MARCH 1990 VOL. 13 NO. 1

a quarterly bulletin of the
IEEE Computer Society
technical committee on

Data Engineering

CONTENTS

SPECIAL ISSUE ON DOCUMENT RETRIEVAL

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC
IEEE

IEEE COMPUTER SOCIETY

Data Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Data Engineering . Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Data Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Data Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both full members and participating members of the TC are entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Letter from the Editor

Document retrieval deals with the capture, storage, and retrieval of natural language texts, which could range from short bibliographic records to full text documents. Document retrieval has been investigated for over three decades, but its application has thus far been limited to library systems. The proliferation of PCs, workstations, online databases, and hypertext systems has presented new challenges and opportunities to this research area. Researches in this area not only are of interest to large-scale systems such as library systems and news databases but have profound impacts on the way we manage our personal, day-to-day, data.

The special issue has assembled eight papers examining various aspects of this important topic.

The first paper, by Salton, describes the SMART system, which is perhaps one of the most thoroughly studied document retrieval system so far, and discusses the potential of knowledge bases in document retrieval. He then describes a simple term weight strategy for the analysis of local document structures.

Smith's paper discusses the expertise required for an effective search and describes a knowledge-based system, called EP-X, which can help the users to refine their queries.

Croft gives an overview of the research being conducted in his research group at the University of Massachusetts, covering a wide range of research from text representation, to retrieval model, to user modeling and interface. The main concern of the research is the effectiveness of the retrieval.

The next paper, by Faloutsos, addresses the other end of the search problem – how to efficiently search a large number of documents. The paper is focused on one particular text access technique, namely, the signature file. Variants of the signature file technique are presented and analyzed.

Along the same line, Stanfill describes a parallel retrieval system based on the signature file. The system runs on a Connection Machine and implements a simple document ranking and relevance feedback strategy. He provides justifications for the use of large-scale parallel systems for document retrieval.

Hollaar discusses his experience in the design and development of the partitioned finite state automaton (PFSA). He describes a prototype based on the PFSA concept and discusses the needs and potentials of special-purpose pattern matchers in light of the rapidly lowering costs of general-purpose processors.

McGill and Dillon describe several major projects being conducted in OCLC. The projects include research prototypes as well as field experiments. One of the concerns in their research is the conversion of paper documents to an electronic form and to provide real services to a large user community.

Last but not least, Lee and Woelk describe their work in integrating text management capability in the object-oriented database ORION developed at MCC. They describe the class hierarchy for organizing textual objects and the search capability of the system.

I would like to thank the authors for accepting my invitation to contribute to this special issue. Many of them have to make time from their busy schedules in order to meet our deadline. The suggestions from Dr. Won Kim, the Editor-in-Chief, were crucial in making my task as enjoyable as it was. I hope this special issue will bring this important subject to a wider audience and you will find the articles stimulating and interesting.

Dik L. Lee
Ohio State University

Full Text Information Processing Using the Smart System

Gerard Salton *

Abstract

The Smart information retrieval project was started in 1961. During the past 30 years methods have been developed for the automatic term assignment to natural-language texts (automatic indexing), automatic document clustering, collection searching, and the automatic reformulation of search queries using relevance feedback. Many of these procedures have been incorporated into practical retrieval settings.

Although there is no hope of solving the content analysis problem for natural-language texts completely satisfactorily, the possibility of automatically analyzing very large text samples offers new approaches for automatic text processing and information retrieval. Some methods for the massive analysis of natural language text are outlined together with applications in information retrieval.

1 The Vector Processing System

Conventional information retrieval systems are based on Boolean query formulations where keywords are used together with connecting Boolean operators. By constructing large so-called *inverted* indexes that contain for each allowable keyword the lists of addresses of all documents indexed by that keyword, it is possible to determine the set of documents corresponding to a given Boolean query formulation from the information stored directly in the index. This implies that rapid responses can be provided in a conventional retrieval setting using standard Boolean processing methods.

The conventional Boolean search system does, however suffer from a number of serious disadvantages: First, the Boolean logic remains inaccessible to many

*Department of Computer Science, Cornell University, Ithaca, NY 14853-7501. This study was supported in part by the National Science Foundation under grant IST 84-02735.

untrained users, so that query formulations and user-system interactions must be delegated by the end user to trained search intermediaries; second, the conventional Boolean logic does not accommodate weighted terms used as part of the query formulations; third, the output produced by a Boolean search is not ranked in any decreasing order of presumed usefulness; finally, the size of the output produced by Boolean searches is difficult to control by untrained personnel. Typically, a search could retrieve far more documents than the user can tolerate, or too few items might be retrieved to satisfy the user needs. In any case, the unranked retrieved materials are difficult to utilize in an interactive environment.

Various solutions have been proposed, including in particular the introduction of new retrieval models not based on the Boolean paradigm. The best known of the alternative retrieval models is the *vector processing* system [1,2]. In vector processing, both the querries and the documents are represented by sets, or vectors, of terms. Given two term vectors $Q = (q_1, q_2, ..., q_t)$ and $D_i = (d_{i1}, d_{i2}, ..., d_{it})$ representing respectively query Q and document D_i, it is easy to compute a vector similarity measure such as, for example, the cosine coefficient as follows:

$$\mathrm{Sim}(Q, D_i) = \frac{\sum_{k=1}^{t} q_k d_{ik}}{\sqrt{\sum_{k=1}^{t} (q_k)^2 \cdot \sum_{k=1}^{t} (d_{ik})^2}} \cdot \qquad (1)$$

In expression (1), q_k and d_{ik} represent the weight or importance of term k in query Q and document D_i, respectively, and a total of t different terms are potentially assigned to each text item. (In the vector system, a positive term weight is used for terms that are present, and a zero weight represents a term that is absent from a particular item.)

In vector processing, variable coefficients are used to represent the similarity between queries and documents, and the documents can be arranged for the user in decreasing order of the corresponding query-document similarities. The output ranking helps the user in dealing with the retrieved materials, because the more important items are seen by the user early in a search. Furthermore, an iterative search strategy, known as *relevance feedback*, is easily implemented where the query statements are automatically improved following an initial retrieval step, by incorporating into the query terms from previously retrieved relevant documents. Effectively this moves the query in the direction of previously retrieved relevant items, and additional relevant items may then be retrieved in the next search iteration. The vector processing model is useful also for generating clustered file organizations where documents represented by similar term vectors are placed in the same groups, or clusters.

Another possibility for refining the conventional Boolean retrieval system

consists in introducing extended, relaxed interpretations of the Boolean operations. In that case, processing systems intermediate between the ordinary Boolean system and the vector processing system are obtained that accommodate term weights for both queries and documents and furnish ranked retrieval output, as well as much improved retrieval effectiveness. [3]

2 Dictionaries and Knowledge Bases

In the vector processing system, both documents and queries are transformed into sets of keywords, sometimes composed of words or word stems occurring in the corresponding document or query texts. The assumption is that no relationship exists between the terms assigned to each particular text item. In fact, of course, it is difficult to maintain that sets of individual terms extracted from query and document texts properly represent text content. For this reason, various refinements have been proposed for content analysis, normally consisting in the introduction of complex text identifiers, such as term phrases, and the addition of relationship indicators between terms. One possibility consists in using the term descriptions contained in *machine-readable dictionaries* and thesauruses to help in term phrase formation. The thesaurus information may be used to disambiguate the meaning of terms and to generate groups of similar, or related, terms by identifying relationships using the contexts provided by the dictionary entries.

Several attempts have been made to extract useful information from machine-readable dictionaries, and the experience indicates that some term relationships are relatively easy to obtain: notably certain synonym relations that are often explicitly identified in the dictionary, and hierarchical, taxonomic relations between terms that are identifiable following analysis of the dictionary definitions. [4] On the other hand, many complications also arise:

- many terms carry several defining statements in the dictionary, and the definition actually applicable in a given case may not be easily found;
- the printed definition may be difficult to parse, in which case the meaning of the defining statement may remain obscure;
- the relationships between different defining statements may be hard to assess.

Overall the accuracy of interpretation of dictionary definitions determined by Fox and coworkers varied between 60 and 77 percent, and several acceptable analyses were frequently generated for a given dictionary definition. [4] These results show that dictionary information is not easily incorporated into automatic text analysis systems.

An alternative solution to the text-indexing and retrieval problem is provided by the use of so-called *knowledge bases* that accurately reflect the structure and

the relationships valid in a given area of discourse. [5] Given such a knowledge base, the content of the various information items can be related to the content of the corresponding knowledge base in order to generate valid content representations. Typical knowledge bases describe the entities and concepts of interest in a given area of discourse, as well as the attributes characterizing these entities, and the relationships – hierarchical or otherwise – that exist between entities. In addition, knowledge bases often include systems of rules used to control the operations performed with the stored knowledge.

When a knowledge base is available, representing a particular subject area, the following extended retrieval strategies can be used:

a) The available search requests and document texts are transformed into formal representations similar to those used in the knowledge base.

b) Fuzzy matching operations are performed to compare the formal representations of search requests and document surrogates.

c) Answers to the requests are constructed by using information provided in the knowledge base if the degree of match between the formal representations of queries and documents is sufficiently great.

Unfortunately, very little is known about the design of knowledge-bases that are valid in open-ended areas of discourse of the kind found in most document collections. In fact, the indications are that the know-how needed to analyze even somewhat specialized documents is vast, and that a good deal of context is needed to interpret document content. This context cannot be expected to be specified in restricted knowledge bases. The knowledge-base approach remains to prove itself in information retrieval environments.

3 Massive Text Analysis

Modern theories of text analysis indicate that ultimately the meaning of words in natural language texts depend on the contexts and circumstances in which the words are used, rather than on preconceived dictionary definitions. [6,7] This suggests that the very large text samples that are now available in machine-readable form should be analyzed to determine the importance of the words in the contexts in which they occur. One way in which this might be done is to take large text samples, such as for example sets of books, which are then broken down into individual local documents (book paragraphs). The importance of individual text units (terms and phrases) occuring in the texts might be computable by comparing the local occurrence characterictics in individual book paragraphs with the global characteristics in the complete text collection.

In the Smart system, the following characteristics of term value have been used. [8,9]

a) The number of occurrences of a term in a given local environment (a local book paragraph); formally tf_{ik} is the term frequency *(tf)* of term k in local document i.

b) The number of local documents (paragraphs) n_k in which term k occurs; if there are N local documents in the complete collection, the so-called inverse document frequency *(idf)* factor is computed as $logN/n_k$; this factor provides high values for terms assigned to only a few local documents, and low values for terms occurring everywhere in the collection.

c) The length of the local documents as a function of the number and weights of terms assigned to local documents.

A particular coefficient of term value for term k in document i may then be computed as

$$w_{ik} = \frac{tf_{ik} \cdot \log\ (N / n_k)}{\sqrt{\sum_{\substack{\text{all terms} \\ k}} (tf_{ik} \cdot N/n_k)^2}} \cdot \qquad (2)$$

The w_{ik} weight is known as the $tf \times idf$ (term frequency times inverse document frequency) weight. This coefficient provides high values for terms that occur frequently in individual local documents, but rarely on the outside.

Because the *idf* factors change as the context changes, the same term may receive quite different term values depending on the context taken into account in computing term values. Consider, as an example, the local document of Fig. 1, representing two paragraphs of chapter 5 of reference [10]. A standard indexing system can be applied to the text of Fig. 1(a), consisting of the deletion from the text of certain common function words included on a special list, the removal of suffixes from the remaining text words, and finally the assignment of term weights using the $(tf \times idf)$ term weighting formula of expression (2). [1] When the terms are arranged in decreasing term weighting order, the output of Fig. 1(b) and 1(c) is obtained, where the ten best terms are listed in two different text contexts. In each case, a computer-assigned concept number is shown in Fig. 1 for each term together with the $(tf \times idf)$ weight and the corresponding word stem.

In Fig. 1(b) the terms are weighted using the local context of chapter 5 of [10] only, whereas the global book context is used in Fig. 1(c). This implies that in Fig. 1(b) the term occurrence measurements cover only the 67 local documents of chapter 5, whereas all 1104 local documents for the complete book are used in Fig. 1(c). It is clear that the indexing assignments of Figs. 1(b) and 1(c) are very different. For example, the term "compress" is absent

from the output of Fig. 1(b) because in chapter 5 all local documents deal with text compression; this means that in the context of chapter 5, "compress" is a high-frequency word with a low inverse document frequency *(idf)* factor, and hence a low overall weight. In other word, a term like "compression" is not a good term capable of distinguishing the text of Fig. 1(a) from the other local documents of chapter 5. "Compression" is, however, a very good term in the global book context — the second best, in fact, on the list of Fig. 1(c) — because the overall collection frequency of "compression" is low.

Using the term weighting assignment of expression (2), each local document can then be represented as a term vector $D_i = (w_{i1}, w_{i2}, ..., w_{it})$, and the cosine similarity function of expression (1) can be used to obtain global similarity measures between pairs of local documents.

A length normalization component is included in the term weighting formula of expression (2) to insure that all documents are considered equally important for retrieval purposes. Without the normalization factor, longer documents with more terms would produce higher similarity measures than shorter documents, leading to a greater retrieval likelihood for the longer documents.

The term weighting and contextual document indexing methods described earlier can also be applied to short local documents, such as individual document sentences, leading to the computation of sentence similarity measures. When the formulas of expression (1) and (2) are used for sentence indexing, many short sentences consisting of only 2 or 3 words, including especially section headings and figure tables, will produce very large similarities. In these circumstances, it is better to use a term weighting system based only on the individual term frequencies in the local context (that is, $w_{ik} = tf_{ik}$). When the sentences are represented by term frequencies, that is, $S_i = (tf_{i1}, tf_{i2}, ..., tf_{it})$ a useful sentence similarity measure may be obtained as:

$$\text{Sim}\ (S_i\ ,\ S_j\ ) = \sum_{\substack{\text{matching} \\ \text{terms } k}} \min\ (tf_{ik}\ ,\ tf_{jk}\ ) \qquad (3)$$

One may expect that documents that include sentences with large pairwise sentence similarities in addition to exhibiting large global document similarities may cover similar subject matter with a high degree of certainty.

The text analysis and document comparison methods described in this note, are usable to obtain representations of the local and global structure of document content. The procedures may also help in obtaining answers to search requests in the form of linked structures of local documents. [11]

REFERENCES

1. G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, McGraw Hill, New York, 1983.

2. C. J. van Rijsbergen, *Information Retrieval*, 2nd ed., Butterworths, London, 1979.

3. G. Salton, E. A. Fox, and H. Wu, "Extended Boolean Information Retrieval," *Communications ACM*, 26:11, 1022-1036, November 1983.

4. E. A. Fox, J. T. Nutter, T. Ahlswede, M. Evens and J. Markowitz, "Building a Large Thesaurus for Information Retrieval", *Proc. Second Conference on Applied Natural Language Processing*, Association for Computationsl Linguistics, Austin, TX, Feb. 1988, 101-108.

5. N. J. Belkin et al., "Distributed Expert-Based Information Systems: An Interdisciplinary Approach", *Information Processing and Management*, 23:5, 395-409, 1987.

6. L. Wittgenstein, *Philosophical Investigations*, Basil Blackwell and Mott, Ltd., Oxford, England 1953.

7. S. C. Levinson, *Pragmatics*, Cambridge University Press, Cambridge, England 1983.

8. K. Sparck Jones, "A Statistical Interpretation of Term Specificity and its Application in Retrieval, *Journal of Documentation*, 28:1, March 1972, 11-21.

9. G. Salton and C.S. Yang, "On the Specification of Term Values in Automatic Indexing" *Journal of Documentation*, 29:4, December 1973, 351-372.

10. G. Salton, *Automatic Text Processing - The Transformation, Analysis and Retrieval of Information by Computer*, Addison Wesley Publishing Co., Reading MA, 1989.

11. G. Salton and C. Buckley, "Approaches to Text Retrieval for Structural Documents" Technical Report TR. 90-1083, Computer Science Department, Cornell University, Ithaca, NY January 1990.

.I 254

Chapter 5 Text Compression

The usefulness and efficiency of text-processing systems can often be improved greatly by converting normal natural-language text representations into a new form better adapted to computer manipulation. For example, storage space and processing time are saved in many applications by using short document abstracts, or summaries, instead of full document texts. Alternatively, the texts can be stored and processed in encrypted form, rather than the usual format, to preserve the secrecy of the content.

.I 255

One obvious factor usable in text transformations is the redundancy built into normal natural-language representation. By eliminating redundancies – a method known as text compression – it is often possible to reduce text sizes considerably without any loss of text content. Compression was especially attractive in earlier years, when computers of restricted size and capability were used to manipulate text. Today large disk arrays are usually available, but using short texts and small dictionary sizes saves processing time in addition to storage space and still remains attractive.

a) Local Document Consisting of Two Paragraphs from Chapter 5 of [10]

3521	0.26873	text	437	0.36273	text
3936	0.23112	save	7652	0.24997	compress
4318	0.22514	stor	8796	0.24397	attract
2655	0.21177	attract	879	0.22654	save
1957	0.21177	docu	3930	0.22654	redund
2546	0.19675	manipul	3259	0.22539	size
1313	0.19117	size	7612	0.17827	short
4300	0.18448	natur	4611	0.16270	natur
47	0.17410	redund	6264	0.16135	stor
3586	0.17157	process	4855	0.15250	spac

b) Ten Best Terms in Local Context of Chapter 5 (67 docs.)

c) Ten Best Terms in Global Book Context (1104 docs.)

Figure 1: Local Document Indexing

Document Retrieval:Expertise in Identifying Relevant Documents

Philip J. Smith
Cognitive Systems Engineering Laboratory
The Ohio State University
210 Baker Systems, 1971 Neil Avenue
Columbus, OH 43210

Introduction

Advances in computer hardware, software and communications capabilities offer the potential to revolutionize access to the information in published documents. It is realistic to start talking about providing widespread computer access to the full text of documents. Equally important, it is realistic to expect workstations that provide tools for exploring, annotating and storing this full text. Thus, in theory these advances will provide a person with far greater access to the documents that are relevant to her needs.

There are two potential pitfalls to this notion. The first is that the information seeker must first identify the documents relevant to her interests before she can retrieve them. As research on document retrieval has long made clear, this is not a trivial problem (Salton and McGill, 1985). The second potential pitfall is cost. Efforts to improve access to information (either in terms of the quantity of information available or in terms of the ease or effectiveness of finding relevant information) are not free. Someone must pay for the improved access.

This paper focuses on the first potential pitfall, the difficulty of finding documents relevant to some topic of interest. These difficulties will be highlighted by looking at studies of online search intermediaries, and at efforts to capture the expertise of these intermediaries in the form of a knowledge-based system. In terms of the second pitfall, cost, two points will be implicit in this discussion:

1. It is not sufficient to simply provide access to increased quantities of information (e.g., the full text of documents). Information seekers need help in finding the relevant documents as well;
2. Computer systems that aid people in finding relevant documents will not be cheap to develop. Thus, for different application areas, we will need to carefully consider the cost effectiveness of investing money in alternative methods for aiding people to find relevant documents, as well as the costs and benefits of providing access to increased quantities of information.

Background

For several decades, researchers and practitioners in information retrieval have sought to develop methods to give people easy access to the world's literature through the use of computers. A number of the resultant methods have been developed commercially and have received widespread usage. Included are the development of online library catalogs to retrieve bibliographic information

about published books and journals. Also included are bibliographic databases describing the contents of individual journal articles.

Three primary methods have been used to identify documents in these bibliographic databases:

1. specification of a particular document in terms of its author or title;
2. use of character string matching techniques to find potentially relevant documents based on words found in titles, abstracts or descriptive keyword lists;
3. retrieval based on citation links (retrieving a new document that is contained in the reference list of an already retrieved document).

The use of such bibliographic databases often requires considerable expertise, particularly when conducting subject searches. Part of this expertise involves clearly defining the subject or topic of interest. Part of it involves translating this topic of interest into a query the computer can understand. Finally, part of it concerns interacting with the computer itself, entering appropriate commands. As a result, information seekers often need the assistance of a human intermediary to make effective use of these databases (Marcus, 1983; Pollitt, 1987).

Considerable improvements can now be made in the design of the interfaces to such computer systems. The use of multiple-window displays and communication by direct manipulation methods can help considerably to reduce the expertise needed to enter commands. Figure 1 provides an illustration of such a system. This is the screen displayed by a prototype system called ELSA (Smith, 1989) that we have developed when the searcher wants to enter the author, title, etc. for a specific book, journal or journal article.

These improvements in interface design, however, only solve the easy problems. The truly difficult problems involve expertise in clearly defining a topic and expressing it in a form the computer can understand. These problems are discussed further below.

Current Retrieval Methods

Using most current retrieval systems, subject searches are conducted by specifying combinations of keyword strings. To search for a document on bioindicators for heavy metals, for instance, a searcher might enter:

(Bioindic? or Accumul? or Bioaccumul?)
and (Heavy (w) Metal# or Mercur? or Lead or Cadmium).

