Direct file organization for lemmatized text retrieval

K Devine and F J Smith describe a system based on hash ordering and recording of semantic relationships between words

This paper discusses the design of a text retrieval system based on a direct file with hash ordering and including facilities for the explicit recording of semantic relationships between words, including grouping into lemmas. This design offers fast and accurate retrieval of groups of related words in return for an extra effort on the part of the indexer in defining the relations. Methods based on stem searching in an alphabetically-ordered file of indexed-sequential type are slow and imprecise by comparison. Two simultaneous indexes, one incorporating the semantic relations and the other omitting them, produce a particularly flexible retrieval environment, at acceptable cost with present-day hardware. The design has been proved in a working system, QUILL, which is in everyday use at Queen's University.

Keywords: information retrieval, file organization, lemmas, indexing

INVERTED FILES

Software systems for text retrieval have been in existence since the late 1960s. There have been numerous surveys of the field, including those by Martin and Parker[1] and by Hall[2]. A recent comparison of four systems is given by Goldsmith[3].

The present paper is concerned with systems which store material in full text form, and permit access to it through the original uncontrolled vocabulary, and which work in interactive mode, using an inverted file structure. We will discuss the implications of including in such systems semantic relations such as synonymy and homography, and we describe the design options for a system where such linguistic features are to be included.

Department of Computer Science, The Queen's University of Belfast, Belfast BT7 1NN, UK
Received 7 October 1983

Typically, an inverted file is located on two separate physical files. A word-list file (or index file, or types file) usually contains a fixed-length record for each type (i.e., distinct graphic word), and is organized in such a way that the record for any given word type can be quickly located. Each word record in the word-list file contains (amongst other things) a pointer to an area of the second file, the reference file (also known as the tokens or postings or concordance file), where a list of references is held to the locations of the word in the document texts.

Each physical disc block of the word-list file will have the capacity to hold a number of word records (typically 20–40), and the retrieval response time of the system will be largely determined by the number of blocks which have to be read in order to find a particular word record. There are two fundamentally different ways in which the word-list file may be structured to permit fast retrieval of the record for a specific word.

The first method is based on sorting the word records into alphabetic order. They may then be scanned with a binary search; or for greater efficiency, a single-level or multi-level index may be constructed. This is the popular indexed-sequential structure. If the ordering is achieved by means of pointers from index blocks to lower-level index blocks and ultimately to datablocks rather than by physically ordering the records on the file, we have B trees[4]. B tree structure has been proposed for the SIFT system[5]. At any time, most blocks are not full; if a block becomes full, half the records are moved to a new block and the index is amended.

HASHING

An alternative method of organization for the word-list file is division hashing[6], which has been used in systems

 0144–817X/84/010025–09$03.00

designed at Queen's University since 1969[7]; the latest of these systems is QUILL[8]. In hashing, we estimate in advance how many disc blocks will be required to store the word-list file. We form a numerical representation of the word (for example, using the ordinal values of its characters in the ASCII sequence), and divide this by the total number of blocks. The remainder is taken to be the number of the block in which this word record should be stored. We found from practical experience that it is satisfactory to use eight letters of each word in our hashing algorithm; hash keys based on the first four letters give a poor distribution[9].

The same range of options presents itself with regard to the order of records within a block: a second hash process, alphabetic ordering, or no ordering (i.e., as encountered in actual text). In the last case (which is the method used in QUILL), the block has to be scanned sequentially during retrieval; the optimal order is then decreasing frequency order, to which the text-derived order ought to be a reasonable approximation.

OVERFLOW

As with the B trees, hash blocks are filled gradually, and an overflow situation arises when a new word record is directed to a block which is already full. One solution (employed in QUILL) is to reserve a set of overflow blocks. An alternative method is to redirect the record by algorithm to a different primary hash block. We have found[10] that the linear quotient method[11] is convenient. If the number of blocks is a prime, this method will probe every block without repetition.

A hash file of insufficient size will result in much overflow and may require to be reorganized. If overflow is handled by reusing the primary hash blocks, a high degree of overflow will slow down retrieval, and eventually the file will be completely filled. On the other hand, when a separate overflow area is used, this may grow until it becomes comparable in length with the hash area, and again retrieval performance will deteriorate. A utility program is therefore required which can extend the hash file and recalculate block addresses for all records. Such a module is provided as part of QUILL.

