A COMPARISON OF A NETWORK STRUCTURE AND A DATABASE SYSTEM USED FOR DOCUMENT RETRIEVAL

W. Bruce Croft
Thomas J. Parenty

COINS Technical Report 85-09

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

ABSTRACT

Database systems have many advantages for implementing document retrieval systems. One of the main advantages would be the integration of data and text handling in a single information system. However, it has not been clear how much a database implementation would cost in terms of efficiency. In this paper, we compare a database implementation and a standalone implementation of a flexible representation of the content of documents and the associated search strategies. The representation used is a network of document and index term nodes. The comparison shows that certain features of a database system can have a significant effect on the efficiency of the implementation. Despite this, it appears that a database implementation of a sophisticated document retrieval system can be competitive with a standalone implementation.

1.0 INTRODUCTION

The task of a document retrieval system is to retrieve relevant text documents, or references to those documents, in response to users' queries. To improve the effectiveness of these systems, a great deal of research has been done on the indexing and retrieval components [1,2]. Indexing refers to the process of representing the content of the text in the documents, and the retrieval process involves the comparison of queries to document representatives in order to decide which documents should be retrieved. One type of system that has performed well in retrieval experiments [1,3] uses a simple automatic indexing technique that represents a document's content by a set of important word stems (or index terms) from the text together with their frequency of occurrence. The retrieval strategies in this system are based on probabilistic models of document relevance and, for a given query, produce a list of documents ranked in order of their probability of relevance. A variety of retrieval strategies are possible depending on the type of probabilistic retrieval model used.

Document retrieval systems have typically been implemented as standalone systems designed for the efficient storage and retrieval of large numbers of documents. However, both the cost of implementing these special purpose systems and the demands of new applications have led researchers to consider the integration of document retrieval with database systems [4,5,6,7]. Office systems, in particular, contain large amounts of information represented as forms that have both fixed format data fields and variable length textual fields. Implementing a document retrieval system with a database system also provides features such as concurrent access and recovery that either are not available or are available only in rudimentary forms in most standalone document retrieval systems.

One aspect of the integration of the two types of system is the modification of the database system's data model to incorporate the structures and operators needed to handle variable length text fields [7]. Although this aspect is important, another of more vital concern for document retrieval system designers is how the representations and algorithms used by sophisticated document retrieval systems can be implemented in a database system. McCleod's work [4,8] addresses this issue to some extent, however only limited and generally simple search strategies are used. Porter [5] considers the implementation of more sophisticated search strategies with a relational database system but no information on the efficiency of the implementation is provided. In this paper, we shall compare a standalone implementation and a database implementation of a sophisticated document retrieval system in order to answer the following questions.

1. How do the two implementations compare in terms of their overall efficiency? The cost measures used include the number of calculations and the number of disk accesses made during the execution of tasks such as searching and updating. Many of the experimental results for the two implementations are difficult to compare directly and, for this reason, the comparisons made are only approximate.
2. Can a document retrieval system be efficiently implemented as an application on a standard database system, or do fundamental changes have to be made to the data model (as proposed in [6])?
3. How should document retrieval search strategies be implemented in a database system?

4. What aspects of a database system are the most crucial for implementing a document retrieval system?

The document retrieval system that is implemented consists of a general representation of the document contents and a variety of search strategies that use different aspects of the representation. The document contents are represented as a network of document and index term nodes with connections between each type of node. The connections represent relationships between these nodes. For example, a document node is connected to the index term nodes that are used to represent it and to other document nodes that have similar contents. A detailed description of this approach to document retrieval and its advantages are given in the next section.

Whereas most of the previous work on text/database system integration has been done with relational systems, the implementation reported here uses a CODASYL system [9]. This choice was made on the basis of availability. Although the type of database system used has a significant effect on the implementation of the search strategies, many of the observations reported here will apply to any database system.

As mentioned previously, the next section presents a detailed discussion of the features of the network-based document retrieval system. It should be emphasised here that the term "network-based" refers to the network of document and term nodes used to represent document contents rather than to the CODASYL implementation of the document retrieval system. Section 3 outlines the important operations needed for implementing search strategies and for maintenance of the network. In section 4, both the standalone implementation of the network structure and the database implementation are described. This description includes algorithms for constructing, updating and searching the network.

The algorithms differ for the two implementations because they attempt to exploit particular features to enhance efficiency. The fifth section reports and compares the results of experiments with the two implementations. The conclusions section summarizes these results.

2.0 A NETWORK-BASED DOCUMENT RETRIEVAL SYSTEM

Different retrieval strategies can require different types of information about the content of the documents. The aim of the network representation of documents is to capture the majority of this information within a concise framework. The main advantage of this representation is the ability to support a variety of search strategies, including some that would not be possible in conventional environments. Retrieval experiments have provided evidence that being able to choose a search strategy appropriate for a given situation will increase the overall performance relative to a system restricted to a single strategy [10].

At the conceptual level, the proposed representation can be viewed in its simplest form as a network of document and term nodes connected by links that represent associations between them (Figure 1). A weight on each link is used to indicate the strength of the association. The links and the weights can be classified as follows;

a. Document-Term Link: Indicates how important a term is in the representation of a document.

b. Term-Term Link: Indicates how closely two terms are related.

c. Document-Document Link: Indicates the similarity of the content of the documents.

Other types of links, such as document-author and document-citation, could be included in this network in order to extend the range of search strategies available. However, in this paper, we shall concentrate on the representation of document content using index terms and assume that other bibliographic information is available but not used. The contents of the document and term nodes will be discussed in section 4.

The document-term links are the links that will be used most often by the search strategies. These links are bi-directional and can be thought of as having two forms; document-term and term-document. The document-term links identify the terms that represent the content of a document. This information is equivalent to that contained in a conventional serial file of documents. The term-document links identify the documents that are described by a particular term. This is equivalent to an inverted file of the documents. Taken together, these parts of the network can be used to implement many of the

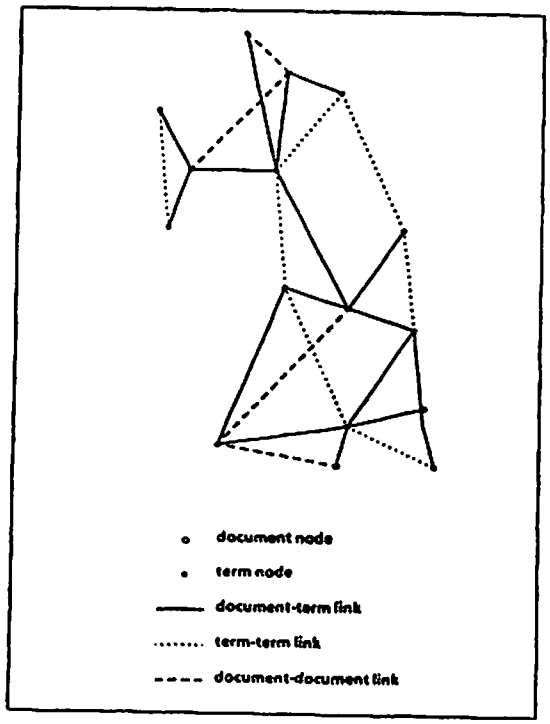


Figure 1: A network organization.

probabilistic strategies mentioned in the literature [1,11]. The particular form of the document-term link weight used here is the within-document frequency of the term.

