6 Search functions and search trees

This chapter describes the search functions available in Okapi from the system's point of view, and how its use of them is controlled by search trees. The topics discussed include string comparison, the possible results of an index search, conditional searching, Booleans, "hyper-Booleans", string searching, and search trees. These functions are used automatically, rather than at the request of the user, although the user is kept informed (see Section 7.5.2).

6.1 String comparison

When the system compares two strings it makes use of a special string comparison routine. This routine is essential. A simple character-by-character comparison would find neither of the following pairs of strings equal:

BBC B.B.C.
SMITH A.B. Smith, A B

Okapi's string comparison routine ignores case (i.e. it treats upper and lower case letters as if they were identical) and it ignores most non-alphanumeric characters such as punctuation. The characters are ordered in the following sequence:

space and hyphen (equal)
0-9
A-Z and a-z (equivalent).

The routine will also accept question marks in the sought term to indicate a match with any character in the index term. For example: SM?TH would be matched by SMITH or SMYTH. This feature has not been used yet, for user interaction reasons (see Section 7.4.3).

The routine returns one of four possible results. For example, the sought term "WOOD" is:

(i) higher than the index term "WOMB"
(ii) equal to "WOOD"
(iii) left-substring of "WOODS"
(iv) lower than "WOOL" (with mismatch before the end of the sought term)

The comparison routine also returns the lengths of the unmatched parts of the strings, which can be used to find the "nearest" term (higher or lower) where there is no exact match.

6.2 Possible results of an index search

When the index is searched for the presence of a given term there are three possible results:

— exact match
— partial match
— non-match

In the case of an exact match or a partial match the result of the index search will include the number of postings.

A partial match means that the sought term has no exact match in the index but there is at least one index term of which it is a left-substring.

Examples: "catalog" partially matches "catalogue"
"Smith A" partially matches "Smith A B"

Note that the string comparison result (i) (higher) listed in Section 6.1 does not correspond to any of the three possible results of searching the index. If the string comparison result is (i) (higher) the search continues forwards through the index until the result is (ii), (iii) or (iv).

It could be argued that a more useful type of result than "no match" could be returned when there is no exact or partial match. As soon as the current index term becomes greater than the sought term the system could go back one index term and decide which index term is "nearest" to the sought term.

This nearest term feature might be useful, for example, in searching for "WOODLOUSE" in an index containing the following terms:

WOOD
WOODLICE
WOODS
WOODWORKING
WOODWORM

there is no exact match and no partial match but "WOODLICE" matches on five characters whereas "WOODS" matches on only four characters. "WOODLICE" could therefore be returned as the nearest match.

In addition to regarding the longer matching length as better, a refinement might be to regard a match which terminated at a word boundary as more significant. Further rules would have to deal with cases where the end of the match was not the end of a word in both the sought and the index term, or where more than one index term matched the sought term up to a word boundary.

Imprecise string matching is discussed further in Section 9.4.1.

6.3 Conditional searching

An index search can be made conditional and be restricted to one or more types of data using "beasts". The concept of beasts was introduced in 5.2.3. Using the beast indicators in the index, searches can be restricted to a specified data type or combination of data types. There are several situations in which this is necessary or useful.

For example if the user is looking for items containing the personal name "Wood" the system will ignore postings for beasts such as "title word" or "subject word", and only select postings for "author word", "author phrase" or "general person". If it knows that "Wood" is an author (rather than a person as a subject) then it only selects postings for "author word" or "author phrase" beasts.

6.4 Booleans

Of the three basic Boolean operations AND, OR and NOT, only the more useful pair, AND and OR, are currently implemented in Okapi. They are

not explicitly available to the user but are called by the system when appropriate.

AND

For example, if the user requests a "known item" search for a book entitled "Lovers and Sons", and there is no exact match of this title phrase, the system will automatically carry out a Boolean operation searching for postings of "lovers" AND "sons", which should successfully find the title "Sons and lovers".

OR

One purpose of the OR function would be to achieve truncation, by OR-ing all the partial matches of a sought term. For example, "comput" partially matches "computable", "computation", "compute", "computer", "computing", OR-ing these terms would be the result of searching for "comput.....".