The main advantage of hash files is their intrinsic speed of retrieval. In the absence of overflow, only one disc access is required to retrieve a word record. Even with overflow, if the most frequent words can be placed in the primary hash blocks, the average number of disc accesses can normally be kept below 1.1[10, 12]. The main disadvantages relate to storage space: hash files may be rather sparsely populated, and furthermore, it is necessary to keep an alphabetically-ordered version for user information purposes.

SEMANTIC RELATIONS

The static nature of hash files (in contrast to indexed-sequential files, or even B trees) is particularly useful if semantic relations between words, such as lemma-membership, synonymy, homography or hierarchical relations, are to be built into a database. Then pointers ('thesaurus pointers') must be set up between word records in the inverted file. For example, 'mouse' and 'mice' belong to the same lemma, but would be stored in unrelated locations. So with each of these words we would have to store the address of the other. With hash files such pointers are easily maintained. The only occasion when hash records are relocated is during extension of the file, and the QUILL module for this purpose preserves the semantic linkages.

We now discuss the usefulness and implementation of some linguistic facilities.

SYNONYMS AND LEMMAS

There are two principal ways in which word records may enter the word-list file: they may be deliberately inserted independently of any text, or they may be encountered in the process of scanning new text. In either case, words which are morphologically related (e.g., man, men or sing, sang, sung) will be treated as distinct by the system.

However, a user searching for a particular word will generally wish to retrieve references to plural forms, past tenses, etc. as well as to the cited form. QUILL accordingly allows a group of word types to be declared as equivalent; in effect, a single list of references is maintained for each whole group so defined, and this list is produced in response to a search request for any of the words in the group. (Homographs require different treatment as we describe in the next section.) QUILL accords no special status to any word in the group; all are treated as equal.

The choice of which words are declared to be equivalent is left to the user. It is envisaged that the facility will be mainly used to gather together members of lexical lemmas, as in the examples quoted above, but it could equally well be applied to words which are formally unrelated (e.g., buy, purchase). This is perhaps closer to what is usually understood by synonyms. However, we will confine ourselves to the lemma application in the remainder of this paper.

The ability to retrieve with individual words belonging to a lemma may be retained by creating two indexes within the same database, based on the same text material, with the lemma structure built into one index (which may be described as 'lemmatized') and with each word type separately indexed in the other ('unlemmatized') index.

To set up a lemma group, the user simply supplies a list of the words, which will generally be derived from inspection of the vocabulary accumulated from scanned text, although the list is allowed to contain words which are new to the database. Then, each word of the lemma will have, in its word record, a pointer to the word record of the next word of the lemma, and so on, until the circle is completed by a pointer back to the first word record. Further, each word record of the lemma points to the combined list on the reference file.

Each word record also contains the frequency of the word, or if the word belongs to a lemma, the frequency of

the lemma. This is convenient when we want to retrieve the frequency without the other reference information, as it avoids consultation of the reference file, but it is a drawback for indexing, since when a reference is added for one of the words in a lemma each word record of the lemma must be updated.

For example, to add an occurrence of a word which belongs in a lemma with five members will require two accesses (read and write) to the reference file and ten accesses to the word-list file, which amounts to an increase of 200% in disc activity simply to update the stored frequencies. The alternative is to store the frequency on the reference file only, although this will cost an extra disc access during retrieval if the frequency alone is required to be displayed.

A similar consideration applies to the pointer to the reference file which is stored in the word record. Indexing requires that we access the end of the list, not the start, but if we store the end position in each word record of a group, we have to change it whenever a reference is added. Instead in QUILL we store the start position (which is unchanged by updating), and the current next free position in the list is stored at the start of the list itself on the reference file. Note that the length of the reference list is not necessarily simply related to the frequency if references have been deleted by editing, leaving 'holes'.

HOMOGRAPHS

Homographs are ambiguous words, that is, word types belonging to more than one lemma. For example, the type 'lead' may mean 'to go ahead of' and share a lemma with types such as 'leads', 'leading', 'led', etc.; or it may refer to a metal and share a lemma with 'leaden'. Homographs are especially difficult to deal with in an information retrieval system. Automatic disambiguation using syntactic or semantic context may be possible in some cases, but not in general with present knowledge. We therefore choose to separate multiple meanings manually, and to provide in QUILL a mechanism to do this as easily as possible.

