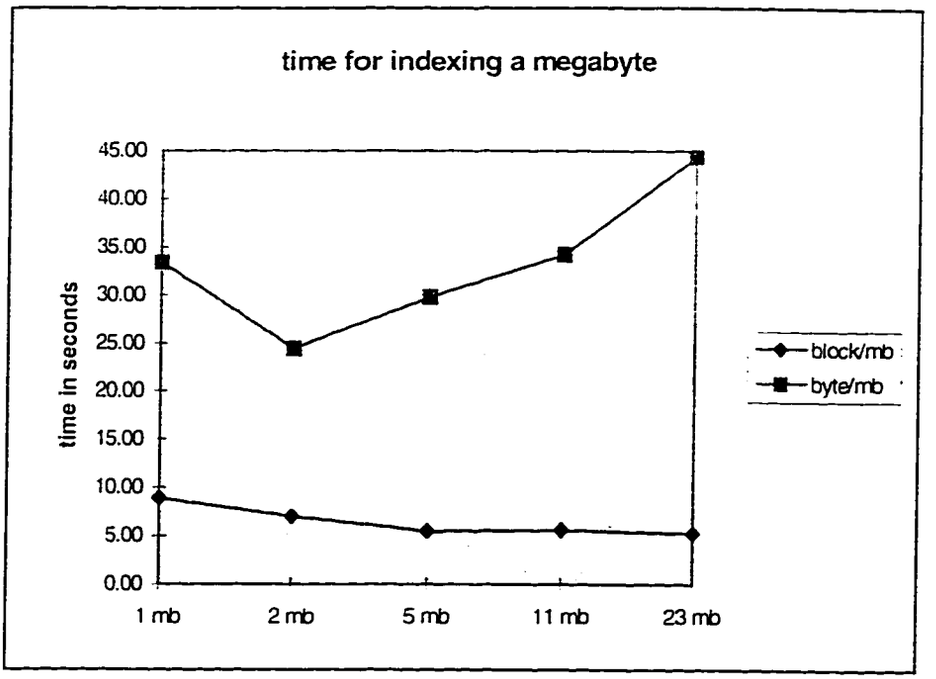


Figure 4: block level versus byte level per megabyte of text in real time

This graph depicts the indexing cost per megabyte. At the block level we notice that the cost of indexing a megabyte is roughly constant. The decline of cost in the first few megabytes is due to the initial fixed cost of initiating the indexing process (loading the programs and other housekeeping) and as the number of megabytes increases the contribution of the initial costs decreases. As stated by Manber we can safely say that a megabyte would cost less than 7-8 second on average when using the block level indexing [Manber 94].

In the byte level, we first notice that there is a high initial cost which drops quickly after the first megabyte. However, the major observation is about the constant increase in the cost of indexing a megabyte of text. A megabyte costs 25 seconds when the amount of text to be indexed is equal 2 Mb, but it jumps to 45 seconds when the amount of text to be indexed is equal to 23 Mb.

The behavior of the two indexing options can be easily explained by our previous observation: fits in memory versus does not fit. That is, at the block level the index contexts always fits totally

in memory. So we expects a constant cost as long as the context fits. Whereas at the byte level we notice a different phenomenon: as the size of the text to be indexed increases as the indexing becomes more and more costly. In other words, the increase in the size of the text to be indexed leads to a corresponding increase in the contributions of disk access (swapping and demand paging) in the indexing process. Correspondingly, this will cause a slower performance due to the large gap of performance between main memory and the secondary storage.

3.6.3 Experiments on sequential searching

In order to have a better insight into searching time, we carried out experiments to measure the search time for "agrep"-- the sequential search utility in Glimpse. We have chosen to test the "agrep" function instead of the two level Glimpse search which is based on the small index because the latter is highly dependent on the choice of the query and its relation to the text such that it is difficult to build a generic figure about the performance. On the other hand, "agrep" is virtually independent from the query content and from the type of text. Therefore, "agrep" would give a better indication about the performance of the search in general.

Text size	one term	three terms
1mb	1.3	1.4
2 mb	2.3	3
5 mb	5.6	6.6
11 mb	12.5	14.2

Table 4: Agrep results in real (wall clock) time in seconds

We measured performance both for one term per query and for three terms. Note that in agrep, the algorithm for multiple terms is different than that for one term.

Text size	one term	three terms
1mb	1.3	1.4
2 mb	2.3	3
5 mb	5.6	6.6
11 mb	12.5	14.2

Table 4: Agrep results in real (wall clock) time in seconds

We measured performance both for one term per query and for three terms. Note that in agrep, the algorithm for multiple terms is different than that for one term.

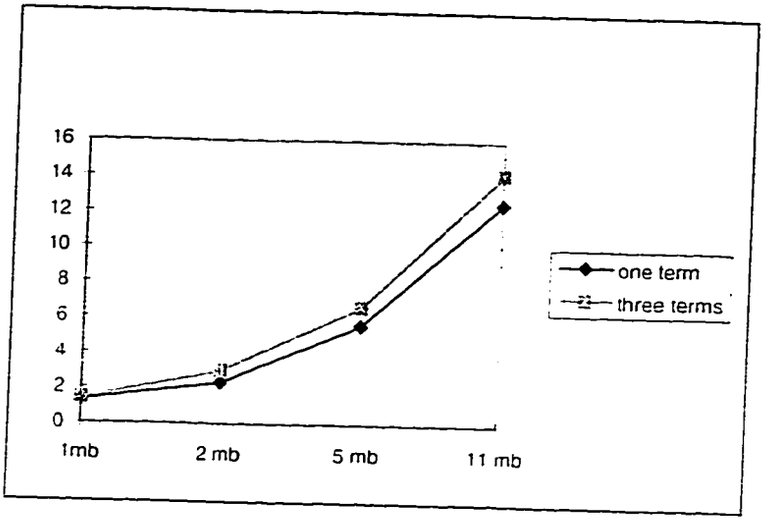


Figure 5: sequential search time in seconds

	one term	three terms	one term with caching	three terms with caching
System time	0.05	0.14	0.09	0.09
Real time	1.14	1.37	0.18	0.34

Table 5: Average time per megabyte of text in second

tried the number of terms to be more than three (up to twenty) but the performance results remained the same as for three terms, under a precision of 0.1 second.

The second reason for this result can be concluded from results shown in Table 5. In this table, the system time (corresponding to the CPU time spent on processing the search) constitute only a small fraction of the real time, as clearly shown in the last two columns of Table 5. Caching (which excludes the I/O time as the data is already in memory) allows us to conclude that the dominant cost of searching is due to I/O or the loading time. This can be seen by the difference in time between search with caching (1.14 second) and without caching (0.18 second), which means that more than 90% of time is spent on loading the text (and the searching program) from the disk.

4. Chapter 4 : The World Wide Web

In this chapter we will discuss the World Wide Web, and its relation to different information models. Also, research ideas and available tools related to information management on the Web will be surveyed.

4.1 Introduction to the World Wide Web

Often confused, the World Wide Web (WWW or simply the Web) and the Internet are different. The Internet (literally means a network of networks) refers to a massive world wide network of computers [Hughes 93]. That is, the Internet refers to the physical side of the global network in addition to the software necessary to implement its communication protocols. On the other hand, the Web is a body of information, an abstract space of knowledge which operates on the Internet. The Web can be viewed as an information layer on top of another physical and communication layer, the Internet. Note that other distinct information layers also exist on top of the Internet such as Gopher and FTP. But the Web, which has the ability to access virtually all the information on Internet, is definitely the most popular mainly because it is easy to use, and because it has a graphical interface.

4.1.1 Definition

The Web is officially described as a "wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents" [Hughes 93]. From an information model perspective, the WWW is a hypertext system which keeps the node and links model intact but provides distributed hypertext by extending the node addressing mechanism and defining a node transport mechanism.

The Web is based on a client-server model, in which a standard transfer protocol (HTTP, or Hypertext Transfer Protocol) is used to communicate hypertext documents in a standard format (HTML, or Hypertext Markup Language). A Web server is a program which runs on a computer connected to the Internet, and this program keeps listening to requests incoming from a specific port number of the computer (usually port # 80) for a HTTP request. HTTP is an Internet protocol which runs on top of the TCP/IP protocol which is the native Internet communication protocol. The Web server replies to a request by sending back the required documents or an error message.

The client, after receiving the data, simply interprets the document's HTML markup to provide a visual rendering of the document, to maintain a history list of the user's recent session, and to enter a dialogue with a remote server to obtain a document when the user activates a link. The client is normally called a Web browser. The "Lynx" browser was among the earliest browsers but it was only able to display characters. The "Mosaic" browser pioneered the graphic interface, while recently "Netscape" dominates the Web browser market.

4.1.2 A brief history

The World-Wide Web began in March 1989, when Tim Berners-Lee of the European Particle Physics Laboratory (known as CERN, a collective of European high-energy physics researchers) proposed the project to be used as a means of transporting research and ideas effectively throughout the organization. Effective communications was a goal of CERN's for many years, as its members were located in a number of countries.

The initial project proposal outlined a simple system for using networked hypertext to transmit documents and communicate among members in the high-energy physics community. From January to December 1993, the amount of network traffic (in bytes) across the Web multiplied by 187 times. In December 1993 the Web was ranked 11th of all network services in terms of sheer byte traffic - just twelve months earlier, its rank was 127.

In June 1993, a small program called "World-Wide Web Wanderer" found around 100 sites that month and over two hundred thousand documents. In March 1994 this program found over 1,200 unique sites. In mid-May 1994 another program found over 3,800 unique Web sites. In May 1994, there were at least 4,500 hypertext Web servers in use throughout the world. It is very difficult to keep up with the growth rate of the Web, but it is certainly extraordinary [Venditto 96].

4.2 Description of the WWW components

The WWW is a synergy of multiple components and technologies. In particular, HTML provided the Web with document structure and the URL addressing mechanism that provided a high degree of flexibility in addressing. Following is a description of these components:

4.2.1 HTML

The hypertext Markup Language (HTML) is the standard language the Web uses for creating and recognizing documents [Hughes 93]. HTML inherited much of its design philosophy from the Standard Generalized Markup Language (SGML). SGML is a meta-language that enables the structure of information to be described[1]. The separation between information and its representation is also inherited from SGML. HTML presentation (formatting) tags are generic and thus can be interpreted differently depending on the HTML browser policy.

HTML encompasses several functions. First, it is a page layout language which allows the user, by specifying appropriate tags, to control visual elements such as fonts, font size and paragraph spacing without changing the original information.

Second, HTML is used to mark the structural parts of a document. In this sense, HTML provides a simple document architecture which consists of several levels of heading, various lists, paragraphs, different kinds of emphasized text.

[1] A brief introduction to SGML is at: http://www.cfi.joensuu.fi/~t_dgreen/sql/sgml.html

Third, HTML -- due to its linking capabilities-- constitutes a distributed hypertext and hypermedia model and provides a rich set of structuring options. Text can be written as a single, coherent document, with internal cross-references. Alternatively, the text can be split into many documents (nodes) and the text's structure would be inferred from the links between the nodes. The major advantage of this scheme is that the logical coherence of text persists, while the physical distribution of the text among different smaller files can increase efficiency in navigation and retrieval.

4.2.2 URLs

The World-Wide Web uses what are called Uniform Resource Locators (URLs) to represent hypermedia links and links to network services within HTML documents. It is possible to specify nearly any file or service on the Internet with a URL.

The URL is formed in a particular syntax to express how a resource can be retrieved. The first part of the URL (before the two slashes) specifies the method of access, the communication protocol such as HTTP or Gopher. The second is the address of the computer host on the Internet where the resource is located. Further parts may specify the paths and names of files and the port number to connect to.

On most Web browsers, a user can specify directly a URL to connect to a document or service. Also, by selecting a hypertext link in an HTML document, the user would be actually sending a request to open a resource pointed to by the URL embedded in the link. Hyperlinks can be made not only to other texts and media, but also to other network services like a telnet session. Moreover, a part of the name space of the URL can be used to specify a program to be run and the arguments to be passed to it (see Chapter 5 on CGI). The "Web server" will execute the program and return to user its output as a document composed on the fly.

4.3 The information model of the Web

Originally intended to be a distributed hypertext information model, the Web is creating a new breed of information model. The Web information model inherited characteristics from many parents, in particular -- the hypertext model, the classical information retrieval model and the Internet.

The Web inherited from information retrieval the concepts of a document base. The Web can be viewed as large distributed collection of textual documents. This collection can be indexed, queried and retrieved in a similar fashion to the information retrieval model. This is actually the view which is used by Web search engines.

From the Internet, there was the concept of the seamless networking of distributed information objects. More importantly, the Web inherited from the Internet a universal information sharing and accessing mechanism.

Also, one of the major advantages of the Web which makes it so successful and powerful is its adoption of the hypertext model by using HTML as the standard format for encoding information. Hypertext has been defined as "an approach to information management in which data is stored in a network of nodes connected by links. Nodes can contain text, graphics, audio, video as well as source code or other forms of data" [Smith & Weiss, 1988]. The linking facilities of HTML which have been integrated with the HTTP communication protocol made the Web a hypertext model where documents represented the nodes and URL's represented the links.

Indeed, in the late 1980s, hypertext emerged as an alternative to classical information retrieval. It offered the semantic connections among logical units of information that information retrieval found so elusive. It incorporated the notion of logical structure where a collection of physically separate documents can preserve proximity and semantic coherence through the hypertext linking facilities. It also introduced an interactive, dynamic user interface style and did away with complicated query syntax typical of information retrieval systems.

Moreover, hypertext as a way of expressing knowledge is much more powerful than plain text. Hypertext parallels human cognition in providing a non-sequential method of accessing and expressing information. We think in nonlinear chunks which we try to associate with each other and build a network of concepts. When we read a book, we go back and forth a number of times to refer to previously read material and to jump to topics using the table of contents or the index. When we set out to write a document we first develop an outline of ideas. Then, we brainstorm, write down on paper, organize, revise, reorganize and repeat the cycle till we are satisfied with the outcome - a coherent document.

4.4 Information retrieval and hypertext

The Web information management paradigm can be seen as a marriage between the classical information retrieval model and the hypertext model. While hypertext is only suitable to navigate a relatively small collection of nodes, classical information retrieval is very useful for large collections. However, information retrieval systems ignore the hypertext features such as links and logical structure; they treats the Web as a flat collection of documents. Therefore, none of these paradigms alone is ideal to handle effectively the problem of information management on the Web. Next we will describe some of the proposed ideas for extending these paradigms and later we will discuss the available tools.

4.4.1 Two level architecture

Bruza [Bruza 90] proposed to separate the hypertext space into two levels: one for index (hyperindex) and the other for documents (hyperbase). The index can be considered as pre-compiled links providing immediate access to required information without navigating through the document space (the hyperbase). When an index term describing the required information is found, the object from the hyperbase is retrieved for examination.

This idea is similar to what has been implemented in virtual libraries (e.g. Yahoo[2], see below the section on Virtual libraries) with great success. However HTML does not explicitly support the distinction between a document intended to be an index and a document intended to have information content. In the absence of such a distinction, the two types of documents need to be manually separated.

4.4.2 Structural queries

A special kind of query called a "Structural query" -- opposed to the classical query called "content query"-- has been proposed to address the logical structured nature of hypertext [Halaz 88]. Instead of returning the matched documents as in content search, a structural query yields the hypertext sub-network that matches a given pattern. In conjunction with a structural query, a content query can filter the sub-network such that the interface will display only those nodes that match the query.

A structural query can be constructed as an extension of a content query by including the notion of quantifiers, recursive operators and aggregation.

In a query language developed for hypertext (HDM2), a query produces a list of references to documents [Garzotto et al., 1993]. These can be used as a dynamic access structure (or virtual structure) from which further navigation can originate. The navigation space resulting from such a query or filter is called a hyperview. All subsequent requests (either queries or navigation commands) will be interpreted only within this restricted space.

4.4.3 Combination of navigation and search

Combining browsing and searching as a new paradigm for finding information is the natural step towards exploiting the marriage between the two paradigms, information retrieval and hypertext. A simple implementation of this paradigm is to treat a search engine result as a starting point for

[2] Yahoo is accessible at : http://www.yahoo.com

further navigation. In this sense the browsing complements the searching which was carried out first. This is how search engines such as Alta Vista are used today.

In another approach implemented in the "WebGlimpse" system[3] [Manber 96], searching was to complement browsing which comes first. In this approach, the system automatically modifies existing Web pages to add search facilities such that the search space for a given page expands to its page neighborhood. A typical neighborhood may be the list of all pages linked from a given page, all pages within distance 2, all pages in the same site, all pages pointing to that page, etc. In short, WebGlimpse allows any Web site to offer a combination of browsing and searching by automatically analyzing the site, computing neighborhoods, and attaching search interfaces to existing pages.