Several types of expertise are involved in generating such queries. First, there is the "art" of using logical operators. One expert we studied gave us an illustration of a rule she used in selecting logical operators:

> "[I] narrow according to what the patron wants. . . . I start with AND because that is the least restrictive of proximities . . . . Least restrictive is AND, then LINK, then the A operator, then the W operator is most restrictive."

Additional expertise is involved identifying appropriate terms to include in the query. This involves generating synonyms or near-synonyms (e.g., radioactive and radioisotope) and terms for

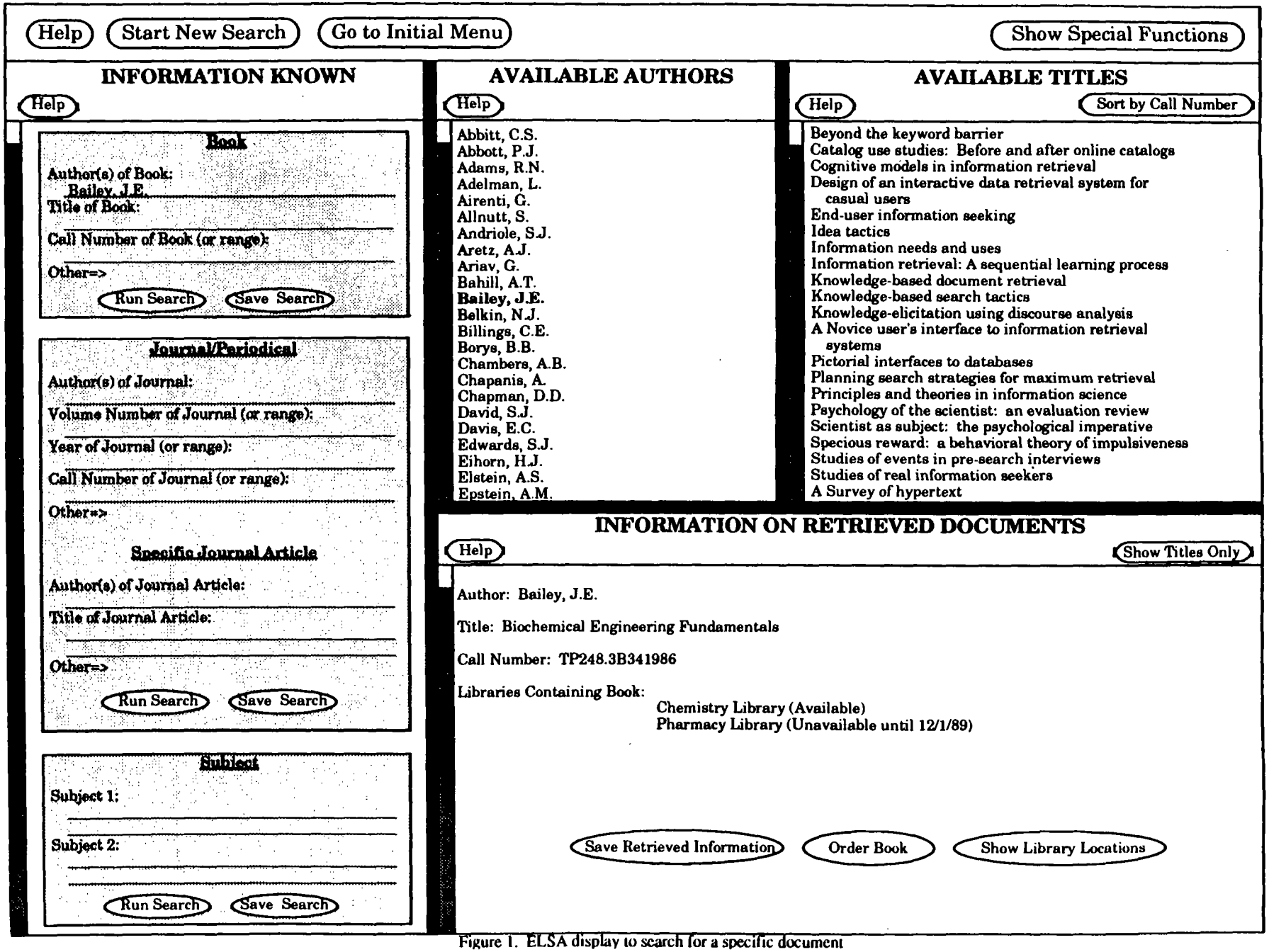


Figure 1. ELSA display to search for a specific document

specific cases of a general concept (e.g., lakes, rivers and streams as specific types of natural bodies of water), as well as removing ambiguities.

The above forms of expertise assume the searcher knows the specific topic she is interested in, that she simply needs to express this topic in a form the computer can deal with. Often this is not the case, however. Often, the searcher needs to learn more about the topic as she conducts the search, so that she can more clearly define and refine her topic.

In several studies of human search intermediaries (Smith and Chignell, 1984; Smith, Krawczak, Shute and Chignell, 1985; Smith, Krawczak, Shute and Chignell, 1987; Smith, Shute, Chignell and Krawczak, 1989; Smith, Shute, Galdes and Chignell, 1989), we have found that these intermediaries play a very active role in this learning process. These intermediaries actively suggested topic refinements to information seekers, such as changing:

"control of acid rain in the United States"

to:

"prevention of nitrogen and sulfur oxides as air pollutants in the United States."

Indeed, generation of such topic refinement suggestions appeared to be one of the primary functions of such intermediaries. In a study of one intermediary, for instance, she generated a total of 361 such suggestions over the course of 17 searches (for 17 different information seekers).

Types of Computer Aids

A variety of solutions have been proposed to replace the expertise of these human intermediaries. Some solutions propose alternatives to the use of logical operators based on statistical word associations or term weightings (Dumais, Furnas, Landauer, Deerwester and Harshman, 1988; Giuliano, 1963; Salton, 1968; Tong and Shapiro, 1985). Others center on the use of thesauri to assist in identifying appropriate terms (Pollitt, 1987; Rada and Martin, 1987). Finally, some solutions focus on the development of semantically-based search techniques, with the representation of document contents and search requests in terms of meaning (Monarch and Carbonell, 1987; Vickery, 1987).

The capabilities of such computer aids vary tremendously. Our own work, focusing on semantically-based search techniques, serves to demonstrate some of the functions that such systems could serve. Illustrations are given below.

Semantically-Based Search

We have been developing a knowledge-based system to assist searchers. This system, EP-X (Environmental Pollution eXpert), helps information seekers by:

1. Translating a searcher's list of keyword phrases into a natural language topic statement;
2. Identifying ambiguities in the intending meaning of the searcher's entry;
3. Automatically including specific cases (e.g., DDT or malathion) in the search when the searcher enters a general concept (e.g. pesticides);
4. Actively helping the searcher to explore a topic area in order to learn about it and to refine her topic.

To accomplish these functions, EP-X uses frames and hierarchically-defined semantic primitives to represent domain knowledge and document contents (Smith, Shute, Chignell and Krawczak, 1989; Smith, Shute, Galdes and Chignell, 1989). Figure 2 gives an example of an interaction with EP-X.

More specifically, EP-X uses a repertoire of knowledge-based search tactics to generate suggestions for alternative topics. These suggestions can help the searcher to broaden, narrow or re-define her topic.

EP-X, their, serves to emphasize some of the areas where information seekers need help. EP-X also illustrates one approach to meeting these needs.

Conclusion

Our research, then, suggests that it is not enough to provide easy access to greater quantities of information. Effective direct end-user searching will not suddenly result from simply providing access to the full-text of documents, or from the use of multiple-window displays and improved command languages. While these are valuable improvements, they only address some of the needs of information seekers.

In particular, the greatest needs are likely to continue to concern the difficulties people have in defining and expressing their topics of interest. These difficulties arise in part because many searchers do not really have a clear idea of what they are looking for. They need to learn more about the subject as they are searching, so that they can formulate a topic. Difficulties also arise because of the subtleties of expressing a topic in a form that the computer can understand.

Thus, as we begin to invest in the next generation of document retrieval systems, we must be sure we understand the real needs of information seekers. Based on this understanding, we can then begin to assess the most cost-effective solutions. These solutions may furthermore vary from one application area to another. What is clear, however, is that it is not enough to simply provide access to greater quantities of information.

Acknowledgements

This research has been supported by Chemical Abstracts Service, the Applied Information Technologies Research Center and the U. S. Department of Education.

KEYWORD LIST

Your keyword list currently consists of the following:

BIOINDICATION
PESTICIDES
MOLLUSKS

INTERPRETATION

18 documents are available on the use of mollusks as bioindicators for pesticides.

SUGGESTIONS FOR BROADENING

104 documents are available on the use of clams, fish, fungi, insects or mosses as bioindicators for pesticides. Thus, you will add 86 documents to your set if you broaden mollusks to include these other bioindicators for pesticides.

Do you want to:

1 BROADEN your topic as suggested above.

Figure 2. Knowledge-based use of the tactic PARALLEL (from Smith, Shute, Galdes and Chignell, 1989).

References

Dumais, S.T., Furnas, G.W., Landauer, T.K., Deerwester, S., and Harshman, R. Using latent semantic analysis to improve access to textual information. In CHI '88 Conference Proceedings (May 15-19, 1988, Washington, D.C.). ACM, New York, 1988, pp. 281-285.

Giuliano, V.E. Analog networks for word association. IEEE Trans. Military Electronics, 7 (1963), 221-234.

Marcus, R. An experimental comparison of the effectiveness of computers and humans as search intermediaries. J. Am. Soc. Inf. Sci., 34, (1983), 381-404.

Pollitt, S. CANSEARCH: An expert systems approach to document retrieval. Inf. Process. Manage, 23 (1987), 119-138.

Rada, R., and Martin, B.K. Augmenting thesauri for information systems. ACM Trans. Off. Inf. Syst., 5 (1987), 378-392.

Salton, G. Automatic Information Organization and Retrieval. McGraw-Hill, New York, 1988.

Salton, G., and McGill, M.P. Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.

Smith, P. J. (1989). Interface Design Issues in the Development of ELSA, an Electronic Library Search Assistant. Technical Report CSEL-123, Cognitive Systems Engineering Laboratory, The Ohio State University.

Smith, P.J., and Chignell, M. Development of an expert system to aid in searches of the Chemical Abstracts. In Proceedings of the 47th ASIS Annual Meeting (Philadelphia, Pa., Oct. 21-25). Knowledge Industry Publications, White Plains, 1984, pp. 99-102.

Smith, P.J., Krawczak, D., Shute, S.J., and Chignell, M.H. Cognitive engineering issues in the design of a knowledge-based information retrieval system. In Proceedings of the Human Factors Society-29th Annual Meeting (Baltimore, Md., Sept. 29-Oct. 3). Human Factors Society, 1985, pp. 362-366.

Smith, P.J., Krawczak, D., Shute, S.J., and Chignell, M.H. Bibliographic information retrieval systems: Increasing cognitive compatibility. Inf. Serv. Use, 7 (1987), 95-102.

Smith, P.J., Shute, S.J., Chignell, M.H., and Krawczak, D. Bibliographic information retrieval: Developing semantically-based search systems. In Advances in Man-Machine Systems Research, 5, W. Rouse, Ed. JAI Press, Greenwhich, Conn., 1989, pp. 93-117.

Smith, P.J., Shute, S.J., Galdes, D., and Chigell, M.H. Knowledge-Based search tactics for an intelligent intermediary systems. ACM Transactions on Information Systems, 7 (1989), 246-270.

Tong, R.M., and Shapiro, D.G. Experimental investigations of uncertainty in a rule-based system for information retrieval. Int. J. Man-Mach. Stud., 22 (1985), 265-282.

Vickery, A., and Brooks, H.M. PLEXUS-the expert system for referral. Inf. Process. Manage., 23, (1987), 99-117.

Towards Intelligent Information Retrieval: An Overview of IR Research at U.Mass.

W.B. Croft
Computer and Information Science Department
University of Massachusetts, Amherst, MA 01003

1 Introduction

Information Retrieval (IR) has been studied for some time and quite a lot is known about how to build systems that provide effective, efficient access to large amounts of text. There are still, however, many unanswered questions about the fundamental nature of the text retrieval process and the limits of performance of IR systems. Statistical IR systems are efficient, domain-independent, and achieve reasonable levels of retrieval effectiveness, as measured by the usual criteria of recall and precision (Van Rijsbergen, 1979; Salton and McGill, 1983; Belkin and Croft, 1987). The major question that is being addressed by many researchers currently is whether significantly better effectiveness can be obtained though the use of "intelligent" IR techniques. This covers a wide variety of techniques which can perhaps be best described by a list of the issues that must be addressed in building an intelligent IR system:

Text Representation: The primary issue in an IR system is the choice of a representation of text. This representation is used by the system to determine which text passages satisfy an information need. Despite many years of experimental studies, little is understood about the limitations of different types of representations and the characteristics of text that determine relevance. It is known that simple representations can perform surprisingly well, but we do not know whether more complex representations, such as those produced by natural language processing techniques, could improve performance significantly or even if it is possible to achieve significant improvements.

Retrieval Models: A retrieval model is a formal description of the retrieval process in a text-based system. The search strategies that are used to rank text documents or passages in response to a particular query are based on a retrieval model. Much research has been done on statistical retrieval models and a lot is known about effective ranking strategies and techniques such as relevance feedback. All of these models, however, are based on a limited view of the retrieval process and, in particular, the types of text representation available. More complex text representations that make use of domain knowledge will need retrieval models that emphasize inference and evidential reasoning.

User Modeling and Interfaces: In order to perform well, a text-based system must be able to acquire an accurate representation of a user's information need. There is

evidence that there are generic classes of information needs and goals, but it has not been demonstrated that this knowledge can be combined with representations of individual needs to improve performance. We also do not know much about the user's mental model of text retrieval and what we do know has rarely been used to design and evaluate interfaces for acquiring the information need and displaying results.

Evaluation: In terms of designing a system, this may appear to be a secondary issue. In order for our knowledge of information retrieval to progress, however, evaluation is perhaps the most critical issue. It is essential that evaluation methodology and the test collections that are used for experimental studies keep pace with the techniques that they are being used to evaluate. The limitations of recall and precision are well-known and have been described many times. It is not clear, however, that there are better measures. Factors that need to be taken into consideration are the highly interactive nature of the proposed text-based systems, the lack of exhaustive relevance judgments, the complexity of the systems, and the impact of the interface on performance.

In the rest of the paper, we describe research on these topics that is underway in the Information Retrieval Laboratory at the University of Massachusetts.

2 Representation of Text

Much of our research addresses the question of whether complex representations of text can achieve better levels of retrieval effectiveness than simple representations. This is a fundamental question and one that is crucial to the development of intelligent retrieval systems. To make this issue more specific, we have to define what we mean by a complex and a simple representation. This is not easy to do; it is, however, easy to define the baselines for simple representations. The simplest representation of text is the text itself; this is the basis of full text systems. Although this representation requires no effort to produce, it is hard to design systems that can produce effective results from it. In systems that use statistical techniques, the basic representation is produced by removing common words and counting occurrences of word stems. This representation is combined with a word or index term weighting scheme based on the within-document term frequency (tf) and the collection frequency (idf). We shall refer to this representation and weighting scheme as simple statistical. It has been difficult to show that any other representation, regardless of its complexity, is more effective than simple statistical (Sparck Jones, 1974; Salton, 1986). The major categories of complex representations being studied are:

- **Enhanced Statistical vs. Simple Statistical:** A variety of statistical techniques are known for enhancing the simple statistical representations. The most important techniques appear to be statistical phrases (Fagan, 1987) and statistical thesaurus classes (Sparck Jones, 1974; Van Rijsbergen, 1979; Salton, 1986). We have developed probabilistic models that make use of enhanced representations and this work is continuing (Croft, 1983,1986).

- **Natural Language Processing (NLP) vs. No NLP:** A number of attempts have been made to incorporate some level of NLP into the process of producing a text representation (e.g. Dillon, 1983; Fagan, 1987; Sparck Jones and Tait, 1984; Lewis, Croft and Bhandaru, 1989). In general, these experiments have had very mixed results. In the following section, we describe our current approaches to using NLP.

- **Domain Knowledge vs. No Domain Knowledge:** Domain knowledge in a text retrieval system can take the form of a thesaurus or, in the case of a knowledge-based system, some more sophisticated representation of the domain of discourse of the documents. Domain knowledge is essential in a system that uses NLP to do semantic analysis of the text (e.g. Sparck Jones and Tait, 1984; Lewis, Croft and Bhandaru, 1989), but it can also be an important part of systems that do not use NLP. Even the controlled vocabularies used in manual indexing can be regarded as a form of domain knowledge. Domain knowledge bases are known to be expensive to produce, but there is very little evidence concerning the levels of performance improvement that can be expected if they are available. We are currently beginning experiments with knowledge-based indexing.

- **Multiple Representations vs. Single Representations:** There is growing evidence that significant effectiveness improvements can be obtained by combining the evidence for relevance provided by multiple representations (Croft et al, 1990). All of the representations mentioned above could potentially be combined into a single, complex representation together with additional information such as manual indexing, citations, and even hypertext links (Croft and Turtle, 1989). This work is described further in the section on retrieval models.

In the following subsections, we describe this research in more detail.

2.1 Syntax-Based Representations

Past research such as that described by Fagan (1987) have reported inconclusive results with phrases derived from using a syntactic parser on the documents and queries, despite their desirable properties of low ambiguity and high specificity. We take the view that much of the problem with syntactic phrases is their low frequency (most syntactic phrases occur in few or no documents in any particular collection) and high redundancy (there are many phrases with the same or very similar meaning). These essentially statistical problems suggest the use of dimensionality reduction, in particular, term clustering, to improve the representation.

Like syntactic phrases, term clustering has not been shown to provide reliable improvements to retrieval performance (Sparck Jones, 1974). We are addressing this problem by clustering relatively unambiguous phrases, rather than ambiguous words. We have demonstrated that substantial clusters of phrases with the same meaning exist in test collections and this suggests that very substantial improvements of a phrasal representation are possible.

The low frequency and large number of terms in a syntactic phrase representation makes the traditional similarity measure used in term clustering (co-occurrence in documents) inappropriate. We are using two kinds of novel similarity information. The first is co-occurrence in sets of documents assigned to the same manual indexing category. The second is linguistic knowledge, such as knowledge of shared words in phrases, of morphologically related words in phrases, and of syntactic structures that tend to express related meanings. We are experimenting with two cluster formation strategies using this information: nearest neighbor clustering (Willett, 1988) and a variant that incorporates linguistic similarity.

2.2 Representations Based on Machine-Readable Dictionaries

Word sense ambiguity has been viewed as a significant problem in information retrieval systems for some time. However, previous approaches have been handicapped by small lexicons which often do not take adequate account of the senses a word can have. Recent advances in theoretical and computational linguistics have led to a great deal of new research on the role of the lexicon. At the same time, increased use of computerized typesetting tapes have made machine-readable dictionaries much more available. These dictionaries have been used for such purposes as: spelling correction, thesaurus construction, machine translation, speech recognition, and lexical analysis. Relatively little work has been done, however, with what most people would consider the principle use of dictionaries, namely as a source of information about word senses. We propose that word senses should be used to index documents, and that these senses be taken from a machine-readable dictionary.

Given our desire to index by word senses, how should we do so, and what dictionary should we use? Dictionaries vary widely in the information they contain and the number of senses they enumerate. At one extreme we have pocket dictionaries with about 30,000-40,000 senses, and at the other the Oxford English Dictionary with over half a million senses, and in which a single definition can go on for several pages. There are seven major dictionaries that are now available in machine-readable form: Webster's Seventh New Collegiate Dictionary (W7), the Merriam-Webster Pocket Dictionary (MWPD), the Oxford English Dictionary (OED), the Collins Dictionary of English (COLL), the Oxford Advanced Learners Dictionary (OALD), the Collins Birmingham University International Language Database (COBUILD), and the Longman Dictionary of Contemporary English (LDOCE).

The dictionary we are using in our research, the Longman Dictionary of Contemporary English (LDOCE), is a 'learner's dictionary' (i.e., a dictionary which is intended for people whose native language is not English) and has a number of useful features such as a restricted vocabulary extensive information about word subcategorization, and many example sentences.

Our approach to word sense disambiguation is based on treating the information associated with the senses as multiple sources of evidence (Krovetz and Croft, 1989). We are essentially trying to infer which sense of a given word is more likely to be correct based on the information associated with that sense in LDOCE. Each type of information associated with that sense will be considered as a potential source of evidence. The more consensus

we have about a given sense, the more likely it is to be correct.

A simple approach to disambiguation has been taken by Lesk in his work with the Oxford Advanced Learners Dictionary (OALD). In this project, words are disambiguated by counting the overlap between words used in the definitions of the senses. Lesk gives a success rate of fifty to seventy percent in disambiguating the words over a small collection of text (Lesk, 1986). More experimentation is needed to see how well this approach would work on a larger scale. A similar approach to that used by Lesk has been used by Wilks and his students in disambiguating the text of the definitions in LDOCE (Wilks et al, 1989). These experiments provide encouraging evidence that accurate disambiguation may be possible.