An ambiguous word is not assigned to a lemma group when using QUILL, but is declared to be ambiguous, and a list of 'alternative meanings' is provided for it by the user: these are terms, one for each distinct sense of the ambiguous word. These alternative terms may be words already in the inverted file, or if not, they will be introduced at this stage. Alternatives are permitted to belong to a lemma. For example, in the case of 'lead', a suitable declaration might be

Lead: lead-led, lead-leaden

The list of alternatives is stored as a piece of free text on a file called the 'comments file', which is also used to hold other textual information about the database. A pointer is set in the word record of the ambiguous word, to give the address on the comments file at which the list of alternatives is stored. During retrieval, if the user asks for an ambiguous word, the list of alternatives is printed out, and the user is asked to choose one of them instead.

While indexing new text, occurrences of an ambiguous word are formed into a reference list in the normal way, but access to this list is denied during retrieval. Instead, the indexer must periodically process the references stored for ambiguous words, and assign them manually to one or other of the declared alternative terms. Lists of the unprocessed references to ambiguous words with their contexts may be obtained to assist in this process.

We have found that this technique may not be practicable with large databases because of the human time required to distribute the ambiguous occurrences. We now believe that a better solution in such cases may be simply to associate each occurrence of an ambiguous word with all of its alternative terms. This will maintain high recall, but at the expense of precision. This is preferable to having a database in which the ambiguous words have their occurrences only partially distributed, because the indexer is short of time, thus losing recall. We plan to introduce this additional facility to QUILL.

WORDS WITH COMMENTS

Any piece of textual information may be associated with a word in the form of a free-text comment. This is stored on the comments file, and a pointer to it is set in the word record. During retrieval, the user can ask if a certain word has a comment associated with it, and if so the comment can be displayed. The comment may contain any chosen information; examples might be a list of near synonyms or a list of hierarchically superior or inferior terms.

If parts of this comments file are given a formal structure, with a precise syntax, the file can be used for purposes other than comments. For example, it might be divided into separate sections, e.g., for synonyms, for alternative meanings, for superior terms or for inferior terms. Each section can be introduced by a keyword, which could be detected by the retrieval program. The comment would be thus interpretable both to the user, if it is displayed, and to the computer.

For example:

cow: synonyms: cattle, kine, bovine;
superior terms: mammal, animal, livestock;
inferior terms: bull, heifer, bullock, calf;
comment: this word belongs to several semantic fields.

This example shows how the important semantic relations vary with the orientation of the database, in this case, whether towards zoology or agriculture. A mechanism of this sort was successfully implemented in an earlier retrieval system (QUOBIRD 2) in Belfast.

For synonyms, this method could complement the mechanism of lemmas, if it was felt that irreversible merging of some reference lists of individual words was not acceptable. For hierarchic relations, the use of comments will be much simpler than setting up many-to-many pointers between word records, and deciding whether or not to merge permanently the lists of references.

STEMS

It is useful in any text retrieval system to be able to access the word types in alphabetic order. One use of such a facility would be the production of complete printed lists or concordances; another would be the provision of online assistance in the form of lists of words with a common stem. As a result of hash ordering it is no longer possible to do this directly.

For this reason, QUILL maintains a second copy of the word-list file, sorted in alphabetic order. This file is purely for user information and for offline listing purposes; it is not involved in the online search process. It may be significantly smaller than the hash-ordered copy, since the latter may have a low packing density. In any case, the word-list file will be the smallest of the main files (smaller than the text file, or the reference file), for any but the smallest database.

Systems which have their word-list file in alphabetic form may use stems as an approximation to searching for a lemma, and this is in general a good approximation for a language like English. Cases like 'man' and 'men', where a stem cannot be found to include all the members of a lemma without excessive irrelevant material, are the exception. But searches with short stems can give low precision; for example, the stem 'cat' will retrieve many words of widely differing meanings (including for example 'catalogue') in a reasonably-sized English database, while words like 'kitten', which ought to be included for high recall, will be missed.