4.4.4 Vector space

Researchers have suggested using the vector space model to organize a hypertext collection into clustered hierarchies [Crouch 1989]. In this model, the content of each node or document is represented by a set of possibly weighted terms. Thus, each document can be represented by a term vector and the complete document collection can be represented by a vector space whose dimension is equal to the number of distinct terms to identify the documents in the collection. Similar or related documents are represented by similar multidimensional term vectors. Such a model facilitates clustering documents based on their similarity and ranking retrieved documents in decreasing order of their similarity to the query vector. Hence, the user can readily focus the search on those clusters that are likely to contain documents which are similar to the query. Comparisons are generally made between the query vector and the document vectors using one of the standard measures of similarity. Clustering is also helpful in locating neighboring nodes which discuss related topics. The user can incrementally refine the query vector to retrieve the desired documents.

[3] See WebGlimpse home page at http://glimpse.cs.arizona.edu/Webglimpse/

4.5 Available information retrieval tools on the Web

Many approaches and tools have been implemented to address the problem of information management and exploration on the Web. Available approaches differ in terms of coverage i.e. some are intended for the whole Web while others are intended for a site, a group of sites or even for a single collection of information. We can also see a difference in terms of targeted users: most of the tools are open to the public and can be used by any person to locate any kind of information (e.g. search engines and virtual libraries). Others are intended for subscribers only (e.g. "Electric Library[4] ") and often are more focused in terms of information domain. Finally, we have tools like WebCompass and EchoSearch (discussed below) which are sold as a software package to be used by individuals.

We will begin discussing the search engines, the most prominent and popular tool for finding information on the Web.

4.5.1 Search engines

Search engines are tools for finding information by maintaining an index of Web resources[5]. They are systems that apply almost the same techniques of classical information retrieval. That is, gathering data, building and updating an index, querying, searching , then retrieving and presenting results.

4.5.1.1 A general procedure

Gathering information is usually done through an indexer "robot". An indexer robot is a program which traverses the Web by recursively following links. Each encountered node would be sent to the indexing engine to be indexed. Some search engines accept updates from people who want their documents to appear in search engine results.

---

[4] Electric Library is accessible at: http://www2.elibrary.com/

[5] Article on search engine is at http://www.pitpint.com/English/Search/cusiksk.htm

Search engine companies provide little information on the details of indexing for search engines. We suspect that some simply use an implementation of the PAT index for search engines which handle phrase searching. Those that do not handle phrase searching would be most likely using an inverted file index. But this has to be implemented on machines with a very large hardware configuration in order to handle the huge size of data indexed and the number of requests from users worldwide.

Although each search engine (unfortunately) defines its own query language; in general, search engines queries are Boolean queries but with different syntax. Some engines permit extra conditions like requiring the match to be in a particular part of the document such as the title or even the URL. Query interface is done via a Web page provided by the search engine. Users fill in the query in a form which is submitted to the engine via encoding it within the URL of the provided Web page.

After the query is submitted, the search engine returns results by displaying a new Web page containing links to the matched documents. Often, extra information is provided along with the links such as a) a small abstract typically made of the first few lines of the documents b) the title of the document c) its size and creation date d) the URLs from which the documents can be retrieved as the document may be stored on many different locations (sites). Some search engines like "Excite" presents a numeric relevance indicator which determines the ranking of returned documents.

4.5.1.2 Ranking and relevance

Most search engines apply ranking techniques on the returned documents. Ranking is a method of approximating the degree of relevance of documents vis-à-vis the user's query. Ranking is especially important in this context because it practically determines what portion of the end result would be seen by the user. In "Alta Vista", for example, only the first 200 matched documents out of thousands are returned to the user. The user's query terms often represents the seed for the ranking algorithm; however, some engines like "Alta Vista" allow the user to define a different set of terms for ranking. Probably, the vector space model is used for ranking.

especially in the case of engines which returns numeric relevance indicator, or those which permit locating similar documents to a returned one, like "OpenText". In general several heuristics are used determine relevance. Often it pays to have an idea about these heuristics. Heuristics are mainly based on relevance weighting of terms, based on some combination of:

1. frequency of occurrence of the search terms overall,
2. proximity of terms (for multiple search terms),
3. occurrence of terms within the first n words of the content,
4. occurrence of the search term(s) in the title element
5. occurrence of the search term(s) in heading elements
6. the ratio of search term frequency to overall document length .

4.5.1.3 Conclusion:

Search engines probably constitute the most comprehensive information exploration tool on the Web. Yet, they suffer from some shortcomings especially, their low precision as discussed in Chapter 1.

A major shortcoming is the lack of a single complete Web index. The indexed document space for each search engine differs. Since these start from different base documents and work in different ways none of the resulting indexes are comprehensive and nor are the resources listed completely duplicated. To complete an extensive Web search for a specific topic, one must submit the same query to multiple search engine sites. Initially to assist in this situation, CUSI or the Configurable Unified Search Interface was developed. This allowed Web users to create on a single Web page, a form to easily submit a query to a variety of search engines. The user could enter the query, select a search engine server from a list and submit it for processing. After viewing the results, the user could then keep the query intact and simply select a different search engine for processing. However this method leads to a loss of flexibility especially in formulating sophisticated queries which vary in syntax between different engines.

Recently new tools which alleviate this shortcoming such as WebCompass and EchoSearch (discussed in section 4.5.5) have been introduced. In these tools, the single query would be automatically translated and passed to each one of multiple search engines. The results of all engines would be next merged, organized then classified.

Finally, it is probably safe to say that most people using the Web are not experienced in formulating queries. Boolean queries, largely used by search engines, are subtle and need significant effort to master in order to get full use of their capabilities.

4.5.2 Virtual Libraries

Virtual libraries represent a more structured and focused way to address the Web information management problem. Virtual libraries at their infancy were merely an organized set of links to various resources on the Web. Later, they emerged to be a more organized and comprehensive database of links and abstracts associated with powerful navigation features. Links are often arranged by topics in a hierarchy [Green 94].

Links are presented, possibly with small abstracts written by the virtual library librarian. These abstracts are more meaningful than the automatically generated abstract of search engines. In addition, searching capabilities are defined with options to filter results or to limit the scope of search to a particular sub-hierarchy related to a topic.

Information gathering is done largely manually. Librarians have to continuously surf the Web to collect new resources. Next, they have to evaluate these resources, write abstracts describing them, then categorize or graft them into their appropriate place in the hierarchy. Librarians have also to repeat the same process on resources suggested by other users.

Yahoo represents the most popular example of a virtual library. Other examples includes the WWW virtual Library[6], Galaxy[7] and the Electric Library.

---

[6] WWW virtual library is at: http://www.w3.org/pub/DataSources/bySubject/Overview.html

4.5.2.1 Analysis

Virtual libraries may represent a better solution to information retrieval on the Web than search engines in many situations. They are very effective when the required information belongs to a well defined topic i.e. when it is clear in the user's mind what he or she is looking for. This will make easy the traversal of the hierarchy in order to get to the required information. Moreover, when the required information belongs to a popular topic then there would be more chance having a well populated sub-hierarchy. However, these advantages resulting from the human processing and compiling of resources also result in many disadvantages:

- The coverage of virtual libraries is limited by human efforts and thus they are likely to miss many important resources, especially new or unpopular topics. It is very difficult, by manual means, to achieve the comprehensive level of tools which use automatic means to gather data as is the case of search engines.
- Resource references become stale (cited resources are either withdrawn or moved to another location)
- As the virtual library grows, it is increasingly difficult to categorize new references.
- Large amounts of time are required to add links and respond to email messages and evaluating incoming items which is done manually.

4.5.3 The Harvest system

Harvest[8] is an integrated set of tools to gather, extract, organize, search, cache, and replicate relevant information across the Internet [Bowman 95]. Harvest features also include digesting information stored in non-HTML formats which are usually inconvenient to use from within the Web such as compressed and PostScript files. Moreover, Harvest use a caching technology in order to minimize the network delay.

---

[8] Galaxy is at: http://www.einet.net/galaxy.html

A Harvest system consists of many sites. Each site can be described as having a server called 'the Gatherer' and one or more clients called 'the Broker'. The Gatherer is responsible for indexing and accessing local resources, communicating with other remote servers (Gatherers) to access their resources, and providing common services required by its clients (Broker). The Harvest Gatherer use a site-resident software optimized for indexing. The Gatherer scans the indexed objects periodically and maintains a cache of indexing information. It addresses network transmission inefficiencies by providing remote access to pre-filtered, incrementally updated indexing information, and transmitting this information in a compressed stream. It also increases the accessibility of the Web through the "Essence" customizable information extraction system. Essence can expand presentation layer encoding (such as compressed "tar" files) into constituent files, recognize each file, and extract information in different ways depending on the file types. More importantly, it provides content summaries of indexed information which is passed between the different components of the system.

The client (the Broker) provides a mechanism for the user to discover and access resources. The broker can be further decomposed into two major components: a user part which provides an interface specially designed for the end-user environment; and an "Agent" part, which provides user specific services and communications with the local server. The Broker provides an indexed query interface to gathered information. Brokers retrieve information from one or more Gatherers or other Brokers, and incrementally update their indexes. The Broker can collect objects directly from another Broker using a bulk transfer protocol. It also supports a remote administration interface.

4.5.3.1 Analysis

Although Harvest does not provide a complete solution for the Web, it alleviates many of the Web dominant problems. As for the balance of precision and comprehension, it constitutes a compromise between the flat, open and unorganized approach of search engines and the humanly screened and organized approach of virtual libraries. It provides more control on the retrieved

---

[1] Harvest home page is at: http://harvest.transarc.com/

information by assigning a sub index to serve a sub-domain and a network to integrate these sub indexes to achieve more comprehension.

In addition, it provides a caching mechanism which significantly reduces navigation time. In particular, we should appreciate its ability to access, in a seamless way, the legacy documents which are encoded in a format that is unreadable by a Web browser and thus invisible to normal search engines (e.g. Postscript and compressed files). These documents still constitute a significant portion of the information available on the Internet.

4.5.4 WebCompass

WebCompass is a program that runs locally (e.g. under Windows 95) and accesses the Web. It is described as a World Wide Web information access and discovery tool, that can be used to build and update custom indexes of the World Wide Web automatically[9] . WebCompass includes features to gather information from the Web and then to organize this information for further use.

In WebCompass, two methods are used for gathering. First, WebCompass can query multiple search engines simultaneously. It also collates and organizes their results and eliminates any redundancy (the same document returned by more than one search engine).

Second, WebCompass include a gatherer agent (a robot which wanders the Web). The agent after being launched, will collect documents and return an index of these documents organized by topic and subtopic. Each document will be accompanied by an automatically generated short summary and a list of keywords and phrases. These are meant to help the user to determine whether a Web document is worth downloading.

---

[9] See the WebCompass home page at http://www.hotfiles.com/swbrowse/000/A/N/swlib-000ANP.html

Moreover, WebCompass is equipped with a built-in topic database of over 40.000 entries. The topic database is organized by a system of cross-referenced hypertext links that allows to traverse related topics by clicking on them. The WebCompass index is organized by a network of relations among the topics: these relations include the "is a" relation, "opposites are", "specific kinds are", "see also".

More than a container for organizing gathered documents, the topics database can also enhance the gathering performance. Users can choose and activate topics from the database of topics in order to guide the gathering process. In this sense, the topic database incorporates a knowledge base-like functionality. Using the stored knowledge in the database and inference rules on relation, WebCompass provides automatic query expansion when querying search engines, e.g. all spelling variants and synonymous terms relating to topics.

4.5.4.1 Analysis

WebCompass provides a significant step towards alleviating the information problems of the Web. With many respects, it succeeds in making information gathering a more convenient and time efficient process. Following is a list of major accomplishments:

- Eliminated redundancy and inconvenience in querying multiple search engines and thus increased comprehension.
- Similar to our system, it addresses the need to have a 'personal' repository of information. This repository will save time by accumulating the information gathering efforts.
- Saved relevance determination time by: a) increasing the precision of returned documents via incorporated external knowledge (form the topic database) into the querying mechanism, b) using a background agent which generates summaries of collected documents such that, presumably, the user will not need to examine the whole document .

However, the user still has to deal with whole documents most of the time. The summaries returned by agents fails in many occasions to be sufficient for determining relevance. More

importantly. WebCompass did very little to exploiting the gathered information (other than organizing it). That is because WebCompass does not download the actual gathered documents but only save their summaries; useful documents have to saved manually.

4.5.5 EchoSearch

EchoSearch[11], like WebCompass, is a tool used on top of (or powered by) the Web search engines and intended to alleviate some of their problems. EchoSearch allows to query multiple search engines simultaneously for information on the Web. EchoSearch then downloads and reads documents selected by the search engines and creates a document index, a "concept index" -- an index of main concepts in documents--.and summaries of the downloaded documents that can be viewed in a Web browser.

4.5.5.1 Analysis

Using EchoSearch is certainly advantageous over directly using individual search engines for locating information. EchoSearch merges the results of many search engines and arranges them in an index which allows rapid access to returned documents. Yet, the concept index is questionable as a serious useful feature. In fact, this index is very shallow in term of grasping document concepts. It appears to be using simple term adjacency -- locating adjacent terms to the query terms and treating the combination as a concept. Summaries are less meaningful of that of WebCompass. A summary often looks like a clipping of certain parts of the document in a non-cohesive way. Moreover, EchoSearch fails to generate summaries for more than half of the downloaded documents.

A major disadvantage of EchoSearch is that it is not a pure Web application. Its main interface is located in a conventional program. EchoSearch executes first from the program which runs on Windows or on Mac but it outputs results in HTML into a Web browser. This violates the Web portability and eliminates the seamless integration of Web based systems.

As compared to our WIC system, EchoSearch is similar in that both systems download and organize collected documents. Yet, unlike WIC, EchoSearch is not aimed at composing a permanent useful repository of information. Instead, EchoSearch downloading is more focused on making the result navigation more effective and efficient. We must also note, that even at the level of downloading EchoSearch is only limited to download the direct search engines results. In WIC, one can use the results to reach different documents and then "recursively" download whatever is found to be relevant (see Chapter 5).

[10] See EchoSearch home page at: http://www.iconovex.com/ECHO/ECHOS.HTM

5. Chapter 5 : Description of the WIC system

In this chapter we will introduce the Web Information Collector (WIC). We will first describe its development, then examine its functions in detail, explaining how each was implemented and the motivation behind implementing it. Finally, will describe an experiment carried out using the WIC.

5.1 Introduction

The WIC is a Web based system; it uses an HTML user interface that can be accessed from any Web browser. Programs are developed in PERL under UNIX and communicate with the HTML interface via CGI (see below for the discussion on CGI). Also, the WIC uses some freeware utilities and systems, most importantly, it uses Glimpse as a search engine to search the local document base.

We will first discuss the details of Web based systems in general, then begin the description of the system.

5.1.1 The Web paradigm

The Web is a new paradigm (or a new approach) for software development which has a fundamental difference with conventional software development. The major difference lies in the topology of the system; a Web system is a distributed client- server model. The client is a user anywhere on the Web using a Web browser to invoke some programs on a remote Web site usually via invoking a URL. In this context the term Web site corresponds to an HTTP server on a machine serving an HTTP request for a document on the same machine. Programs need not reside on the same computer as the server. Programs may be sent to the client from the server along with the requested document to run within the browser. In other cases, invoked programs could simply run on the server and send back results as HTML documents. The former model (client side execution) is

implemented in languages such as Java and JavaScript. We have used JavaScript for our system in a limited way mainly for creating a more dynamic user interface. Programs written in the PERL programming language are typically used to implement the server side model. Most of our programs have been implemented in PERL: such programs are called CGI programs because they use the CGI protocols to communicate with HTTP Web server.