2.3 Evaluation of Knowledge-Based Indexing

Another representation with considerable promise for improving IR performance is manual indexing of texts with concept descriptions based on a large knowledge base. Such techniques assume the use of inference in comparing queries to documents (Van Rijsbergen, 1987; Croft and Turtle, 1989). However, there is very little experimental data on the potential effectiveness improvements that are possible. The GRANT system and associated test collection, developed by Cohen (Cohen and Kjeldsen, 1987), provides a useful testbed for exploring knowledge-based indexing. Past work on manual indexing in IR, along with what is known about knowledge-based representations in machine learning, has led us to design experiments to test the following hypotheses:

1. Initial attempts (such as GRANT) to create knowledge-based text representations, especially by personnel who are not professional indexers, will lead to performance that is no better than that of conventional manual indexing and free-text indexing.

2. The sophistication of inferences in knowledge-based text retrieval systems is, given current AI technology, quite limited. The benefits of query-time inference can be duplicated by the off-line use of inference to augment document representations, allowing efficient matching functions to be used at query time. Furthermore, the techniques of probabilistic retrieval can be used to improve the performance of these augmented representations.

3 Retrieval Models

In some recent papers (Croft et al, 1990, Croft and Turtle, 1989) a retrieval model based on combining multiple sources of evidence has been presented, along with retrieval results that indicate the potential for significant performance improvements. The model is based on earlier experimental work which showed that different representations and search strategies tend to retrieve different relevant documents, and on the I^3R system (Croft and Thompson, 1987; Croft et al, 1990).

The basis of our retrieval model is viewing retrieval as a process of inference. For example, in a database system that uses relational calculus queries, tuples are retrieved that can be shown to satisfy the query. The inference process is even more clear in an "expert"

database system built using PROLOG, where the objects that are retrieved are those for which the proof of the query succeeds. The expert database approach is more general than relational calculus because the proof of the query may involve domain knowledge in the form of rules.

The queries in a text-based retrieval system can also be viewed as assertions about documents in the database. It is possible, then, to think of constructing an expert database for document retrieval that consists of assertions about the presence of concepts in particular documents, relationships between concepts, and rules that allow us to infer the presence of concepts and relationships. This is not very different from using a sophisticated form of indexing together with a thesaurus of domain concepts. Deductive inference could then be used to retrieve documents that satisfy a query.

Experimental evidence tells us, however, that this is not an effective way to build a document retrieval system. There are a number of reasons for this. One of these is that the relevant documents, or in other words, the documents in which the user is interested, will be those documents that satisfy the query. That is, we are assuming that satisfying the query implies relevance. In general this implication does not strictly hold. The main reason for this is that the query is not accurately specified by the user. Techniques such as relevance feedback (Salton and McGill, 1983) and query formulation assistance (Croft and Thompson, 1987) are designed to alleviate this problem. Since the system cannot access the definition of the information need directly, it must deal with the best possible description of the query.

Another source of problems is the inaccuracies and omissions in the descriptions of the documents, the domain knowledge, and the inference rules. Documents that do not satisfy the query may still be relevant. Strict adherence to deductive inference will result in poor performance. Instead, retrieval must be viewed as a process of plausible inference where, for a particular document, the satisfaction of each proposition in the query contributes to the overall plausibility of the inference. Another way of expressing this is that there are multiple sources of evidence as to whether a document satisfies a query. The task of the retrieval system is to quantify the evidence, combine the different sources of evidence, and retrieve documents in order of this overall measure.

There are many ways to approach the formalization of plausible inference (Pearl, 1989, provides a good overview). In the area of information retrieval, Van Rijsbergen has developed a form of uncertain logic, and the probabilistic models of retrieval use a form of plausible inference (Van Rijsbergen, 1987). There was also significant early work that defined relevance in terms of inference (Cooper, 1971; Wilson, 1973). Another example of this type of approach is the RUBRIC system (Tong, 1987), which uses certainty values attached to inference rules. The approach we are pursuing is to extend the basic probabilistic model used in IR using the network formalism developed by Pearl (1989).

4 User Modeling

In the work on the I^{3}R system (Croft and Thompson, 1987; Thompson and Croft, 1989), a number of questions were raised about which form of user model improves retrieval effectiveness and how the knowledge in these models is acquired. Specifically, the I^{3}R system uses stereotypes activated by simple questions at the start of a session, and then builds a view of the domain knowledge of individual users by interactive dialogue during query formulation and evaluation of retrieved documents. The hypothesis is that each user may have an individual perspective on a domain area and that they are able to describe parts of a domain to the system. We are currently conducting a series of experiments to determine:

1. Are people able to provide descriptions of knowledge relevant to an information need.
2. Can the system make effective use of the knowledge provided by the users.

These studies have involved interesting experimental design issues dealing with evaluation in realistic environments and the impact of interfaces on performance.

Acknowledgments

The description of this research was put together with the help of students in the IR Laboratory: David Lewis, Bob Krovetz, Howard Turtle, and Raj Das. The research is supported by AFOSR, NSF and DEC.

References

[1] Belkin, N. and Croft, W.B., "Retrieval Techniques". *Annual Review of Information Science and Technology*, Edited by M.E. Williams, Elsevier Science Publishers, 22, 110-145, 1987.

[2] Cohen, P.R.; Kjeldsen, R. "Information Retrieval by Constrained Spreading Activation in Semantic Networks", *Information Processing and Management*, 23, 255-268, 1987.

[3] Cooper, W.S. "A Definition of Relevance for Information Retrieval", *Information Storage and Retrieval*, 7, 19-37, 1971.

[4] Croft, W.B., "Experiments with representation in a document retrieval system". *Information Technology*, 2, 1-22, 1983.

[5] Croft, W.B., "Boolean queries and term dependencies in probabilistic retrieval models". *Journal of the American Society for Information Science*, 37, 71-77, 1986.

[6] Croft, W. B.; Thompson, R., "I^{3}R: A New Approach to the Design of Document Retrieval Systems", *Journal of the American Society for Information Science*, 38, 389-404, 1987.

[7] Croft, W.B.; Turtle, H., "A Retrieval Model Incorporating Hypertext Links", *Proceedings of Hypertext 89*, 213-224, 1989.

[8] Croft, W.B., Lucia, T.J., Cringean, J, and Willett, P. "Retrieving Documents by Plausible Inference: An Experimental Study", *Information Processing and Management*, (in press).

[9] Dillon, M; Gray, A.S. "FASIT: A fully automatic syntactically based indexing system", *Journal of the American Society for Information Science*, 34, 99-108, 1983.

[10] Fagan, J.L., *Experiments in Automatic Phrase Indexing for Document Retrieval: A Comparison of Syntactic and Non-Syntactic Methods*, Ph.D. Thesis, Department of Computer Science, Cornell University, 1987.

[11] R. Krovetz and W.B. Croft, "Word Sense Disambiguation Using a Machine-Readable Dictionary", Proceedings of the 12th International Conference on Research and Development in Information Retrieval, 127-136, 1989.

[12] Lesk M., "Automatic Sense Disambiguation using Machine Readable Dictionaries: How to tell a Pine Cone from an Ice Cream Cone", Proceedings of SIGDOC, pp. 24-26, 1986.

[13] Lewis, D., Croft, W.B. and Bhandaru, N., "Language-Oriented Information Retrieval", *International Journal of Intelligent Systems*, 4, 285-318 (1989).

[14] Pearl, J., *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, California, 1989.

[15] Van Rijsbergen, C.J., *Information Retrieval.* Butterworths, London, 1979.

[16] Van Rijsbergen, C.J.. "A Non-Classical Logic for Information Retrieval". *Computer Journal*, 29, 481-48, 1986.

[17] Salton, G. and McGill, M., *An Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.

[18] Salton, G. "Another Look at Automatic Text Retrieval Systems", *Communications of the ACM*, 29, 648-656, 1986.

[19] Sparck Jones, K. *Automatic Keyword Classification for Information Retrieval.* Butterworths, London, 1970.

[20] Sparck Jones, K. "Automatic Indexing", *Journal of Documentation*, 30, 393-432, 1974.

[21] Sparck Jones, K. and Tait, J.I. "Automatic Search Term Variant Generation", *Journal of Documentation*, 40, 50-66, 1984.

[22] Thompson, R. and Croft, W.B., "Support for Browsing in an Intelligent Text Retrieval System", *International Journal of Man-Machine Studies*, 30, 639-668, 1989.

[23] Tong, R.; Appelbaum, L.; Askman, V.; Cunningham, J. "Conceptual Information Retrieval Using RUBRIC", *Proceedings of ACM SIGIR Conference*, 247-253, 1987.

[24] Wilks Y., Fass D., Guo C-M., McDonald J., Plate T., and Slator B., "A Tractable Machine Dictionary as a Resource for Computational Semantics", in Computational Lexicography for Natural Language Processing, Briscoe and Boguraev (eds), Longman, 1989.

[25] Willett, P., 'Recent Trends in Hierarchic Document Clustering: A Critical Review', *Information Processing and Management*, 24 (5), 577-598, 1988.

[26] Wilson, P. "Situational Relevance", *Information Storage and Retrieval*, 9, 457-471, 1973.

Signature–Based Text Retrieval Methods: A Survey

Christos Faloutsos
Univ. of Maryland, College Park
and UMIACS

1. Introduction

There are numerous applications involving storage and retrieval of textual data, including: Electronic office filing [36], [4]; computerized libraries [28], [26], [35]; automated law [16] and patent offices [17]; electronic encyclopedias [21], [12]; indexing of software components to enhance reusability [32]; searching in DNA databases [23]. Common operational characteristics in all these applications are: (a) Text databases are traditionally large and (b) they have archival nature: deletions and updates are rare.

Text retrieval methods form the following large classes [8]: Full text scanning, inversion, and signature files, which we shall focus next. Signature files constitute an inexact filter: They provide a quick test, which discards many of the non–qualifying items. Compared to full text scanning, the signature–based methods are much faster by 1 or 2 orders of magnitude, depending on the individual signature method. Compared to inversion, the signature–based methods are slower, but they require a modest space overhead (typically $\approx$ 10%–15% [3], as opposed to 50%–300% that inversion requires [14]); also, they can handle insertions more easily than inversion: they usually require fewer disk accesses, and they need "append–only" operations, thus working well on Write–Once–Read–Many (WORM) optical disks, which constitute an excellent archival medium [11], [2].

The paper is organized as follows: In section 2 we present the basic concepts in signature files and superimposed coding. In sections 3–6 we discuss several classes of signature methods. In section 7 we give the conclusions.

2. Basic Concepts

Signature files typically use superimposed coding [25] to create the signature of a document. A stop–list of common words is maintained; using hashing, every non–common word of the document yields a "word signature", which is a bit pattern of size F, with m bits set to "1" (see Figure 2.1). F and m are design parameters. The word signatures are OR–ed together to form the document signature. Searching for a word is handled by creating the signature of the word (query signature) and by examining each document signature for "1"'s in those bit positions that the signature of the search word has a "1".

To avoid having document signatures that are flooded with "1"'s, long documents are divided into "logical blocks", that is, pieces of text that contain a constant number D of distinct, non–common words [3]. Each logical block of a document gives a block signature; block signatures are concatenated, to form the document signature.

Word	Signature
free	001 000 110 010
text	000 010 101 001
block signature	001 010 111 011

Figure 2.1. Illustration of the superimposed coding method.
D=2 words per document; F=12 bits; m=4 bits per word.

The false drop probability F_d plays an important role in signature files:
DEFINITION: F_d, is the probability that a block signature seems to qualify, ***given that the block does not actually qualify*** (thus creating a "false drop" or "false alarm" or "false hit"). For the rest of this

This research was sponsored partially by the National Science Foundation under the grants DCR–86–16833, IRI–8719458 and IRI–8958546.

paper, F_d refers to single–word queries, unless explicitly mentioned otherwise.

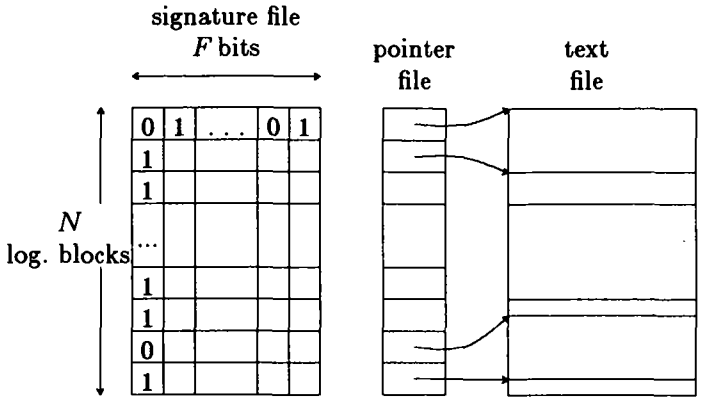


Figure 2.2. File structure for SSF

The signature file is an $F \times N$ binary matrix (called **signature matrix**). The value of m that minimizes F_d for a given value of F is [34]:

$$F \ln 2 = mD \tag{2.1}$$

In this case, each document signature is half–full with "1"s, conveying maximum information (entropy).

Symbol	Definition
F	signature size in bits
m	number of bits per word
D	number of distinct non–common words per document
F_d	false drop probability
O_n	space overhead of the signature file

The simplest signature method, the Sequential Signature File (**SSF**), stores the signature matrix sequentially, row by row. Figure 2.2 illustrates the file structure used: the so–called "pointer file" stores pointers to the beginnings of the logical blocks (or documents). SSF may be slow for large databases. Next, we examine alternative signature methods that trade off space or insertion simplicity for speed. Figure 2.3 shows a classification of these methods. All these methods use one or more of the following ideas:

1. Compression. If the signature matrix is deliberately sparse, it can be compressed.
2. Vertical partitioning. Storing the signature matrix column–wise improves the response time on the expense of insertion time.
3. Horizontal partitioning. Grouping similar signatures together and/or providing an index on the signature matrix may result in better–than–linear search.

Sequential storage of the signature matrix
  without compression: sequential signature files (SSF)
  with compression: bit–block compression (BC and VBC)
Vertical partitioning
  without compression: bit–sliced (BSSF, B'SSF), frame sliced (FSSF, GFSSF)
  with compression: compressed bit slices (CBS, DCBS, NFD)
Horizontal partitioning
  data independent: Gustafson's method; Partitioned signature files
  data dependent: 2–level signature files; S–trees

Figure 2.3. Classification of the signature–based methods

3. Compression

In this section we examine a family of methods suggested in [10]. These methods create sparse document signatures on purpose, and then compress them before storing them sequentially.

Using run–length encoding [24] to compress the sparse document signatures results in slow searching. The proposed **Bit–block Compression (BC)** method accelerates the search at the expense of space. It divides the sparse vector into groups of consecutive bits (bit–blocks) and encodes each bit–block.

The **Variable Bit–block Compression (VBC)** method uses a different value for the bit–block size b_{opt} for each document, according to the number W of bits set to "1" in the sparse vector. Thus, documents do not need to be split into logical blocks. This simplifies and accelerates the searching, especially on multi–term conjunctive queries on long documents.

Analysis in [10] shows that the best value for m is 1, when compression is used. The two methods (BC and VBC) require less space than SSF; thus, they are slightly faster than SSF, due to the decreased I/O requirements. Insertions are as easy as in SSF.

4. Vertical Partitioning

The idea behind the vertical partitioning is to avoid bringing useless portions of the document signature in main memory; this can be achieved by storing the signature file in a bit–sliced form [29], [9], or in a "frame–sliced" form [22].

The **Bit–Sliced Signature Files (BSSF)** store the signature matrix (see Figure 3.2) in a column–wise form. To allow insertions, F different files can be used, one per each bit position, which will be referred to by "bit–files". Searching for a single word requires the retrieval of $m(\approx 10)$ bit vectors, instead of all of the $F(\approx 1000)$ bit vectors. Thus, the method requires significantly less I/O than SSF. The retrieved bit vectors are subsequently ANDed together; the resulting bit vector has N bits, with "1"'s at the positions of the qualifying logical blocks. An insertion of a new logical block requires no rewriting – just F disk accesses, one for each bit–file.

B'SSF suggests using a value for m that is smaller than the optimal (Eq. (2.1)). Thus, the number of random disk accesses upon searching decreases. The drawback is that the document signatures have to be longer, to maintain the same false drop probability.

The **Frame–sliced signature file (FSSF)** forces each word to hash into bit positions that are close to each other in the document signature. Then, these columns of the signature matrix are stored in the same file and can be retrieved with few random disk accesses. Figure 4.1 gives an example for this method. The document signature (F bits long) is divided into k frames of s consecutive bits each. Each word in the document hashes to one of the k frames; using another hash function, the word sets m (not necessarily distinct) bits in that frame. F,k,s,m are design parameters. The signature matrix is stored frame–wise, using k "frame files". Ideally, each frame file could be stored on consecutive disk blocks. Since only one frame has to be retrieved for a single word query, as few as only one random disk access is required. Thus, compared to BSSF, the method saves random disk accesses (which are expensive – 18ms–200ms) at the cost of more sequential disk accesses. Insertion is much faster than BSSF since only $k(\approx 20)$ frame files need to be appended to, instead of $F(\approx 1000)$ bit files.

Word	Signature
free	000000 110010
text	010110 000000
doc. signature	010110 110010

Figure 4.1 D = 2 words, F=12 bits, k=2 frames, m=3 bits per word.
"*free*" hashes into the second frame; "*text*" into the first one.

The **Generalized Frame–Sliced Signature File (GFSSF)** allows each word to hash to $n \geq 1$ frames, setting m bits in each of these frames [22]. Notice that BSSF, B'SSF, FSSF and SSF are actually special cases of GFSSF: For $k=F$, $n=m$, GFSSF reduces to the BSSF or B'SSF method; for $n=1$, it

reduces to the FSSF method and for k=1, n= 1, it reduces to the SSF method.

Performance: We have carried out experiments [22] on a 2.8Mb database with average document size $\approx$ 1Kb and D=58 distinct non-common words per document. The experiments run on a SUN 3/50 with a disk, when the load was light (no other user, for most of the time). Averaged over 1000 single-word queries, the response time ("real time") was 420 ms for FSSF with s=63, m=8, and O_v = 18%, and 480 ms for GFSSF with s=15, n=3, m=3, and O_v = 18%. Full text scanning with UNIX's "grep" requires $\approx$45 sec for the same queries, i.e., two order of magnitudes slower. SSF is expected to be $\approx$10 times faster than "grep".

5. Vertical Partitioning and Compression

The idea in all the methods in this class [9] is to create a very sparse signature matrix, to store it in a bit sliced form, and compress each bit slice by storing the position of the "1"s in the slice. The methods in this class are closely related to inversion with a hash table.

The **Compressed Bit Slices (CBS)** method tries to accelerate the BSSF method, by setting m=1. Thus, it requires fewer disk accesses on searching. As in B'SSF, to maintain the same false drop probability, F has to be increased (to $\approx 2^{16}$). The easiest way to compress the resulting sparse bit file is to store the positions of the "1"'s. Since the size of each bit file after compression is unpredictable, use of a chain of buckets is suggested. The size B_p of a bucket is a design parameter. We also need a directory (hash table) with F pointers, one for each bit slice. Notice that there is no need to split documents into logical blocks any more; also, the pointer file can be eliminated: Instead of storing the position of each "1" in a (compressed) bit file, we can store a pointer to the document in the text file.

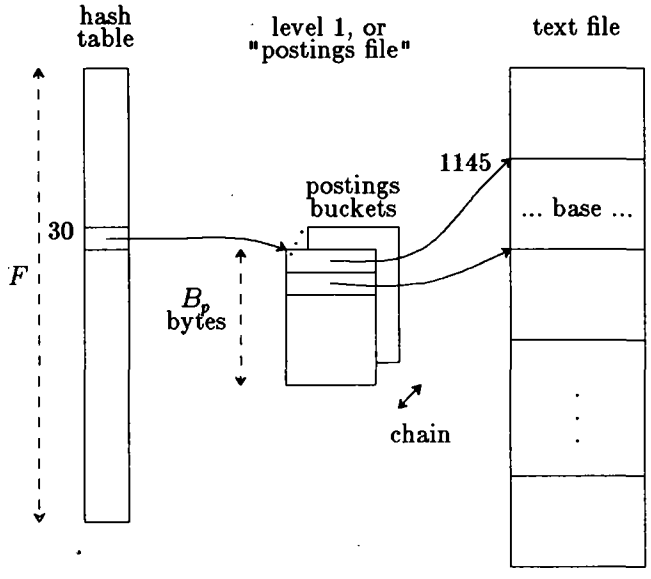


Figure 5.1
Illustration of CBS

Figure 5.1 illustrates the proposed file structure, and gives an example, assuming that the word "base" hashes to the 30-th position (h("base")=30), and that it appears in the document starting at the 1145-th byte of the text file. Notice that the method requires no re-writing. It is very similar to hash-based inverted files, with the following differences: (a) The directory (hash table) is sparse; traditional hashing schemes require loads of 80-90% (b) The actual word is not stored in the index. Since the hash table is sparse, there will be few collisions. Thus, we save space and maintain a simple file structure.

The **Doubly Compressed Bit Slices (DCBS)** method tries to compress the sparse directory of CBS. The idea is to use a shorter hash table, and to use a short (1 byte) code to distinguish between the synonyms. This short code is decided by using a second hashing function. The detailed file structure is in

[9]; the method still has the append–only property.