A search strategy that will be used in the experiments reported in section 5 ranks the documents according to the following score (which is an estimate of their probability of relevance);

$$\Sigma\ w_i x_i \qquad (1)$$

In this score, w_i is a weight for query term i and x_i is a document term (in this case, assumed to be either 1 or 0). One form of the weight w_i that is derived from a probabilistic model [3,12] is $\log p_i(1-q_i)/(1-p_i)q_i$, where p_i is the probability of term i occurring in a relevant document and q_i is the probability of term i occurring in a nonrelevant document. The q_i probabilities can be estimated from the statistics of the entire collection of documents. Estimating p_i is more difficult. Initially, p_i can be assumed to be a constant for the query terms. This gives a weight that is inversely proportional to the collection frequency of the term (also known as the inverse document frequency weight or IDF)[13]. After the user has evaluated some of the retrieved documents, better estimates of p_i can be made.

Other search strategies are based on the assumption that the occurrence of terms is dependent on other terms [14,15,16]. These strategies can make use of the information contained in the term-term links. The association between terms represented by the links can derived statistically or manually using a thesaurus. A method of generating the statistical links is described in the next paragraph.

Document-document links are used to provide the information for a cluster-based search strategy [1,17]. In this strategy, clusters or groups of related documents are ranked in order of their similarity with the query. Many previous experiments have been done with

a single-link cluster hierarchy of documents that can also be represented in the network by a minimum spanning tree of links (MST)[18]. The generation of the MST (for both documents and terms) is essentially an $O(n^2)$ process although it can be made reasonably efficient [19]. The main drawback with the MST is that it is expensive to maintain when new documents are included. A more efficient approach results from experimental evidence that indicates that only the strongest document-document similarities are useful for retrieval [17,20]. Many documents are only weakly connected to each other so their document-document links are not significant and need not be represented. In terms of the single-link hierarchy, this means that only the lowest level (smallest) clusters are required. Therefore, rather than generating the MST as a representation of the document-document links, each document need only be connected to its nearest neighbours (defined in terms of a similarity measure). A similar statement can be made for the construction of term-term links. In both cases, the network can be restricted to contain links for the single nearest neighbour only (or the set of equal nearest neighbours). The restriction of clusters to those involving nearest neighbours can result in significant efficiency benefits with no loss in effectiveness.

The document-document (or term-term) links formed by the nearest neighbour process can be thought of as *star* clusters. A star is a cluster in which every member is related to a distinguished central member of the cluster [20]. Each document in the network representation serves as the central member of a star cluster formed by connecting that document node to its nearest neighbour document nodes (Figure 2).

In general, the formation of star clusters is highly order dependent in that a different clustering is constructed depending upon which nodes are regarded as central nodes. Since we generate a separate star cluster using each node as a central node, the generation of

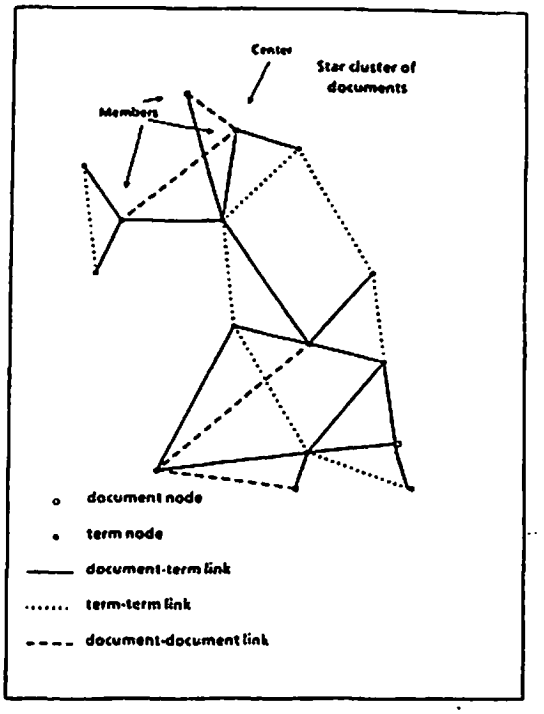


Figure 2: A star cluster in the network representation.

star clusters is order independent in the network organization.

The star clusters formed in the network organization are small overlapping clusters of highly related documents or terms. This means that, in contrast to the document-term links, the term-term and the document-document links are directed in the sense that they define two different relationships. These relationships can be called the document-to-document, term-to-term, document-from-document and term-from-term links. One set of links defines the other members of the cluster having the current document (or term) as the central member. The other set of links defines the clusters that the current document (or term) belongs to as a non-central member.

Cluster searching uses the document-document links from the star clusters. In general, this type of strategy requires cluster representatives for comparison to the queries. However, for maximum flexibility and storage savings, the cluster representatives are not

stored in the network as separate entities, but are generated dynamically. The algorithms for cluster searching as well as for the probabilistic search will be described in section 4.

As well as the probabilistic and cluster-based searches, the network organization could allow the user to follow any links in the network while searching for relevant documents. A special retrieval strategy, called *browsing*, could be based on this ability. A browsing strategy will be particularly useful as an alternative to more formal strategies or as a means of assisting the user with the query formulation process. In some related work, Oddy proposed a simple network organization which was used to implement a system entirely based on a browsing strategy [21].

An obvious implementation of the document and term network would be to directly represent the nodes and links in a graph data structure [22]. In addition to the probable efficiency benefits of such an implementation, it would also be able to exploit the grouping inherent in the clusters constructed from the terms and documents. Since these clusters contain either terms or documents with similar contents, it could be assumed that storing the members of a cluster physically close to each other would yield additional efficiency benefits.

The advantages of a database implementation of the network are also significant. The most important of these advantages is the simplicity and flexibility inherent in the separation of the logical representation from the physical representation. The crucial question is whether this flexibility results in much lower efficiency. The experiments reported in this paper are designed to address this question.

3.0 THE BASIC NETWORK OPERATIONS

A number of basic operations are required when the network representation is searched and updated. The same operations are also a part of the algorithms used to construct the initial network. One basis for comparison of the network structure and database implementation is to contrast the relative efficiency of these operations. The operations are as follows,

1. Given a node identifier (for example, a document or term number), retrieve the contents of the node.
2. Given a node identifier and a link type (document-term, term-document, term-to-term, term-from-term, document-to-document or document-from-document), retrieve all nodes connected to the specified node by the specified link type.
3. Given the content information, create a new node.
4. Given a link description, create a new link.
5. Given a node identifier and new information, update the contents of an existing node (similar to operation 1).

The first two of these operations are the most important for our experimental implementations. These operations will be referred to in the following sections.

4.0 THE TWO IMPLEMENTATIONS

The following discussion covers various aspects of the two implementations of the network representation. After a general description of the implementation, the algorithms for the initial construction of the network are given. This primarily involves the generation of the document and term nearest neighbours used for the document-document and term-term links. The remaining sections discuss, for each implementation, the following

topics;

a. The use of the physical grouping inherent in the document and term clusters.

b. The search algorithms for the probabilistic and cluster searches. The browsing strategy is implemented using basic operations 1 and 2.

c. The method of updating the network. If the network is to be used in a real system existing in a dynamic environment, updating must be efficient and must not degrade the effectiveness of the structure. Insertion of a document into the network establishes links to the related documents and terms, creates additional term nodes if new terms are introduced, and modifies any term-term or document-document links that are affected.