5.1.1.1 Server side program execution : CGI

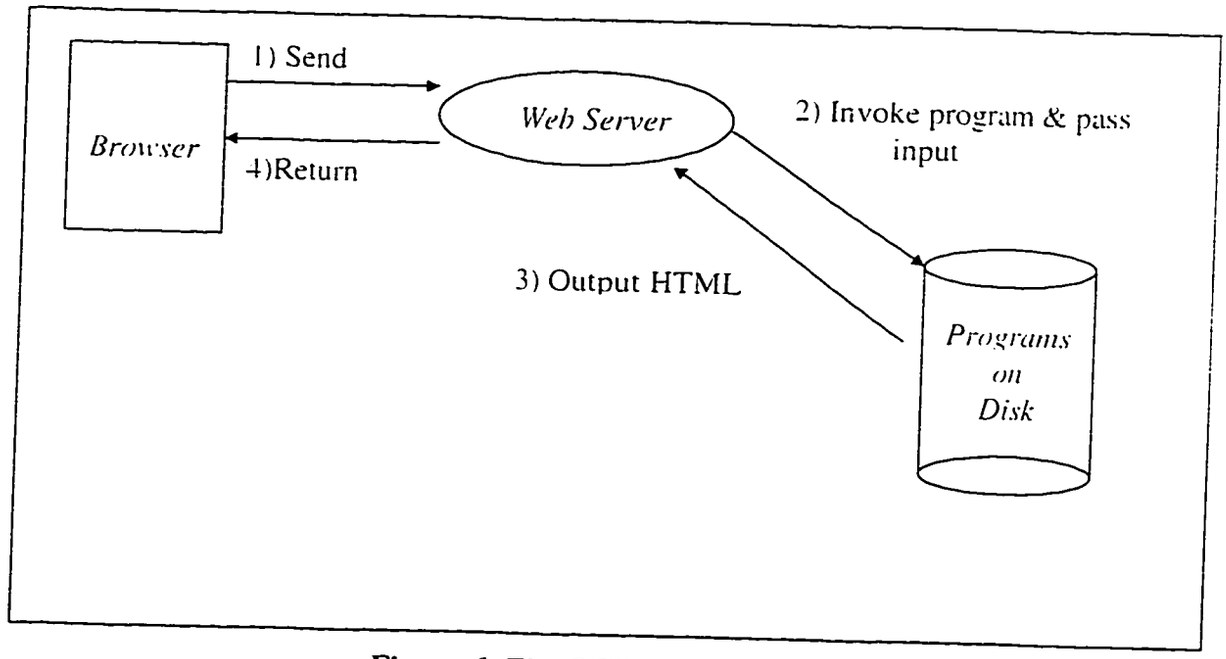


Figure 6: The CGI mechanism

By definition, CGI (Common Gate Interface) is a means for an HTTP server to "talk" to programs on a machine [December 95]. In this sense, CGI allows for the incorporation of programs into the World Wide Web. With a CGI program, it is possible to input data remotely via a browser (a Web client) to a program and receive back the program output. Such output would allow for dynamically generated documents.

To invoke a CGI program, the client sends a request conforming to URL standards to the server. Attached to this request is a "header" of data submitted by the client to be input to the CGI program. The data header can either be part of the URL or automatically concatenated to it (by the browser) at the form submission time. A form is an HTML feature which allows for a user to input data in a fill-in-the-blank interface (it also can contain different control entities such as pop up menu and check box). When the server receives the request, it parses it and if it recognizes the file type of the URL as an executable program, rather then returning it, it will run the program and keep listening for its output. The program's output is then received by the server and sent back to the client as a normal HTML document.

In summary, what a CGI defines is how data are passed (in both directions) between a client and a server and between the server and the CGI programs. Most importantly, CGI also defines how data should be encoded, decoded, and in which format it should be transferred.

5.1.1.2 Client side program execution: JavaScript

In client side program execution model, programs would be initially stored at the server. They are embedded inside the HTML documents and passed along when the documents are transferred to the client. Java and JavaScript are currently the two languages that belong to this model. In Java, embedded programs are encoded in a compiled format, ready to be executed by the browser. We have used JavaScript mainly in implementing some of the user interface in our system. JavaScript statements are embedded like other HTML statements into a special tag and they are interpreted by the browser.

5.2 Description of the WIC system

In general, the main functionality of the system is divided into two phases 1) collecting documents from the Web and storing them into a hierarchical local document base 2) querying the local document base and then ranking and presenting the results.

In what will follow, we will describe the sequence of steps corresponding to a typical scenario for using the system. This will serve both as a user manual and will help in understanding and learning the system. We will also describe implementation details.

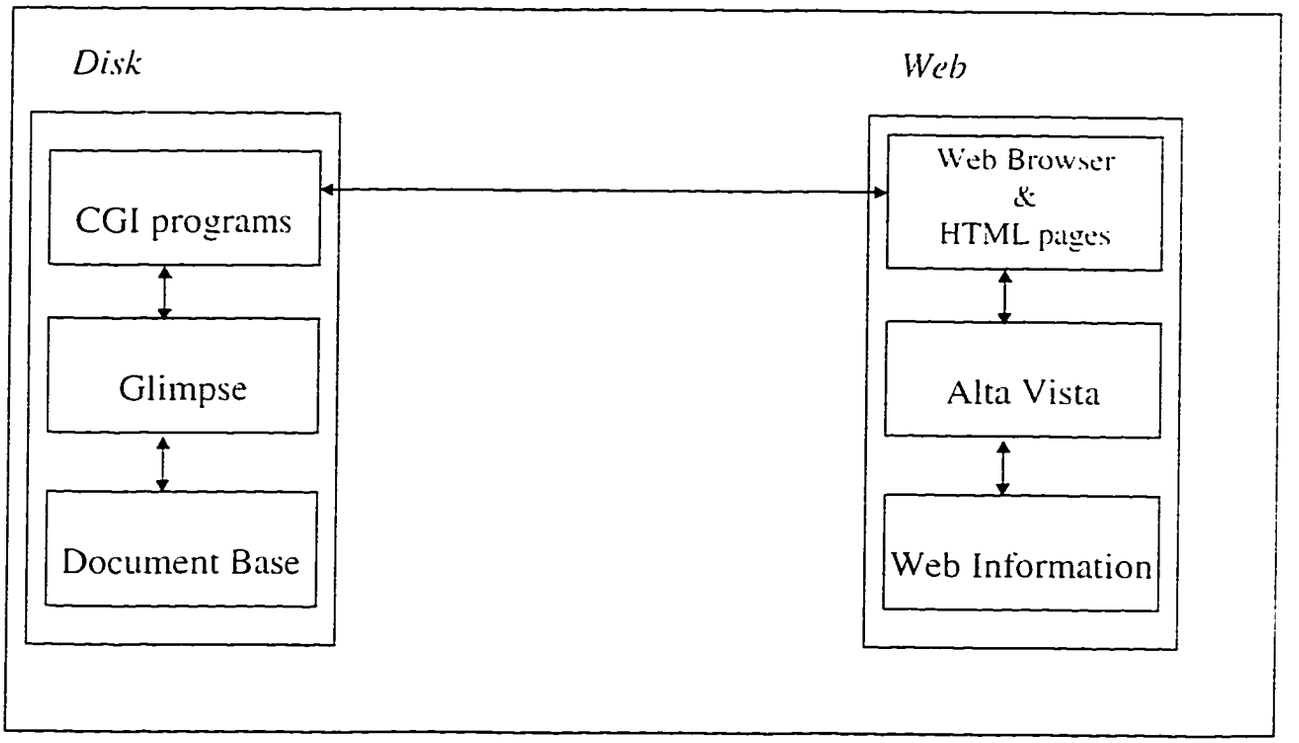


Figure 7: Main components of the WIC

5.2.1 Collecting

The first phase is to collect information from the Web and to store it locally. That is, to construct a local document base to be the subject for further querying and retrieving by WIC users.

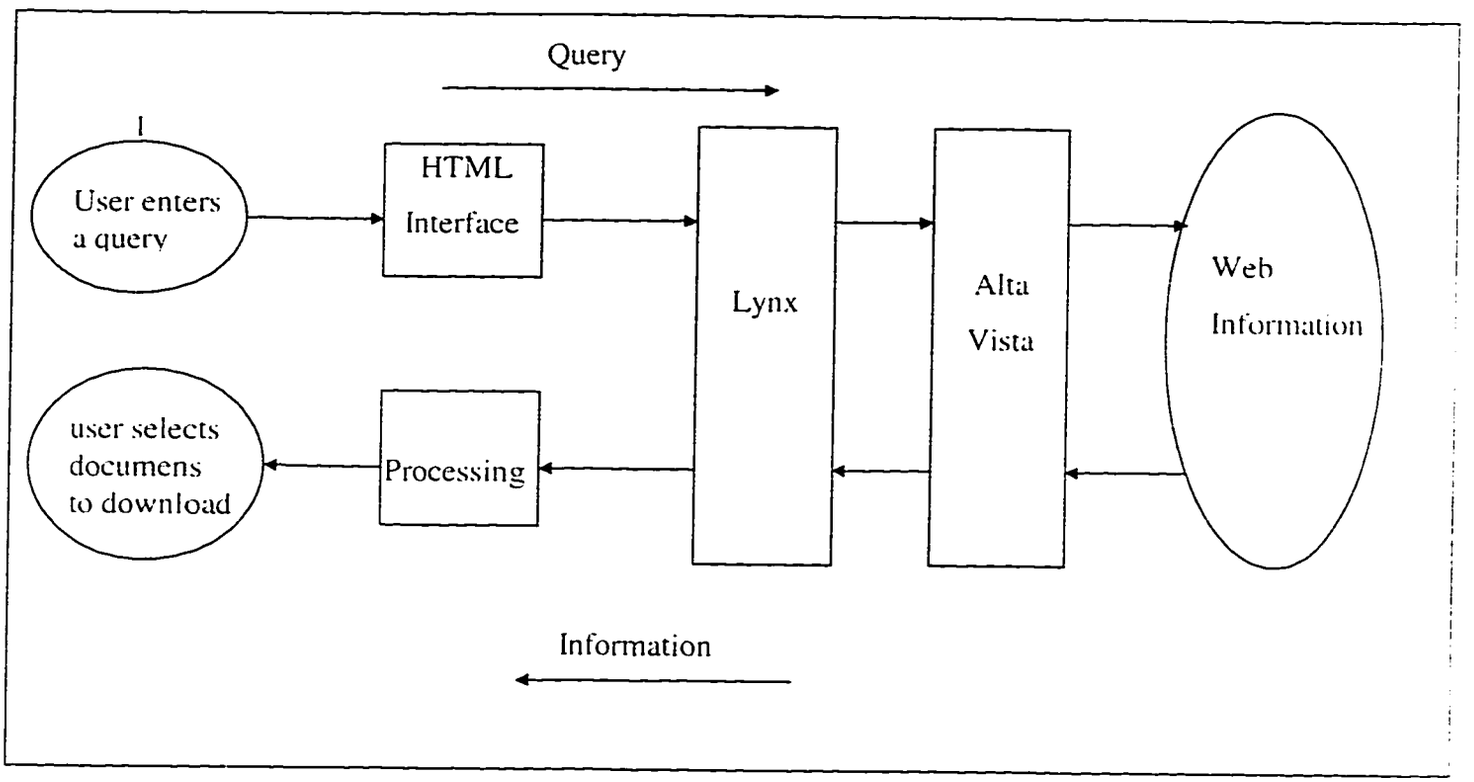


Figure 8: A general diagram for the collecting procedure

5.2.1.1 Querying Alta Vista

The WIC system uses the Alta Vista Web search engine as a means to collect documents from the Web for particular topics. We have developed an HTML interface to Alta Vista such that the user can enter a query (complying with the Alta Vista syntax) within the main page of WIC. The query will be passed to Alta Vista which will perform the corresponding search. The returned results of the Alta Vista search are intercepted and processed before being returned to the WIC.

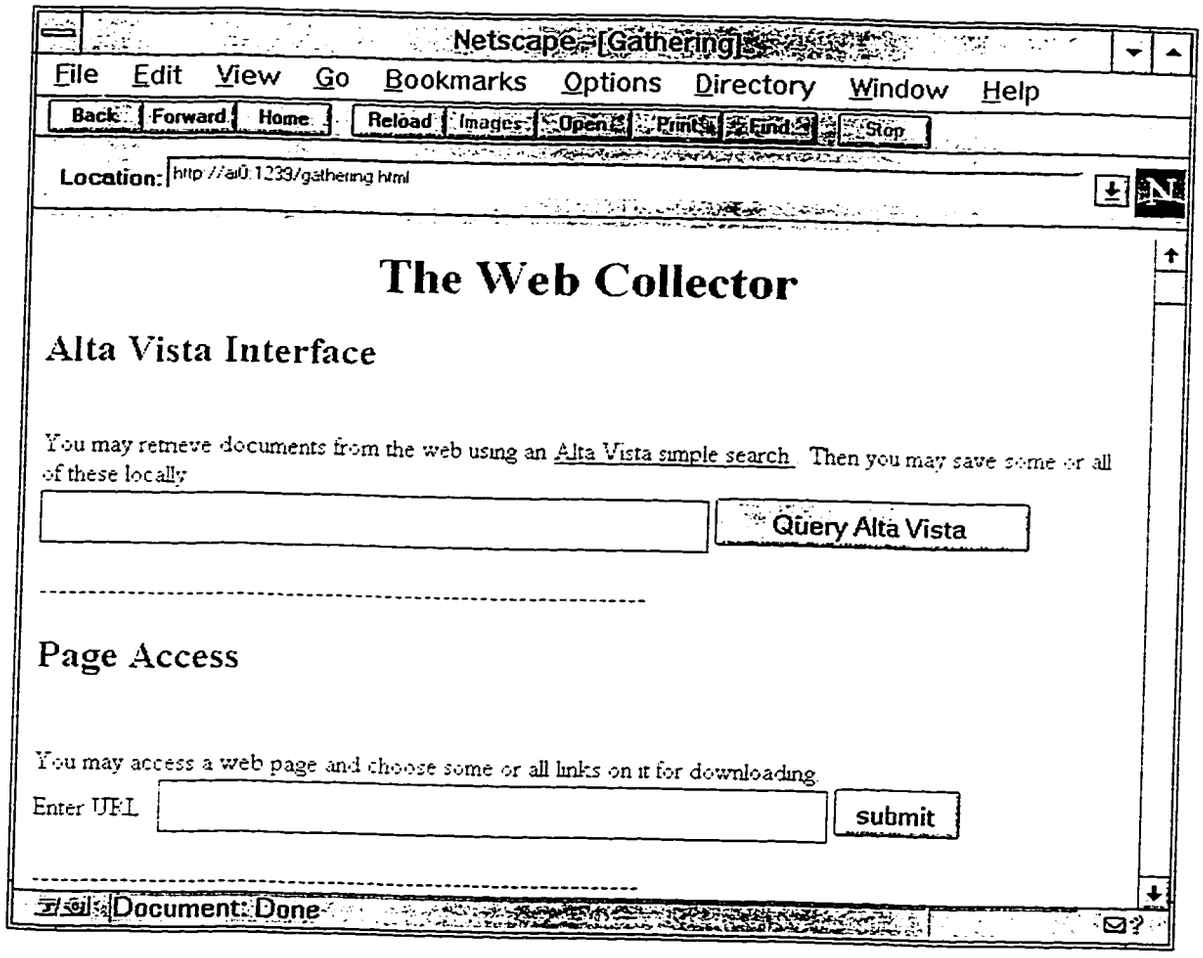


Figure 9: This page is used for gathering

In order to be able to indirectly query Alta Vista, we have first to decipher the URL syntax used by Alta Vista to communicate the query to the Alta Vista remote server. Deciphering was done by observing the syntax of multiple instances of encoded queries (in the URL used by Alta Vista to invoke the search) and inferring scheme. We used this scheme later in order to encode our own queries and pass them to Alta Vista.

Second, after passing the query, we have to intercept the returned results of Alta Vista in order to be able to process these results later on. We used the "Lynx" browser as a utility which can be called from a CGI program.. Lynx, when given the appropriate parameters and the URL, returns the text of the corresponding remote document on the Web. Thus ,

we were able to emulate the URL syntax of Alta Vista, input the emulated URL to Lynx and redirect the Lynx output (the document) to another program which processes these results and send them back to the WIC.

5.2.1.2 Processing the Alta Vista results

By processing the Alta Vista results we were able to create features to facilitate their browsing. Instead of having the results distributed over 20 separate pages and where an HTTP request is necessary to retrieve each one of these pages, we concatenated all results in one long page. Therefore, the user only has to wait for the transfer of the first page. The user can work on the transferred pages while the remainder is transferred in the background. Moreover, page flipping is replaced by scrolling up and down the result stream. This feature adds convenience and saves considerable time given the network delay for each HTTP request.