The No False Drops method (NFD) avoids the false drops completely without storing the actual words in the index structure; instead, it stores a pointer which points to the first occurrence of the word in the text file. This way each word can be completely distinguished from its synonyms, using less space: one pointer (usually, 4 bytes) instead of the full word (a word from the dictionary is ≈ 8 characters long [27]). Moreover, this approach avoids problems with variable–length records in the index. Like the previous methods, NFD requires no rewriting on insertions.

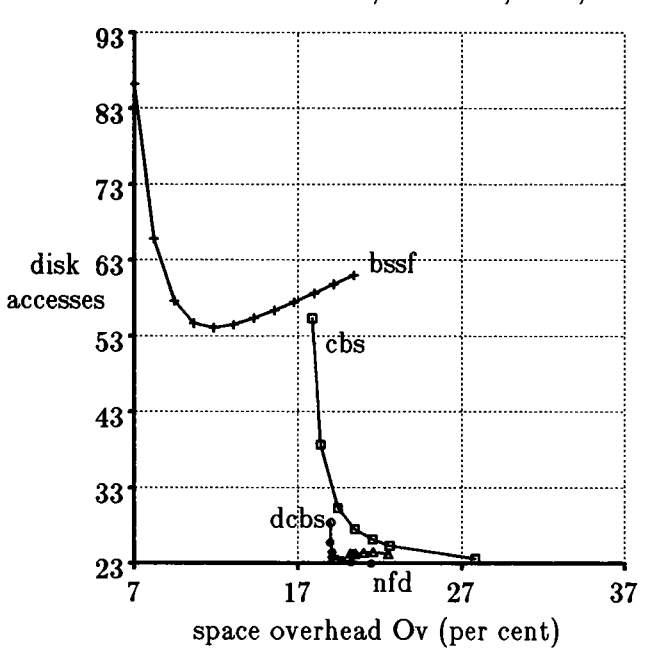


Figure 5.2. Total disk accesses on successful search versus space overhead.
Analytical results for the 2.8 Mb data base, with p=3 bytes per pointer.
Squares correspond to the CBS method, circles to DCBS and triangles to NFD.

Performance: In [9] an analytical model is developed for the performance of each of the above methods. Experiments on the same database that was used in Section 4 showed that the model is accurate. Figure 5.2 plots the theoretical performance of the methods (search time as a function of the overhead). The final conclusion is that these methods require few disk accesses, they introduce 20–25% space overhead and they still need append–only operations on insertion.

6. Horizontal partitioning

The motivation behind all these methods is to avoid the sequential scanning of the signature file (or its bit–slices), to achieve better than $O(N)$ search time. Thus, they group the signatures into sets, partitioning the signature matrix horizontally. The grouping criterion can be decided before hand, in the form of a hashing function $h(S)$, where S is a document signature (data–independent case). Alternatively, the groups can be determined on the fly, using a hierarchical structure (e.g., like a B–tree) (data dependent case).

6.1. Data independent case.

Gustafson's method [13] is best illustrated with an example ([19] p. 562): Consider bibliographic titles (records) with, say, 6 keywords (attributes) each. The method uses superimposed coding with F=16 bits and m=1 bit per keyword, to map each title into a 16–bit bit pattern. If k<6 bits are set in a record signature, additional $6-k$ bits are set by some random method. Thus, there are $C(16,6)$= 8008

possible distinct record signatures (where $C(m,n)$ denotes the combinations of m choose n items). Using a hash table with 8008 slots, we can map each record signature uniquely to one such slot (see [19]). Notice that the extent of the search decreases quickly (almost exponentially) with the number of terms in the (conjunctive) query: Single word queries touch $C(15,5)=3003$ slots of the hash table, two–word queries touch $C(14,4)=1001$ slots etc. Although elegant, Gustafson's method suffers from some practical problems [8], the most important being that the method can not handle documents with many keywords, because it needs a huge hash table then.

Partitioned signature files: D. Lee and C. Leng [20] proposed a family of methods that can be applied for longer documents. They suggested using a portion of a document signature as a key to partition the signature file. For example, we can choose the first 10 bits of a signature as its key; all the signatures with the same key will be grouped into a so–called "module". In this example we have 2^{10} modules. When searching for a word, we examine the first 10 bits of its signature and avoid searching the modules that don't match the key of the search word. Lee and Leng suggested additional, improved methods for extracting keys. In their simulation experiments, they reported 15 to 85 percent speed ups over SSF, depending on the number of bits being specified in the query signature.

6.2. Data dependent case.

Two level signature files: Sacks–Davis and his colleagues [31], [30] suggested using a two levels of signatures. Their documents are bibliographic records of variable length. The first level of signatures consists of document signatures that are stored sequentially, as in the SSF method. The second level consists of "block signatures"; each such signature corresponds to one block (group) of bibliographic records, and is created by superimposing the signatures of all the words in this block, ignoring the record boundaries. The second level is stored in a bit–sliced form. Each level has its own hashing functions that map words to bit positions. Searching is performed by scanning the block signatures first, and then concentrating on these portions of the first level signature file that seem promising. A subtle problem arises when multi–term conjunctive queries are asked: A block may result in an unsuccessful block match, because it may contain the desired terms, but not within the same record. The inventors propose a variety of clever ways to minimize these block matches.

Analysis on a database with $N \approx 10^6$ records (with 128 bytes per record on the average) reported response times as low as 0.3 seconds for single word queries, when 1 record matched the query. The BSSF method required $\approx$ 1–5 seconds for the same situation.

S–tree. Deppisch [7] proposed a B–tree like structure to facilitate fast access to the records (which are signatures) in a signature file. The leaf of a S–tree consists of k "similar" (i.e. with small Hamming distance) document signatures along with the document identifiers. The OR–ing of these k document signatures forms the "key" of an entry in an upper level node, which serves as a directory for the leaves. Higher–level nodes are constructed recursively, in the same way. Like a B–tree, the S–tree is kept balanced: when a leaf node overflows it is split in two groups of "similar" signatures; the father node is changed appropriately to reflect the new situation. Splits may propagate upwards.

The method requires small space overhead; the response time on queries is difficult to estimate analytically. The insertion requires a few disk accesses (proportional to the height of the tree at worst), but the append–only property is lost. Another problem is that higher level nodes may contain keys that have many 1's and thus become useless.

7. Discussion.

Signature files provide a space–time trade–off between the two extremes: full text scanning, which is slow but requires no overheads, and inversion, which is fast but requires expensive insertions and needs significant space overhead. Thus, signature–based methods have been applied in the following environments:

1) Medium size databases. "Hyperties" [21], a commercial hypertext package for IBM PCs and SUNs, uses the SSF method. Another product [18] also uses the signature approach.

2) Message filing in office automation. Several prototype systems use signatures, such as the Office Filing Project (OFP) [37] at the University of Toronto, MINOS [5] at the University of Waterloo, the

MULTOS project [6] funded by the ESPRIT project of the European Economic Community (EEC).

3) Thanks to the append–only insertion, signature–based methods can be used on WORM optical disks.

4) Signature files can easily benefit from parallelism. Stanfill and Kahle [33] used signature files on the Connection Machine [15].

5) As a simple and flexible access method for records and "long fields" (=text and complex objects) in extensible DBMSs. Chang and Schek [1] use signature files in IBM's STARBUST system.

References

1. Chang, W.W. and H.J. Schek, "A Signature Access Method for the Starbust Database System," *Proc. VLDB Conference*, pp. 145–153, Amsterdam, Netherlands, Aug. 22–25 1989.
2. Christodoulakis, S., "Analysis of Retrieval Performance for Records and Objects Using Optical Disk Technology," *ACM TODS*, vol. 12, no. 2, pp. 137–169, June 1987.
3. Christodoulakis, S. and C. Faloutsos, "Design Considerations for a Message File Server," *IEEE Trans. on Software Engineering*, vol. SE–10, no. 2, pp. 201–210, March 1984.
4. Christodoulakis, S., F. Ho, and M. Theodoridou, "The Multimedia Object Presentation Manager in MINOS: A Symmetric Approach," *Proc. ACM SIGMOD*, May 1986.
5. Christodoulakis, S., M. Theodoridou, F. Ho, M. Papa, and A. Pathria, "Multimedia Document Presentation, Information Extraction and Document Formation in MINOS: A Model and a System," *ACM TOOIS*, vol. 4, no. 4, Oct. 1986.
6. Croft, W.B. and P. Savino, "Implementing Ranking Strategies Using Text Signatures," *ACM Trans. on Office Informations Systems (TOOIS)*, vol. 6, no. 1, pp. 42–62, Jan. 1988.
7. Deppisch, U., "S–tree: A Dynamic Balanced Signature Index for Office Retrieval," *Proc. of ACM "Research and Development in Information Retrieval"*, pp. 77–87, Pisa, Italy, Sept. 8–10, 1986.
8. Faloutsos, C., "Access Methods for Text," *ACM Computing Surveys*, vol. 17, no. 1, pp. 49–74, March 1985.
9. Faloutsos, C. and R. Chan, "Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison," *Proc. 14th International Conf. on VLDB* , pp. 280–293, Long Beach, California, Aug. 1988.
10. Faloutsos, C. and S. Christodoulakis, "Description and Performance Analysis of Signature File Methods," *ACM TOOIS*, vol. 5, no. 3, pp. 237–257, 1987.
11. Fujitani, L., "Laser Optical Disk: The Coming Revolution in On–Line Storage," *CACM*, vol. 27, no. 6, pp. 546–554, June 1984.
12. Gonnet, G.H. and F.W. Tompa, "Mind your grammar: a new approach to modelling text," *Proc. of the Thirteenth Int. Conf. on Very Large Data Bases*, pp. 339–346, Brighton, England, Sept. 1–4, 1987.
13. Gustafson, R. A., "Elements of the Randomized Combinatorial File Structure," *ACM SIGIR, Proc. of the Symposium on Information Storage and Retrieval*, pp. 163–174, Univ. of Maryland, Apr. 1971.
14. Haskin, R.L., "Special–Purpose Processors for Text Retrieval," *Database Engineering*, vol. 4, no. 1, pp. 16–29, Sept. 1981.
15. Hillis, D., *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.
16. Hollaar, L.A., "Text Retrieval Computers," *IEEE Computer Magazine*, vol. 12, no. 3, pp. 40–50, March 1979.
17. Hollaar, L.A., K.F. Smith, W.H. Chow, P.A. Emrath, and R.L. Haskin, "Architecture and Operation of a Large, Full–Text Information–Retrieval System," in *Advanced Database Machine Architecture*, ed. D.K. Hsiao, pp. 256–299, Prentice–Hall, Englewood Cliffs, New Jersey, 1983.
18. Kimbrell, R.E., "Searching for Text? Send an N–gram!," *Byte*, vol. 13, no. 5, pp. 297–312, May 1988.

19. Knuth, D.E., *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison–Wesley, Reading, Mass, 1973.

20. Lee, D.L. and C.–W. Leng, "Partitioned Signature File: Designs and Performance Evaluation," *ACM Trans. on Information Systems (TOIS)*, vol. 7, no. 2, pp. 158–180, April 1989.

21. Lee, R., C. Faloutsos, C. Plaisant, and B. Shneiderman, "Incorporating String Search in a Hypertext System: User Interface and Physical Design Issues," Dept. of Computer Science, Univ. of Maryland, College Park, 1988. working paper

22. Lin, Z. and C. Faloutsos, "Frame Sliced Signature Files," CS–TR–2146 and UMIACS–TR–88–88, Dept. of Computer Science, Univ. of Maryland, Dec. 1988.

23. Lipman, D.J. and W.R. Pearson, "Rapid and Sensitive Protein Similarity Searches," *Science*, vol. 227, pp. 1435–1441, American Association for the Advancement of Science, March 22, 1985.

24. McIlroy, M.D., "Development of a Spelling List," *IEEE Trans. on Communications*, vol. COM–30, no. 1, pp. 91–99, Jan. 1982.

25. Mooers, C., "Application of Random Codes to the Gathering of Statistical Information," Bulletin 31, Zator Co, Cambridge, Mass, 1949. based on M.S. thesis, MIT, January 1948

26. Nofel, P.J., "40 Million Hits on Optical Disk," *Modern Office Technology*, pp. 84–88, March 1986.

27. Peterson, J.L., "Computer Programs for Detecting and Correcting Spelling Errors," *CACM*, vol. 23, no. 12, pp. 676–687, Dec. 1980.

28. Price, Joseph, "The Optical Disk Pilot Project At the Library of Congress," *Videodisc and Optical Disk*, vol. 4, no. 6, pp. 424–432, Nov.–Dec. 1984.

29. Roberts, C.S., "Partial–Match Retrieval via the Method of Superimposed Codes," *Proc. IEEE*, vol. 67, no. 12, pp. 1624–1642, Dec. 1979.

30. Sacks–Davis, R., A. Kent, and K. Ramamohanarao, "Multikey Access Methods Based on Superimposed Coding Techniques," *ACM Trans. on Database Systems (TODS)*, vol. 12, no. 4, pp. 655–696, Dec. 1987.

31. Sacks–Davis, R. and K. Ramamohanarao, "A Two Level Superimposed Coding Scheme for Partial Match Retrieval," *Information Systems*, vol. 8, no. 4, pp. 273–280, 1983.

32. Standish, T.A., "An Essay on Software Reuse," *IEEE Trans. on Software Engineering*, vol. SE–10, no. 5, pp. 494–497, Sept. 1984.

33. Stanfill, C. and B. Kahle, "Parallel Free–Text Search on the Connection Machine System," *CACM*, vol. 29, no. 12, pp. 1229–1239, Dec. 1986.

34. Stiassny, S., "Mathematical Analysis of Various Superimposed Coding Methods," *American Documentation*, vol. 11, no. 2, pp. 155–169, Feb. 1960.

35. Thoma, G.R., S. Suthasinekul, F.A. Walker, J. Cookson, and M. Rashidian, "A Prototype System for the Electronic Storage and Retrieval of Document Images," *ACM TOOIS*, vol. 3, no. 3, July 1985.

36. Tsichritzis, D. and S. Christodoulakis, "Message Files," *ACM Trans. on Office Information Systems*, vol. 1, no. 1, pp. 88–98, Jan. 1983.

37. Tsichritzis, D., S. Christodoulakis, P. Economopoulos, C. Faloutsos, A. Lee, D. Lee, and J. Vandenbroek, and C. Woo, "A Multimedia Office Filing System," *Proc. 9th International Conference on VLDB*, Florence, Italy, Oct.–Nov. 1983.

Information Retrieval Using Parallel Signature Files

Craig Stanfill
Thinking Machines Corporation
245 First Street
Cambridge MA 02142

1. Introduction

Over the past three years, Thinking Machines Corporation has explored the application of a parallel computer, the Connection Machine®, to Information Retrieval. The hope has been that the Connection Machine could deliver fast responses on information retrieval algorithms previously thought too slow for interactive access to large databases. This paper will present the results of work to date in this area.

1.1. Document Ranking and Relevance Feedback

A database is a set of documents. Each document is represented in the database as a set of tokens, which generally correspond to words or to word–stems. In some representations, the tokens will have real–valued weights representing the importance of the token in representing the content of the document; in other representations tokens are unweighted. A query is a set of tokens, each of which has a weight representing the importance of that token in determining what is to be retrieved. Retrieval is accomplished by a two–stage process of scoring and ranking. In the scoring phase, each document is assigned a score according to the overlap between the document's tokens and the tokens in the query. The exact manner in which the score is computed will be described below. Once the documents are scored, the documents with the highest scores will be extracted, sorted, and presented to the user.

Queries can be produced by several methods. Typically, the user will start by entering a set of words which are likely to be contained in relevant documents. The system will automatically determine weights for those words, and apply the resulting query to the database. The documents that are retrieved will be presented to the user, who will then read them and determine whether they are, in fact, relevant. He will then inform the system as to his relevance judgements. The system will scan the text of these documents, extract the most important words from them, and formulate a new query which is a composite of the user's original query and words from the relevant document. The resulting query may contain a hundred terms or more. In this way, documents which are similar to the relevant documents but which were not retrieved by the original manually–entered query will be found. This retrieval method is called *relevance feedback.*

These methods – document ranking and relevance feedback – have been known for nearly 20 years [1][2]. However, they have not met with widespread use. The majority of the online database

---

Connection Machine is a registered trademark of Thinking Machines Corporation.
VAX 8800 is a registered trademark of Digital Equipment Corporation.
Sun–4 is a registered trademark of Sun Microsystems.
DowQuest is a registered trademark of Dow Jones and Company, Inc.

industry remains dominated by a very different model, in which the user produces a boolean query, which retrieves documents containing specific combinations of words.

The aim of the work presented in this paper has been to deliver interactive access to very large databases using document ranking and relevance feedback, in the hope that this would remove obstacles to their commercial application. Judged by this standard, the project has been successful. The remainder of the paper will describe the system which has evolved over the past several years.

2. The Connection Machine

The Connection Machine model CM–2 is a massively parallel, fine grained, SIMD computer [3]. A full description of the architecture is beyond the scope of this paper, but the following discussion will characterize the architecture in sufficient detail to allow the remainder of the paper to be understood. The machine consists of up to 64K (65,536) bit–serial processing elements (PE's). Smaller configurations having 4K, 8K, 16K, and 32K PE's are also available.

Each PE in the Connection Machine has a local memory. The initial release of the CM–2 had 64K bits per processor. The most recently introduced model (CM–2a) has 256K bits per processor. CM memory is divided in software into *fields*. A field can be thought of as a vector having one element per PE. Local computation is accomplished by performing arithmetic operations on fields (e.g. addition, logical AND). All operations on the CM may be masked by use of a context flag (processors for which the context flag is 0 will ignore most operations).

The Connection Machine operates under the control of a host computer; typical front ends include the Sun–4® workstation and the VAX 8800® minicomputer. The user's program runs on the host, and has full access to the host's scalar instruction set, I/O devices, and file system. A library of routines provides a software interface between the CM's parallel instruction set (PARIS) and the user program.

The Connection Machine supports a number of non–local operations: operations in which data moves either from the PE's to the scalar processor, or from PE to PE. The simplest of these are the global–reduce operations. For example, the global–maximum operation takes all elements of a field, determines the largest value, and stores the result in the host as a scalar. A second non–local operation is SENDing data. In this operation, one field contains data, and a second field contains addresses of processing elements. The contents of the data field are permuted according to the contents of the address field. There are a great number of non–local operations; they will be explained as required.

3. The Signature File Implementation

3.1. Properties of Parallel Signature Files

The current CM implementation of information retrieval is based on a parallel implementation of overlap encoding [4][5]. The details of this implementation are well explained in the literature, and will not be covered in detail here. For the moment, the following properties of signatures are salient:

1. A signature is a structure for representing sets of words. Words may be inserted in signatures. Signatures may be probed for the presence of words.

2. The representation is probabilistic. If a word is inserted in the signature, subsequent probes for that word are guaranteed to return PRESENT. However, the algorithm sometimes returns PRESENT for words that were never inserted.

3. The probability of such errors may be kept arbitrarily low by limiting the number of words to be inserted into each signature. For 4K–bit signatures, limiting the number of words to 120 keeps the probability of a false positive below 1E–6; this is adequate for most purposes.

4. Probing of signatures on Connection Machines is extremely fast. Assuming one signature per processing element, each probe takes 5 microseconds. Thus, on a 64K–processor system it is possible to perform 13 billion probes per second.

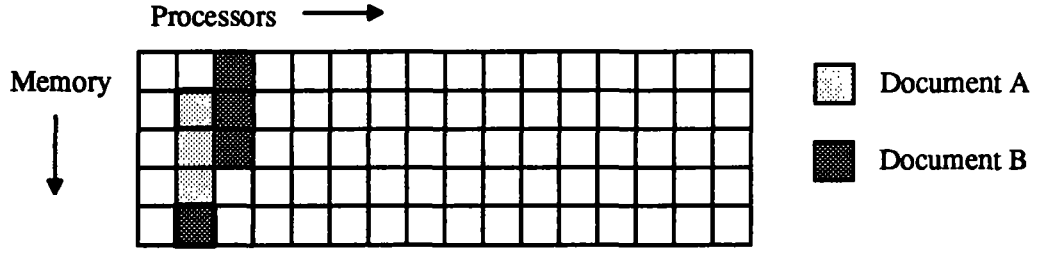
5. In practice, each processor of the Connection Machine is given several signatures. For a CM with 64K–bits of memory per processor and 4K–bit signatures, 14 signatures per processor are generally used (leaving sufficient room for scratch space). For newer models with 256K–bits per processor, 56 signatures are used.

6. The Connection Machine iterates through the signatures in each processor. Thus, with 14 signatures per processor, probes take 70 microseconds; with 56 per processor this takes 280 microseconds.

7. The use of signatures does not permit document representations in which document–terms are weighted. This has implications in terms of which retrieval methods may be implemented.

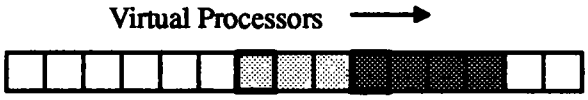
8. In databases for which the size of the documents is variable, documents must be represented by groups of signatures. In the parallel implementation, this requires a method for combining of probes to a number of signatures. These combination operations will be discussed in some detail below.