The nearest neighbour and probabilistic search algorithms used for the network structure are very similar to algorithms described in previous studies [23,24]. For the database implementation, however, these algorithms proved to be very inefficient and had to be modified. The modifications are discussed both in this section and in section 5.

4.1 The network structure

The representation of the network as a graph structure is accomplished by six arrays storing the document nodes, term nodes, and the four varieties of links. The varieties are Doc-Doc, Doc-Term, Term-Doc, and Term-Term links. The arrays are divided into fixed length "pages" for simulating page access on a virtual memory system. The nodes contain the minimal amount of information to define a term or document, which consists of the pointers to the links, the number of the links of each type (also called the postings), and some other details required for the operation of the grouping algorithm. Having only this information allows us to keep the nodes small so we can place a number of them on a

512 byte page. A link consists of a node number, which corresponds to an index into one of the node arrays, and a weight. The main disadvantage of separating the links from the nodes is that it will generate a fault as we follow the links from node to node. The storage overhead required for this organization is very similar to that required by a combined serial/inverted file which also contains document-document and term-term links.

Information such as the text of the document, title, author(s), and the words corresponding to term identifiers could be stored in a separate area. Access to this information will generate a page fault. In our experiments, this information is omitted.

The nodes (documents and terms) are grouped and the lists of links are also grouped, so that lists of links of similar nodes are stored contiguously. Using this technique it is possible to fit many nodes on one page, but few lists of links on one page, since each list of links will take up a good portion of a page.

Indexing into the network is performed by hashing. Hashing will usually cost one page fault to locate a document or term node, given the document or term identifier (basic operation 1). The location of nodes connected to a particular node (operation 2) is carried out in the following way. First, the node with the given identifier is found. Then the pointer to the links of the specified type is used to access the appropriate part of the link array. Finally, the node numbers in the link array are used to access the connected nodes. The overall efficiency of this operation will depend on the effectiveness of the physical grouping algorithms.

Constructing the network.

The main problem with the construction of a network for a given set of documents and terms is the generation of the document and term nearest neighbours. Various methods for efficiently finding nearest neighbours in the information retrieval environment

have been proposed [23,24]. These methods use the following basic algorithm (we shall discuss only document nearest neighbours, term nearest neighbours are found in a similar way);

To find the nearest neighbour for document D, calculate similarity values between D and documents having at least one term in common. The document-document pairs for the similarity calculations are found by first finding the set of terms that describe D and then using the inverted lists to retrieve the documents associated with those terms. Documents which have been seen in a previous inverted list are ignored. The similarity measure used in the experiments reported here is Dice's coefficient [1], which can be represented as $2|D_i \cap D_j|/(|D_i|+|D_j|)$. In this expression, |.| is the modulus operator and D represents a set of binary index terms.

Smeaton and Van Rijsbergen modified this procedure to avoid processing all of the inverted lists associated with the terms in document D. After each inverted list is processed, an upper bound (U1) is calculated for the maximum similarity value that could be obtained by comparing D to documents in the remaining inverted lists. The upper bound is calculated by assuming that any document not seen could have all the remaining terms in common with the document D. The assumption is also made that only N nearest neighbours are required. A ranked list of the top N document scores is maintained and when the lowest score on this list is greater than U1, processing stops. Since we only require the nearest neighbour, N is 1 in the experiments reported here. The use of U1 avoids many unnecessary calculations but guarantees correct results. If the inverted lists were organized in order of frequency of use of the associated terms, this method will also avoid using the longest lists.

Murtagh used an upper bound (U2) which is calculated for each candidate document on a new inverted list in order to determine whether the actual similarity calculation for the document should be made. This upperbound is a more precise form of U1 in that it says that the maximum number of terms that D could have in common with a particular document D′ not seen before is *either* the remaining number of terms or the number of terms in D′, whichever is smaller. The number of terms in D′ is also used to provide a more accurate upperbound for Dice's coefficient. The use of U2 avoids many calculations of the number of co-occurring terms between two documents which is the most expensive part of the similarity calculation.

The method used in the experiments reported in section 5 is a combination of these two approaches with some modifications which are designed specifically for the task of finding nearest neighbours for all documents in a collection. Upper bound U1 is used, as described previously, to determine if more inverted lists should be processed. Upper bound U2 is used to determine if documents within these inverted lists should be considered. A minimum threshold T is also placed on the similarity value. The upperbounds U1 and U2 are compared to the threshold T before comparing them to the lowest score on the list of N nearest neighbours. Processing stops when no document can achieve a score greater than T or when no more nearest neighbours will be found. Any document score actually calculated that is less than T is ignored. The use of the threshold value affects mainly those documents that are only weakly related to other documents. These weak links are not regarded as significant and this results in documents with no nearest neighbours.

While a nearest neighbour for a particular document is being calculated, a record is kept, for every other document, of the highest similarity value that the document was involved in. For example, if while finding the nearest neighbour for document 100, a

similarity value of 0.53 was calculated for the document pair 100,123, this similarity value is compared to the previous highest value seen for document 123 and stored if it is higher. This means that when the nearest neighbour for document N is to be found, the calculations start with the similarity value found from the calculation of nearest neighbours for documents 1 through N-1. That is, upperbounds U1 and U2 must be greater than this previously calculated similarity value for processing to continue. This procedure significantly reduces the number of calculations requires and it ensures that document pairs containing documents 1 through N-1 need not be considered when calculating nearest neighbours for document N. An outline of the overall nearest neighbour algorithm appears in Figure 3.

This algorithm for finding nearest neighbours relies heavily on the ability of the network structure to quickly locate the list of terms connected to a particular document and the list of documents connected to a particular term. In fact, for this task only the node numbers stored in the link arrays need to be retrieved. As we will see in section 4.2, the characteristics of the database implementation require modifications to the nearest neighbour algorithm.

Physical grouping.

Because of the declining cost of disk storage, large collections of documents will continue for some time to be stored on disk, a cellular memory device. Grouping of closely related items in the same cell to reduce inter-cellular references can significantly reduce the time required to respond to user queries. Grouping is similar in concept to the use of document (and term) clustering to organize a document collection. Although grouping shares some of the same attributes and techniques of clustering, the two are not identical. In particular, the groups into which a document collection is partitioned need not correspond exactly to the clusters produced by document clustering.

1: Get terms in next document (D)
|
Put document with highest previously calculated similarity to D into nearest neighbour list
|
Rank terms in D in order of increasing frequency
|
2: Calculate U1
|
If U1 < T or U1 < current nearest neighbour similarity then goto 1 else
|
Get next term from ranked list
|
Get inverted list for term
|
3: Get next document (D') from list where D' > D
If no more documents goto 2
|
Calculate U2
|
If U2 < T or U2 < current nearest neighbour similarity then goto 3 else
|
Calculate similarity for (D,D')
|
If similarity value > previous high similarity for (D,D') replace
|
If similarity value >= current nearest neighbour similarity then replace nearest neighbour or add to list of equal nearest neighbours
|
Goto 3

Figure 3: Finding nearest neighbours for documents 1 to N.