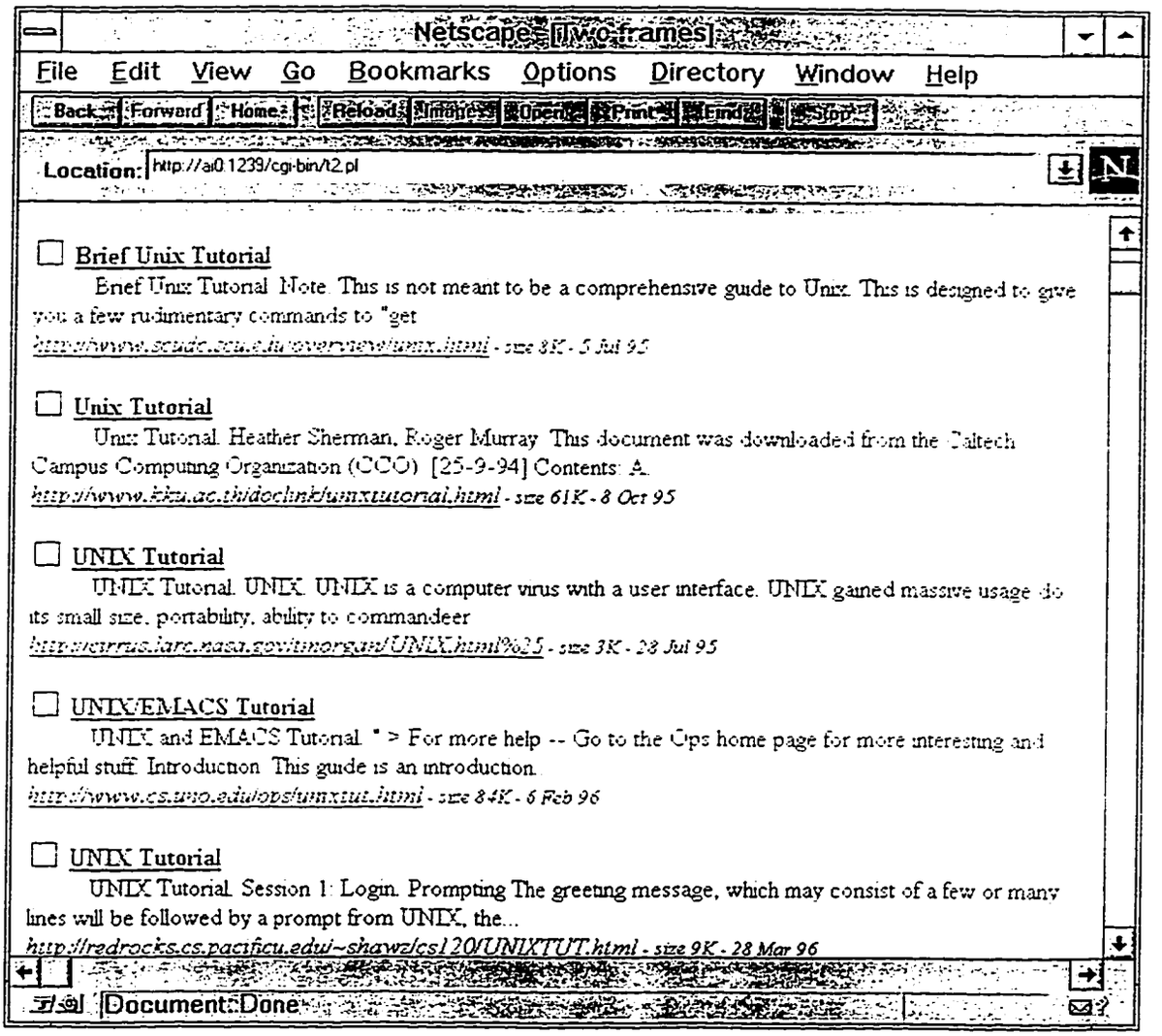


Figure 10: The processed results of Alta Vista

Second, we added a "download" check box in front of the title of each returned document. Alta Vista arranges the returned results as chunks of text separated by a special HTML tag. A chunk corresponding to a returned document contains the document title, its URL, abstract, size and its creation date. We modified this chunk so it contains a check box. When this box is checked, the corresponding document will be added to the list of documents to be downloaded. Thus the user, while reading the abstracts of transferred results, can check any document he or she thinks is worth downloading.

Third, we embedded a "target" statement in the URL of the returned documents. The target statement will cause a document to open in a new browser window (instance of the browser) when its URL is invoked. Hence while examining the returned results, and if not satisfied with the associated abstract, the user can examine the document itself in the second browser instance. More importantly, the user can also follows links originating from the first returned document until they get to a more useful document. All this can be done without interrupting the transfer of the Alta Vista results stream and without having to make any flip back. From our experience, we found that link following, originating from returned result, is of great value and often leads to more useful documents.

5.2.1.3 Selecting URL's to download

Checking the boxes is not the only way to select documents to download. The user can navigate in the separate window (created by the "target" statement) and when the user finds a document worth downloading he or she may select the document. Selecting is done by placing the URL into the text field of the upper frame of the WIC result navigation screen (see Figure 11), then clicking on either "download this page" or "download this tree" (discussed below). Thus this URL will be added to the list of documents to be downloaded. Copying and pasting is certainly not the most elegant way for accomplishing this task, however, the security restrictions of currently available versions of Netscape (version 3.0 beta 5) prevents any more automated way of doing this. Note that because Netscape is still in its infancy as a development environment, many design flaws are still present.

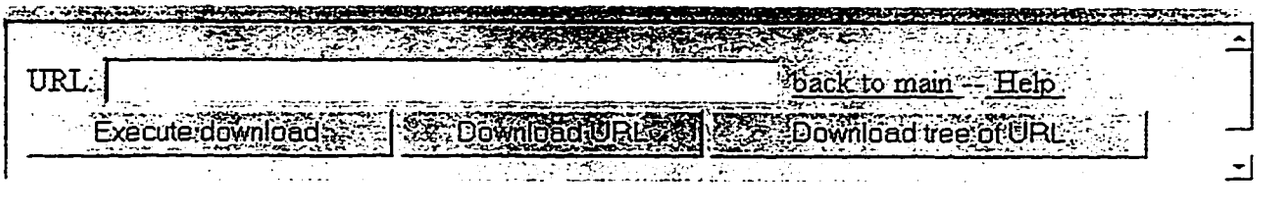


Figure 11: interface for selecting a document by placing its URL into the text field

Note that in the case of box checking, the URL's of the checked document would be transferred via encoding and concatenating them to the URL which invokes the downloading program (a CGI mechanism for passing data). In the case of using "Download this page", the URL is transferred to the downloading program via a text file. Also, when the "Download this tree" button is used, the URL is passed via the text file with a flag indicating that the document should be recursively downloaded (see below section 5.2.1.5 on recursive downloading).

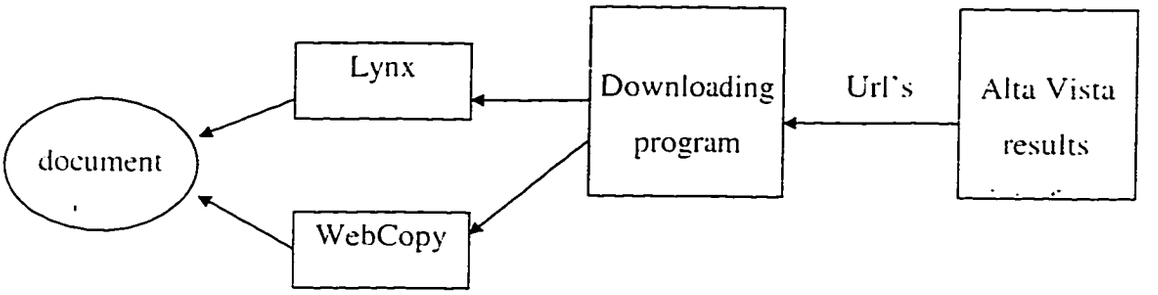


Figure 12: Diagram for the downloading process

5.2.1.4 Choosing the directory to download into

When the user finishes selecting documents to download, they next have to choose the directory into which the selected documents should be stored. Clicking on the "Execute download" button (see Figure 13) will bring up a window in which the directory hierarchical structure corresponding to the document base will be displayed.

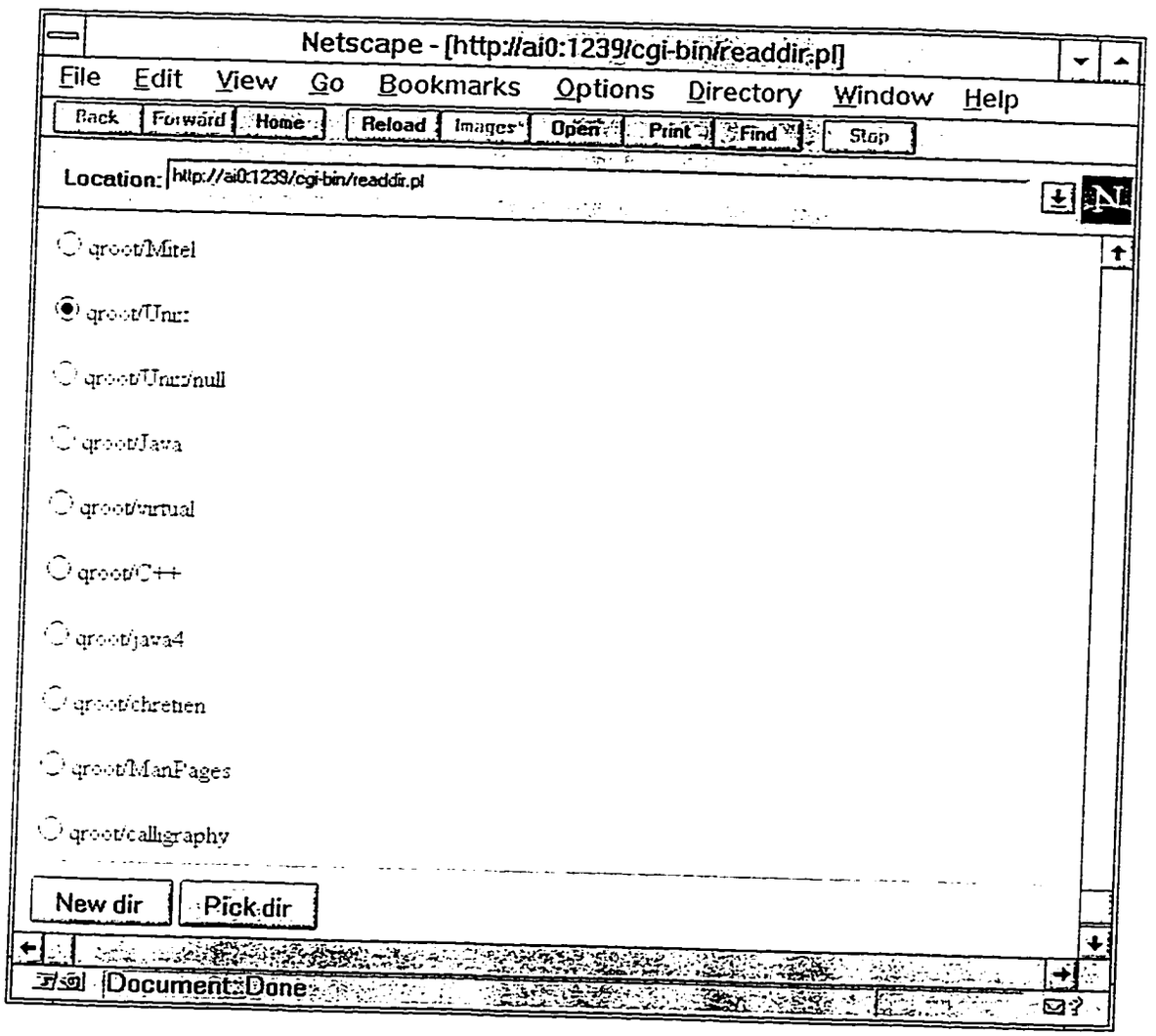


Figure 13: The window to select or to create e a directory

In the window in Figure 13 the user can choose a directory to contain the downloaded documents or create a new one. After choosing the directory (selecting the corresponding radio button) the user can click on the "New directory" button which will pop up a JavaScript prompt window to enter the name of a new sub-directory to the chosen one.

5.2.1.5 Downloading

Documents are downloaded using two mechanisms, simple and recursive. In a simple download the user selects documents to be downloaded individually, while in a recursive

download the user selects a document to be the origin of a recursive traversal which downloads a directory tree of documents.

Simple download

Lynx is also used to carry out the simple download of documents. The URLs of documents, both those encoded into the URL (checked box) and in the text file (copy and paste the URL), will be decoded and passed to Lynx. Next, Lynx will retrieve the corresponding documents and it will create new files in the previously selected directory. The downloaded file will be named differently from their names in their original locations. The reason is that if the same directory may store many files from many sites, there would be a chance to have the same name for multiple files and thus some files would be overwritten. The new file names are generated as a combination of both the URL of documents and their original filenames. For example, a file of name 'file1' and URL 'http://www.csi.uottawa.ca' would have the new name 'http_www_csi_uottawa_ca_file1'. This will minimize the possibility of files being overwritten and yield more information about the content of a file.

Recursive download

For URL's flagged to be recursively downloaded we used a utility called "WebCopy" to recursively download a set of documents.

WebCopy

WebCopy is a freeware utility written in PERL and can be downloaded from the Web (both the executable file and the source code[16]). WebCopy takes a URL of a document, downloads it, parses it to locates all internal links, then recursively follows those links which satisfy some conditions aimed at preventing an endless recursion. By following links, WebCopy traverses new documents where the same process of downloading, parsing and links following would recur on the newly reached document. WebCopy also

[16] WebCopy can be downloaded from: http://www.inf.utfsm.cl/~vparada/webcopy.html

mirrors locally any remote directory in which a document has been traversed and downloaded.

The original WebCopy utility uses as a criterion to prevent endless recursion (or more precisely the unwanted expansion in downloading) a condition on the URL of the referenced document in order to qualify for downloading. The condition is that the traversed document has to belong to the same server as the first document (the origin). That is, its URL has to have the same computer host name and address as the origin document. If not for this condition, WebCopy may enter an uncontrolled expansion in the Web space.

Tree downloading

We have modified WebCopy such that it does a "tree downloading". We used the term "tree" because in this type of download mechanism only "lower" documents in a directory tree can be traversed. That is, the new condition for a document to be traversed would be that it should belong to the same directory of the origin document or to any of its sub-directories. This kind of traversal leads to much more focused results and it was especially suitable for large documents which are broken physically into many smaller documents for efficiency reasons(e.g. tutorials). Often these documents are physically stored in a tree-like directory structure where the physical files are spread among sub-directories. On the other hand, the original version of WebCopy when used to download a document would often cause the whole site to be downloaded (including unrelated documents). This is because most of the time, it is possible to find a reference in a traversed document to the home page of the site and consequently every document can be reached from the home page using this original type of recursive traversal.

Note that mirroring the directory structure of the remote site is also necessary in order to maintain valid the internal links in the downloaded files. We must note that an internal link (URL embedded in a downloaded document) contains a directory path relative to the server when it references another document on the same site. Also, when a directory tree

is mirrored, it will be placed inside a separate directory which would be created for this purpose and named by a name generated from the URL of the origin document. This separation would also help in distinguishing the automatically created directories (the mirrored ones) from other manually created directories. The latter directories typically reflect the author's way of organizing topics in the document base.

5.2.1.6 Inserting URL's

After all the documents have been successfully downloaded, a small program is executed to visit each newly downloaded file in order to insert in the beginning of the file its original URL. The URLs would be extracted and used later for informative and navigation features such as showing the remote version of a downloaded document.

5.2.1.7 Intermediate format

After being downloaded, the HTML documents need to be processed in order to be suitable for paragraph browsing. Remember that in our system the granularity of retrieval is the paragraph. The local document base is queried and the Glimpse search engine will return paragraphs extracted from multiple files and concatenated in one stream to be presented to the user. The concatenation may cause problems because many tags in HTML are stream based i.e. a formatting tag may have effects until it is explicitly disabled e.g. a "" tag will bold all following characters until a closing "" is encountered. If the closing part of a tag is missing because it was not contained in the retrieved paragraph, the effect of the unclosed tag will carry on to the rest of the stream of paragraphs and thus causing a mal-presentation.

In order to eliminate this presentation problem we have to create a new format in which a paragraph can be extracted and displayed without affecting the presentation of other paragraphs. In this format, tags are cleaned on a paragraph basis: each tag is checked for its closing part and if it is missing the whole tag is removed causing a minor loss in the formatting details. Also, some tags which are unsuitable for the paragraph display are

completely removed like image reference tags which displays a question mark icon instead of the image itself because images are not downloaded along with the HTML documents.

In addition, at this stage some information necessary for navigation is embedded in each paragraph as a part of the new format. This information contains the name and path of the file from which the paragraph originated, the URL of that file, and the paragraph location or index within that file (used for the Show surrounding command). Also, the HTML statements necessary for these navigation features are embedded in each paragraph at this stage.