Signatures and documents have the following arrangement in the Connection Machine's memory:



Documents occupy contiguous groups of signatures within each processor (e.g. Document A). In some cases documents will span processor boundaries (Document B). The first signature in each document (indicated by heavy boxes in the diagram above) is called the *head signature*. In some cases we will use the view shown above, in which the database is viewed as a 2–dimensional array of signatures, with columns corresponding to processors and rows corresponding to memory locations within processors. However, for most purposes it is expedient to view the database as a linear sequence of *virtual processors*. The exact mechanisms which allow this to be done are beyond the

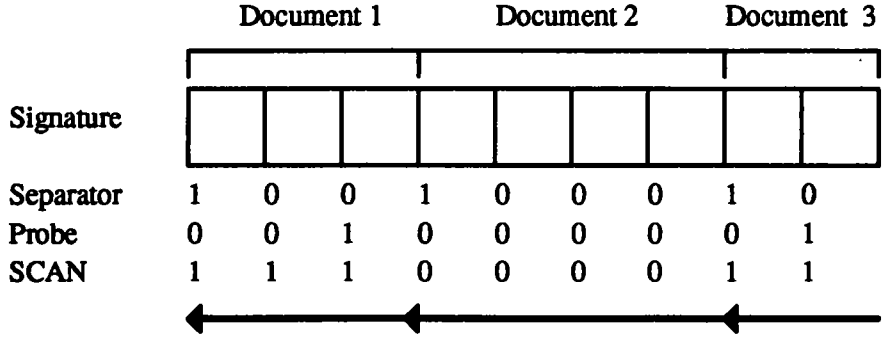
scope of this paper. The majority of the discussion presented below will use the virtual processor model. The ratio of virtual to physical processors is called the *VP ratio*, and is denoted by the symbol *V*.



3.2. The Document Scoring Algorithm

A query consists of a set of words with associated weights. The basic retrieval algorithm scores each document by summing the weights of the query–words which it contains, then presents the user with those documents having the highest scores.

The first step is for each virtual processor to allocate a field to contain its document's score, and initialize it to 0. The score field is valid for virtual processors containing head signatures; other processors ignore its value. The algorithm then proceeds to iterate through the terms in the query. First, each processing element will probe its signature for the query term. The next step is to OR together these result bits for all the segments in a document. A single non–local operation called a *segmented OR–scan* is used to perform this operation:



Those virtual processors for which the result of the SCAN is 1 will then add the term's weight to their score field.

This scoring method requires one PROBE, one SCAN, and one ADD for each query term. The total time for these operations is 20 microseconds times the virtual processing ratio.

If there are Q terms in the query, then the above operation must be performed Q times. Once this has been done, each of the head processors will contain the total score for a complete document.

3.3. The Document Ranking Algorithm

It is now necessary to rank the documents. The first step is to append the score and an identifier for the document. If document 1 has a score of 3, the token 3,1 will result. Non–head processors will receive 0. Ignoring the virtual processing model and viewing the CM as a 2–dimensional array of processors, the following data structure has been prepared:

3,1	0	30,5	0	30,9
0	6,3	0	5,7	0
0	18,4	0	0	0
4,2	0	17,6	0	17,10
0	0	0	8,8	0

The first step in ranking the documents is to find the largest value in each row. This requires that the *global maximum* operation be repetitively applied. The largest values are stored in an array on the host processor. This yields the following situation:

					MAX
3,1	0	30,5	0	30,9	30,9
0	6,3	0	5,7	0	6,3
0	18,4	0	0	0	18,4
4,2	0	17,6	0	17,10	17,10
0	0	0	8,8	0	8,8

The front end scans this array to find the largest token (in this case, 30,9). Once this is done, the CM memory location having that token is 0'ed out, and the MAX for that row is re–computed:

					MAX
3,1	0	30,5	0	0	30,5
0	6,3	0	5,7	0	6,3
0	18,4	0	0	0	18,4
4,2	0	17,6	0	17,10	17,10
0	0	0	8,8	0	8,8

This step is repeated once for every document to be presented to the user. If V is the VP ratio and D is the number of documents to be presented, then $(V + D - 1)$ global maximum operations are required. Each global–maximum operation takes 600 microseconds.

3.4. Performance

The following parameters affecting performance have thus far been introduced:

V	Virtual Processing Ratio
Q	Number of query terms
D	Number of documents to be retrieved.

The scoring phase takes time $Q * V * 20E{-}6$. The ranking phase takes time $(D + V - 1) * 600E{-}6$. Some representative query times are shown below (times are in seconds):

Q	V	D	Time
10	14	20	.022
10	56	20	.056

100	14	20	.047
100	56	20	.157

The size of database which may be represented by a memory–resident signature file is governed by: 1) the number of document–terms stored in each signature; 2) the total number of signatures in the machine; and 3) the number of document–terms which result from processing 1 MB of raw text. As noted above, a 4 K–bit signature comfortably holds 120 document terms. Assuming 56 signatures per processor and a 64K–processor machine, one gets a capacity of 440 million document terms. On the databases for which the Connection Machine has been used, there are generally about 80,000 document terms in a megabyte of text (after stop–words such as *the* and *it* have been removed). This yields a total capacity of 5500 megabytes of raw text.

In summary, a 64K processor Connection Machine model CM–2a can store signatures for a 5.5 Gigabyte database, and apply a 100–term query to it in .157 second. It can apply a 10–term query in .056 second.

4. Responses to Questions

Since the original publication of the parallel signature algorithm, two papers have questioned its usefulness. The first, due to Stone, criticized an early proposal to use a high–speed disk system to support sequential searches of databases too large to fit into the Connection Machine's memory. The second, due to Salton, asserted that 1) the CM was no faster than a SUN–3 using inverted indexes; 2) even if it was, there was no need for such fast response times; and 3) the signature algorithm was fatally flawed in that it did not support document–term weighting.

Before responding to these in detail, it is important to understand the difference between what was described in the original paper and what has been presented above. When this work was first published in *CACM*, only the Connection Machine model CM–1 was available. This machine had 4K bits per processor, in contrast to the current 256K–bits. The CM–2 has a somewhat faster clock, but in most other respects is similar to the CM–1.

The problem with the CM–1 in this application was that, in spite of its high performance, it had a small memory capacity and was thus unable to hold a database of a realistic size in–memory. In practice, it was only possible to fit 3 1K–bit signatures in memory. A 1K–bit signature holds 30 document terms. Results were reported for a 16K–processor machine. Repeating the computations outlined above, this gives a capacity of 1.5 million document–terms, corresponding to a full–text database of approximately 19 megabytes.

In order to extend the size of the database, it was proposed to repeatedly load the contents of memory from a high–performance disk array and execute queries against each memory–load in turn. Stone[6] pointed out that such a system, when applied to queries with less than several thousand terms, did much more I/O than a conventional system based on inverted indexes. He did not offer any criticism of the in–memory performance of the system. In practice, the need for the disk–based system has been obviated by the availability of larger main memories.

Salton went further, claiming that inverted index algorithms on serial machines were generally superior to the parallel signature algorithm[7].

His first claim was that the parallel signature algorithm is no faster than the well known serial inverted index algorithm when the database is memory–resident. He offered as evidence a computation claiming that a 3.5 MIP Sun–3, evaluating a query with 200 terms against a database with 16,000 documents could process a query in 40 milliseconds, which is the same time as was quoted in [4] for a 16K–processor Connection Machine running against a 20 megabyte database. This argument is, however, weak in that it pits a theoretical computation based on a theoretical database against a real computation on a real database. Additionally, it includes only the time to score documents, omitting time required to rank them. Finally, whatever this result may have said about the CM–1, it is at this time obsolete as the CM–2 is a much more capable machine.

Running a real query against a real database, it was found that a Sun–4 could evaluate a 100–term query against a 13 Megabyte database in an average of .037 seconds[8]. By way of comparison, the most recent version of the CM–2 can execute a 100–term query against a 5.5 Gigabyte database in .147 seconds, which amounts to a 100–fold advantage in performance.

Salton's second claim was that the parallel signature algorithm's performance was simply not needed for retrieval. For databases small enough to fit into the memory of the CM–1, this is perhaps true; a serial machine can deliver adequate performance on 10 Megabytes of data. The performance of parallel algorithm is, however, needed for large databases. If the serial algorithm takes .037 seconds on a 13 Megabyte database, then it will take 16 seconds on a 5.5 gigabyte database (compared with .157 seconds using the parallel algorithm). Given the choice of sub–second response and 16–second response, most users will choose the former. A further advantage of the sub–second response speed is that, rather than servicing 4 requests per minute, the system will be able to handle over 6 per second. Thus, the higher performance of the parallel algorithm translates both into faster response against large databases and into higher throughput.

Salton's final claim is that, because the signature algorithm prevents the use of document–term weighting, it will deliver much less effective searches than implementations that support document–term weighs. While it is true that document–term weighting is desirable, it is clear that the query–term weighting supported by the signature algorithm is superior to the boolean queries which are still the basis of most commercial retrieval systems[9].

5. Summary and Conclusions

In summary, the following can be safely said about the parallel signature algorithm:

1. For databases which fit in main memory, the parallel signature algorithm is much faster than the serial inverted index algorithm.

2. There is no acceptable method for extending the algorithm to secondary storage.

3. Parallel signatures do not support document–term weighting. The result is that they offer somewhat lower search effectiveness than methods that do support document term weighting, but higher search effectiveness than implementations which support only boolean queries.

In June 1989, Dow Jones Inc. introduced a full text retrieval service, called DowQuest®, based on the algorithms described above. Their system includes a 32K–processor CM–2 with 64K–bits of

memory per processor, holding a total of approximately 800 MB of data. Data from a large number of publications, such as the Wall Street Journal and the Washington Post, is included in this database. The user interface supports both user–supplied queries and relevance feedback. This system has attracted wide attention in the online database industry; its progress should be carefully watched over the next several years to see how users react to document ranking and relevance feedback.

Nevertheless, the limitations of the signature file algorithm noted above do exist. Recent work has focused on the development of parallel inverted index algorithms. Such algorithms should give excellent performance on disk–resident databases up to 1000 GB, while supporting the full document–term weighting model. Preliminary results are encouraging[10]; within a few years this family of algorithms should support a full range of information retrieval methods on the largest databases currently in existence.

REFERENCES

[1] Salton, G.J., *The SMART Retrieval System — Experiment in Automatic Document Processing*, Prentice–Hall, Englewood Cliffs NJ, 1971.

[2] Salton, G.J., *Automatic Text Processing*, Addison–Wesley Publishing Company, Reading, MA, 1989.

[3] "CM–2 Technical Summary," Thinking Machines Corporation Technical Report, 1987.

[4] Stanfill, C. and B. Kahle, "Parallel Free–Text Search on the Connection Machine System," *Communications of the ACM*, Volume 29 Number 12, December 1986, pp. 1229–1239.

[5] Pogue, C. and P. Willett, "Use of Text Signatures for Document Retrieval in a Highly Parallel Environment," *Parallel Computing* Volume 4, 1987, pp. 259–268.

[6] Stone, H., "Parallel Querying of Large Databases: A Case Study," *IEEE Computer*, October 1987, pp. 11–21.

[7] Salton, G., and C. Buckley, "Parallel Text Search Methods," *Communications of the ACM*, Volume 31 Number 2, February 1988, pp. 202–215.

[8] Stanfill, C., "Parallel Computing for Information Retrieval: Recent Developments," Technical Report DR88–1, Thinking Machines Corporation, Cambridge MA, January 1988.

[9] Croft, B., "Implementing Ranking Strategies Using Text Signatures," *ACM Transactions on Office Information Systems*, Volume 6 Number 1, January 1988, pp. 42–62.

[10] Stanfill, C., R. Thau, and D. Waltz, "A Parallel Indexed Algorithm for Information Retrieval," *Proceedings, ACM Conference on Research and Development in Information Retrieval*, June 1988, pp 88–97.

Special-Purpose Hardware For Text Searching: Past Experience, Future Potential

Lee A. Hollaar
Department of Computer Science
University of Utah
and
Contexture, Inc.

The partitioned finite state automaton (PFSA) was proposed ten years ago as a new way of implementing a hardware-based pattern matcher for text information retrieval systems. The pattern matcher was part of a special-purpose search processor to be attached to each disk of a large information retrieval system. Since then, there have been dramatic changes in microprocessor, memory, and disk technologies. After an overview of the experiences implementing and evaluating the PFSA searcher, the applicability of special purpose hardware for text searching in the future is discussed.

Introduction

Over a decade ago, I proposed a special-purpose processor attached to a disk drive to improve the performance of a text information retrieval system. The proposed searcher differed from others that had been discussed at that time in a number of important ways. Because the searcher was to be used with very large databases, contained on a hundred or more disk drives, much of the speedup over a conventional computer or other hardware-based searchers would come from the natural parallelism that results from having a searcher attached to each disk drive. This, of course, means that the cost of the searcher should not be substantially more than the cost of the disk.

As part of the research into the searcher design, Roger Haskin developed a new technique for implementing a finite state character recognizer. The partitioned finite state automaton (PFSA) was based on dividing the state table that describes the terms to be matched into groups of compatible states. Two states are compatible if there is no possible input sequence that would cause them to be active at the same time, so that a circuit corresponding to a partition need do only one comparison for each input character from the disk. This, and its modest memory requirements, made the PFSA particularly attractive for implementation as a custom integrated circuit. PFSA operation has been detailed in an article by Haskin and Hollaar[1] and a United States Patent[2].

At the time the searcher was proposed, microprocessors were just becoming available, and certainly weren't fast enough to keep up with a disk drive when doing a search with many terms. In fact, high-end mainframes were not fast enough, searching at the rate of only about 100,000 characters per second[3]. The PFSA gave the necessary speed.

Memory size was also a consideration. While workstations with eight megabytes of memory or more are now commonplace, ten years ago 16 kilobit dynamic RAMs were just becoming widely used. The high cost of memory not only influenced the design of the PFSA matcher (which avoids a large state table memory) but resulted in a design that searched the data as soon as it was read from the disk drive, eliminating the need for memory to hold the document being searched. This required a search technique that did not require going back to a previous character in the document when a mismatch was detected.

The proposed searcher was designed to be part of a retrieval system that also used a surrogate of a document (a partially-inverted file, although other techniques are possible) as an

initial filter to determine the documents to be searched. Other searchers at that time had been designed to scan the entire database[4, 5], so that a search would take minutes for each disk of the database. If a reasonable percentage of the documents can be eliminated through the surrogate, the resulting search can be done in a few seconds. This meant that in many cases it is not necessary to batch a number of queries to give a reasonable response time, simplifying the control of the searcher, but made its performance more sensitive to the seek time of the disk.

Because each search engine was a self-contained unit with its own disk, it did not impact the host computer's I/O and memory systems. Lower cost disks, controllers, and memory could be used since they were all dedicated to searching, not supporting an operating system and other programs. This both lowered the cost of the system and improved its efficiency. Other search processors had been designed to search as rapidly as possible, requiring elaborate (and expensive) parallel transfer disk systems to match their character comparison rate[4].

The Prototype Implementation

A prototype of the search engine has been operational since 1985. It is based on a 190 megabyte ST-506-type disk drive. The entire search engine, including disk controller, search controller, query resolver, and PFSA character matchers fits on a single printed circuit card approximately 10 inches by 11 inches in size. It is packaged in a 5-1/4 inch high rack-mount chassis, connects to its host processor using a GPIB network, and uses less than 100 watts.

The circuit card contains 78 integrated circuits. An Intel 80186 microprocessor, used to control the searcher and for query resolution, uses 21 integrated circuits, primarily bus drivers and memory (256 KBytes of static RAM and 16 KBytes of EPROM, the latter holding bootstrap code, device support routines, and a low-level debugger). Test points are provided for attaching a logic analyzer to monitor microprocessor bus transactions. An additional 9 integrated circuits are used for the disk controller and interface, and 9 more circuits for the GPIB host interface and a serial diagnostic port.

The PFSA Matcher

The remaining 39 integrated circuits are used to implement the PFSA matcher. The prototype unit contains eight character matchers, each on a separate IC. The remaining integrated circuits are used for the matcher data paths (13 packages) and timing and control of the matcher (18 SSI circuits). Each character matcher contains a 64 state transition memory (16 bits per state), 16 states with forking extensions (14 bits per extension), and a startup table handling six bit characters (11 bits per character). The total memory requirement for each character matcher is 1,952 bits. This gives a total of 512 states, or approximately 75 normal search terms. The character matcher is implemented using 4 micron NMOS, although the memory is actually designed on a 6 micron pitch because it was first designed before a 4 micron process was available to us. Most of the area of the chip is taken up by the memory, memory address decoders, input/output pads, and busses. Less than 10 percent of the area consists of random logic.

To save memory (primarily in the startup table, but also in the comparand field of the transition table), the character matchers work with six-bit input characters. Since the data is stored on the disk in eight-bit ASCII, a mapping RAM converts each disk character to a six-bit character before it is presented to the PFSA. This mapping also gives the same representation to upper- and lower-case letters, if the search is to be case insensitive, or maps the two cases into six-bit characters differing in the high-order bit. A control flag in the transition table state determines whether this case bit should be considered in the comparison, allowing case sensitivity on a character-by-character basis.

The use of six-bit characters, saving over a factor of two in the CM memory requirements and allowing it to fit in a 40 pin package, proved to be a good decision, but one that constantly requires justification. Because the mapping function from disk characters to PFSA input is set based on each query, it permits 60 distinct interesting characters. The remaining four character patterns represent non-interesting alphabetic, numeric, and punctuation characters, as well as a word delimiter. With case insensitivity being the default, there has never been a time when a query was close to using the full 64 possible input characters.

Query Resolution

While the PFSA is capable of recognizing regular expressions, it is difficult to represent some common query forms as regular expressions. A good example of this is a query that specifies that five different terms must occur in the same paragraph, in any order. Rather than transform queries like this into regular expressions, the pattern matching process is divided into two parts. The first, performed by the PFSA, is the recognition of the individual terms of the pattern. The second combines the PFSA results indicating that particular terms have been detected to determine if the pattern has been matched. Because this latter processing is required only when a term has been found, its realtime requirements are considerably less than that of comparing characters from the disk, and it can be handled by a microprocessor.

The PFSA matcher uses the microprocessor DMA capabilities to write a sixteen entry describing a term match into the microprocessor memory. The entry contains three bits indicating which of the eight character matchers found the match, five bits indicating the transition table address (the low-order bit is dropped so that everything fits in a single microprocessor word), and an eight bit counter value indicating the word within the document where the match was found. It is possible for two search terms to match at the same point in the search, with the CM number reported being the lowest CM with a match, and the address the OR of all the state addresses with a match. Careful assignment assures that this composite term match report is distinct from any single term report.

The document word counter included in the hit entry comes from a counter that is advanced whenever a word delimiter is detected. This is based on a bit that comes from the mapping RAM, indicating that the current character is a delimiter. The counter logic advances only after all delimiters following a word have been seen. Whenever the eight bit counter overflows, a special hit entry is written. This is used to increment a software counter in the query resolution code, so an arbitrary length counter can be formed.

In effect, the PFSA term matcher converts a document into a list of entries that are a shorthand description of the document as it pertains to the query. The query resolution program can then process this greatly reduced information according to whatever scheme is desired. The prototype implementation uses an enhanced Boolean query language, with contexts and word location proximity, although weighted term document scoring or other techniques could easily be implemented. The query resolution is done on the microprocessor, and is a small C subroutine to the search control program. The search programs are written and debugged on a workstation, using a simulator and a conventional window-oriented debugger.

Prototype Performance

The search speed is determined by the slowest component, with other parts of the searcher balanced to that speed. In the case of the prototype searcher, the NMOS searchers operated with a 1400 ns cycle time (an implementation using more current CMOS technology and a better memory design would operate considerably faster, say with a 100 ns cycle time), which was a good match to a 5 megabit per second (1600 ns per character) ST-506-type disk drive and the worst-case DMA speed an 8 MHz Intel 80186 microprocessor. The reading of data from the disk and mapping to six bit characters is pipelined with the matching of the

previous disk character. No part of the system costs more than necessary, because its speed is matched to all other components.

Its performance on small documents and simple queries is about 2 to 3 times that of a 68020-based workstation; that increases to 100 times for larger documents (so that the overhead for positioning to the next document and starting its processing is smaller compared to the search times) or queries with many terms since, unlike the program on the workstation, the search time is independent of the number of search terms. In particular, search of 628 documents for a two word phrase takes over 90 seconds on a 25 MHz Apollo workstation, and 34 seconds on the prototype searcher. For a search of 7 documents for a query with 25 terms, the Apollo takes about 23 seconds and the searcher under three seconds.

Future Directions

Based on the experiences with the prototype PFSA searcher, how would I implement a text searcher given today's faster processors, larger and faster disks, and lower-cost memory? Disk drives holding over 700 MBytes of data now cost under $2000, a change in cost per bit of about 50 from ten years ago. One megabit dynamic RAM costs under $10, and 4 megabit RAM is now available. A RISC microprocessor costing under $10,000 runs at 12 MIPS, faster than large mainframes of a decade ago.