Unlike document clustering, where a document may belong to more than one cluster, a document may belong to only one group, since it is stored in one location in physical memory. A method of grouping is required that constructs non-overlapping groups. In addition, document cluster sizes may vary whereas grouping attempts to produce groups that are of fixed size, the size of a secondary storage cell (disk track or page), since the entire cell will be transferred into primary memory when a reference to any member of that cell

occurs. A method of grouping is required, then, that constructs fixed size groups.

Groups are constructed using *strings*, a clustering technique similar to stars. Strings are clusters in which strongly connected objects are linked up to the natural cutoff of a loop or maximum length [20]. A string is a group of objects that are related transitively as nearest neighbours (Figure 4).

Once an object has been placed in a string, it is not eligible for membership in another string. If in the course of constructing a string it leads into another string, the two strings are joined together as one. If the size of a string becomes larger than the maximum size allowed, the string is split into two.

The use of strings for grouping suffers from the problem that the construction of strings is dependent upon the order in which objects are inserted into the network. Another problem is that it is difficult to generate fixed size groups since many strings will complete

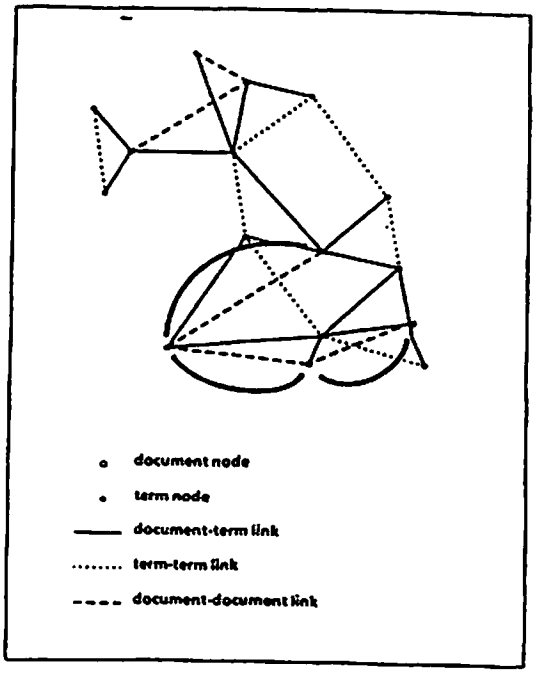


Figure 4: A string in the network structure.

before the specified size is reached. To overcome this, small strings will be stored together in cells until the desired size is obtained. Some space will be left in each cell for updating. The properties of strings used for grouping will be studied in section 5.

Search algorithms.

The probabilistic search strategy described in section 2 is implemented in the network structure using an algorithm based on the nearest neighbour algorithm. The algorithm assumes that only a fixed number N (say 20) of the top ranked documents are retrieved initially. For each query term, the weight w_i is calculated using the information stored in the term nodes. Then, for each term in order of decreasing weight, the set of documents connected to that term (the inverted list) is found. For each document that has not been seen before, the set of terms connected to it is found and used to calculate the document's total score. Before processing a new inverted list, a check is made on the number of documents seen. If this number is greater than N and it is not possible for a document that has not been seen to have a higher score than those already seen, processing stops [19]. The upper bound used to stop processing is calculated in a similar way to the U1 bound used in the nearest neighbour algorithm. Because terms used in many documents have low weights, this algorithm avoids processing many of the large inverted lists.

The cluster search is implemented in the following manner. First, the documents that contain query terms are located using the document-term links (term-document form) of the network. The document-document links are then followed from these documents to form cluster representatives and calculate the cluster scores. More details are given in the section on the database implementation.

Updating the network.

The insertion of new documents into the network structure is performed as follows (Figure 5). First, space is allocated for a new document node and the document-term forms of the links are constructed. The document-term information is taken from the document representative provided as input to the insertion operation. Next the term-document form of the link is constructed for every term of the document representative. If a new term is used, space is allocated for that term in the network. The nearest neighbour(s) of the new document are then determined to construct the document-document links. Insertion of a new document may require modification of existing document-document links, in which case the document-document links of the nearest neighbours of the new document are reconstructed. Insertion of a new document may also modify existing term-term links, so for each term of the new document, the term-term links are reconstructed. The reconstruction of links is not propagated further in the network than the nearest neighbour documents or terms, for reasons of efficiency. The nearest neighbours of the new document and terms should be used to guide the placement of the new node in the physical grouping of the network. That is, a new document node should be stored as close as possible to its nearest neighbour.

4.2 The database implementation

The CODASYL implementation of the network is based on the logical schema design shown in Figure 6(a). Each of the relationships shown in this diagram are many-to-many and intersection records must be used to implement the schema using CODASYL sets. The CODASYL schema is shown in Figure 6(b). This schema, although straightforward, leads to major efficiency problems. In particular, the extra level of complexity that is introduced by the dt-link intersection records produces unacceptable response times when the documents

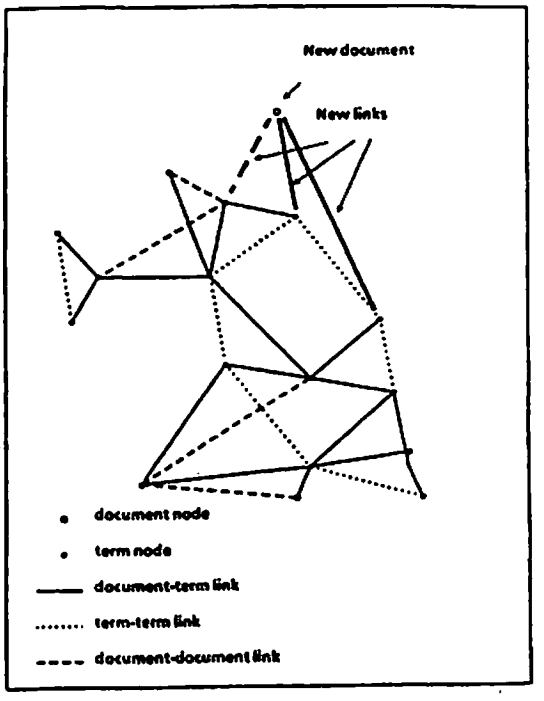


Figure 5: Insertion of a new document in the network.

connected to a particular term (or the opposite) need to be found. For example, to find the documents connected to term x, each of the member dt_link records in the TERM_DOC set must be retrieved using a "FIND NEXT" DML statement. Then, for each of these dt_link records, the owner in the DOC_TERM set is retrieved. We will see in section 5 that a major component of the overhead in processing the CODASYL sets is the repeated execution of the DML statements. As the document-term links are the most crucial for implementing the search strategies, some redundancy was introduced into the schema to overcome this problem.