The figure below depicts the HTML content of a paragraph in intermediate format where the upper part corresponds to the included header and the lower corresponds to the text of paragraph. The header is encoded as an HTML "form" composed of several fields, each field holding a piece of the embedded information. These fields along with their values would be passed to the program (the program in the action field "ret.pl") which would be invoked when the form is submitted. A form is submitted when one of its buttons is pushed e.g. the "Show Surrounding" button.

```
<hr> <p>
<form action=/cgi-bin/ret.pl method=post>
<a href= http://ai0:1239/cgibin/qroot/perl/unix.html> Show
local doc </a>
<input type = submit value = "Show Surrounding">
<input name= adir type=hidden
value=/usr4/iyad/public_html/cgi-bin/qroot/perl/>
<input name= afile type=hidden value= unix.html>
<input name= parnum type=hidden value= 1>
<a href=http://www.bluemarble.net/~scotty/Perl/unix.html>
Show remote doc </a>
</form>

<h2> Using <strong> Unix </strong> from Within Perl</h2>
<h3>Writing to a <strong> Process</h3> </strong> Watch
this: What did that do? It wrote to a <strong> process
</strong> rather than a file. That little pipe (|) on the
first line tells any output to the filehandle MAIL to go to
a sendmail <strong> process; </strong> when you close up
the <strong> process, </strong> with the close statement--
presto, you've sent yourself a mail message.
```

Figure 14: HTML contents of a paragraph in the intermediate format

The final result of the intermediate format process is that for each directory of the document base, an HTML file containing all paragraphs of that directory will be created. This file contains a concatenation of all paragraphs prepared to be extracted and displayed individually (containing all information and statements for further navigation features).

5.2.1.8 Indexing

Finally, after the intermediate format files are created, the newly downloaded files are indexed by Glimpse. We used the incremental indexing feature of Glimpse which proved to be very efficient compared to reindexing the whole document base at each modification. Only the files containing paragraphs in the intermediate format are indexed by Glimpse because only the paragraphs out of these files shall be retrieved in response to a Glimpse search. These files are created when new directories are created; otherwise, the paragraphs of the newly downloaded documents would be appended to these files at each time documents are downloaded into an existing directory.

5.2.2 Information retrieval with the WIC

The second major function of the WIC is to access the document base. Similarly to the classical information retrieval, this is done by querying and retrieving results. We have made the retrieval unit to be a paragraph instead of a document. We have also added a ranking capability in addition to some other enhanced presentation and browsing features. Moreover, under our system it is possible to browse the document base on a directory and file basis.

5.2.2.1 Searching

Glimpse is the only means for searching the document base. The motivation to use Glimpse is mainly its efficiency as discussed in Chapter 3. Additional advantages are that it is a freeware product and that it has a high degree of flexibility, comprehension and ability to be customized. For instance, it is possible to set a user defined record delimiter, where a record is the retrieving unit and the record delimiter is the character sequence which signals the boundaries of the record. In our case we set the record delimiter to the "<p>" tag which is used to denote paragraphs in HTML in order to retrieve on paragraph basis.

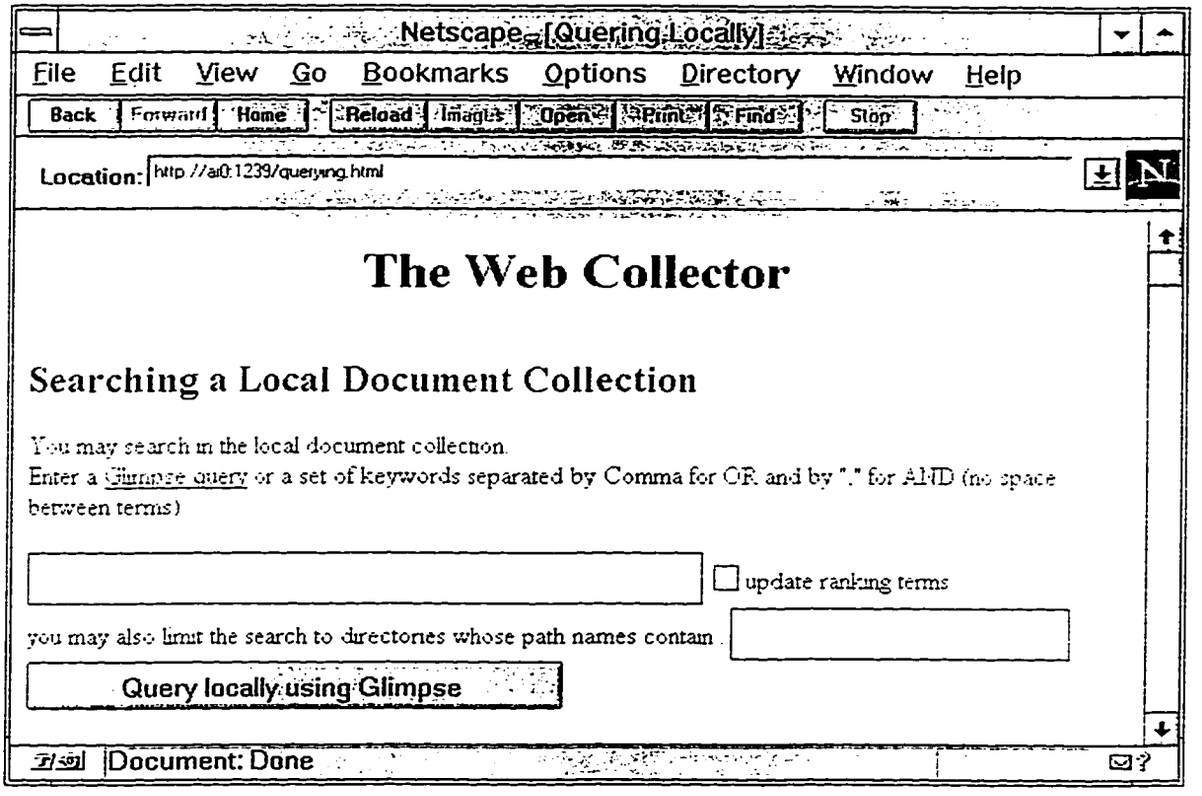


Figure 15: The Glimpse interface

In Glimpse it is also possible to limit the search scope to a sub-branch of the document base. The user may enter a string which would cause the search to be limited to the files whose path name contains that string. Practically, this feature is used to limit the search scope to a certain topic in a document base. A document base would be typically partitioned into many branches, where each branch would be contained in a separate sub-directory and correspond to a particular topic.

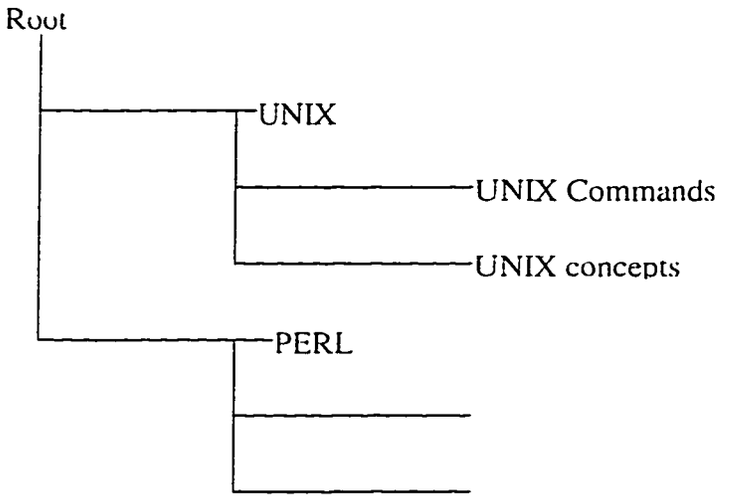


Figure 16: Directory structure corresponding to document base.

For example, in the above figure, if a user types "UNIX" in the limiting search field (second text field in Figure 15) only the directories under the UNIX directory would be searched.

5.2.2.2 Paragraph Extraction

Similarly to what we have done with Alta Vista, we have also developed an HTML interface to Glimpse (see Figure 15). From the main page of the WIC, the user can enter a query (in Glimpse syntax) which would be passed to the Glimpse search engine. Next, Glimpse performs the search on the indexed data in the local document base and returns the results which satisfy the query.

The results are next routed to a ranking program. The ranking program approximates the relevance of each paragraph to the query, then outputs the paragraphs in their relevance order.

The ranking algorithm looks at the occurrences of each one of the "ranking terms" used within individual paragraphs. The ranking terms used by default for ranking are the terms contained in the query. The ranking terms may also be modified by the user. Checking the appropriate check box (note the check box in Figure 15) will cause a window to open in which the user can change, delete or add terms which will be used as the ranking terms for the ranking algorithm (see Figure 17).

In order to denote the ranking value, a numeric relevance value is computed equal to the total number of matches (between ranking terms and document terms) in a paragraph and this value is assigned to the paragraph. These values are then sorted by their decreasing order and the corresponding paragraphs are sorted and displayed in that order.

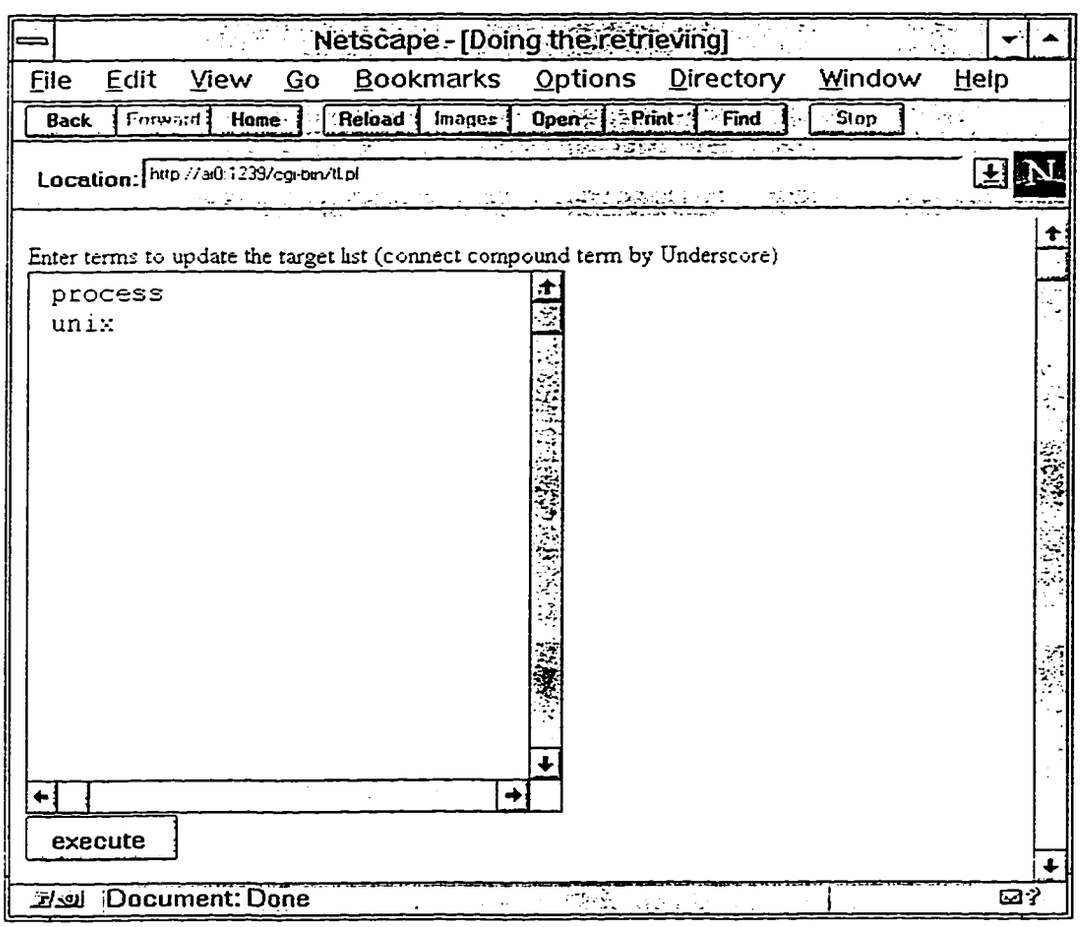


Figure 17: ranking terms modification window, for query on UNIX process

5.2.2.3 Presentation

At each occurrence of a match between a ranking term and a paragraph term a bolding tag is inserted in order to emphasize the match for the user convenience. The bolding of matches facilitates navigation and relevance determination. An instance of the ranked output is shown in Figure 18.

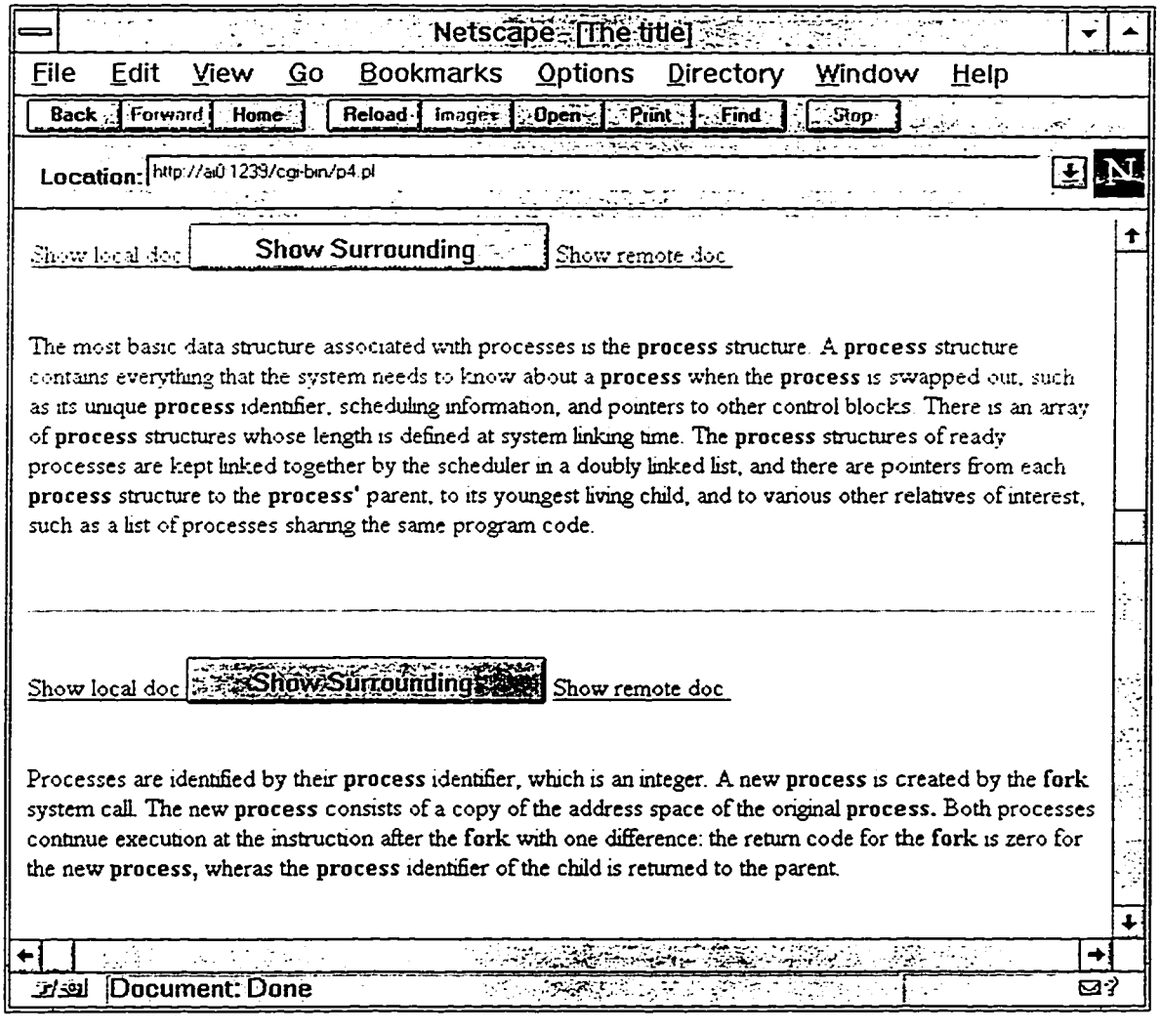


Figure 18: Two paragraphs of an output of query on "process"

As shown in Figure 18, the displayed paragraphs are padded with a button and two links. These are the results of the HTML tags (statements) which were embedded during the preparation of the intermediate format along with other information (e.g. file name, paragraph number).

The button "Show surrounding" permits the user to see the upper and the lower paragraphs surrounding the displayed paragraph (inclusive) in the original document from which the designated paragraph has been extracted. This is useful when a single paragraph does not convey enough information to tell about the document relevance. The embedded file name of the document and the paragraph number are also used in conjunction with a paragraph index stored in the directory. The index file stores the location for each paragraph within its file in order to permit locating and retrieving the surrounding paragraphs.