Need For Searching

The rational for needing a special search engine is that a conventional processor cannot do the job. For searching large databases, this is still the case and will continue to be true in the future. If a large database resides on a number of disk drives, it will still be faster and more efficient not to bring the raw information from the disk into a central processor, but instead search it with a processor closer to the disk. In this way, the search time will remain the same as the database size increases, since each search processor will continue to handle a single disk.

However, because of the slow searching speeds for a central processor, most commercial information retrieval systems use some sort of inverted file index rather than actually searching the documents. The argument in the past against inverted files was that they substantially increase the storage requirements of a database. The index may take more disks than the database itself, but this may be less of a concern with lower-cost disk drives. This means that to be competitive, the cost of searching must be comparable to the cost of a disk drive. This was one of the design goals for the original searcher, and obviously remains one in the future.

However, there are a number of problems associated with inverted files or other document surrogates besides their storage requirements. In most cases information is discarded to reduce the size of the surrogate. For inverted files, commonly-occurring words are generally not included in the index. This means that a search for a phrase like "changing of the guard" really finds anything that has "changing", any two words, and "guard". The phrase "To be or not to be, that is the question" matches any document containing the word "question", since all other words in the phrase are discarded. A similar problem exists for superimposed code words, where the combining of the codes for two words makes it seems like another word is present when it is not.

Our experience is that these artifacts of the use of a surrogate confuse a user because it is not clear why a document that doesn't match the query was retrieved. Time is spent trying to understand an irrelevant document, perhaps more time than would be spent reviewing a relevant one. But even if this were not the case, there are other problems associated with the use of a surrogate for locating documents. Time must be spent building and maintaining the surrogate, a non-trivial amount of time for very large databases. Also, a document is not retrievable until it has been entered into the surrogate data structure.

Searching Alone Is Not The Answer

If there are problems with the use of a surrogate, there are probably more problems not using some technique to reduce the amount of data that must be searched. If an initial filter is not used to limit the search, then the time necessary to complete a user's query is the time it takes to read all the data from a disk drive. While this obviously varies with the type of disk, for most low-cost, high-capacity disks it is about five minutes. This is far to long for an interactive system, and complicates the search control by requiring that a number of different user queries must be combined in a batch to give reasonable performance.

One solution to this problem is to make the search go faster by using higher performance disks, but this substantially increases the system cost because of the low production volumes for such disks. Furthermore, the speedup is on the order of 10, while a factor of 100 is desirable for an interactive system.

A more reasonable solution is to combine the attributes of using a surrogate and searching to overcome the difficulties with each approach. An inexact surrogate can be used to eliminate documents that have no hope of matching a query. Superimposed codewords will provide a list of documents that is a superset of the documents containing the search terms; if the term doesn't really occur in the document, but is an artifact of the superimposed codeword scheme, it will be eliminated by the search. A fully inverted file, where every term is indexed with its location within a document is not necessary; phrases, contexts, and word location proximities can be handled by the following search.

The partial inverted file that we have used adds less than 20 percent to the size of the database. Two lists of documents are returned by the surrogate operation: documents that match the query and documents that need further searching (a "maybe" list). For example, if the query were to find documents containing "beagle or basset hound" and no phrases were indexed, the first list would contain those documents that have the word "beagle" and the maybe list would be those documents that have "basset" and "hound" but not "beagle". Including common phrases in the index increases its performance with only a slight performance and storage penalty. Moreover, it is not necessary to index a document before it is available. As long as the number of unindexed documents remains low relative to an average search, they can simply be added to the maybe list for every search, making them available as soon as the text is loaded.

Seek and Search Mode

When a surrogate is used to reduce the number of documents to be searched, the search goes from a scan to a seek and search mode. This changes the critical disk parameter from its transfer rate to its seek time relative to document transfer time. To see why this is so, consider an ESDI drive like the Maxtor XT-8760, where the seek time and rotational latency to position to the start of a document is about 20 milliseconds. It takes another 25 msec to read 35,000 characters (the size of the average United States Patent) off two tracks, for a total of 45 msec. A disk drive with the same seek characteristics, but which could read the data in zero time, would be only 2.25 times faster. For smaller documents (say, those that fit on a single track), if it takes 25 milliseconds to reach the start of a document (8 ms average latency, 17 ms average seek), the effective transfer rate is 590 KBytes per second, a little over 30 percent of the nominal transfer rate of 1,875 KBytes per second.

Just as parallel transfer drives are not particularly effective in a seek and search mode, optical drives with their high seek times are even more devastating to search performance. If the seek time in the small document example above were changed to a 150 ms positioning time (typical of today's optical disks), the effective transfer rate is only 148 KBytes per second, less than 8 percent of the nominal transfer rate.

Substantial improvements can be made by selecting an appropriate file system. A randomly organized file system, where blocks are placed in any convenient free location on the disk and a pointer (either in the previous block or in a master block) is used to locate the next data block (such as used in most file systems) is convenient for a timesharing system, where files come and go. However, using our example ESDI disk drive, if we have to do an average seek before reading a 512 byte block, we will have an effective transfer rate of only 20 KBytes per second. The use of a large blocksize improves this, at the expense of higher unusable disk capacity due to internal fragmentation.

The use of a contiguous file system, where each document is stored in consecutive disk blocks, substantially improves disk performance. In one disk revolution after positioning, almost 50 blocks can be read, rather than just one. Since the documents are seldom removed (or expanded) after they are loaded into an archival text database, the problems of file reorganization and lost disk space are minimal.

Two other improvements to the file system can also substantially improve the effective transfer rate by reducing the number of seeks necessary to access a document. The first is to use an array, rather than a conventional directory, to store the location information for the documents. Since documents are generally accessed by some code number, rather than by a mnemonic name, the code number can be used to directly compute the location on the disk of the proper directory entry, rather than having to search all the directory blocks (and the directories would be large for a large database). The second improvement is to look up the directory information for a number of files at the same time, so the directory does not have to be accessed for every file. Getting the directory information for 100 documents at the same time can double the performance of the file system (and the search) for track-sized documents.

Processor Speed Requirements

If commodity disk drives are used to keep costs low, rather than special drives like those with parallel reads, the disk will determine the speed of the searcher. There are two possible options: match the term comparator to the nominal transfer rate of the disk, or match it to the effective transfer rate. In either case, the same performance will result, since it is based on how many characters come from the disk in a unit of time. Matching to the effective transfer rate permits a lower comparator speed at the expense of buffer memory. How much lower depends on the seek characteristics for the typical or worst-case queries. Matching to the nominal transfer rate was used for the prototype PFSA implementation, because of the (formerly) high cost of memory for buffers and to avoid determining what the seek characteristics were.

While it was clear a decade ago that available microprocessors could not keep up with the disk for a multi-term match, is it still true today? Looking at the inner loop for the various grep programs, egrep appears to have the tightest loop for complex multi-term searches. On a Sun SPARC, an optimized version takes 10 instructions and 13 cycles. This means that the inner loop searching rate for a 25 MHz SPARCstation-1 is about 1.92 MBytes per second, approximately the nominal disk rate of 1.875 MBytes per second for a 15 MHz drive. A 40 MHz SPARC will run at over 3 MBytes per second.

Obviously, though, the processor must do more than handle the inner loop of the term matching. It must also do file control, query compilation, and query resolution. On a test database of 7 documents totalling about 220,000 characters (and with the files buffered in memory), for a SPARCstation-1 egrep processed 550 KBytes per second when searching for a single term, and 250 KBytes per second for a fifteen term search. So, while a current microprocessor can keep up with the disk in its inner loop, when all aspects of the search are considered, it is still not fast enough.

But even as microprocessors become faster, they may still not be a good choice for handling the basic term matching operation. To achieve a high MIPS rate, the program for the microprocessor must be stored in a high speed memory. For a 25 MHz machine, it must have a cycle time of less than 40 ns. This often means the use of a cache memory, although for small programs a fast SRAM could be used. Such a microprocessor and memory can cost over $500 for parts alone.

If we look at the inner loop for egrep, we find that it implements a finite state recognizer, using two arrays. The first one is indexed by the current state and the input character and gives the next state. The second indicates whether a match has been found. For 256 states (about 32 nominal search terms), a similar FSA can be implemented using 9 64 Kbit RAMs and an eight bit holding register, for a cost of about $30. A few more gates will allow the state table to be loaded via DMA or as addressable memory by the microprocessor. Such an inexpensive matcher would substantially increase the performance of the system, allowing the microprocessor to concentrate on tasks difficult to implement in hardwired logic, such as query compilation, disk control, or query resolution. A lower priced microprocessor (say, a 286 which costs about $25) and slower memory can be used for these tasks, further lowering the searcher costs.

For a system handling more terms (say, 300 terms to accommodate large searches generated by a thesaurus), about 2400 states would be necessary. For a system of this size, techniques that result in a smaller memory requirement for the state table, such as the PFSA or mapping the eight-bit characters from the disk to six bits, will continue to be useful to lower the implementation cost of the hardware-based term comparator. Search processors attached to each disk drive of the database, operating in parallel, used in conjunction with an index or other document surrogate to reduce the required search, continue to be a cost-effective, high-performance means of implementing a very large text database. Special purpose logic allows the use of more reasonably priced microprocessors and memories, rather than using a very fast processor just to perform a simple match at disk speeds.

References

1. R L Haskin and L A Hollaar, "Operational Characteristics of a Hardware-based Pattern Matcher", *ACM Trans. Database Systems*, Vol. 8, No. 1, March 1983.

2. L A Hollaar and R L Haskin, "Method and System for Matching Encoded Characters", U. S. Patent 4,450,520.

3. D C Roberts, "A Specialized Computer Architecture for Text Retrieval", *Fourth Workshop on Computer Architecture for Non-Numeric Processing*, August 1978, pp. 51-59.

4. K-I Yu, S-P Hsu, R E Heiss Jr, and L Z Hasiuk, "Pipelined for Speed: The Fast Data Finder System", *Quest, Technology at TRW*, Vol. 9, No. 2, Winter 1986/1987, pp. 4-19.

5. "Look-up Chips Check Entire Data Base Fast", *Electronics*, Vol. 54, No. 22, November 3 1981, pp. 42-46.

LIBRARY RESEARCH ACTIVITIES AT OCLC ONLINE COMPUTER LIBRARY CENTER

Michael McGill
Vice President, Planning
OCLC

Martin Dillon
Director of Research
OCLC

Introduction

Libraries, considered by many to be among the earliest data management organizations, have vast data requirements. Their "data" is language based and inherently complex; it encompasses the world's knowledge and library users are potentially the population of the world. Thus, the data and processing requirements for the library environment are particularly challenging.

OCLC the Online Computer Library Center was founded in 1967 to meet one aspect of the library's needs--the need to catalog books. Every library must create a cataloging record for each and every item that it acquires. The state of the art prior to OCLC was to individually catalog each item as it was received by a library. This is an expensive, labor intensive, and intellectually challenging task. OCLC brought into existence the concept of shared cataloging. The idea is simple: a book is cataloged only once and entered into OCLC's database. Each succeeding library that acquires this same item looks up the record from the database and uses that record with the addition of its local information. This greatly reduces the cost and significantly increases both the speed and the accuracy of the catalog that they create.

OCLC's system came online in 1971 and was available only to the member libraries within the state of Ohio. Today OCLC's database has over 20 million records and is used across the United States and in 26 other countries. The OCLC system has over 10,000 member libraries and processes about 80 transactions every second. Each transaction may search, edit, update, or delete an entity from the database. These activities occur over OCLC's dedicated telecommunications network or via several dial access mechanisms.

The database is used by librarians as a means of cataloging the materials acquired for their library. However, a database of this magnitude has obvious value well beyond the cataloging process. In fact, the demand for reference use, by library staff and patrons, has just resulted in the creation of OCLC's EPIC Service. This will give full subject access to the database to answer patron questions and for direct patron use wherever reasonable.

The OCLC Office of Research

The libraries of the world are primarily service organizations with tremendous research needs but very few research opportunities or facilities. The creation of OCLC leveraged the resources from these institutions and enabled the establishment of a research group that would focus on the needs of libraries and particularly on problems in evolving systems to manage library information resources.

OCLC research is focused on discovering practical solutions to the challenges that face the producers, providers, and users of the world's information. From published research findings to prototype systems, OCLC research finds its fruition in the products and services used by librarians, information professionals, and information users. An overview of its current research emphasis is provided in Figure 1. The block of activities to the left in the figure focus on document analysis and database creation; the central block explore innovative database reorganization and retrieval activities; over on the right, the primary focus is on interface problems. Although all projects deal at least tangentially with document retrieval, Projects ADAPT, DIADEM, Mercury and CORE are of particular interest.

Overview of Projects

AUTOMATED DOCUMENT PROCESSING

Project ADAPT

Despite the increasing availability of information in machine readable form, the bulk of the world's knowledge remains recorded on paper. Conversion of this material to digital form is a high priority for the

next decade. At present, conversion requires either re-keying or optical character recognition (OCR) of scanned bit images. Because of the relatively high error rates in OCR, which require human operators to correct, both approaches are costly. Project ADAPT (Automated Document Architecture Processing and Tagging) is conducting research into automating the conversion of paper documents into electronic form for use in advanced information systems. Its focus is on both improving the results of OCR and on analyzing the structure of the resulting texts.[1]

Project ADAPT prototypes techniques to increase the efficiency of OCR conversion and to structure the resulting information in a form useful for information retrieval and display systems.

The approach to the problem is organized into three phases:

1. Image preprocessing
2. Optical character recognition
3. Postprocessing and tagging of semantic, syntactic, and document layout information

Figure 2 sketches these phases. The first phase involves the preprocessing of the document as an image in order to automatically distinguish among text objects, graphic objects of various types, and extraneous noise. OCR performance benefits from separation of these object classes. Graphic objects can then be processed and tagged by appropriate subsystems for subsequent linking to appropriate text passages.

OCR processing of text images results in ASCII text and associated attributes and layout information (e.g., location of the text on the page, information about font size). This information provides a basis for further processing and tagging of text.

Postprocessing of text spans a wide range of potentially useful activities. At the simplest level, application of spelling checkers optimized to understand the error patterns of OCR technology will result in improved OCR conversion. More advanced syntactic analysis is likely to result in further improvement of imperfect OCR output.

Even perfect OCR conversion (an unlikely result) does not provide a machine-readable database of optimal usefulness. Identification and descriptive tagging of bibliographic elements (e.g., title, author, publishing agent) and document structures (i.e., abstracts, text, graphs, tables, and the like) are needed to enhance access to these elemnts and facilitate handling in electronic information systems.

Project ADAPT will also explore the application of rule-based systems to automatically add SGML (Standard Generalized Markup Language) tagging to OCR-converted documents. SGML is becoming the de facto standard for the descriptive markup of documents. It is anticipated that retrieval will be enhanced by such markup, but an additional benefit is the ability to use descriptive markup for driving display devices.

The goal of completely automated document conversion and markup is tantalizing but unrealistic. Nonetheless, the economic value of certain documents is such that conversion with human intervention is now justified; facilitating this conversion by partial automation will bring more difficult document collections into the range of economic conversion. Incremental advances in these techniques will gradually push the boundary of economic conversion into new areas. The goal of Project ADAPT is to build a framework for exploring and implementing such advances.

Neural Nets

One aspect of the ADAPT process is distinguishing in the scanned page between text proper and other objects on the page such as graphics, tables or images. The process is called segmentation and is described by Wong, Casey, and Wahl (1982).[2] In a controlled experiment using images of catalog cards, the ADAPT group evaluated neural net technology as a means of differentiating between text and noise in the images. Two classification techniques were compared, both using 14 features extracted from the original card image. In the first, a classification tree is built using the Classification and Regression Trees (CART) technique (Breiman et al. 1984).[3] To classify an object, the classification tree examines one feature at a time and either classifies the object as noise or text or chooses another feature to look at. Eventually the object falls into one of the two categories.

The second classification technique uses a back-propagation neural network that learns to classify the objects by repetitive exposure to a training set. Back-propagation refers to a learning rule that specifies

how connection weight information is propagated throughout the network (Rumelhart and McClelland 1986).[4] The network configuration consisted of 14 input nodes (one for each of the 14 statistical features of the document image blocks), 29 hidden nodes, and a single output node indicating whether the block was a text item or extraneous noise. The results indicated in Figure 3 required 1,750 training passes during which the network was exposed to 1,808 text objects and 2,176 noise objects. Total misclassification error was 2.3% for CART and 2.1% for the neural network.[5]

Graph-Text

The end product of the ADAPT process is best exemplified by the attributes of the Graph-Text System, the output of an earlier project at OCLC. The Graph-Text project goal was to develop a useful, complete and inexpensive system to deliver technical articles and reference works to both working scientists and general library users.[6]

Early in the project, in work with the American Chemical Society, the focus was on delivering chemical journal articles using the Kirk-Othmer Encyclopedia of Chemical Technology[7] as a test database. The chemical literature offers a variety of challenges for electronic document delivery. Chemical articles include a variety of unusual characters and symbols, complex equations, large tables, and graphics ranging from line drawings to full-color photographs. The retrieval and display system was designed for an IBM AT with a CD-ROM drive for the database and a Wyse 700 high-resolution display running under the IBM PC DOS operating system. Figure 4 is an example of a Graph-Text display and shows a page of the Kirk-Othmer Encyclopedia of Chemical Technology overlaid by the window for the retrieval engine interface.

The basic approach to the document conversion process, as represented in Figure 5, is to write programs to convert the source text into SGML, then index the SGML file for full-text retrieval and convert the file into a typesetting language for formatting. This process involves modifying the ASCII text and then parsing and converting the text into SGML. Graphics are scanned from the original pages and coordinated with the text by a graphics control file generated from the ASCII text. The Graph-Text project uses TeX for the typesetting language. The files used to generate the enclopedia article display files are standard TeX output files with the SGML tags embedded in them, providing a way to mark equations, tables, text and graphics placement. This method of marking elements of the document provides some simple hypertext capabilities such as using figure references to move around within the articles.

FIELD EXPERIMENTS

Two important efforts at OCLC involve collaborative projects with universities. Project Mercury at Carnegie Mellon University is a concerted effort to create an electronic library for a subset of its constituency. Its purpose is to study questions and problems related to establishing and administering such a library with a primary emphasis on distributed systems and on economic issues.[8] CORE (Chemical Online Retrieval Experiment) is being conducted at the Albert R. Mann Library at Cornell University. Its aim is to deliver to library users a collection of the full text of journals of the American Chemical Society in order to study problems associated with the use of computer-supplied full text for research.[9]

Project Mercury

The goal of the Mercury project is to build a prototype electronic library. This library will not only serve as a testbed for research in the areas of information retrieval and economics of electronic publishing, but it will also offer a range of real services to academic users.

The electronic library is not an electronic book nor even an electronic bookshelf. The full vision is as far from a single CD-ROM as the modern library is from Gutenberg's first printed book. The electronic library is not only much larger but it is a genuine library bringing different materials to all scholars on campus in a way that integrates well with their working environment. This view of a broad service to an entire community is a massive jump in scale of numbers, geographic distribution, subject coverage, and range of resources. Thousands of information producers will have to provide products that can be delivered coherently through a single interface. Despite technical advances, the combination of these factors make the accomplishment of the electronic library a monumental undertaking with many complex interrelationships.

Mercury has three purposes:

- Demonstration. Mercury will demonstrate that a large scale distributed library can be built with today's technology.

- o Laboratory. Mercury will be a laboratory to be used for a wide array of studies in handling information electronically.

- o Library. Mercury will be a real library for computer science research. It will contain a large percentage of the information that computer scientists use and will deliver it to them at their desks.

CORE

A portion of the X-memex system developed at OCLC will be used as a front-end for the CORE project early in 1990 at Cornell University. CORE is a joint project of Cornell University, OCLC, Bellcore, and the American Chemical Society which will provide a comprehensive electronic library of chemical journals to approximately 90 chemistry scholars in various Cornell departments. Librarians, as information intermediaries, will also participate as users. The electronic library will support both browsing and searching modes of access.

Conclusion

OCLC's mission--to further ease of access to and use of the ever-expanding body of worldwide scientific, literary and educational knowledge and information--requires an ongoing commitment to research. Much of this research is focused on the future needs and expectations of libraries and their patrons. The users of information systems are increasingly sophisticated and require increased and improved access to larger and larger quantities of textual, graphic, and audio information from larger and larger stores of information. The researchers at OCLC are methodically laying the groundwork to provide this access.

Mercury is typical of OCLC's interest in pulling together the result of many lines of research into a prototypic experiment that is providing a service to the Carnegie Mellon University campus as well as establishing a unique laboratory setting for the development and evaluation of information systems of the future.

References

[1]Weibel, Stuart, John Handley, and Charles Huff. 1989. "Automated Document Structure Architecture Processing and Tagging." In Proceedings of the Conference on Application of Scanning Methodologies in Libraries, Nov. 17-18, 1988, National Agricultural Library, Beltsville, Maryland, ed. Donald L. Blamberg, Carol L. Dowling, Claudia V. Weston, 3-15. Beltsville, Maryland: National Agricultural Library.