The revised schema, shown in Figure 7, replaces the dt-link record with two record types, link_document and link_term. These records contain either the document or term identifiers, respectively. As most of the search strategies require only the identifiers to produce a ranked list of documents, this modification results in considerable savings in

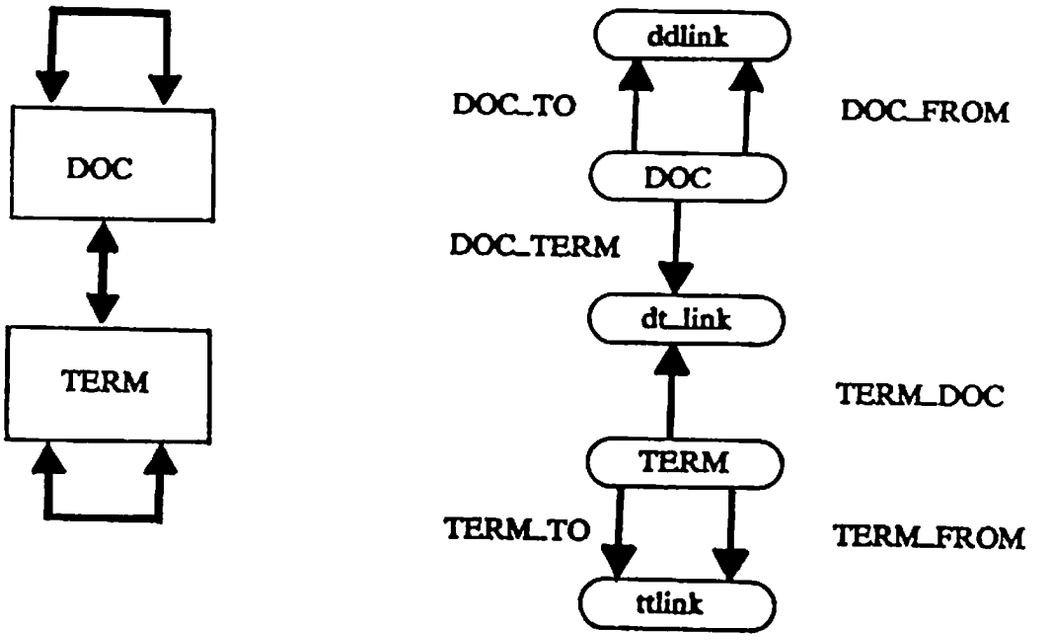


Figure 6: A CODASYL schema for the network.

processing time at the expense of the redundant storage of the identifiers. If the full document or term information is required, the identifier in the link_document or link_term record can be used to access the node through the system sets. These sets, which are shown in Figure 7, allow direct access to document or term nodes in a similar manner to the hash algorithm used in the network structure. The records representing document and term nodes contain identifiers and postings information. The intersection records representing the links contain a weight that measures the strength of the link.

The basic operations (1 and 2) are implemented through the system sets and intersection records using the CODASYL Data Manipulation Language (DML) [9]. The algorithms that use the schema to find nearest neighbours or carry out search strategies are

written as application programs with embedded DML statements.

Constructing the network.

As before, the main problem to be addressed is the generation of the document and term nearest neighbours. The algorithm described in section 4.1 must be modified for the database implementation. This is necessary because the efficiency of the database implementation is heavily dependent on the number of record accesses made through the linked lists that represent the CODASYL sets. Each execution of a CODASYL DML statement retrieves a single record which represents a single node connected to the original node. The previous algorithm causes a large number of record accesses because both the term-document and document-term links are used. The modified algorithm avoids using the document-term links by accumulating partial document scores as each list is processed. This means that the upper bound cutoff values used in the network structure algorithm either cannot be used in the database algorithm or, if they are, could lead to inaccurate results. If the upperbound comparison stops processing before all inverted lists are used, either the

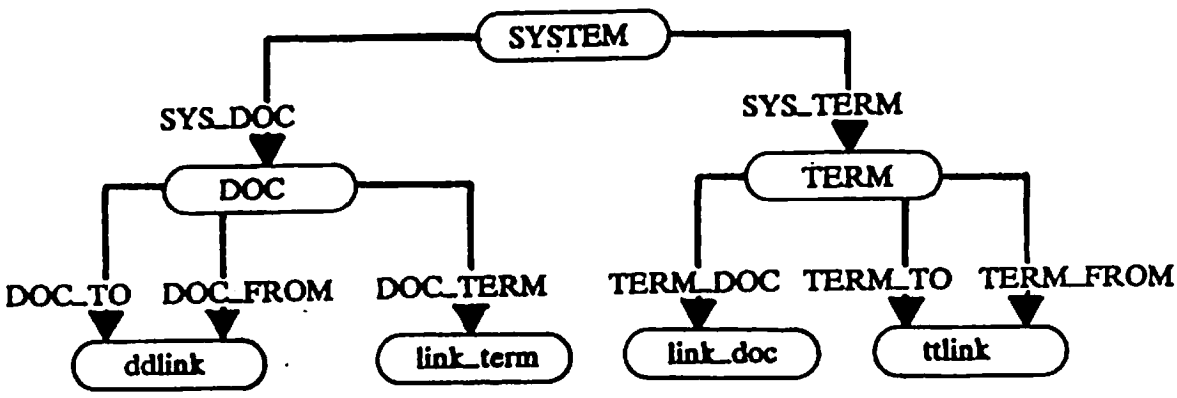


Figure 7: The revised CODASYL schema.

similarity value for the nearest neighbour will be incomplete or the wrong nearest neighbour will be selected. The modified algorithm is not as appropriate for the network structure because a single record access in this structure can retrieve an entire document-term or term-document list.

To find document D's nearest neighbour(s), each of the terms in the document are first ranked in decreasing order of their IDF weight. These terms are then expanded in order by finding the documents connected to them. A hash table is used to count the number of term lists in which a document occurs. An update is applied to the hash table for each document in the current term list. If the document has occurred previously, its score is incremented by one, otherwise an entry for that document is inserted.

Although the use of an upperbound cutoff could lead to inaccurate results, the potential efficiency savings (in processing time) led to the use of a simple upperbound based on the number of terms in common between the document D and the other documents. This upperbound is used as follows. A record is kept of the document with the most terms in common with document D (referred to as MAXDOC). Term lists are expanded until the number of lists MAXDOC has occurred in is greater than the number of unexpanded term lists. This cutoff ensures that no documents that have not been seen will have more terms in common with document D than MAXDOC. However, because Dice's coefficient depends on the number of terms in a document as well as the number of terms in common with D, this cutoff does not guarantee that a document's actual nearest neighbour will be inserted into the hash table. Because the terms are processed in decreasing order of importance, documents that are missed in this way will tend to be less significant for retrieval. The effect of this possible source of error will be evaluated in section 5.

After the cutoff has occurred, the documents in the hash table must be processed further to calculate the exact similarity values. This stage is not required for the network structure algorithm because exact values are calculated while processing the term lists. The hash table entries are divided into five possibly empty groups. The first group contains those documents with the same number of terms in common with document D as MAXDOC. Documents in the second, third and fourth groups occurred in one, two and three less term lists, respectively. The last group contains the remaining documents in the hash table.

Starting with the first group and proceeding to the fourth, each document is checked to see if the maximum possible similarity value that could be obtained with the document is greater than a minimum threshold value (0.3). The calculation of the maximum possible similarity (which is similar to U2) is done by assuming that the document will occur in all the term lists which were not expanded. If the maximum possible similarity value is greater than the threshold, then the actual similarity value is calculated and stored in a sorted list. Documents are checked this way until either the first four groups are exhausted or ten similarity values are put into the sorted list. The highest scoring document in the sorted list is then stored in the database as the nearest neighbour. Any other documents in the list with equal scores are considered equal nearest neighbours and are also stored in the database. This version of the nearest neighbour algorithm does not make use of previously calculated similarities because very few complete similarity values are calculated and ıtoffs based on similarity values (such as U1, U2) are not appropriate when only partial similarity values are available. This would not be a drawback in operational systems where the database is added to incrementally and previously calculated similarity values are not available. An outline of this version of the nearest neighbour

algorithm appears in Figure 8.