The "Show local copy" link permits the user to see the whole local document from which the particular paragraph has originated. We have included this feature because we expect that when a user finds a paragraph of interest, he or she would most likely want to take a look at the whole document for any associated information.

The "Show remote" link retrieves the document containing the paragraph from its original location on the Web. This feature is useful for the user who wants to see the document images (they are not downloaded and thus do not appear in the local document), to see if the document is updated, or to browse the original "neighborhood" of that document e.g. related documents in the same site.

5.2.2.4 File browsing

File browsing permits exploring the content of the document i.e. navigating the directories of the document base and browsing through files stored in those directories. Initially, the top level directory name is displayed as a set of links, then the user can choose the exploration path by clicking on a directory name. Doing so will cause the sub-

directories of the selected one to be displayed also as links along with any files at the same level of the sub-directories. Then the same process can be repeated recursively for these sub-directories.

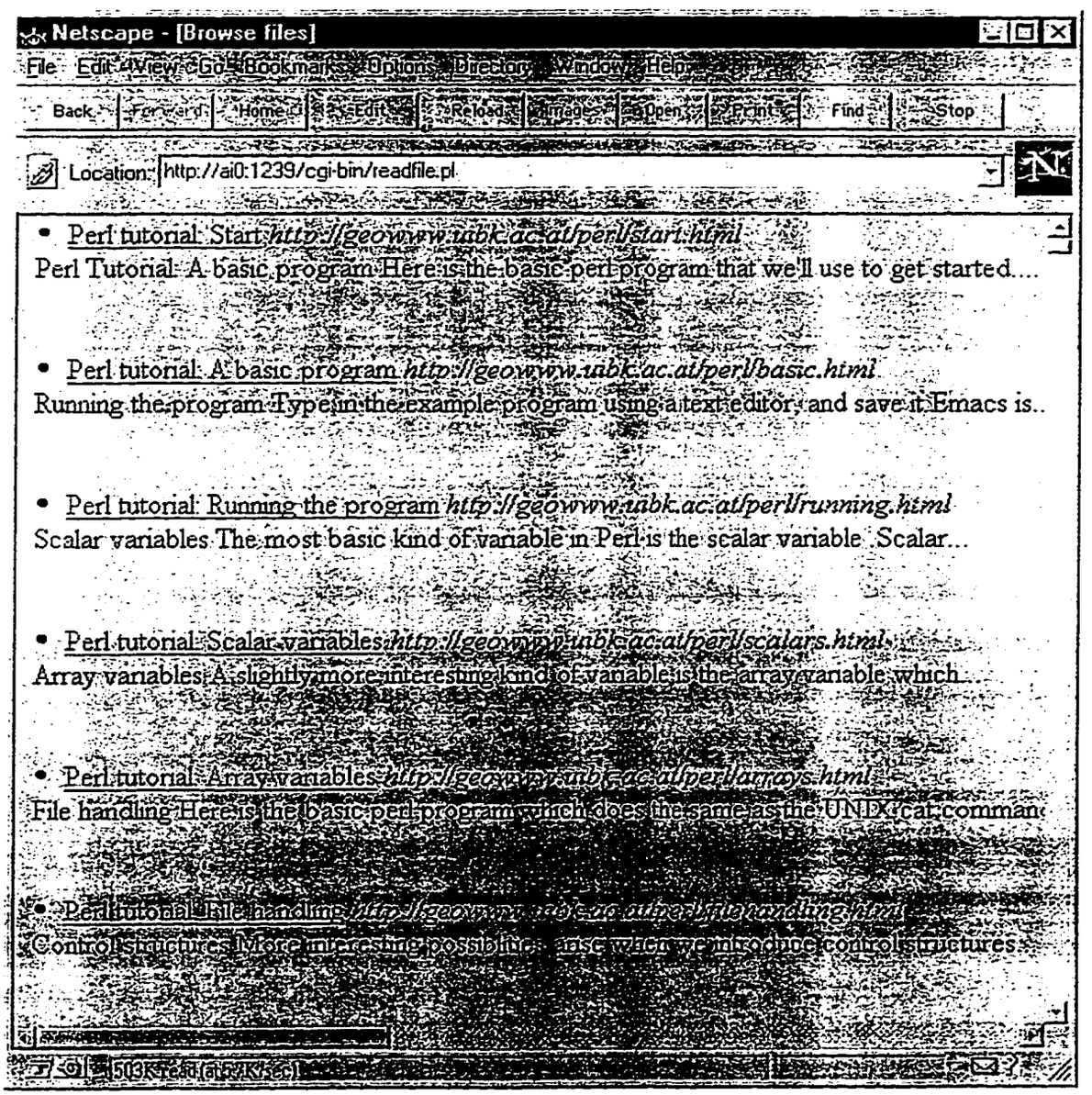


Figure 19: File browsing where the length of file abstract is set to one line

The file names are also displayed as links so the user can see the content of a file by clicking on the link. The URL's are also shown with the file names because the parts of

the URL may convey important information about the nature of the file (e.g. if it originated from a university or from a company). Along the file name, a small abstract formed from the first few lines of the document file is also displayed. The user can control the number of displayed lines in the abstract document before invoking a file browsing session.

5.3 ***Experiments and results***

5.3.1 **Gathering**

The WIC was tested by gathering documents from the Web in order to build a local document base. The topics of interest were : UNIX, Java, and PERL. We have selected these topics because we are familiar with their concepts.

5.3.1.1 Querying and downloading

We began by using the Alta Vista query interface, inputting several forms of queries. The initial queries yielded few useful documents among much more irrelevant ones. It took many iterations of query calibration in order to begin achieving better results.

Some of the queries which yielded the best results were in the form "introduction to <topic>" (e.g. "Introduction to UNIX") or "<topic> tutorial". These patterns appeared to be typical in "content rich" documents. This demonstrates the importance of proficiency in formulating queries, or in other words how elusive "good" are queries when searching the Web.

Even after using all the time saving features of WIC , gathering a megabyte of information by downloading individual documents was a relatively lengthy process. The delay is not only because good queries or good documents are hard to find, but also because usually the documents on the Web are relatively small. The typical document size was in the order of few kilobytes. On the other hand, we have found that the

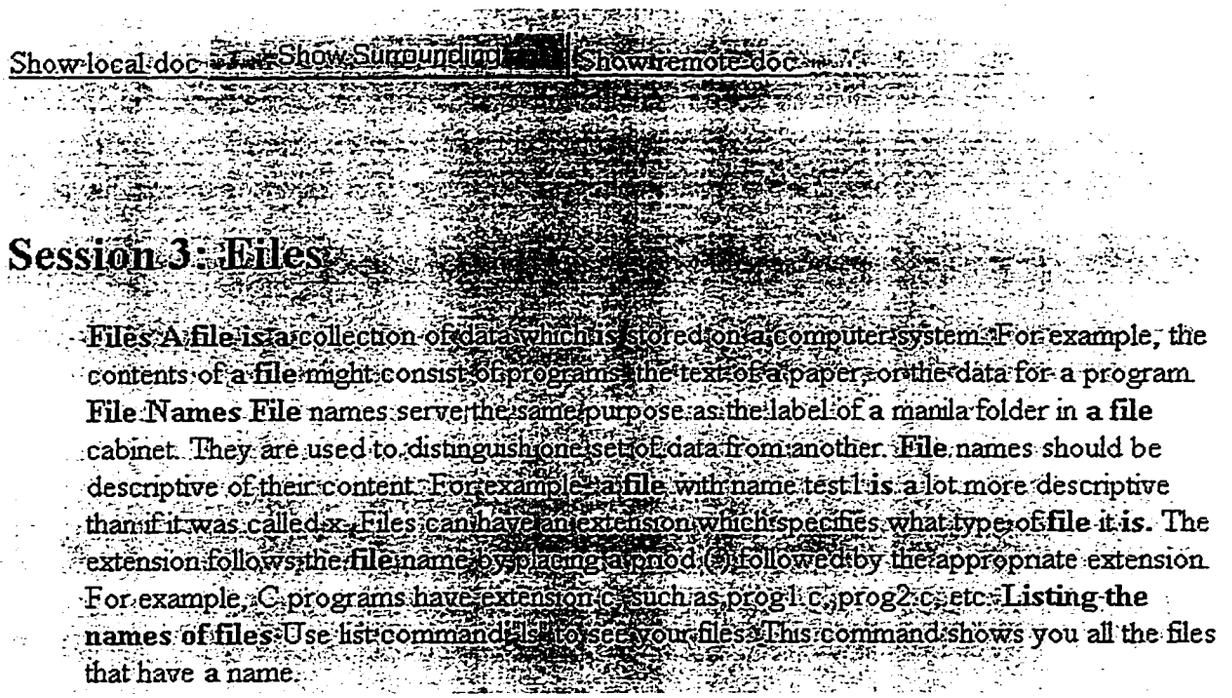
recursive downloading is much more efficient. Typically a recursive download for a document yielded between 20 and 50 documents. Most of the recursively downloaded documents were tutorials on particular topics. Often, tutorials are stored in a way such that choosing the top page as an origin for recursive download would cause all the tutorial documents to be downloaded. Therefore, in order to locate such top pages, we began including in our queries terms which typically occur in such pages (e.g. the term "table of contents"). We have also came across other useful types of "cluster of documents" which have similar structure to tutorials e.g. thesis, papers and Frequently Asked Questions (FAQ).

5.3.1.2 The document base

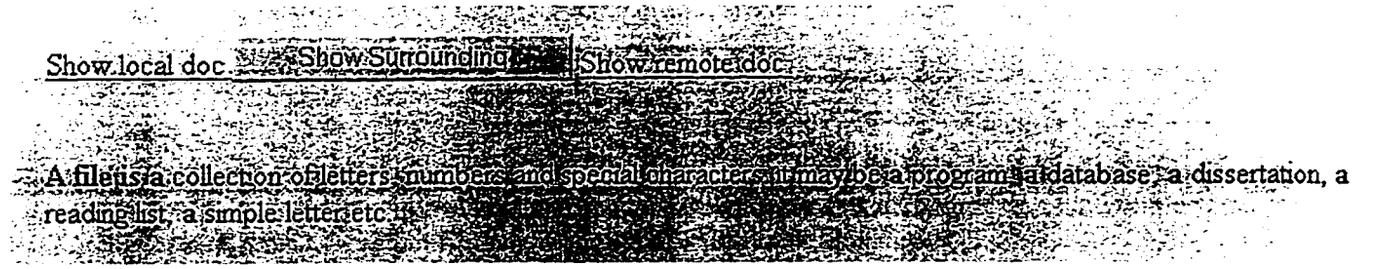
The resulting document base was made of 10 megabytes of data on the three topics (3 Mb on UNIX, 2.5 Mb on PERL and 4.5 Mb on Java). The document base has been gathered in multiple gathering sessions. Around 50 recursive download operations have been carried out on the three different topics. In order to enhance the comprehension of the document base, we have also included a subset of the UNIX man pages after converting them to HTML format.

5.3.2 **Querying the document base**

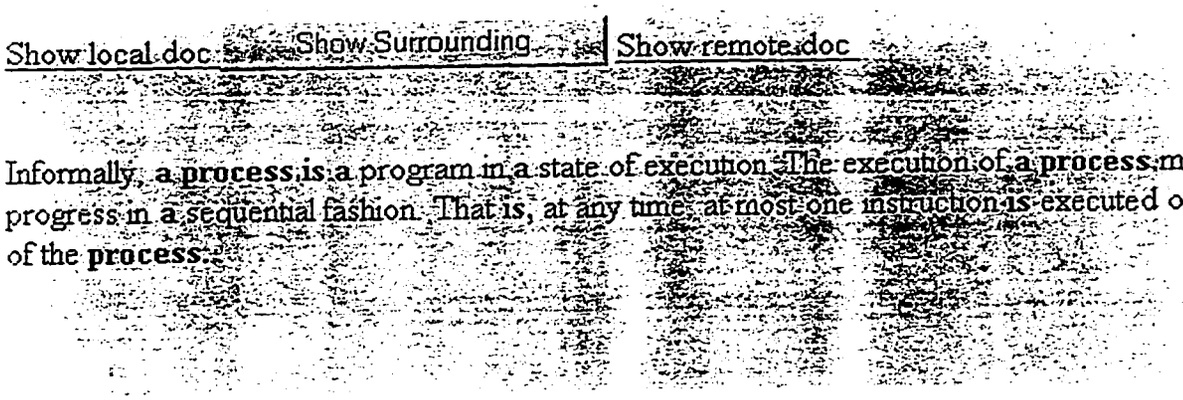
The second part of experimentation was to query the document base. Our goal was to determine how successful the document base would be in answering common queries. We have chosen to query the UNIX parts for some UNIX concept definitions such as the definition of a file or a process. Using Glimpse query syntax we began by using simple query such as looking for "file". The Glimpse output was fast and abundant, but most of the returned paragraphs were not definitions. By quickly scanning the output, it was easy to note that the query was too general. We chose the second query to be "file is a", the definition showed up as the first returned paragraph:



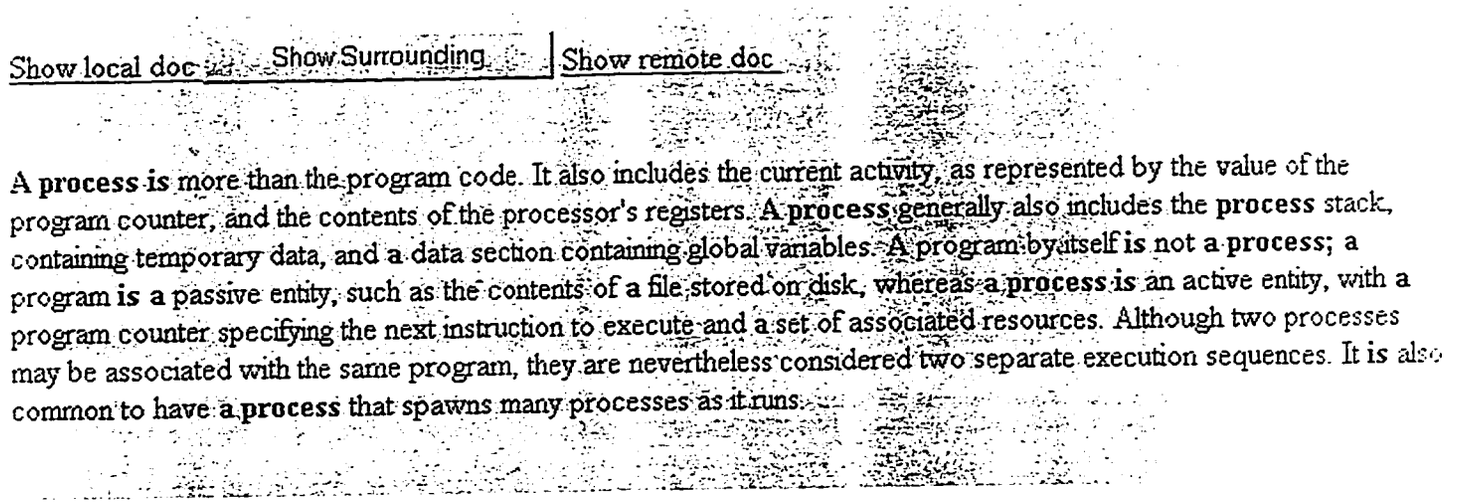
Other useful paragraphs followed (after some irrelevant ones):



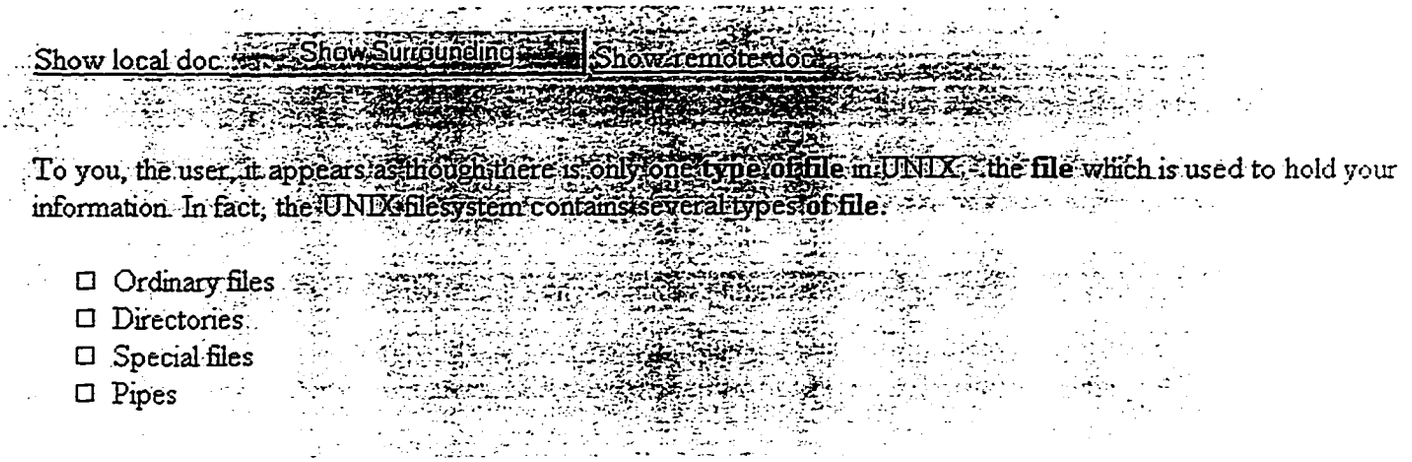
Learning from the success of the first query, we tried the query "process is a" to find the definition of a process. One paragraph was returned which clearly was a definition:



Relaxing the query to "a process is" yielded another definition:

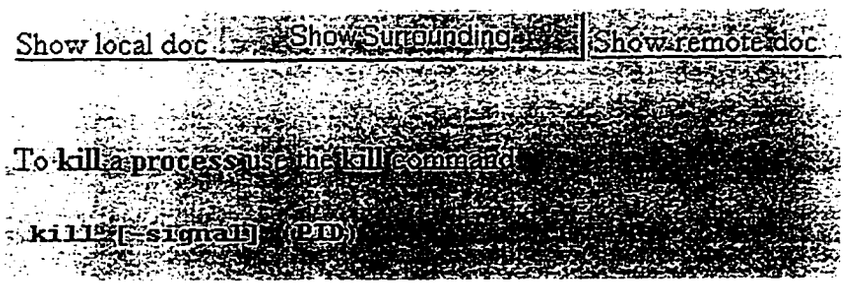


Next we moved to locating more information about the designated concepts. We tried to find out the types of files and processes. For files we use the query "type of file" , 3 out of 15 returned paragraphs were relevant, the most useful one was:



When we tried to do the same on process, no paragraph was returned even when we used different variations for the query such as "kind of process", "process type", "process of type".