Similarly, suppose that there is no exact match for the author "SMITH A". In this case it might be helpful if the system were to OR all partially matching authors: "SMITH A B", "SMITH A D", "SMITH A J", "SMITH A J D", etc.

6.5 Hyper-Booleans

In some cases a simple Boolean operation is not appropriate or results in no postings. At this point the system can automatically perform what can be called a "hyper-Boolean", i.e. an operation that combines a number of simple ORs and/or ANDs.

Two such hyper-Booleans have currently been implemented. One is the "hyper-OR", which is a quorum search [1] incorporating term weighting and a cut-off rule. The other is the "H-AND" operation: a combination of a simple AND with a hyper-OR.

Hyper-OR

A number of different algorithms have been suggested for a quorum or weighted search. In most cases they are intended to form part of a front end to an existing IR system. One method is to save the set of postings for each term, and then to perform successive ANDs, of all possible

combinations of terms. The ANDs are performed on decreasing numbers of terms, and the results are presumed to be of decreasing relevance. This can be very costly and time-consuming for large sets if there are more than a few terms. Robertson and Bovey [2] have implemented quite an effective version of this approach in a recent project.

An OPAC is a somewhat different environment. Quick response is a high priority, but there is an advantage in being able to design the system to incorporate the algorithm, rather than having to graft it on.

The project team decided to implement a "hyper-OR", based on a very simple and elegant algorithm given by Harper in [3, Chapter 7].

The hyper-OR is called automatically by Okapi in certain cases where the user's original request cannot be matched exactly, and there are more than two words in the user input. It is designed to retrieve relevant, though not exactly matching, material without further intervention from the user.

In the hyper-OR the postings for several terms are merged, i.e. they are OR-ed, and a weight is calculated for each posting. The value of this weight is the sum of the weights assigned to the terms to which the posting relates (see below). Records can then be displayed to the user in weight order, which it is hoped will have some correspondence with their degree of relevance to the user's request.

For example, the user requests a subject search for:

"the use of microcomputers to teach the mentally handicapped"

this phrase contains five keywords or terms, for which there are the following postings:

mentally	(58)	107, 135, ...
microcomputers	(87)	104, 121, 135,138, ...
teach	(114)	107, 122, 145, 196,.....
handicapped	(132)	104,107, 135, 196,.....
use	(863)	102,104, 121,122, 150, ...

In the absence of relevance feedback, a reasonable weighting rule is one which assigns a weight to a term based on the inverse of the number of postings for that term [4]. The reasoning is that rare terms are better for

discriminating between relevant and non-relevant records than are common terms. In the example, the terms "mentally" and "microcomputers" would be assigned relatively high weights compared with "use" which occurs much more frequently.

The postings shown above would be displayed to the user in the following order:

```
135 (mentally, microcomputers,        handicapped     )
107 (mentally,                 teach, handicapped,    )
104 (          microcomputers,        handicapped, use)
196 (                          teach, handicapped,    )
etc
```

Using a weighting system can mean that postings for n rare terms are displayed before postings for n+1 common terms. In a strict quorum search, postings for n+1 terms are always regarded as more relevant than postings for only n terms. Note that a hyper-OR becomes a quorum search if all terms are assigned equal weights.

The hyper-OR can put quite a load on the system if some of the terms have many postings. This load can be reduced by implementing a cut-off rule. The terms are merged in weight order, i.e. rarest first. The cut-off rule prevents new postings being saved if they cannot score the minimum weight already achieved by the best **n** postings (where **n** is some predetermined minimum number of records to be retrieved). This helps to reduce the size of the merged set, since many postings have only to be read to see if they are new. If they are not new, they add to the weight of existing postings in the merged set.

H-AND

The H-AND is a combination of the hyper-OR (on up to eight terms) with a single AND. It can therefore be regarded as a hyper-OR with one compulsory term.

This function has not been used in Okapi yet, for lack of time. It could be used in a known item search, making the author the compulsory term and performing a hyper-OR on the title words. It could also be used in a

subject search where one topic is known to be regarded as essential by the user.

6.6 Computational aspects

Online reference retrieval systems invariably make use of temporary files. A temporary file is storage space, usually on disc, which is used as working space at run-time but does not contain any permanent information. Temporary files are typically used for saving sets of postings and for saving intermediate results of searches.