Physical grouping.

The network structure implementation allows direct control over the physical placement of the document and term nodes. Although this degree of control is not possible in a CODASYL database, some facilities do exist for specifying where records should be

---

```
Get terms in next document D
  |
Rank terms  in order of increasing frequency (or decreasing IDF weight)
  |
1: If MAXDOC > number of remaining lists then goto 3
  |
Get next term from ranked list
  |
Get inverted list for term
  |
2: Get next document from list
If no more goto 1
  |
Add 1 to score in hash table
If score > MAXDOC, update MAXDOC
  |
Goto 2


[Process documents in hash table in order of decreasing number of
terms in common with D]
  |
3: Get next document from hash table group
  |
If maximum possible similarity < threshold then goto 3
  |
Calculate actual similarity
  |
Add to list of nearest neighbours
  |
If list = 10 stop
else goto 3
```

Figure 8: Finding nearest neighbours using partial scores.

stored. In our implementation, the database is divided into two physical areas, term-area and doc-area. The main records in these areas are the term and document records, respectively. The term records are scattered throughout term-area and are accessed directly (via the system set). The link_doc records that specify which documents contain a particular term are clustered around the owner term records. The records representing term-term links are loaded after the link_doc records and are also clustered, as much as possible, around the owner records. The link_doc records are given priority in the physical placement because they are the most frequently used in the system. The doc-area is set up in a similar fashion with the link_term records clustered around their owners, followed by the records representing the document-document links.

The main difference between the physical grouping in the network structure and the database system is that the document and term records (or nodes) are clustered in the network structure. The importance of this clustering is studied in section 5.

Search Algorithms.

The probabilistic and cluster search algorithms are modified for the database implementation for the same reason as the nearest neighbour algorithm - the efficiency of the database implementation is strongly dependent on the number of record accesses required, whereas this is a less important factor for the network structure implementation. Both of these algorithms accumulate document (or cluster) scores in a similar way to the nearest neighbour algorithm for the database implementation.

The probabilistic search starts by sorting the query terms in o ler of decreasing IDF weight. Then, starting with the first term, all of the documents connected to that term are retrieved and inserted into a hash table along with a score. At this point, the score is simply the weight of the first term. The second query term is then expanded in a similar

manner, except that in the case of a document that had already been inserted into the hash table, the score is incremented by the current term's weight. The rest of the query terms are expanded in the same way. The documents in the hash table are sorted by their score and the top ranked documents are returned to the user. It can usually be assumed that only a certain number (say 10 or 20) of the top ranked documents are required.

Since the terms are expanded in order of decreasing importance, it is possible to determine the best and worst possible scores for new documents. For any document that has not been seen before, the best possible score for that document is the sum of the term weights from the current term to the last query term, and the worst possible score is just the weight of the current term. If, when expanding a term, the number of documents in the hash table exceeds the number of documents to be retrieved (usually 10 or 20), then that term's weight is recorded. This means that the worst possible score for a document in the hash table is the weight of that term. From that point, documents are no longer inserted into the hash table if their best possible score is less than the recorded weight.

In keeping with the fact that the more common a term is the less it contributes to a document's score, a second cut-off has been included. After the number of documents in the hash table exceeds the number required for retrieval, each term is tested to see if it should be expanded. If a term's weight is greater than the difference between it and the weight of the previous term then the term is expanded, otherwise term expansion stops. This cut-off helps to eliminate processing of the long lists associated with common terms. Unlike the previous cut-off, this does not guarantee optimal results, but in practice there is no noticeable change in the documents retrieved (see section 5).

The cluster search operates along the same general lines as the probabilistic search. A score for a cluster is calculated by comparing the union of all its documents' terms with the terms in the query. This union is called a cluster representative. In previous studies, the cluster representatives were calculated and stored prior to searching. This implementation reduces storage overhead significantly by generating cluster representatives dynamically. At present, the number of times a term occurs in a cluster is not taken into consideration, but will be in a later version of the search.

After the query terms have been sorted on decreasing IDF weight, the document lists for each of the terms are processed. Because of the nature of star clusters, each document must be considered as the central member of one cluster and a secondary member of a number of other clusters. As each document is processed, it is inserted into a hash table as the representative for the cluster of which it is the central member. If the document has not been seen before, the cluster's score is the weight of the current term. If the document has been seen before, the cluster's score is incremented by the current term's weight. The scores are then updated for the clusters in which the document is a secondary member. This is done by keeping a list of those documents which are the central members of the clusters in which the current document is a member. The list is generated and stored in the hash table the first time a document is encountered in an inverted list. The owner documents for the list are found by following links in the database schema. For each of these owner documents, if the document is not in the hash table then it is inserted with the current term's weight as the score; otherwise, the score for the document's cluster is incremented by the current term's weight. As the cluster representative being used is the union of the terms in the member documents, each term's weight is used at most once in the calculation of a cluster's score. Note that it is possible to have a cluster whose score is

not dependent upon its central member, if the central member never appears in a document list, but one of the secondary documents does. This is a crucial feature of the cluster search and makes it possible to retrieve different documents than the probabilistic search.

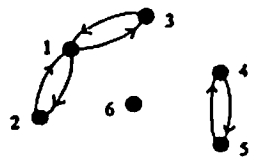
Figure 9 gives an example of the cluster search algorithm. In this example, the documents are in tightly connected clusters that results in some redundant calculations of cluster scores (for example, the document cluster 4,5). This has no effect on retrieval and only a small effect on efficiency. Note that the cluster (4,5) is an example of a document being retrieved by virtue of its relationship with another document rather than with the query.

The cut-offs used in the probabilistic search are also used in the cluster search. Clusters are not inserted into the hash table when their best possible scores are too low for them to be included in the set of retrieved clusters. The expansion of query terms is stopped according to the same criteria used in the probabilistic search. In this case, the search retrieves the top-ranked clusters, rather than documents.

Updating the network.

The algorithms used for updating the database implementation of the network are very similar to those described for the graph structure. The main difference is that the actual operations of creating new records and adding them to CODASYL set instances are done by the database system. The update algorithms include CODASYL DML commands that specify the operations to be carried out.

Example documents with nearest neighbour links



Query terms 8, 10, 35
IDF weights 40, 70, 10

Inverted lists

Term	Documents
10	1,2,3,5
8	2,3,5 [more]
35	1,2,6 [more]

Hash table during processing of inverted lists

Central cluster member	Owner documents	Cluster Score After term 10	After term 8	After term 35
1	2,3	70	90	100
2	1	70	90	100
3	1	70	70	80
4	5	70	90	90
5	4	70	90	90
6	-	-	-	10

Figure 9: A cluster search example.

5.0 THE EXPERIMENTS

In order to provide a firmer basis for comparing the two implementations, a series of experiments using test collections of documents and queries were carried out. The documents and queries in the Cranfield and NPL collections have been indexed by various methods [3] and their statistics appear in Table 1.