[2]Wong, K.Y., R.G. Casey, and F.M. Wahl. 1982. "Document Analysis System." IBM Journal of Research and Development. 26:647-56.

[3]Breiman, L., et al. 1984. Classification and Regression Trees. Belmont, Calif.: Wadsworth Intl. Group.

[4]Rumelhart, D.E., and J.L. McClelland. 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1. Cambridge, Mass.: MIT Press.

[5]Weibel, Stuart. 1989. "Applying Neural Networks to Classification Problems." In Annual Review of OCLC Research, July 1988-June 1989. Dublin, OH: OCLC Online Computer Library Center, Inc.

[6]Hickey, Thomas B. 1989. "Using SGML and TeX for an Interactive Chemical Encyclopedia." In National Online Meeting Proceedings-1989, New York, May 9-11, 1989, compiled by C. Nixon and L. Padgett, 187-195. Medford, NJ: Learned Information, Inc.

[7]Grayson, Martin, ed. 1978. Kirk-Othmer Encyclopedia of Chemical Technology. 24 vols. New York: John Wiley & Sons.

[8]Kibbey, Mark, and Nancy H. Evans. 1989. "The Network Is the Library." EDUCOM Review 24(3): 15-20 (Fall, 1989).

[9]Lindeman, Martha J. 1989. "Design of Interfaces and Databases for Electronic Media (DIADEM)." In Annaul Review of OCLC Research, July 1988-June 1989. Dublin, OH: OCLC Online Computer Library Center, Inc.

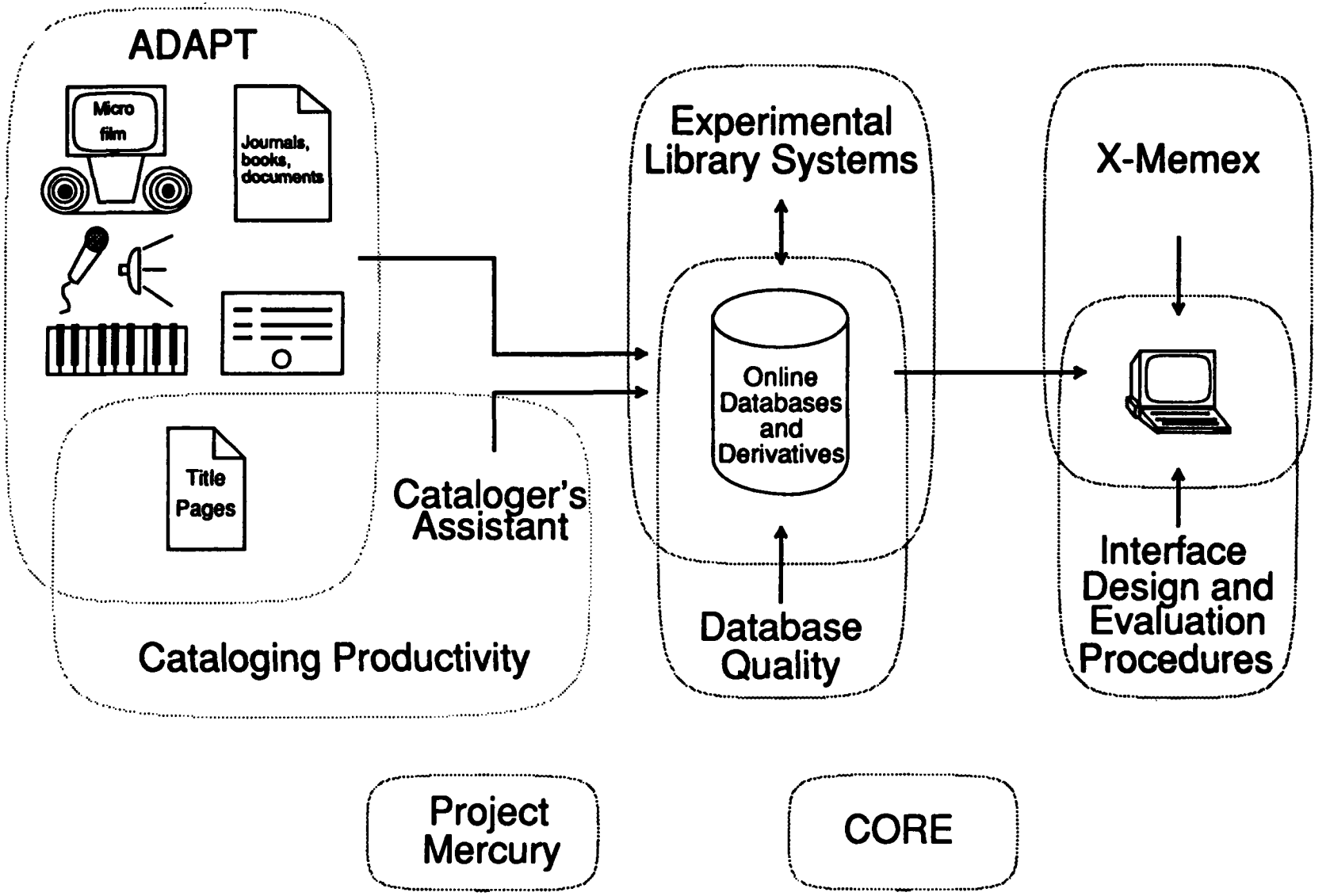


Figure 1: OCLC Office of Research Project Overview

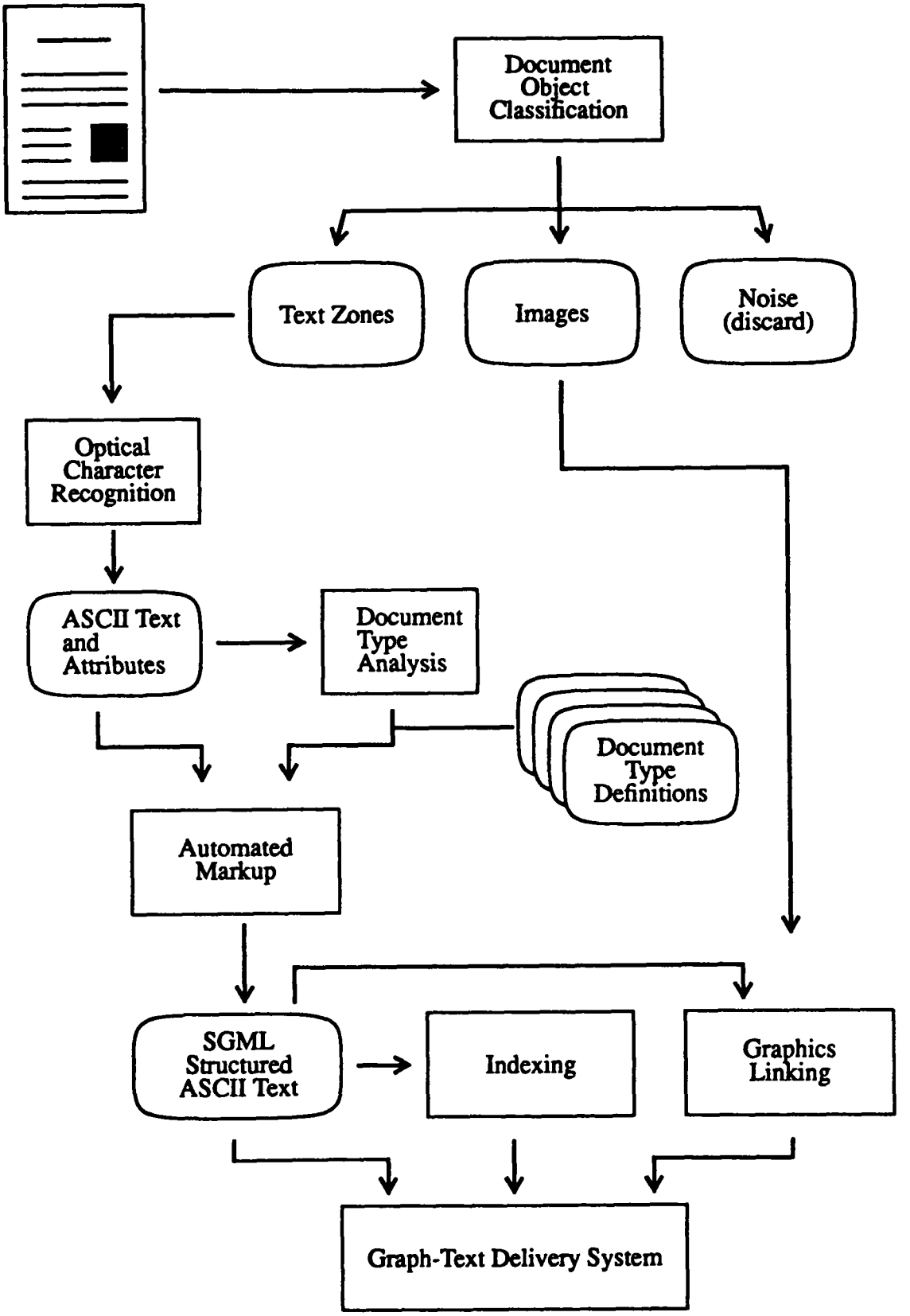


Figure 2: Project ADAPT

CART

Predicted	Actual: text	Actual: noise
text	1792 (99.1%)	30 (1.4%)
noise	16 (0.9%)	2146 (98.6%)

NEURAL NET

Predicted	Actual: text	Actual: noise
text	1798 (99.5%)	35 (1.6%)
noise	10 (0.5%)	2141 (98.4%)

Figure 3: Segment Discrimination: Classification of text and noise segments using CART methodology and back-propagation neural network

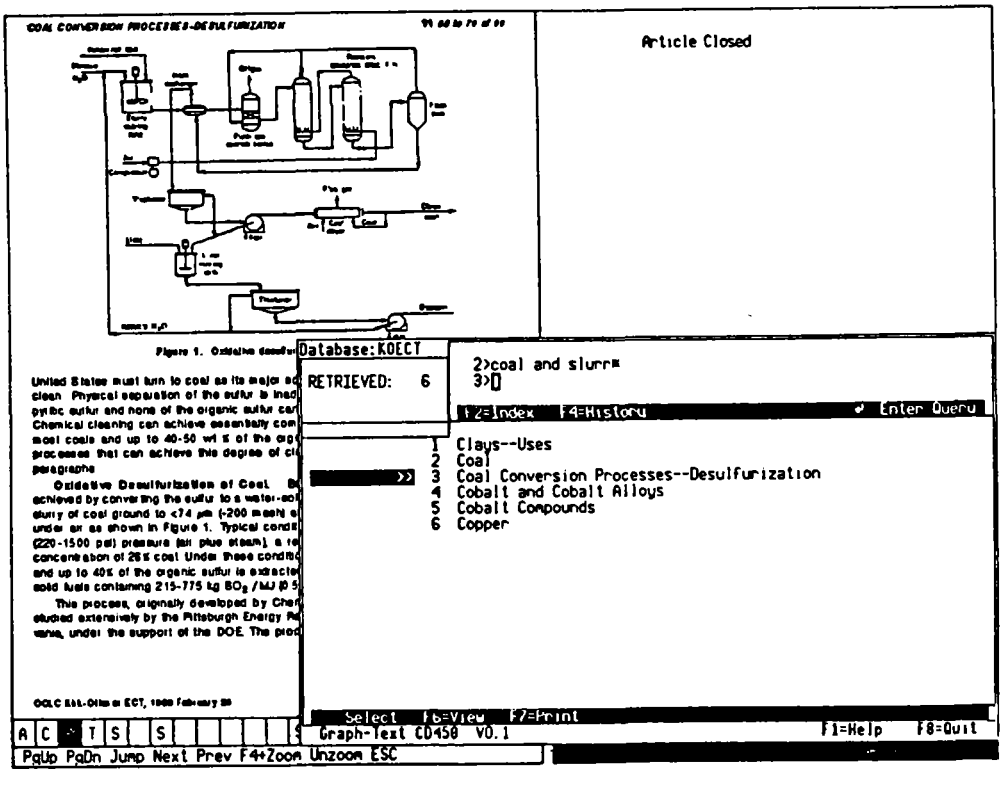


Figure 4: Graph-Text Display

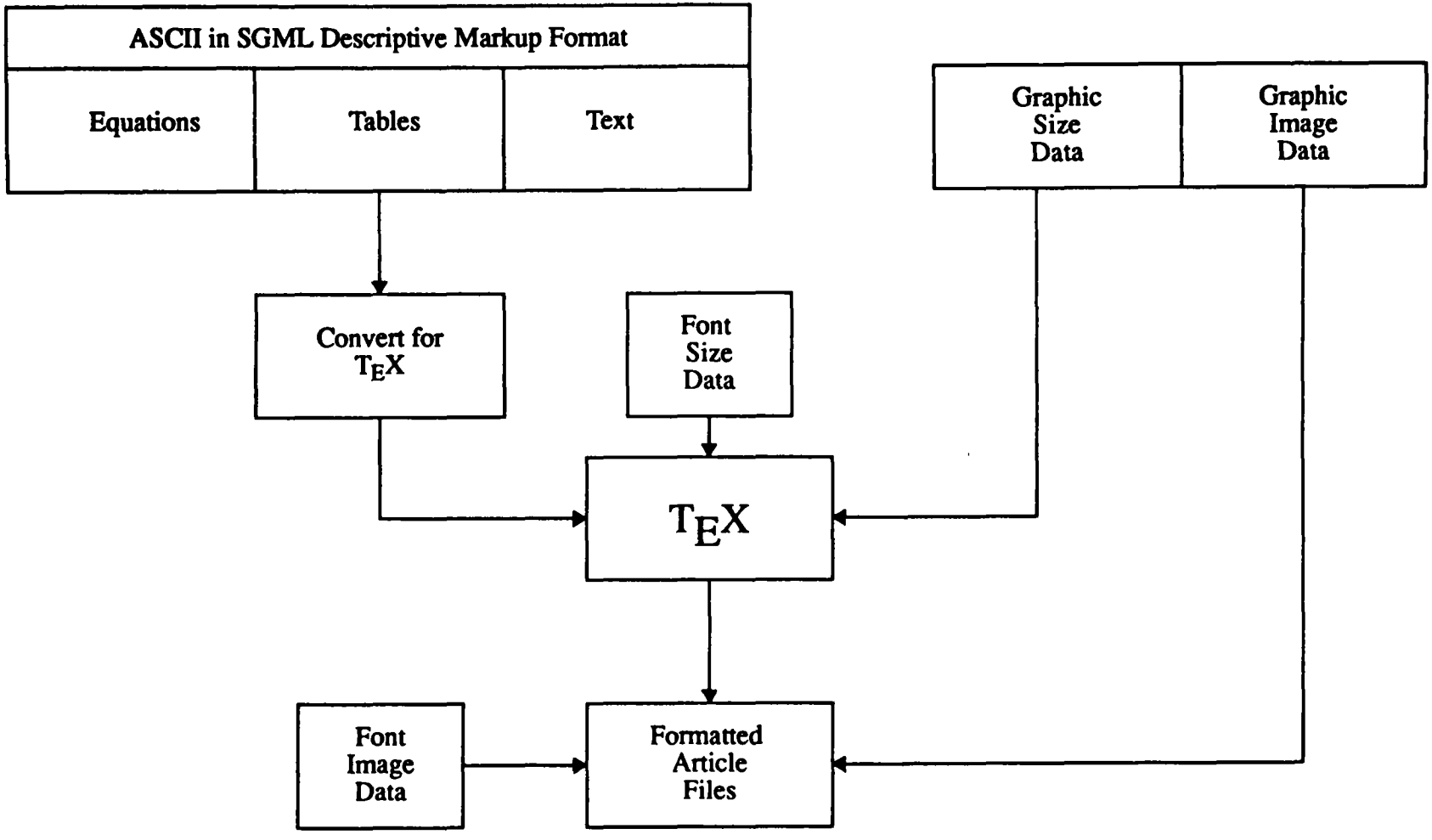


Figure 5: Graph-Text Document Conversion Process

Integration of Text Search with ORION

Wan Lik Lee and Darrell Woelk

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78759

ABSTRACT

The ORION object-oriented database system supports the capture, storage and presentation of text, images and audio information. This support is in the form of an extensible object-oriented framework that includes definitions of classes and a message passing protocol for capture devices, presentation devices, and captured multimedia objects. This paper describes the portion of the framework that supports the capture, storage and presentation of textual objects. Further, it describes an implementation of text search capability within this framework that is compatible with the general ORION query processing functionality. The text search algorithm which we have implemented supports Boolean predicates on string patterns specified as regular expressions.

1. Introduction

The ORION object-oriented database system, developed at MCC, has directly implemented the object-oriented paradigm adding persistence and shareability to objects through transaction support. Advanced functions in ORION include versions and change notification, composite objects, dynamic schema evolution, transaction management, queries, and multimedia data management [KIM90]. In [WOEL87], we described the ORION support for the capture, storage and presentation of images and audio information. This support is in the form of an extensible object-oriented framework that includes definitions of classes and a message passing protocol for capture devices, presentation devices and captured multimedia objects. This paper describes an extension to the framework to support the capture, storage, and presentation of textual objects. Further, it describes an implementation of text search capability within the framework that is compatible with the general ORION query processing functionality. The text search algorithm which we have implemented accepts Boolean expressions as search patterns. Each search pattern can be as simple as a string of characters, or as complicated as a general regular expression.

2. ORION Interface for Text Capture, Storage, and Presentation

This section will review the ORION multimedia framework and describe how the framework has been extended to support capture, storage, and presentation of textual objects. Section 3 will later describe how text search is invoked on these objects.

2.1 Class Definitions

Figures 1, 2, and 3 illustrate the ORION classes for presentation devices, capture devices, and captured objects. The shaded classes are classes that have been added to ORION for the support of textual objects.

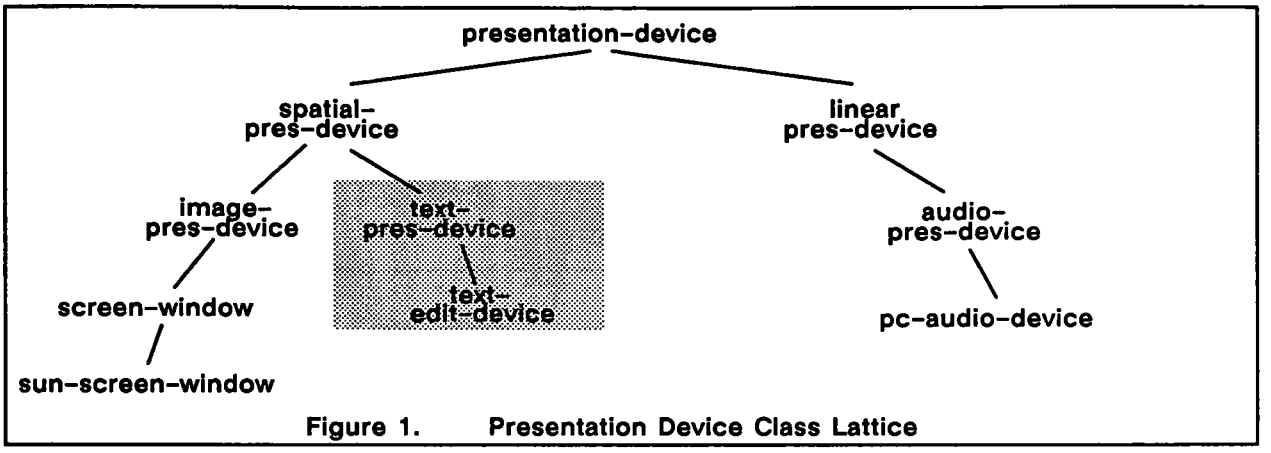


Figure 1. Presentation Device Class Lattice

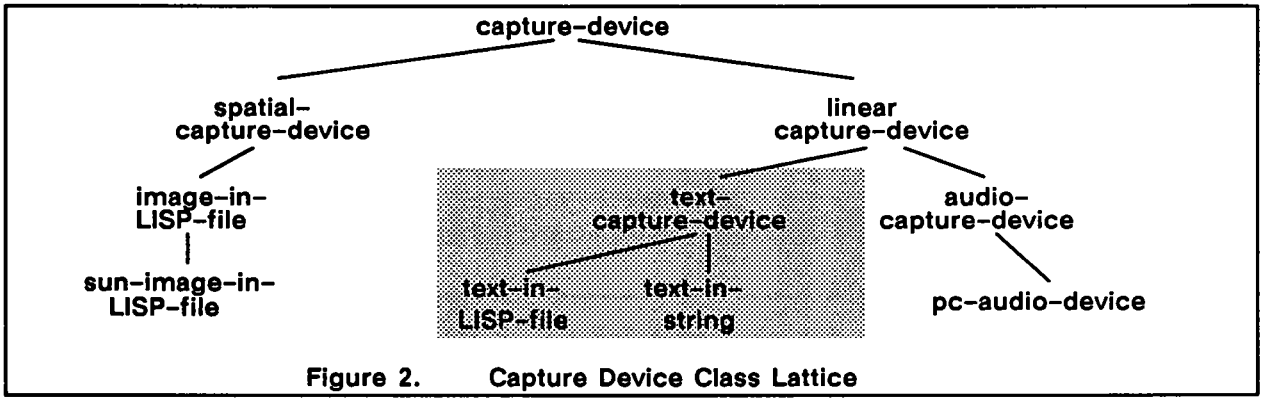


Figure 2. Capture Device Class Lattice

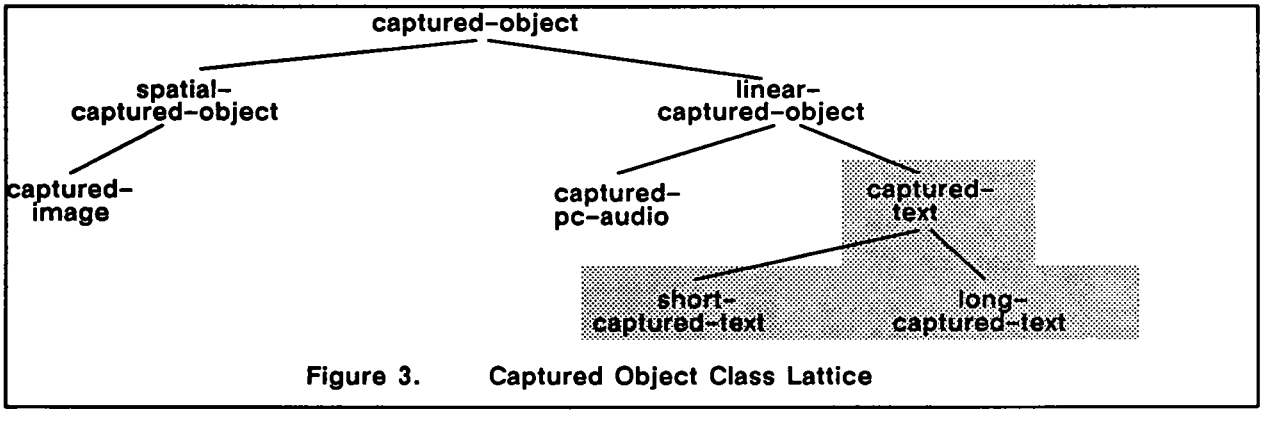


Figure 3. Captured Object Class Lattice

The class lattice in Figure 1 represents the presentation devices available on the system. An *instance* of one of these classes, however, represents more than just the identity of a physical presentation device. Each instance also has attributes that further specify, for example, where on the device a multimedia object is to be presented and what portion of a multimedia object is to be presented. These pre-defined **presentation-device** instances can be stored in the database and used for presenting the same multimedia object using different presentation formats. Methods associated with a class are used to initialize parameters of a presentation device and initiate the presentation process.