The implementation of Boolean and hyper-Boolean functions are the only parts of Okapi that currently require the use of temporary files.

Of the three simple Booleans AND, OR and NOT, it is OR that is most likely to create a set of postings too large to be held in core. The number of postings for "X OR Y" can never be less than the number of postings for "X" or "Y", whichever has more. Suppose there are 100 postings for "X" and 60 postings for "Y", then the number of postings for "X OR Y" cannot be less than 100. Conversely the number of postings for "X AND Y" and "X NOT Y" cannot exceeed 60 and 100 respectively. Each new term OR-ed tends to increase the size of a set, whereas each successive term AND-ed or NOT-ed will tend to reduce it. From a purely computational point of view AND and NOT are very similar and will usually be implemented with shared code.

The hyper-OR described in Section 6.5 has been implemented by performing a series of ORs, or two-way merges, which has the advantage of enabling the cut-off rule to be implemented, the result being a list of postings in address order. An alternative approach would be to save the current state of the system and to make use of the extra space to perform a multi-way merge. This could be adopted when large numbers of postings were involved, in order to speed up the process. However, it is in just such cases that a cut-off rule is useful.

Whichever method is used to perform the hyper-OR the resulting postings have to be sorted by weight either before, or while, being displayed to the user.

Okapi currently begins to display the results from a hyper-OR as soon as the merge is complete. It makes a number of passes through the merged list of postings (which is usually stored as a temporary file). On each pass it displays postings with a given weight (starting with the highest), and

looks for the next highest weight which will be used during the subsequent pass. Postings with the same weight will be randomly distributed through the temporary file, which results in random delays between the display of postings. The displayed postings, now in rank order, are also written to a new temporary file so that any user request to backtrack through the set gets a quick response.

The alternative is to sort the results into rank order before displaying any postings to the user. The user will then experience only one (longer) delay.

6.7 String searching

Explicit string searching (sequential scanning of source text) is not currently implemented. There are occasions when it would be of considerable use, for example in limiting by location, or in resolving certain other queries where the index does not contain enough information.

A string searching routine is used, however, during the full record display in order to highlight the first occurrence of the sought term in the actual record text (see Section 7.5.3).

6.8 Search trees

At each stage in a search Okapi makes use of a "search tree" in order to decide what to do next. The action taken by the system depends on what the user has requested, and the result of each index search, including the number of postings.

The search tree is a set of paths with branches or choices, which enables the system to carry out the most sensible search function at each stage of a search. Part of a search tree is shown in Figure 6.1.

The development of search trees is one of the most distinctive features in the design of Okapi. After the original idea had been formulated the search trees evolved through a process of discussion and trial and error. There is an approach to formalisation of the principles involved in a recent paper by Mitev and Walker [5].

An early version of Okapi's beast schedule made a clear distinction between personal and corporate names and this was reflected in early