Table 1 gives some examples of using searches based on stems in a particular database. It can be seen that the stem 'law', for example, retrieves irrelevant items such as 'Lawrence' and 'Lawrie', which reduce the precision, and fails to retrieve 'legal' which may decrease the recall. If an indexer has manually grouped the word types into lemmas, however, these effects can be avoided, and further improvement can be achieved by using a still finer level of distinction: 'law' and 'laws' can be placed in one lemma; 'lawful', 'lawfully' and 'lawfulness' in another; and 'lawyer' and 'lawyers' in a third. In the following section, we will compare searches based on stems with those based on lemmas from the point of view of system efficiency, leaving considerations of precision and recall aside.

Languages other than English may pose additional problems for stem searches. Compound words are found in German and Swedish, for example, Weltanschauung (world view). Unless such words are divided at text input time, they will not be found by a stem search on other than their first element. In fact it is usually the final element of such words which is semantically dominant. In a system such as QUILL they can be treated in the same way as homographs, and their occurrences can be assigned to all of the relevant lemmas.

An interesting problem arises in the Celtic languages, where words may be modified grammatically at the beginning as well as at the end. While the initial mutations are quite regular in form and could easily be programmed, this type of facility is not offered by existing systems. Some systems may offer left-hand truncation, or more general pattern-matching, which will improve recall, but as with stems, precison may be poor. In our design, the mutated and unmutated forms of each word can be placed together in a lemma.

EFFECT OF LEMMAS ON SYSTEM EFFICIENCY

The maintenance of a single merged list of references for a lemma enables those references to be retrieved together in what we will call a lemma search. Alternatively if the reference lists for the individual word types are stored separately they must be merged during retrieval. This is the norm with stem-based methods of retrieval and we will call it a stem search. Clearly, during retrieval, a stem search for a group of words must be more expensive both in processor time and in disc accesses than a lemma search because of the extra merging of lists involved. The cost of this improvement in retrieval performance is an overhead during indexing. Another factor is the human effort in defining the lemmas, but this should result in lemmas which give better precision and recall than the use of stems.

Experimental evidence

We performed some experiments using QUILL on a VAX 11/780 computer to compare the numbers of disc accesses and the amounts of CPU time required for comparable stem and lemma searches. For this we used a database containing the first eight issues of the *Bulletin of Northern Ireland Law*[13] (about 1 Mbyte). A number of stems were chosen and all words in the database with these stems were retrieved by both methods, followed in each case by a print of a minimal amount of information from the last reference. The test data are listed in Table 1. QUILL does not provide automatic stem searching, but the equivalent computations are performed in response to Boolean requests such as

crimes OR crime OR criminal

where the words are placed in ascending frequency order for efficiency. The words in each group are then declared to QUILL to be equivalent, i.e., belonging to one lemma, and retrieval is repeated.

The results are presented in Table 2. The CPU time figures are averaged over several trials, while the disc access figures are invariant. The disc accesses represent the number of times QUILL attempts to read or write a block, which may vary in size between 1024 bytes (word-list file) and 4096 bytes (reference file). No account is taken of economies resulting from the possibility that a block to be read may already be in core.

Theory

While the data in the above examples are interesting, we must ask how typical they are of the resource savings

possible with lemma searching. This requires a theoretical investigation. To this end, suppose a lemma to be retrieved contains four word types, with frequencies f_1, f_2, f_3 and f_4. If the references have already been merged, their retrieval requires that we read

$$f_L^4 = f_1 + f_2 + f_3 + f_4 \quad (1)$$

references. On the other hand, if a stem is specified and causes the four individual reference lists to be read and merged using workfile areas, the number of references read and written is

$$f_S^4 = 7(f_1 + f_2) + 5f_3 + 3f_4 \quad (2)$$

This number includes writing the final merged list to workfile and re-reading it; this will be necessary given the normal practice in online retrieval of responding to the user's search request with summary information only, and allowing the user further passes through the retrieved references (e.g., to print some of them).

The formula for f_S^4 is derived on the assumption that only two lists can be merged at one time, and the strategy is such that at any stage the combined list of references for words already processed is merged with the references for the next word in sequence. For efficiency, the words would be arranged so that $f_1 \leq f_2 \leq f_3 \leq f_4$. The ratio

$$r_4 = f_S^4 / f_L^4 \quad (3)$$

varies between 3 (when one list — the last — is much longer than the others), and 5.5 (when all the lists are of about the same length). Other merging strategies are of course possible.