Two classes have been added to Figure 1 for ORION support of textual objects. Instances of the **text-edit-device** class are used to retrieve text from a **captured-text** object, write the text to a Lisp file, and invoke the Common Lisp [STEE84] editor function to edit this file.

The class lattice in Figure 2 represents the capture devices available on the system. Once again, an instance of one of these classes represents more than just the identity of a physical capture device. Each instance may also have attributes that further specify camera settings or the bit rate for digitizing audio information. These pre-defined **capture-device** instances can be stored in the database and used for capturing a multimedia object using different capture formats. Methods associated with a class are used to initialize parameters of a capture device and initiate the capture process.

Three classes have been added to Figure 2 for ORION support of textual objects. Instances of the **text-in-LISP-file** class are used to capture text from a Lisp file and to store the text in a **captured-text** object. Instances of the **text-in-string** class are used to capture text from a Lisp string and to store the text in a **captured-text** object.

The class lattice in Figure 3 describes the types of multimedia objects that can be stored in the ORION database system. These classes have attributes and methods that describe the size and format of the multimedia object in proper units of measurement.

Three classes have been added to Figure 3 for ORION support of textual objects. Instances of the **long-captured-text** class and the **short-captured-text** class are used to store textual information in two distinct formats. An instance of the **long-captured-text** class stores text in the long data format described in [WOEL87]. The long data format is optimized for storing and retrieving large data objects and should be used if an entire document is to be stored as a single ORION object. An instance of the **short-captured-text** class stores text as a normal Lisp string and is optimized for short text objects, such as chapter names and abstracts.

2.2 Message Passing Protocol for Capture and Presentation of Text

Textual information can be captured and stored in the ORION database using the following steps. First, an instance of one of the **captured-text** subclasses is created using either of the following messages:

```
(make (class-object 'short-captured-text))
(make (class-object 'long-captured-text))
```

The **make** class method [KIM88] for each of these classes creates a new instance and also initializes the instance to prepare it for storing text. The identity of the newly created instance is returned. This instance is then passed as an argument to an instance of one of the **text-capture-device** subclasses, either the **text-in-LISP-file** class or the **text-in-string** class, using one of the following messages:

```
(capture text-in-lisp-file-instance  captured-text-instance  lisp-filename
         [:append t-or-nil])
(capture text-in-string-instance     captured-text-instance     string
         [:append t-or-nil])
```

The **captured-text** instance (actually either a **short-captured-text** instance or a **long-captured-text** instance) now stores the textual information in the database. If the append keyword argument in this message is t, the new text is appended to any text presently stored in the instance. If it is nil, the old text is overwritten. The default value of the append keyword argument is nil.

The text can be displayed for editing by passing it as an argument to an instance of the **text-edit-device** class using the following **present** message format:

(**present** text-edit-device-instance captured-text-instance lisp-file-name)

The instance of the **text-edit-device** class also stores the lisp-file-name in one of its attributes. If the user modifies the text in the Lisp file, the following message may be sent to the **text-edit-device** instance to cause the updating of the **captured-text** instance:

(**capture** text-edit-device-instance captured-text-instance [lisp-file-name])

If the optional lisp-file-name argument is not provided, the Lisp file name passed as an argument in the previous **present** message is used.

3. ORION Interface for Text Search

Text search capability has been implemented as methods defined for each of the subclasses of the **captured-text** class. This implementation allows text search strategies to be optimized based on the storage format of the text. It also allows the system designer to extend the system by adding other text search strategies.

Each subclass of the **captured-text** class must define an **includesp** method that responds to a message of the following format:

(**includesp** captured-text-instance pattern)

This method will return true if the specified pattern is found in the text stored in the **captured-text** instance; otherwise, it will return nil. The **includesp** methods for the **short-captured-text** class and the **long-captured-text** class optimize the search based on the storage format of the text for each class.

The pattern parameter described above consists of Boolean combinations of regular expressions using the Boolean operators OR, AND and NOT [FALO85]. For example, the following message would return t only if both the string "apples" and the string "oranges" were found in the text.

(**includesp** captured-text-instance '(and "apples" "oranges"))

There are two types of regular expressions which can be specified. The first type is the most general regular expression for text search. The syntax for this regular expression includes the unary transitive closure operator "*" and the union operator "+". The wild card character is a ".". For example, ("a" (+ 1 2) "b") is the regular expression denoting all strings beginning with "a" followed by either 1 or 2 and ending in a "b". A second example is the regular expression ("a" (* "-") ">b") that recognizes "a>b" or "a->b" or "a-->b" or "a--->b", ad infinitum.

The second type of regular expression can be used for searching for *words* where the full generality of the first type of regular expression is not needed. A word is defined as contiguous characters bounded by any non-alphabetic characters on both sides. It allows wild card characters where "*" stands for any number of occurrence of an alphabetic character and "." stands for exactly one occurrence of an alphabetic character. The "*" and the "." can be juxtaposed in any manner. An example of this simpler notation would be (word "apples") which would match the word "apples". Another example would be (word "a..b"), which would match any 4-letter word beginning with "a" and ending with "b". To match an arbitrary number (including zero) of alphabetic characters between "a" and "b", the expression (word "a*b") would be used. The expression (word "*ought*") would match the words "thought", "thoughtful", and "ought".

4. Integration of Text Search with ORION Query Processing

Thus far, we have described how a single **captured-text** instance can be searched for a text pattern using the **includesp** message. We will now describe, using the following example, how this capability has been integrated with the ORION query processing functionality. Figure 4 illustrates a database schema for a simple document. Note that the arrows in Figure 4 do not represent the class/subclass relationship but rather indicate that one class is the domain of an attribute of another class. For example, the **document** class has three attributes: **title, text**, and **page-count**. The **title,** and **text** attributes have the **captured-text** class (or one of its subclasses) as a domain. The **page-count** attribute has the Common Lisp type **integer** as its domain.

Once we have populated the database with document instances, we can execute the following query using the syntax described in [KIM90]:

```
(select   'document   '(> page-count 10))
```

This query will return all instances of the **document** class that have an integer greater than 10 stored in the **page-count** attribute. In a like manner, we can execute the following query that will return all instances of the **document** class that have more than 10 pages and that contain the strings "database" and "multimedia" in the text:

```
(select   'document   '(and (includesp text '(and "database" "multimedia"))
                            (> page-count 10)))
```

When the query processor executes this query, it will send the **includesp** message to the object that is stored in the **text** attribute of each **document** instance. The expression (and "database" "multimedia") will be passed as an argument with the message. The object receiving this message will be an instance of one of the subclasses of the **captured-text** class which will execute its **includesp** method and return either t or nil. Thus, the **includesp** message is treated as any other system-defined comparator (such as =, >, etc.).

Figure 5 illustrates a more complex schema for representing a document as an aggregate object. The body of the document has now been divided into chapters. The domain of the **chapters** attribute of the **document** class is a *set of* instances of the **chapter** class. The following query will return all instances of the **document** class that have greater than 10 pages and that have the words "database" and "multimedia" in at least one chapter:

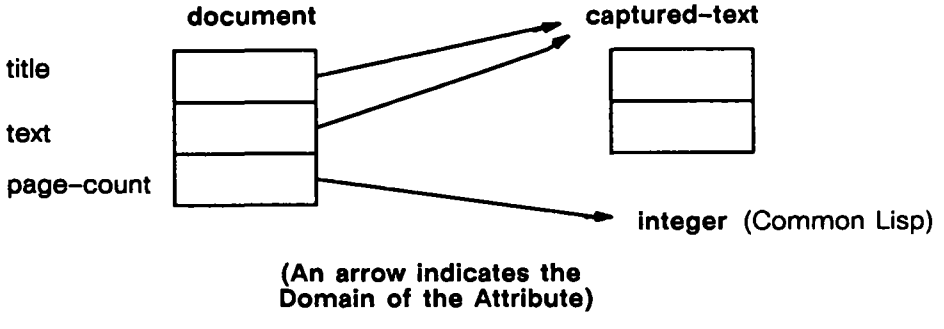


Figure 4. Simple Document Example

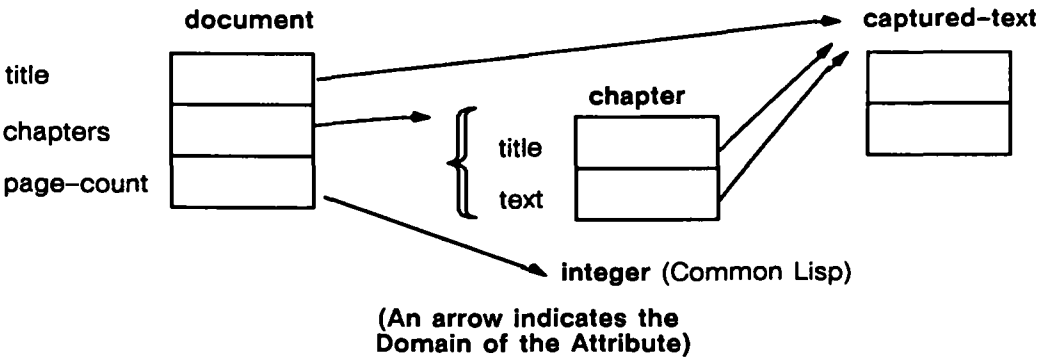


Figure 5. More Complex Document Example

```
(select 'document
        '(and (> page-count 10)
              (path (some chapters) (includesp text '(and "database" "multimedia")))))
```

The ORION query processor recognizes that the includesp operation is slow compared to such comparators as = and <. It rearranges the query for execution so that page-count comparison is always executed first.

5. Text Search Algorithm

Section 3 described the two types of regular expressions that can be used as patterns; a general type of regular expression and a more limited type of regular expression using word boundaries. These two types are distinguished for reasons of efficiency and expressibility.

For the most general case of regular expressions, the approach described in [HOPC79] is followed. First a transition table is created from the regular expression in order to automate the pattern matching process. There are four steps in the creation of the transition table.

1) The regular expression is converted into a nondeterministic finite automaton (NFA).
2) The NFA is then transformed into a deterministic finite automaton (DFA).
3) A minimization procedure is applied to identify and condense equivalence states in order to cut down the size of the DFA.
4) The minimized DFA is encoded as a 2-dimensional transition table indexed by a state number and a character.

For the more limited type of regular expression using word boundaries, each word can be converted directly into a transition table without the intermediate transformations.

The transition table is used to make transitions according to the current state and the next input character from the text. The matching process can terminate in one of two ways. If a final acceptance state is reached, the search succeeds. If the text is exhausted without reaching an acceptance state, the search fails. In either case, only one scan is needed to determine if the text contains a string pattern specified by the regular expression. One major weakness of this approach is that the number of states of the automaton may be exponential on the size of the regular expression. However, the typical regular expressions for a text search query are not expected to be complex.

Boolean combinations of regular expressions are handled using a control structure in addition to the transition tables. First, multiple transitions tables, each corresponding to a regular expression in the query, are applied to the text according to the order determined by the Boolean operators. Following this, the Boolean operators are applied. A potential drawback of this scheme is that each text object may need to be scanned multiple times during the evaluation of a text search query. The potentially excessive I/O cost of this technique is avoided by interleaving the scans on a page-by-page basis. Each disk page containing text is fetched (only once) and fed through the transition tables one at a time. An additional data structure is maintained to keep track of the intermediate results (transition states) associated with the regular expressions.

Alternatively, it would be possible to merge all the transition tables into a single table. This would only require a single pass through each text object. This approach was ruled out because of its computational complexity and the combinatorial growth in the size of the transition table.

6. Summary

This paper has described how textual objects are supported in ORION within the framework of multimedia objects. In particular, it describes a text search capability that has been incorporated into ORION and integrated with the ORION query processing functionality. Internally, the text search algorithm is sufficiently flexible to handle a very general form of text search queries which contain Boolean combinations of regular expressions. Externally, the interface for text search is fully compatible with the existing multimedia framework and query processing. Implemented as methods of the **captured-text** class, the text search capability and strategies can easily be extended by the system designer.

Acknowledgments

The authors would like to acknowledge the contributions of Hong-Tai Chou to the design of the text search algorithm for ORION.

REFERENCES

[FALO85] Faloutsos, C., "Access Methods for Text," *ACM Computing Surveys*, Vol. 17, No. 1, March 1985.

[HOPC79] Hopcroft, J.E. and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 1979.

[KIM88] Kim, W., N. Ballou, H-T. Chou, J. Garza and D. Woelk, "Integrating an Object-Oriented Programming System with a Database System", *Proc. Object-Oriented Programming Systems, Languages, and Applications Conference,* September, 1988, pp. 142-152.

[KIM90] Kim, W., J. Garza, N. Ballou and D. Woelk, "Architecture of the ORION Next-Generation Database System", to appear in IEEE Transactions on Knowledge and Data Engineering, March, 1990.

[STEE84] Steele, G. Jr., Scott E. Fahlman, Richard P. Gabriel, David A. Moon and Daniel L. Weinreb, "Common Lisp," *Digital Press*, 1984.

[WOEL87] Woelk, D. and W. Kim, "Multimedia Information Management in an Object-Oriented Database System," *Proc. 13th Intl Conf. on Very Large Data Bases*, Brighton, England, September 1987, pp. 319-329.

STANDARDS

STANDARDS STATUS REPORT

Standards development in the IEEE Computer Society continues to grow at a breathtaking pace. The Standards Status Report is a comprehensive look at the current status of standards development. The report describes standards projects, summarizes the status of standards projects, includes a directory of working group chairmen and others involved in standards development, lists information on working group meeting dates and more.

To order the Standards Status Report, call the Assistant Director for Standards at (202) 371-0101. The cost is $20.00

DRAFT STANDARDS

Draft standards are available from the IEEE Computer Society. The following is a list of some of the drafts that are available:

P 896.1 – Futurebus +

P 1003 – POSIX

P 802 – Local Area Networks

P 1151 – Modula-2

P 1175 – Standard for Interconnections Among Computing System Engineering Tools

To order draft standards or to obtain price information, contact the Assistant Director for Standards at (202) 371-0101.

IEEE COMPUTER SOCIETY
Technical Committee Membership Application

COMPLETE AND RETURN THIS FORM TO:
IEEE COMPUTER SOCIETY
1730 Massachusetts Ave., NW
Washington, DC 20036-1903

INSTRUCTIONS:
Please print in ink or type, one character per box. INFORMATION OUTSIDE BOXES WILL NOT BE RECORDED. Street addresses are preferable to P.O. boxes for mail delivery. International members are requested to make best use of available space for long addresses.

Last Name | First Name | Initial | Dr./Mr./Mrs./Ms./Miss/Prof.

Company/University/Agency Name | Dept. Mail Stop/Bldg./P.O.Box/Apartment

Check One: ☐ New Application ☐ Information Update

Street Address/P.O. Box | Date: Month | Day | Year

City | State | Postal Code

Country | Office Phone | Home Phone (optional)

E-mail Network | E-mail Address (Mailbox)

Telex Number | IEEE Member/Affiliate Number

I am a member of the Computer Society ☐ Yes ☐ No

The Computer Society shares its mailing lists with other organizations which have information of interest to computer professionals. If you do not wish to be included on those lists, please check here: ☐

If you are presently a member of the IEEE Computer Society, Place an X in the left-side box below corresponding to a TC of which you would like to be a member. If you are not a member of the Computer Society and would like to be a member of a TC, place an X in the right-side box below.

	MEMBER	NON-MEMBER	
01	☐	☐	Computational Medicine
02	☐	☐	Computer Architecture
03	☐	☐	Computer Communications
04	☐	☐	Computer Elements
05	☐	☐	Computer Graphics
06	☐	☐	Computer Languages
07	☐	☐	Computer Packaging
08	☐	☐	Computers in Education
09	☐	☐	Computing and the Handicapped
10	☐	☐	Data Engineering*
11	☐	☐	Design Automation
12	☐	☐	Distributed Processing
13	☐	☐	Fault-Tolerant Computing
14	☐	☐	Mass Storage Systems & Technology
15	☐	☐	Mathematical Foundations of Computing
16	☐	☐	Microprocessors and Microcomputers
17	☐	☐	Microprogramming
18	☐	☐	Multiple-Valued Logic
19	☐	☐	Oceanic Engineering and Technology
20	☐	☐	Office Automation
21	☐	☐	Operating Systems
22	☐	☐	Optical Processing
23	☐	☐	Pattern Analysis and Machine Intelligence
24	☐	☐	Personal Computing
25	☐	☐	Real-Time Systems
26	☐	☐	Robotics
27	☐	☐	Security and Privacy
28	☐	☐	Simulation *
29	☐	☐	Software Engineering
30	☐	☐	Test Technology
31	☐	☐	VLSI
32	☐	☐	Computer and Display Ergonomics
33	☐	☐	Supercomputing

For each committee, please indicate activities of the Technical Committee in which you would like to become involved.

I would like to become involved in the Technical Committee and:

	Conferences	Newsletters	Standards Development	Education
01	☐	☐	☐	☐
02	☐	☐	☐	☐
03	☐	☐	☐	☐
04	☐	☐	☐	☐
05	☐	☐	☐	☐
06	☐	☐	☐	☐
07	☐	☐	☐	☐
08	☐	☐	☐	☐
09	☐	☐	☐	☐
10	☐	☐	☐	☐
11	☐	☐	☐	☐
12	☐	☐	☐	☐
13	☐	☐	☐	☐
14	☐	☐	☐	☐
15	☐	☐	☐	☐
16	☐	☐	☐	☐
17	☐	☐	☐	☐
18	☐	☐	☐	☐
19	☐	☐	☐	☐
20	☐	☐	☐	☐
21	☐	☐	☐	☐
22	☐	☐	☐	☐
23	☐	☐	☐	☐
24	☐	☐	☐	☐
25	☐	☐	☐	☐
26	☐	☐	☐	☐
27	☐	☐	☐	☐
28	☐	☐	☐	☐
29	☐	☐	☐	☐
30	☐	☐	☐	☐
31	☐	☐	☐	☐
32	☐	☐	☐	☐
33	☐	☐	☐	☐

Please note that the Technical Committees on Data Engineering and on Simulation now charge an annual membership fee. The rates are $15 for Computer Society members, and $25 for non-members. Checks should be made out to IEEE Computer Society and sent to the address above. Credit cards are accepted ONLY FROM NON-U.S. MEMBERS!

IEEE Computer Society
1730 Massachusetts Avenue, N W
Washington, DC 20036-1903

Henry F. Korth
University of Texas
Taylor 2124 Dept of CS
Austin, TX 78712
USA