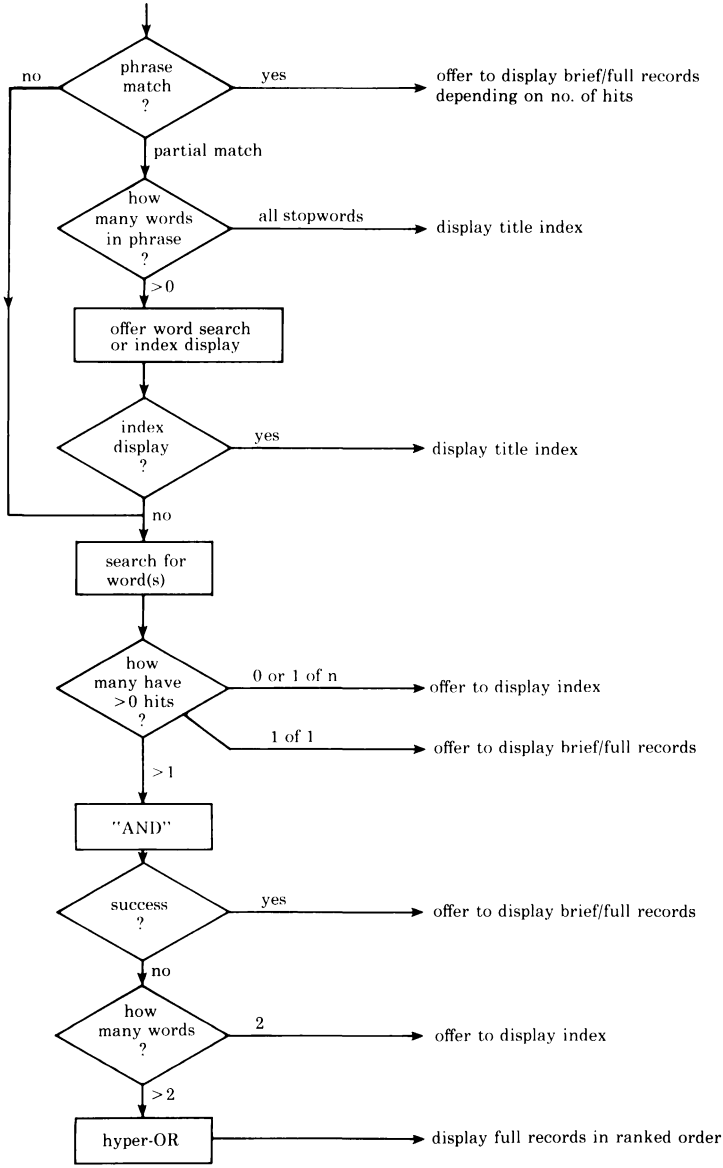

Figure 6.1. Title only search tree

versions of the search decision trees. However, experience showed that the data did not have correspondingly clear-cut categories. For example, the corporate name "OPEN UNIVERSITY" had sometimes been allocated a personal MARC name tag (100, 600 or 700), and such entries were therefore omitted from results of searches restricted to corporate names. This experience led to the adoption of the set of author and name beasts given in Section 5.2.3, which does not distinguish personal and corporate names. Instead there is a name phrase beast (a whole personal or corporate name) and a name word beast (a surname or a word from a corporate name). This practice of combining personal and corporate name data types also simplifies the input of authors (see Chapter 7); it has been followed in several other OPACs.

6.9 Functions not implemented and reasons why

A number of search functions commonly found in reference retrieval systems have not been implemented, namely:

NOT
explicit saving of retrieved sets for re-use
explicit truncation facility
explicit OR and AND
saving of search statements

These functions are less useful in an OPAC and are not really appropriate for the majority of users. The whole approach adopted for Okapi, which is described in detail in Chapter 7, is that the user should be able to express his or her request without having to bother about sets, truncation notation, the explicit use of Booleans, etc. In other words the system is to appear simple. This does not at all mean that the system actually is simple. In some ways the opposite is true. It is simpler to write a system which is commanded to perform each step in a logical sequence, than to write a system that decides for itself what is the best or most helpful course of action at each stage. A natural development of this approach is to engage the user in the process. For example it should be possible for the system to decide for itself that a term should be NOT-ed, as a result of obtaining feedback, or having a simple "conversation" with the user. This is discussed further in Section 9.4.4.

The team had hoped to build another level of interaction to cater for the more experienced or knowledgeable user. This would have probably included some of the omitted functions, perhaps using a command

language (see Section 7.4.1).

References

1 **Cleverdon C.** Optimising convenient online access to bibliographic databases. In: *British Library Research and Development Department. Seminar on Basic Information Research.* Cranfield Institute of Technology, 21-23 July 1983. Report by Mary Rowbottom. p13-15.

2 **Robertson S E** and **Bovey J D.** *A front-end for IR experiments.* Final report to the British Library Research and Development Department on Project Number SI/G/569. December 1983.

3 **Harper D J.** *Relevance feedback in document retrieval.* PhD thesis. University of Cambridge, 1980.

4 **Croft W B** and **Harper D J.** Using probabilistic models of document retrieval without relevance information. *Journal of Documentation 35* (4), 1979, p285-295.

5 **Mitev N N** and **Walker S.** Information retrieval aids in an online public access catalogue: automatic intelligent search sequencing. In: *Informatics 8: Advances in intelligent retrieval.* Proceedings of an Aslib/BCS conference. Oxford, 16-17 April 1985. To be published 1985.