The general forms of equations (1), (2) and (3) for a lemma with n word types are

$$f_L^n = \sum_{i=1}^{n} f_i \quad (4)$$

$$f_S^n = (2n-1)f_1 + \sum_{i=2}^{n} (2(n-i)+3)f_i \quad (5)$$

$$r_n = f_S^n / f_L^n \quad (6)$$

where the f_i have been ordered so that

$$f_1 \leq f_2 \leq f_3 \leq \ldots \leq f_n$$

Limits on r_n are given in Table 3. The upper limits on the ratio (all words equally frequent) are given by

$$r_n \leq n + 2 - 2/n \quad (7)$$

Table 1. Groups of words making up a lemma[1] and their frequencies (f) in a test database. These lemmas are used in later comparisons of stem searches with lemma searches. In a lemma search all words of a lemma have their references stored in a single list

Stem:	JUDG[2]		LAW[3]		TERROR		BENEFIT		CRIM		OFFENCE	
		f		f		f		f		f		f
Types:	JUDGE	138	LAW	155	TERRORISM	13	BENEFIT	147	CRIME	13	OFFENCE	38
	JUDGE'S	5	LAWFUL	8	TERRORIST	10	BENEFITS	53	CRIMES	3	OFFENCES	57
	JUDGED	4	LAWFULLY	3	TERRORIST-TYPE	1	BENEFITTED	1	CRIMINAL	66		
	JUDGES	3	LAWFULNESS	2	TERRORISTS	4	BENEFITTING	1				
	JUDGES'	2	LAWS	6								
	JUDGMENT	34	LAWYERS	4								
	JUDGMENTS	38										

[1]It is not implied that all the words in one of these lists would be assigned to a single lemma in practice; precision and recall could in many cases be improved by other choices of lemma structure.
[2]JUDGEMENT (one instance) was omitted from both stem and lemma searches for a technical reason.
[3]LAWRENCE and LAWRIE (one instance each) were omitted from both stem and lemma searches; it would have been inappropriate to include them in a lemma with the other words, to which they are completely unrelated semantically.

Table 2. Experimental comparison of lemma search and simulated stem search for the lemmas in Table 1 and for a single-member lemma

Stem	No. of types per lemma	Disc accesses			CPU time		
		Stem search	Lemma search	Ratio	Stem search	Lemma search	Ratio
JUDG	7	30	3	10.0	0.61	0.21	2.9
LAW	6	27	3	9.0	0.60	0.25	2.4
TERROR	4	16	3	5.3	0.25	0.11	2.3
BENEFIT	4	15	4	3.8	0.41	0.20	2.1
CRIM	3	13	4	3.3	0.28	0.13	2.2
OFFENCE	2	6	3	2.0	0.19	0.14	1.4
HIRE-PURCHASE	1	3	3	1.0	0.12	0.12	1.0

Table 3. Ratio r_n of reference transactions needed in a stem search divided by reference transactions needed in an equivalent lemma search

Number of types per lemma (n)	Ratio (r_n)	Ratio, Zipf-averaged ($\bar{r}_n$)
1	$r_n = 1$	1.0
2	$r_n = 3$	3.0
3	$3 < r_n \leq 4.3$	3.8
4	$3 < r_n \leq 5.5$	4.4
5	$3 < r_n \leq 6.6$	4.9
6	$3 < r_n \leq 7.7$	5.4
7	$3 < r_n \leq 8.7$	5.9
8	$3 < r_n \leq 9.8$	6.4

The average ratio $\bar{r}_n$ for each n has been calculated, assuming that the frequencies of the n words follow a Zipf distribution[14]

$$f = \frac{K}{t} \tag{8}$$

where t is the rank of the word in order of decreasing frequency. If we substitute the above Zipf formula for the frequencies in equations (4) to (6) for r_n, we obtain r_n in terms of the n ranks, $t_1, t_2, \ldots t_n$. It is independent of the Zipf constant K. This formula may be averaged by an n-dimensional numerical integration over the ranks t_i as variables, each ranging from unity to infinity. In this process the t_i are independently chosen and then placed in order: $t_1 \geq t_2 \geq t_3$, etc., rather than integrating directly over a restricted region. (The results are insensitive to the upper limit as long as it exceeds about 500.)