	Cranfield	NPL
Number of documents	1,400	11,429
Number of terms	4,949	7,491
Number of queries	225	93
Average terms per document	53.6	20.0
Average documents per term	15.2	30.4
Average terms per query	8.9	7.2

Table 1: The test collections

The experiments concentrate on the generation of nearest neighbours, the effect of physical grouping, and the efficiency of the search algorithms. Many of the experimental results for the two implementations are difficult to compare directly, but it should be remembered that one of the major aims of the experiments is to study the efficiency of the database implementation. For this reason, more results are provided for this implementation. All experiments were run on a single-user VAX 750/VMS machine. Statistics for the database experiments were collected with the monitoring utility provided with the DBMS system [25]. The timing figures in the tables are seconds of CPU processing time.

The generation of nearest neighbours, as well as being the main part of constructing an initial network, is the most crucial part of updating the network to include new documents. It is important, therefore, that this operation is relatively efficient. The first set of experiments used the network structure algorithm described in section 4.1 with the Cranfield test collection. If the full n(n-1)/2 possible similarity values between all pairs of documents in a collection were calculated and stored in an array, the nearest neighbour of any document in this collection could be found using this array. This would result in an average figure of 700 similarity calculations to find a document's nearest neighbour. The algorithm using the U1 and U2 cutoffs combined with remembering previously calculated

values required an average of 209 similarity calculations. To find a term's nearest neighbour required only an average of 25 calculations. The efficiency of the term calculations comes from the fact that many of the terms in the collection occur in only one document.

The majority of the savings in calculating document nearest neighbours comes from the previously calculated similarities. In an experiment that used only the U1 upper bound, it took an average of 480 calculations to find the nearest neighbour for a document. In an operational system, the majority of nearest neighbours will be calculated as a result of new documents being inserted into the network. In this case, previously calculated similarities are not available and the higher average number of similarity calculations is a better estimate of the efficiency of this algorithm.

As described in section 4.2, the nearest neighbour algorithm used in the database implementation cannot make use of the U1, U2 cutoffs or the precalculated similarities because only partial similarity values are calculated as the term lists are processed. The algorithm also does not necessarily find the exact nearest neighbour. In order to compare the database algorithm with the previous algorithm, the number of entries in the hash table can be taken as roughly equivalent to the number of similarity calculations made. The average number of hash table entries for the Cranfield collection was 608 without using the term list cutoff. If the term list cutoff is used, the number of similarity calculations required is reduced and the correct nearest neighbour is found in over 85% of the cases. More detailed statistics for the database algorithm used for the 11,429 documents of the NPL collection appear in Table 2. In this table and in the remaining experiments, "records" refers to database logical records (e.g. document, term) and "disk accesses" refers to the number of disk accesses to the actual database. The buffer size, unless otherwise mentioned, is 280 pages (each page is 512 bytes).

	Average per document
Term lists processed	15.7
Term lists avoided	4.3
Entries in hash table	2725
Full similarity calculations	34.3
Disk accesses	81.3
Records retrieved	4060
Number of nearest neighbours	0.94

Table 2: Finding nearest neighbours in the NPL collection

These results can be summarized as follows. The term list cutoff occurs on average after 75% of the term lists for a document have been processed. As terms are processed in order of increasing frequency, significant savings are made by avoiding the processing of the last lists. Taking the number of entries in the hash table to be roughly equivalent to the number of similarity calculations, an average of 2725 calculations were required per document (compared to 6,500 for the full similarity matrix). This number is somewhat misleading, as an average of only 34.3 entries in the hash table had to have exact similarity values calculated in order to find ten values over the threshold value. Most of the overhead involved with this algorithm is accessing the inverted lists corresponding to a document's terms. Since these lists only grow linearly with collection size, the algorithm appears to be appropriate for large document collections.

The nearest neighbour algorithm for the network structure implementation appears to have a significant advantage when all the documents are known and precalculated similarities can be used. However, in the more realistic case when nearest neighbours are calculated for every document as it arrives, there is much less difference between the two implementations. The greatest inefficiency that arose from using the database system was the repeated execution of the CODASYL DML statements. For example, to find the members of a particular link_doc set requires the repeated execution of a FIND NEXT

statement. In our database system, the DML statements are interpreted at run-time and, for large sets, this results in considerable delays. Later experiments provide further evidence of this problem.

The experiments dealing with physical grouping concentrated on measuring the potential benefits of document and term clustering in the network structure implementation. To do this, statistics were taken of simulated page faults during the processing of the 225 queries of the Cranfield collection. The grouping of the document and term nodes was done with the string algorithm described in section 4. Two different variations of this algorithm were used. One variation held the size of a string to a fixed limit. The limited size strings were designed to be more convenient for storing on pages. The other variation started each string on a new page rather than packing strings together as much as possible. Table 3 presents the results of this experiment. The table compares the number of page faults obtained for each type of page (document nodes, term nodes or links). The experiment simulated actual page faults by maintaining a buffer of 64 pages. Whenever a new page was required that was not in this buffer, a page fault was recorded. The page replaced was the least recently used. The results show very little increase in efficiency due to physical grouping. The only significant difference was a 17% decrease in the number of document page faults. This suggests that physical grouping using strings is not necessary and the database implementation will not be adversely affected by the lack of this grouping.

Table 4 gives some indication of the effectiveness of grouping the member records around document and term records in the database implementation. This table gives examples of processing time and disk access figures for basic operation 2 (retrieve all nodes connected to a specific node) on the Cranfield and NPL collections. The actual operation was to retrieve the documents connected to a specific term. These figures show that,

although the sets with many member records can be retrieved efficiently from disk, they do take much longer to process than small sets. For example, in the NPL collection retrieving a set of size 200 required 3 disk accesses and a set of size 805 required 2 accesses. Despite the efficient grouping of member records on disk, the processing times for these two sets differ greatly (5 versus 19 seconds). This result supports the earlier observation that the most inefficient part of the database implementation is executing the DML commands in the application programs.

	Page faults		
	Term	Link	Document
Not grouped	1373	5570	3483
Grouped (Limited size)			
No new page	1304	5570	2909
New page	1314	5548	2926
Grouped (Unlimited size)			
No new page	1350	5521	2861
New page	1346	5567	2905

Table 3: Physical grouping in the Cranfield collection

	Examples of basic operation 2.		
	Size of CODASYL set	Disk accesses	Time(secs)
Cranfield	1	1	0.18
	250	2	5.67
	845	4	18.94
NPL	1	3	0.17
	200	3	4.89
	805	2	19.14
	2506	6	59.54

Average records retrieved per disk access: 113.4 (Cran); 189.4 (NPL)
Average records retrieved per second: 36.2 (Cran); 34.4 (NPL)

Table 4: Using links in the database implementation

The final set of experiments measured the performance of the probabilistic and cluster search algorithms using the database implementation. The results are presented in Tables 5 and 6. The processing times are shown to be heavily dependent on the number of records accessed. The figures for the probabilistic search for Cranfield and NPL, together with the figures in Table 4, indicate that disk accesses are not the major component of the

overhead. That is, although the number of records seen and the processing time goes up considerably from Cranfield to NPL, the number of disk accesses does not increase proportionally. The NPL collection, with its longer term lists, retrieves more desired records with each disk access. The reason for the increase in processing time is mainly due to the overhead, mentioned before, of interpreting a DML command for each record processed.

In order to achieve response times that are acceptable for an operational environment, a database system that does not have this overhead (such as a typical relational system) is needed. The cluster search involves many more disk accesses than the probabilistic search to follow nearest neighbour links. The relatively long processing times required mean that the cluster search will essentially be restricted to use as an alternative strategy.