Next we tried to locate the syntax of some operations. We used query "kill:process " which means that paragraph containing "kill" and "process" will qualify. Fortunately, many good paragraphs were returned, for example:



5.3.2.1 Results & problems

The first observation from the experiments is that results were abundant for certain queries and scarce for others. The query "type of file" returned 15 paragraphs while "type of process" returned none. This raises a serious issue about the quality of collected

information from the Web. In our case, the core of the gathered information originated from tutorials on UNIX (the experiment was only on UNIX information). These tutorials are usually offered by universities as an introductory text for their students on how to use their UNIX operating systems. Therefore, such tutorials would mainly cover "how to do" topics rather then other more advanced issues. Thus all of the gathered information tends to be similar and collecting more would only increase redundancy. The question of why we could not find more advanced information may be due to motivation i.e. why would someone publish such advanced information on the Web instead of selling it as a book?

The second observation is that calibrating queries in order to get to the required results, in the contrast to the Alta Vista case which takes long time, was a very practical process. Glimpse response time is very fast and assessing its output was very convenient. Yet, many returned paragraphs are small in size, often not enough to contain a coherent information unit, thus "Show surrounding" was necessary. Also, the ranking sometimes presents results in an unexpected way. A paragraph may show first not because it is the most relevant but because it is the largest. That is because the ranking algorithm depends on the number of matches, which is likely to increase as the size of a paragraph increases.

Finally, the Glimpse index was found to be brittle, in many occasions it was corrupted. Thus, we have added a reindexing ability which should be used whenever the Glimpse output looks bizarre.

6. Chapter 6: Discussion and Future development

In this Chapter, we will conclude the thesis by discussing the main strengths, weakness, and problems encountered in the WIC. Finally, we will suggest some enhancements for the future.

6.1 Assessment of the WIC

The motivation behind the WIC system was to provide a more effective and time saving solution than the currently available tools for using the information of the Web. In particular, the system addresses cases where users are interested in small facts or answers in restricted domains. We tried mainly to address the problems associated with search engines, the most popular tools for information exploration on the Web. Perhaps, the most important problem is that they leave much work undone; we have tried to automate some of this.

6.1.1 Advantages of the WIC

In this section we will discuss how successful was our solution as an addition to search engines for Web-based information retrieval. In particular, we will describe how it saves times as opposed to relying only on search engines. Afterwards, in the following sections, we will discuss when these time saving features apply i.e. when it begins to be feasible to use the WIC.

1. Precision enhancement

The first problem in search engines that was addressed in the WIC is the low precision -- having to waste time exploring mostly irrelevant information. We can safely claim that by using a WIC document base one can achieve much better precision than by just using "raw" search engines. Instead of having the entire Web as the pool from which information is queried as in search engines, the information source in the WIC is a collection of focused documents, the local document base. In fact, stored information in

the document base has been humanly verified for relevance before being added. This will most likely yield results of high precision and thus be time saving providing that the feasibility of gathering a document base then querying it will hold (see section on cost below).

Moreover, for more precision in the WIC, we have included the ability to tailor the storage of information to be associated with a set of topics: then the user can limit the scope of search to a particular topic.

2. Results navigation

The WIC also addresses another time consuming problem in search engines, the inconvenience or difficulty in browsing the results of search engines. Typically, in search engines ambiguous abstracts are returned with the returned document to give an idea about the document contents. But often one has to spend considerable time to read the document itself because the abstracts are not informative enough. On the other hand, the WIC returns a better window on the returned documents-- the paragraphs in which the matches have occurred. These paragraphs are very likely to be more informative about the contents of a document. The WIC also provides a rapid way to enlarge this window by displaying the surrounding paragraphs of the returned paragraph or by rapidly displaying a local copy of the document to which the paragraph belongs.

More importantly, returning the paragraph in which the matches occurred may even replace the need to deal with the entire document; the answer may lie within the returned paragraph or within its surrounding paragraphs. This means considerable time is saved in finding the required facts or answers. Also, within the returned paragraph, matches are bolded (emphasized) which makes grasping the theme of the paragraph even faster and easier. It is very likely that it would be enough to tell about the theme by reading the sentence in which the match occurs. This sentence is easily distinguishable due to the bolding.

Finally in the WIC, returned results are displayed as one scrollable stream of paragraphs. In search engines results span many pages and considerable time is lost moving between these pages.

4. Network delay

Because all Web documents are remotely stored, the network delay of the Internet plays a major role in consuming time for navigating results. In search engines, there is a network delay at query submission, at each time the user wants to look at a returned document, and each time the user wants to flip from a subset of results to another. In the WIC, all of these steps are done on local information so the network delay of the Internet is eliminated .

5. Time saving in gathering

Even when dealing with search engines (in gathering), the WIC can save time. The Alta Vista interface in the WIC is more convenient and time efficient than directly accessing search engines because it contains two additional features. First, results (intercepted from the Alta Vista search engine) are concatenated in one stream eliminating network delay for flipping between pages of results. Second, an additional browser instance can be opened for exploring returned documents without interrupting the Alta Vista results transfer into the original browser instance. Or even it is possible to examine the Alta Vista results in the original instance while a document is being transferred into the second instance.

Moreover, the downloading scheme in the WIC -- to choose all documents first then to download all of them by one operation -- incurs considerable time saving. Instead of having to wait for downloading a document in order to be able to move to another one, in the WIC one can choose all documents to be downloaded then one command that causes all chosen documents to be downloaded sequentially.

6.1.2 Cost of the WIC

On the other hand, there is a price to pay for using the WIC: the cost of gathering information to build a sufficiently comprehensive document base. The gathering process may be very time consuming as it involves using Web search engines. Additional cost includes the downloading and indexing time for information, deciding on a document base hierarchy and other minor details. In general, using the WIC involves more overhead than using directly search engines, especially because it involves local storage of data.

Also the WIC is much less comprehensive than search engines, the search scope is being limited to what has been manually gathered. Thus a large gathering time is necessary before a useful document base is created (see Chapter 4 on gathering experiments).

6.1.3 When the WIC is better?

In conclusion, we think that using our system is more suitable than search engines for situations where users are interested in specific topics and expect to query on them for small facts frequently and for a long time. A user's community would be able to incrementally form a good repository of information and later use it more efficiently then querying the web directly. In such a situation, the more frequently users use this repository, the more feasible the WIC would be as an alternative for information retrieval. The repository of documents we have collected (on UNIX, Java , and PERL) is a typical example of a frequently used repository.

Otherwise, it is more suitable to use directly search engines to locate casual queries or to collect large information on unpopular topics.

6.2 Web-based development: advantages and disadvantages

As an inventory for the development experience of the WIC system, we will discuss in this section the Web development process and compare it to conventional software development. Remember that we mean by Web-based development the development of

programs which run in a Web browser. On the other hand, a conventional system is a system whose programs receive input and send outputs directly to the local file system (via the operating system).

6.2.1 Advantages

The implementation process of Web-based systems is significantly different from conventional development. New problems that increase the complexity are involved mainly in the area of communications between the program, the server and the user. Yet, the Web-based development has many significant advantages:

- It provides a universal access to a software system: any one who has access to the Web can access and use a Web-based system.

- Due to its client server architecture, the user would not have to buy, store, install a system in order to be able to use it. He or she can pay per use (if any) and access the system simply by invoking its URL from his or her own Web browser.

- Data sharing and accumulation is also a key advantage as demonstrated in our system. The document base can be accessed by many users and each one can contribute by accumulating information where his or her contributions may then be shared by others instantly. Thus all users would be accessing up to date data and instantly disseminating any changes or improvements.

- Using the Web as platform means that no extra hardware, software or labor are necessary to afford the connectivity (networking) of the system.

- HTML provides a common unique interface for all Web-based systems. Most of the users would be familiar with the HTML user interface, thus they will have a good learning curve if any learning is necessary.

6.2.2 Disadvantages

On the other hand, during our development experience, we have found the following disadvantages:

1. Complexity of debugging

Debugging is difficult because of the distributed topology of Web systems. Input data could not be entered directly to the programs; instead, it is inputted in the browser then encoded and transferred to the Web server then to the program. Tracing data is very difficult in such a topology. For example, when data is mal-formed the server often fails, producing short and ambiguous error messages even when the problem is not located at the server.

2. Bad communication between programs

Often done via text files, the communication between the different programs in web systems is very awkward. Typically, a Web system generates a large relatively number of programs which are stored physically in separate files. Programming modules have to be separated physically because of the event driven nature of the Web interface. We needed to have a separate program file for each event handling procedure because of the way events are associated to programs. In CGI programming, an event can only invoke a program file instead of being able to invoke a subroutine within a file as in conventional programming.

3. Unreliable platform

Netscape, the prominent platform for Web systems, is still in its infancy as a platform for applications. Bugs are still frequent in Netscape and versions are being released within short intervals (we were working with version 3 beta 5) causing confusion due to the compatibility problems between different versions. Moreover, Netscape still

suffers from many flaws in its design particularly when dealing with JavaScript and with the other enhanced interface features like frames.

4. Security vulnerability

Security is a traditional concern for the Internet community. As opposed to classical software systems running on private network which is physically isolated from the external world, Web systems are more vulnerable to unauthorized access. There would always be a chance for someone around the world to find a flaw in the software security measures of a Web-based system.

5. Slow processing

Perhaps the execution speed of Web systems is the major disadvantage for this development paradigm. Aside from the network delay which is hardware dependent, we noticed that usually the response time in a Web system is relatively long for many operations compared to the conventional systems. One reason maybe due to the CGI programming language used (PERL) which is an interpreted language. Also, compared to conventional programming, Web systems incorporate much more overhead; particularly because data has to be communicated between three points-- the browser, the program and the server. At each point there would an overhead incurred as a result of encoding, decoding and interpreting the data.

6. Inflexibility of interface

Web systems use normal HTML documents as their user interface. But an HTML based interface is awkward because any modification for an interface element can not be reflected without a full reload of the HTML document. Reloads can be time consuming and thus there is no notion of real time interface update.

Several techniques are being introduced to alleviate this shortcoming of user interface with more or less success. In particular, JavaScript is providing some built-in dialog

boxes, but these are still considered inflexible relative to what is provided in conventional systems. A proper solution is currently evolving with Java.

6.3 Future enhancements

The WIC is after all a prototype, not a perfected and optimized system. Many enhancements and improvements could be done. In particular, we believe that the following are the most relevant enhancements:

6.3.1 Suggested minor enhancements

1. Concurrent downloading

In order to shorten the downloading time of documents, we would like for the concurrent downloading of documents to be investigated. Concurrent downloading can be achieved by creating many instances of the Lynx process, for example. Each instance would be assigned a subset of URL's to download their corresponding documents, thus downloading would be carried out in parallel and not in serial. Because the bottleneck of downloading lies in the Internet and not in the machine resources, we believe that this may reduce the downloading time. However, as always with parallel execution, much more overhead would be necessary to deal with.

2. More intelligent paragraph recognition

In the WIC the only criterion used to recognize a paragraph is to treat some HTML tags like the " <p>" tag as paragraph delimiters. We found that this often would yield inaccurate results; paragraphs are sometimes missed or in other cases a series of paragraphs is treated as a single paragraph.

Ideally, the "<p>" tag, which is designated specifically for the purpose of denoting paragraphs in HTML should be enough to recognize a paragraphs. However, the poor HTML authoring practice deprived us from relying on this tag. Often, HTML authors around the Web do not abide by the semantic meaning of the <p> tag as a tag designated

at identifying a paragraph structure. For example, the "<p>" and "
" (a tag which cause a new line in HTML) tags have been widely used interchangeably because from the presentation perspective both tags have the same functionality. Unfortunately, there is no means in HTML to enforce HTML authors to abide by the proper semantic meaning of the tags.

We would like to see more analysis to be done on recognizing paragraphs. One suggestion is to enforce the length of extracted paragraphs to fall within a certain range when irregularity in authoring is suspected.

3. Querying more than one search engine

Currently we are using only Alta Vista as the search engine for gathering Web documents. But as discussed in Chapter 4, no one search engine has a total coverage of the entire Web. Better results could achieved if more than one search engine are queried and results are merged .

4. Better ranking

The ranking algorithm used in the WIC is very straightforward. Many enhancements could be done to achieve better performance. A good place to begin is to assign weights to the terms used for ranking. A weight -- a numeric value typically between zero and one which is associated with a term -- would reflect the importance or contribution value of a term in determining the relevance of a paragraph. The ranking (or relevance) value of a paragraph would be then equal to the product of the number of occurrences of a term by its weight.

The weight will improve the accuracy of ranking because it is a mechanism by which the difference in importance of the terms is realized. The difference in importance can originate from different sources. Examples of the sources of difference are the user judgment, and the relative location of a term in a document e.g. if a term occurs in the title it should have more weight than if it occurs in the body.

Also, it would be useful to take into consideration the size of a paragraph in order to reduce the bias of the size in the ranking result. As described in the experiments conclusion, the big paragraph is more likely to contain more matches because it contains more terms and not because it is more relevant.

5. Automatic location of top pages

By top page we mean the document which is intended to be an index for a collection of related documents, or at least the top of a hierarchical structure. An example would be the table of contents of a Web tutorial which contains links for all documents of the tutorial. Locating the top page is a very vital operation for gathering information from the Web. In our experiments we have frequently manually followed a long chain of links in order to locate the top page which often constitutes a good candidate for recursive downloading.

It is quite puzzling why HTML does not support a special tag to denote such pages (and any other special purpose kind of pages). But in the absence of such HTML tags, we suggest one could locate top pages automatically by using some heuristics such as searching for keywords typically occurring on top pages e.g. "table of contents" or even keywords in the URL like the keyword "top" .

6.3.2 **Suggested major enhancements**

In the long run, we believe that a more sophisticated WIC should address the following points:

6.3.2.1 Semantic document classification

Although the WIC reduces the need for external search engines, we still rely on external engines for gathering. The Alta Vista HTML interface in the WIC constitutes an improvement over directly querying search engines. Yet the WIC user who needs to

gather information from the Web has still to suffer from the low precision of search engines.

We believe that an important step toward increasing the efficiency of the gathering process would be to incorporate a mechanism that improves the precision of search engines. We suggest doing this by having a semantic based classification of documents, i.e. to be able to distinguish documents by their information content. From our experience, we found that a very desirable classification would be to distinguish between "content rich" or academic documents from commercial advertisements. For example, it would be highly desirable to filter out documents of advertisements on UNIX books when the user is interested in technical information about UNIX.