If in addition to the data in Table 3, we knew the percentages of lemmas of different sizes in a corpus, we could predict an approximate figure for the average ratio of references transferred, for that corpus. We have available one such fully-lemmatized corpus of English-language legal material, containing 3534 types, arranged in 1769 lemmas[15]. The percentages are given in Table 4. Weighting the Zipf-averaged values of $\bar{r}_n$ with these percentages gives an overall mean ratio of 2.4. This figure is derived on the basis of the best lemmatization data available to us, and is probably typical of English text in general. It predicts that retrieval using predefined lemmas is less than half as expensive in terms of disc resources than the use of stems.

Comparison of theory with our experiment

Some of the results obtained in our experiment presented in Table 2 appear to be in disagreement with the theoretical conclusions in Table 3. This is because the theory counts reference transactions while the experiment measures block accesses, and direct proportionality will hold only for long lists.

All our test lemmas produce lists of references which are less than one block in size (which does not make them untypical). This gives rise to two main factors, working in opposite directions. Firstly, no workfile transfers are necessary, as intermediate lists can be held in central store. If all the workfile activity is eliminated from the theory, we find that we read

$$\sum_{i=1}^{n} f_i$$

references in both cases, but as a single list in the lemma case and as n lists in the stem case.

The second factor now enters: any list, however short, requires accesses to read or write it. QUILL needs three accesses to read a list from the word list and reference files, assuming that the primary hash block on the word-list file has not overflowed, and that the list on the reference file does not span blocks. Thus the experimental accesses reflect the numbers of lists read, and the ratio will approximate to n, the lemma size.

Indexing efficiency

A lemma structure involves some overheads when indexing new text, but since this process is not time critical (and may well be performed offline at off-peak times), we believe these costs are usually acceptable. When new text is indexed against a lemmatized word list, this process will take longer. We found by experiment with QUILL that disc accesses increased by a factor of about 2.2, and CPU time by a factor of about 1.2, compared with indexing without lemmatization. In lemmatized indexing, the reference lists being manipulated (though fewer) are longer, and more updating of pointers in word-list records is required.

If two indexes are maintained, one lemmatized and the other unlemmatized, indexing time is roughly the sum of the indexing times for lemmatized and unlemmatized indexes separately. In this case also, the sizes of the word list and reference files will be approximately doubled, compared with those for a single index.

Thus the total indexing resource goes up by a factor of about 3, somewhat greater than the factor by which it is reduced during retrieval. Retrieval, however, generally

Table 4. Lemma size distribution for a legal corpus

No. of types per lemma	% of lemmas
1	48
2	27
3	12
4	7
5	3
6	2
7	1

takes place at peak times while indexing can be at quiet times. Which system is more efficient also depends on the number of queries, and on the weighting attached to user convenience of faster retrieval with improved precision and recall.

USE OF THE QUILL SYSTEM

QUILL (Queen's University Interrogation of Legal Literature) was originally aimed at retrieval of legal documents[17], but there is no domain dependence in its design. It is written in FORTRAN 77 with minor extensions, and has been implemented on VAX 11/780 and ICL 2900 computers.

QUILL operates on free text, in which it recognizes three levels of units: words, sentences and documents. Retrieval criteria are stated in terms of words, and the retrieved segments which satisfy these criteria may be either sentences or documents. Besides the sentences which constitute its text, a document may contain up to nine other fields. Up to three separate indexes may be defined, each being based on the vocabulary encountered in some combination of fields and/or text (i.e., sentences). Alternatively, a finite chosen vocabulary may be input into an index before any text is scanned, and the index closed to any other words subsequently found in the documents. Indexes may also differ in their structuring of the vocabulary.

QUILL is presented to the user as two programs: a management program, which performs all functions required to establish and maintain the database, and a retrieval program. People who wish only to retrieve from the database need only learn to use the second program. The use of the management program will in general be confined to the database administrator.

The management program has modules to perform the following functions:

- to set up the files, and define the document formats (this is the data definition module),
- to input documents via terminals or offline,
- to index documents,
- to merge new material with existing material,
- to edit documents, whether or not already indexed and merged,
- to ascribe properties to words in the indexes, or to change such properties,
- to produce selective or complete lists of the words in the indexes.