The algorithms used in the network structure implementation require more than twice the number of logical record accesses when used in the database implementation because the full similarity values are calculated during the processing of the inverted lists. A more direct comparison of the network structure and database algorithms can be made using Table 3. The figures in this table were generated using the probabilistic search algorithm for the network structure. Each simulated page fault can be taken as equivalent to a disk access. This gives a figure of 43 disk accesses per query. Because the number of disk accesses depends on the size of the buffer storage, a run of the probabilistic search on the database implementation was carried out with a buffer pool of 70 pages (compared to the 64 pages used in the network structure). The average number of disk accesses for this run was 18. This result indicates that the database algorithm can be competitive in terms of efficiency with the network structure algorithm. The documents retrieved by the database algorithm (which uses an inexact cutoff) were virtually identical to those retrieved by the other algorithm.

	Cranfield	NPL
Av. records seen per query	667.7	2954.0
Av. disk accesses	12.6	17.7
Av. time (secs)	17	68

Table 5: The database implementation of the probabilistic search

	Cranfield	NPL
Av. records seen per query	2924	3942
Av. disk accesses	868.5	1252.0
Av. time (secs)	117	162
Av. clusters seen	314	448

Table 6: The database implementation of the cluster search

6.0 CONCLUSIONS

A network representation of documents and words can enhance the effectiveness of a document retrieval system by allowing a variety of search strategies to be implemented. The comparison of the standalone implementation of the network with the database implementation can be summarized as follows;

a. In terms of disk accesses and the number of similarity calculations required for the basic network operations, search strategies and nearest neighbour algorithms, the database implementation was comparable to the standalone implementation.

b. The standalone implementation received no significant benefits by exploiting document and term clustering in the physical organization.

c. The database implementation required long processing times for large CODASYL sets. This is a result of repeated executions of uncompiled DML statements. The processing times for the probabilistic search and, in particular, the cluster search

are too long for an operational system because of this overhead.

Another overhead involved with the database implementation is the storage requirements. In the CODASYL system, the storage overhead was approximately 250% of the storage required for the actual data. This is much worse than the standalone implementation. Most of this overhead comes from representing each network link in a separate CODASYL record.

The experiments also showed that a database system provides a flexible, convenient framework for implementing sophisticated document retrieval algorithms. For example, the same system can be used to implement different versions of probabilistic searches, cluster searches and conventional Boolean query processing. It appears that these algorithms can be implemented without requiring fundamental changes to the database system. The implementation of the search algorithms and the underlying network structure was much more straightforward in the database system than in the standalone system because of the separation of the logical and physical levels of design.

The major components of the database system that are involved with the successful implementation of a document retrieval system are the logical schema, the data manipulation language, and the underlying implementation of both of these. The CODASYL database system used in these experiments suffered from long processing times and large storage overheads due to the large number of DML commands that had to be interpreted and the intersection records that are a necessary consequence of using this data model. These efficiency problems could be largely avoided by using another database system. For example, database systems that precompile DML statements are currently available. Relational database systems, in particular, appear to offer advantages for network-based document retrieval. The DML statements in a relational system are expressed in terms of

larger units than CODASYL records (relations versus records). The links in the network can be represented directly in relations (e.g. a document-term relation) and the system can avoid the extra level of processing required for intersection records. The efficiency of a relational database implementation of a document retrieval system will, however, depend strongly on the underlying file organizations.

Acknowledgments.

This work was supported in part by grant IST-8111108 from the National Science Foundation. Richard Wolf and Roger Thompson made significant contributions to the early experiments. Dr. Jack Orenstein provided valuable advice during the experiments and the preparation of this paper.

REFERENCES

[1] C.J.Van Rijsbergen, *Information Retrieval*. Second Ed., Butterworths, London, (1979).

[2] G.Salton and M.J.McGill. *Introduction to Modern Information Retrieval*. McGruw-Hill, New York, (1983).

[3] K.Sparck Jones and C.A. Webster, *Research on relevance weighting*. British Library Research and Development Report 5553, Computer Laboratory, Cambridge, (1980).

[4] I.A.MacLeod and R.G.Crawford, Document retrieval as a database application. *Information Technology*, 2, 43-60, (1983).

[5] M.F.Porter, Implementing a probabilisitic information retrieval system. *Information Technology*, 1, 131-156, (1982).

[6] H.J.Schek and P.Pistor, Data structures for an integrated data base management and information retrieval system. *Proceedings of the Eighth International Conference on Very Large Data Bases*, 197-207, (1982).

[7] M.Stonebraker, H.Stettner, N.Lynn, J.Kalash, and A.Guttman, Document processing in a relational database system. *ACM Transactions on Office Information Systems*, 1, 143-158, (1983).

[8] I.A.MacLeod, A data base management system for document retrieval applications. *Information Systems*, 6, 131-137, (1981).

[9] C.J.Date, *An introduction to database systems*. 3rd edition, Addison-Wesley, Reading, Ma., (1981).

[10] W.B.Croft and R.Thompson, The use of adaptive mechanisms for selection of search strategies in document retrieval systems. To appear in *Proceedings of the Third ACM/BCS Symposium on Research and Development in Information Retrieval*, (1984).

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

[12] S.E.Robertson and K.Sparck Jones, Relevance weighting of search terms. *Journal of the American Society for Information Science*, **23**, 129-146, (1976).

[13] K.Sparck Jones, A statistical interpretation of term specificity and its application to retrieval. *Journal of Documentation*, **28**, 11-20, (1972).

[14] C.J.Van Rijsbergen, A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation*, **33**, 106-119, (1977).

[15] D.J.Harper and C.J.Van Rijsbergen, An evaluation of feedback in document retrieval using co-occurrence data. *Journal of Documentation*, **34**, 189-216, (1978).

[16] C.T.Yu, D.Buckley, K.Lam and G.Salton, A generalized term dependence model in information retrieval. *Information Technology*, **2**, 129-154, (1983).

[17] W.B.Croft, A model of cluster searching based on classification. *Information Systems*, **5**, 189-195, (1980).

[18] J.C.Gower and G.J.S.Ross, Minimum spanning trees and single-linkage analysis. *Applied Statistics*, **18**, 54-64, (1969).

[19] D.J.Harper, *Relevance feedback in document retrieval systems*. Ph.D. Thesis, University of Cambridge, England, (1980).

[20] K.Sparck Jones and R.G.Bates, *Research on automatic indexing 1974-1976*, British Library Research and Development Report 5464, Computer Laboratory, University of Cambridge (1977).

[21] R.N.Oddy, Information retrieval through man-machine dialogue. *Journal of Documentation*, **33**, 1-14, (1977).

[22] E.Horowitz and S.Sahni, *Fundamentals of data structures*. Computer Science Press, Potomac, Md., (1976).

[23] A.F.Smeaton and C.J.Van Rijsbergen, The nearest neighbour problem in information retrieval. Proceedings of the 4th ACM SIGIR Conference, *SIGIR Forum*, **16**, 83-87, (1981).

[24] F.Murtagh, A very fast, exact nearest neighbour algorithm for use in information retrieval. *Information Technology*, 1, 275-284, (1982).

[25] Digital Equipment Corporation. VAX-11 DBMS Utilities Reference Manual, (1982).