One way to achieve such a classification would be to analyze the contents of documents using natural language processing (NLP) techniques. Such techniques should be linguistically aware and should be able to approximate the meaning of the text content of a document.

In the absence of NLP, we suggest making use of more "shallow" techniques which may yield acceptable results. That is, to use heuristics such as looking for "typically occurring keywords" in each class of documents, e.g. the keywords "price" for advertisement based documents or the keyword "abstract" for academic papers. Other suggested heuristics are to compute the proportionality of images and links versus text (as content rich documents tends to have more proportion of text) and to look into the URL of a document (where URL's containing ".com" tend to be advertisements, for example).

6.3.2.2 Document base administration

Under the current system, the document base can change in one direction only -- to be incremented by new documents. Although that it is possible to bypass the web browser and to operate on the document base files at the operating system level e.g. to delete files, there is still a need to have a better administration of the document base contents.

We believe that it would of great value to have the ability to administrate the information in the document base at a paragraph level, mainly because the paragraph is the unit of retrieval. We mean by the term administration the tailoring and cleaning of the information. That is, to incorporate a mechanism which permits deleting obsolete documents and removing junk paragraphs, and adding other new "good" paragraphs. Such a mechanism should help in removing redundancy and enforcing information integrity. IKARUS, as discussed in the following section, could have the role of keeping track of the "good" paragraphs and providing accessibility to these paragraphs.

We also suggest incorporating another mechanism to increase the visibility of good paragraphs. That is, to bias the ranking algorithm in order to make it increase the relevance of what a user has previously marked as a good paragraph. We must note that the ranking algorithm not only decides on the order of paragraphs but also may determine if a paragraph would be presented to the user or not. Knowing that, only the best N paragraphs (having the N top relevance value) of all retrieved ones will be presented to the user.

Accomplishing these suggestions needs more control over paragraphs. More control means that a more sophisticated user interface is needed which is difficult to implement with CGI or with the current version of JavaScript. Java may be a good candidate, though. Also, it would be very helpful to incorporate a database engine, such that each paragraphs would corresponds to field of a record. Other fields of the record may serve other administration purposes, e.g. to implement the relevance biasing by a user to the ranking algorithm for "good" paragraphs. Adding and removing paragraphs would be certainly easier if a DBMS is involved.

6.3.3 IKARUS and the WIC

As mentioned in Chapter 1, the WIC , other than being a stand alone application, can also be used as a part of IKARUS. Both systems are designed to be integrated together and when they do it would be a major improvement toward better information and knowledge management solution. Following is a description of how integrating IKARUS and the WIC would be done, and what could be offered.

6.3.3.1 What is IKARUS

IKARUS(Intelligent Knowledge Acquisition and Retrieval Universal System) is a system being developed at the LAKE lab in the University of Ottawa. IKARUS is meant to provide a comprehensive solution for knowledge management problems.

IKARUS is a Web-based system which uses a frame based knowledge representation and contains a frame processing engine. The IKARUS knowledge base allows one to arrange concepts in an "isa" hierarchy and to attach (store) statements with these concepts. The "isa" hierarchy allows for some inference features like inheritance of statements between concepts and their descendants. Using IKARUS, knowledge bases can be built on certain domains of knowledge; this knowledge can be queried and displayed in many forms. For more description on the IKARUS project see [Skuce 96].

6.3.3.2 Integrating IKARUS and the WIC

The WIC has been designed to be integrated with IKARUS. For example, the WIC ranking program expects the list of terms used for ranking to be given externally-- to be eventually imported from IKARUS. We envision that the ranking terms list would be passed by IKARUS to the WIC and it would contain an expanded list of terms including synonyms to the query terms, in addition to the original query terms. Possibly, the list may also contains other domain knowledge related terms inferred by the knowledge engine of IKARUS providing that an appropriate knowledge base is in IKARUS. This is supposed to enhance the ranking performance.

IKARUS could also provide facilities for saving queries (that yield good results) within an IKARUS Knowledge base. Also it would be very useful to be able to save the results of a query (when these results are satisfactory) or to save any individual "good" paragraphs out of these results. We have created a paragraph index for the document base in order to make possible relating paragraphs to an IKARUS knowledge base by storing pointers into the knowledge base.

The accumulation of good queries and good results would significantly improve the retrieval effectiveness. Formulating a "good" query is not a trivial task: it needs considerable skill and sometimes luck to formulate a "good" query. Thus it would be very useful for "good" queries to be available to other users.

Saved queries would be available to the user and saved results would be returned first if the user had a similar information need. This accumulation of relevant and focused results is the main design philosophy of the WIC. That is, to narrow the search space, make information more focused to increase precision, and to accumulate and share efforts.

7. Appendix A

In this appendix we will describe the architecture of the WIC system from a programming perspective. The architecture defines what is the function of each program and how the different programs and HTML pages are related.

7.1 Gathering

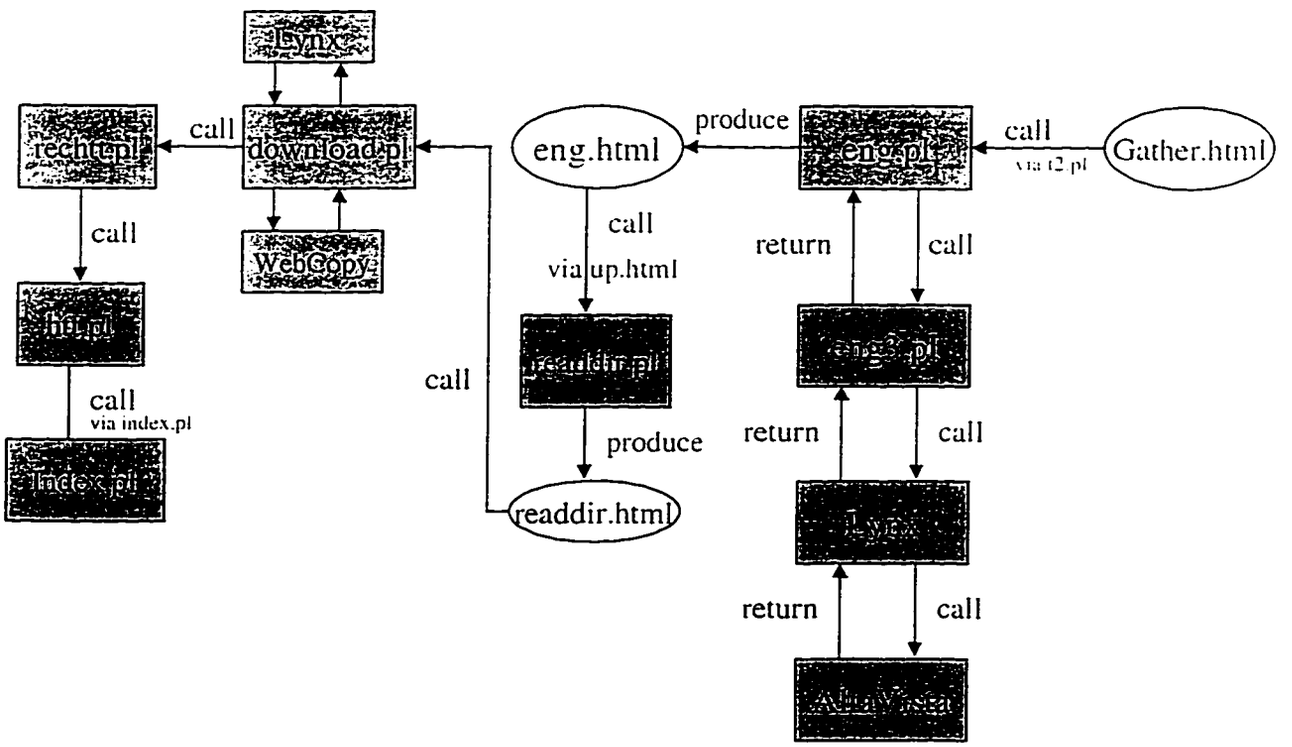


Figure 20: Architecture of the gathering subsystem

In the diagram above, a square corresponds to a program where the string inside the square corresponds to the physical file name of the program. An ellipse corresponds to an

HTML page . this page is either produced by a program or authored by us. Note that JavaScript routines may be embedded in the HTML pages.

Gathering.html	The main interface for doing the gathering tasks-- enter a query to Alta Vista or access an individual pages.
eng.pl	Engine.perl: This program queries Alta Vista and process its results
eng3.pl	This program is like a subroutine for eng.pl, it does the querying of Alta Vista and returns the raw results to eng.pl (via a pipe connection) which does the processing.
eng.html	The output of eng.pl, here the user can select the document they want to download.
readdir.pl	When the user click on "Execute Download", this program is called to recursively read and display the directories of the document base.
readdir.html	Output of "readdir.pl" , the directories name are displayed here in radio buttons so that user can select one or create a new one. This page contains JavaScript routines including a routine to create new directory.
download.pl	The main program for downloading documents . It decoded passed URLs from the server and download the documents. Also it reads url's from the passed text file "urlLists" then download them individually or recursively.
rechtt.pl	Recursive htt.pl, this program recursively traverses a hierarchy of directories and applies the "htt.p" on each traversed one.
htt.pl	HTML To Text , this program creates out of the HTML files in a directory the "intermediate format paragraphs" and stores them in the file

"text" which is later indexed by Glimpse. It also insert in the HTML files their own URL.

7.2 Querying

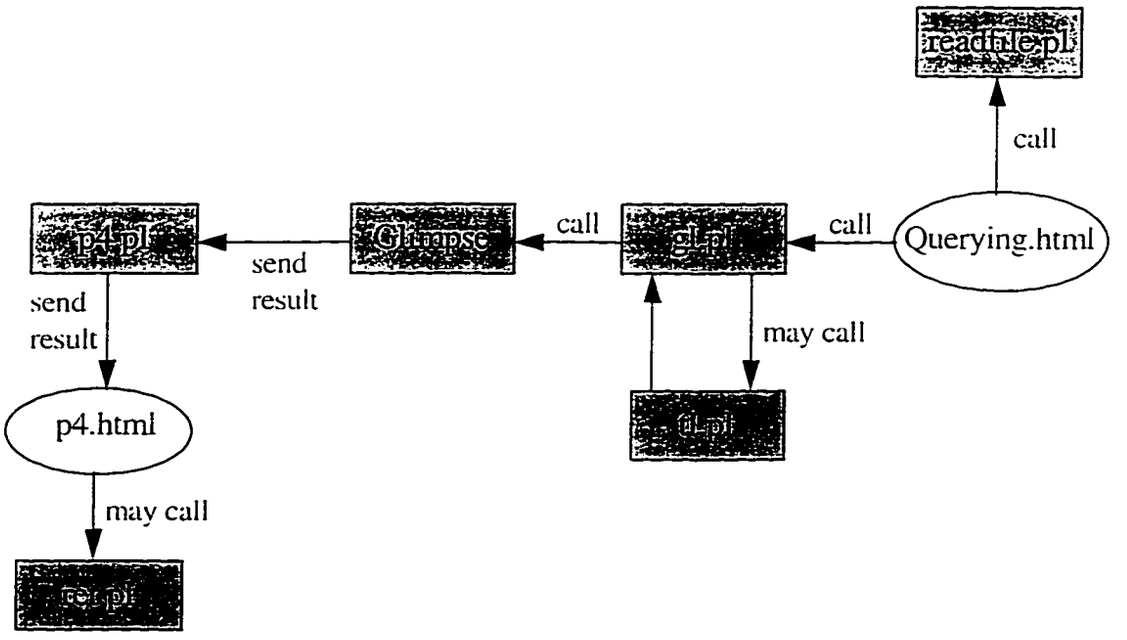


Figure 21: Architecture of the querying subsystem

Querying.html	The main interface for querying and browsing the document base.
readfile.pl	To display the file content of the document base, this program can recursively reads directories and display the file information of a

	directory content.
gl.pl	Glimpse.perl, this program is to call Glimpse after preparing the required parameters.
tl.pl	TargetList, a program to updates the list of terms used for ranking. It would be called only when the user indicates so.
p4.pl	The paragraph ranking program. It reads the result of a Glimpse search, isolates paragraph, matches the paragraph terms with terms of target list then sorts paragraph according to the number of matches. Finally outputs the sorted paragraphs with matches bolded.
p4.html	The output of p4.pl, here the Glimpse extracted paragraphs are presented.
ret.pl	Return the surrounding paragraphs of a paragraph. Invoked when the "Show surrounding" button is clicked in p4.html.

References

[Baeza-Yates & Frakes 92] Baeza-Yates, R. & Frakes, W. In preface of W. Frakes and R.Baeza-Yates, editors, Information Retrieval: Data Structure and Algorithms, Prentice Hall, Englewood, NJ, 1992.

[Baeza-Yates 92] Baeza-Yates, R. "Introduction to Data Structure and Algorithm related to Information Retrieval" . In W. Frakes and R.Baeza-Yates, editors, Information Retrieval: Data Structure and Algorithms. Prentice Hall, Englewood, NJ, 1992.

[Bell 94] Bell, T. "Managing Gigbytes" , pp. 79, New York, Van Neistrand Reinhold, 1994.

[Bowman 95] Bowman, M. et al . "Harvest: a scaleable, customizable discovery and access system", Technical report , 1995.

[Bruza 90] Bruza, P. " A Novel Aid For Searching in Hypermedia", Proceedings of European Conference on Hypertext, 1990.

[Ciaccia & Zezula 93] Ciaccia, P. & Zezula, P. "Estimating accesses in partitioned signature file organizations", ACM Transactions on Information Systems, April 1993.

[Croft 92] Croft, W. "The University of Massachusetts TIPSTER project", SIGIR Forum, 1992.

[Crouch 1989] Crouch, D. et al. "The Use of Cluster Hierarchies in Hypertext Information Retrieval", Proceedings of Hypertext '89, ACM Press, 1989.

[Daoud 93] Daoud, A. "Efficient data structures for information retrieval". Ph.D. dissertation, Virginia state University. 1993

[December 95] December, J. , "HTML & CGI unleashed". Sames Net Publishing, Indianapolis, 1995.

[Faloutsos 92] Faloutsos, C. "Signature Fikes". In W. Frakes and R.Baeza-Yates, editors, Information Retrieval: Data Structure and Algorithms. Prentice Hall, Englewood, NJ, 1992.

[Gonnet 92] Gonnet, G. et al., "New indices for text: PAT trees and PAT arrays" . In W. Frakes and R.Baeza-Yates, editors, Information Retrieval: Data Structure and Algorithms, Prentice Hall, Englewood, NJ, 1992.

[Green 94] Green, D. "Virtual Libraries ", Charles Sturt University, 1994. http://www.efi.joensuu.fi/~i_dgreen/vl.

[Halaz 88] Halasz, F. "Reflections on NoteCards : Seven Issues for the Next Generation of Hypermedia Systems", Communications of the ACM, July 1988.

[Harmon 92] Harmon, D, et al. "Inverted files". In W. Frakes and R.Baeza-Yates, editors, Information Retrieval: Data Structure and Algorithms, Prentice Hall, Englewood, NJ, 1992

[Hughes 93] Hughes, K. "Entering the World-Wide Web: A Guide to Cyberspace", 1993 . http://www.pharmazie.uni-halle.de/lern/guide/www.guide.html

[Manber 94] Manber, U. et al. , "GLIMPSE: A Tool to Search Through

Entire File Systems", Technical Report, 1994. ftp://ftp.cs.arizona.edu/glimpse/glimpse.ps.Z

[Rabbati & Sizka 84] Rabitti, F. & Zizka,J. "Evaluation of access methods to text documents in office systems", Proc. 3rd symposium on research and development in IR. Cambridge University Press. 1984.

[Salton & McGill 83] Salton, J. & McGill, M. "Introduction to Modern Information Retrieval", McGraw-Hill, New York, 1983.

[Skuce 96] Skuce, D. " IKARUS: Intelligent knowledge acquisition and retrieval universal system" , 1996. http://www.csi.uottawa.ca:80/~kavanagh/Ikarus/Ikarus4.html

[Stalling 95] Stalling, W."Operating systems" , Prentice-Hall, Englewood, NJ, 1995.

[Venditto 96] Venditto, G. "Search engine showdown", Internet World . May 1996.

[Wartik 92] Wartik, S. et al. , "Hashing Algorithms" . In W. Frakes and R.Baeza-Yates, editors, Information Retrieval: Data Structure and Algorithms, Prentice Hall, 1992.

[Zeng 96] Zeng, Y. " toward networked resource discovery: A challenging application for information retrieval", 1996. http://quasar.poly.edu/~llin/GISS/sessions/SecI-D.html