The retrieval program is the one most likely to be used by persons with minimal training, and has been designed to be particularly straightforward in operation, with a help facility. LIST commands are available to inspect the words in the indexes (in terms of stems, synonyms, comments, etc.) SEARCH commands retrieve sentences or documents containing specified words in Boolean combination or immediate sequence. Having retrieved some material, a DISPLAY command may be used, or a further search may be made and combined with the existing search. The database administrator may tailor the retrieval language, to suggest more closely the actual content of a particular database.

A more detailed description of the use of the QUILL system is given in a manual by Devine and Smith[8].

CONCLUSION

We have discussed a design differing from the common indexed-sequential file structure for text retrieval. This design is based on a word list in hash order, and provides facilities for handling semantic relations between words. In addition to the speed of retrieval which hashing provides, this system allows a group of related words to be defined by the indexer (for example, this might consist of all the word types belonging to a lemma), and a single merged list of references is then automatically maintained for the whole group. Retrieval from such a list is at once faster and more precise than the use of stems.

It was found that a particularly flexible retrieval environment could be created by establishing both lemmatized and unlemmatized indexes to the same text, where the lemmatized index incorporates the semantic facilities, while the unlemmatized index does not. We believe that the reduced cost of processor time and the increased availability of cheaper disc storage in recent years, as compared with the situation in the 1970s, justifies the use of an advanced design employing multiple indexes, where these result in improved speed and accuracy of retrieval. Indexing is certainly slower, but retrieval can be faster by a factor greater than two.

The QUILL system is the latest in a series of successful practical implementations based on these principles. It has proved that the design we have presented is feasible with present-day hardware, in a variety of application domains including legal records, medieval Latin texts, museum records, gravestone inscriptions, newspaper indexes, phonetic atlases, medical bibliography and software documentation.

REFERENCES

1 **Martin, T H and Parker E B** 'Comparative analysis of interactive retrieval systems' *SIGPLAN Notices* Vol 10 (1975) pp 75–85

2 **Hall, J L** *On-line information retrieval 1965–1976: bibliography with a guide to on-line databases and systems* Aslib, UK (1977)

3 **Goldsmith, N** 'An appraisal of factors affecting the performance of text retrieval systems' *Information Technology: Research and Development* Vol 1 (1982) pp 41–53

4 **Ore, T** 'Structural requirements to a free-text retrieval system' Paper presented at a Symposium on Legal Information Retrieval in Europe, Strasbourg (11–13 June 1979)

5 **GIOM** *SIFT — searching in free text: design specifications* Government Institution of Organization and Management (1980)

6 **Maurer, W D** 'An improved hash code for scatter storage' *Commun. ACM (USA)* Vol 11 (1968) pp 35–38

7 **Higgins, L D and Smith, F J** 'On-line subject indexing and retrieval' *Program (GB)* Vol 3 (1969) pp 147–156

8 **Devine, K and Smith, F J** *QUILL: an on-line text retrieval system* The Queen's University of Belfast, Department of Computer Science Report No. CS026, Belfast, UK (1983)

9 **Carville, M** *A study in on-line text retrieval* PhD thesis, The Queen's University of Belfast, UK (1976)

10 **Jamison, J Q** *Studies in information processing* PhD thesis, The Queen's University of Belfast, UK (1977)

11 **Bell, J R and Kaman, C H 'The linear quotient hash code'** *Commun. ACM (USA)* Vol 13 (1970) pp 675–677

12 **Higgins, L D and Smith, F J** 'Disc access algorithms' *Comput. J. (GB)* Vol 14 (1971) pp 249–253

13 *Bulletin of Northern Ireland Law, Issues 1–8* SLS Legal Publications (NI), Faculty of Law, Queen's University, Belfast, UK (1981)

14 **Zipf, G K** *Human Behaviour and the Principle of Least Effort* Addison-Wesley, USA (1949)

15 **Johnston, J** *A lemmatized dictionary derived from a legal database* The Queen's University of Belfast, Department of Computer Science (in preparation) (1983)

16 **Devine, K, Walsh, M and Smith F J** *The computer database system for the Celtic-Latin dictionary.* To be published by the Royal Irish Academy, Dublin (1983)

17 **Campbell, C M and Smith, F J** 'Students searching legal texts by computer' *Computers and Law* No 11 (1977) pp 6–7