CHAPTER 7

The CAMIR system

CAMIR is a collection of programs for running information retrieval experiments. The files handled by CAMIR consist of binary records in VBS format. Each record consists of N words (i.e. 4N bytes) where N is at least 2 and has no effective upper bound. The first word contains N, the length of the record. The second word contains a numeric identifier for the record in the file. This will be called the id-word. Words 3 to N contain data for the particular record. All, or almost all, data handled by CAMIR goes into this form, even when it not quite convenient. In fact this form of data is quite suitable for almost all of the experimental work that we have been doing.

Representation of documents

A document is represented by a record in the form

$$/ \ d \ t_1 \ w_1 \ t_2 \ w_2 \ \ldots \ t_n \ w_n \ /$$

d is the record id-word, and is simply the document number (1, 2, 3 ...). t_1, t_2, ... t_n are the term numbers for the document, sorted into increasing order, and w_i is a weight for the term t_i. For many experimental purposes the w_i will be irrelevant, and can be set to zero. Occasionally, however, it will have significance. Thus w_i could be the beta weight of Harter's model, which measures the usefulness of term t_i as a discriminator of document d. This form of document representation is not very compact, but can be processed rapidly.

The w_i will almost always be integers, and frequently w_i will be the largest integer contained in $1000f_i$, where f_i is a computed floating point number giving the weight of t_i in d.

If t_n is the last term, the record length will be 2n+2.

Representation of the other common files

The file of documents gives rise to an inverted documents file which is represented in an exactly analogous way.

$$/ \ t \ d_1 \ w_1 \ d_2 \ w_2 \ \ldots \ d_n \ w_n \ /$$

The terms t (1, 2, 3 ...) occur in the given documents with the given weights. A common use of w_i is as a measure of the frequency of occurrence of t in d_i.

The file of queries has records of the form

$$/ \; q \; t_1 \; w_1 \; t_2 \; w_2 \; \ldots \; t_n \; w_n \; /$$

q is the query number, and takes the values 1, 2, 3 ... through the file. Here it is very likely that significance will be attached to the w_i.

The file of relevance assessments, or rels, has the form

$$/ \; q \; d_1 \; w_1 \; d_2 \; w_2 \; \ldots \; d_n \; w_n \; /$$

The d_i are the relevant documents for the query q. The w_i are effectively dummy here, and may have the value zero.

The representation of other files will be described as they occur. The records in the files contain no indication of their type, which is no doubt a mistake. Care must therefore be taken not to confuse, say, a file of queries with a file of rels, since the system will not trap such an error. Errors of this kind can lead to program collapses.

The traditional form of IR data at Cambridge is as a series of numbers in text form. A group of numbers is terminated by a solidus, and a file of numbers by a further solidus. (This is described in [1].) To go from this form to the CAMIR form, use the command

```
    C MFP1.IR:BIN FROM <traditional> TO <CAMIR form>
```

and to go the other way, use

```
    C MFP1.IR:TXT FROM <CAMIR> TO <traditional form>
```

The defaults for FROM and TO are %C and %O respectively. (These commands are given in terms of phoenix, the command language in use on the IBM 370/165 at Cambridge. There is a certain limited dependence on phoenix in the description which follows.)

The CAMIR command language

CAMIR expresses its commands in a language which follows closely the form of the TRIPOS command language [2]. A command is followed by a list of keywords and arguments up to a semicolon. A keyword may have an associated list of qualifiers. The possible qualifiers are:

A - The argument must be present.
K - If the argument is present, the corresponding keyword must be

present.
S - The argument is absent (i.e. the keyword simply supplies a switch).
I - The argument must be an integer. An integer is a digit string, optionally preceded by a minus sign.

Thus, qualifiers AK mean that argument and keyword must always be present. K without A means that the argument together with the keyword may be present, but not the argument without the keyword. If K is absent the keyword may be omitted, in which case the arguments so employed in a command are interpreted positionally: A without K means that the argument must be present but the keyword is optional, and the absence of both A and K means that the argument is optional, and when it is used its keyword is also optional.

In describing a command, the qualifiers are written after their keyword and separated from it by a solidus. Thus the syntax of the EXPAND command may be described as follows:

```
    EXPAND FROM/ TO/ WITH/AK;
```

Here the WITH keyword, together with its argument, must always be present. The FROM keyword is optional, and its argument is also optional. TO is like FROM. The following are therefore equivalent

```
    EXPAND FROM A TO B WITH C;
    EXPAND A TO B WITH C;
    EXPAND FROM A B WITH C;
    EXPAND A B WITH C;
```

The arguments can also be reordered, so the above set is also equivalent to

```
    EXPAND WITH C FROM A TO B;
    EXPAND TO B A WITH C;
```

The arguments may also have default values. For EXPAND, C is the default value of the arguments of keywords FROM and TO. This may be expressed by writing

```
    EXPAND FROM/(C) TO/(C) WITH/AK;
```

And so the following are all equivalent

```
    EXPAND FROM C TO C WITH A;
    EXPAND C C WITH A;
    EXPAND TO C WITH A;
    EXPAND WITH A;
```

A few commands (e.g. DEL) do not quite have this shape. These exceptions will be explained as they occur. Note that an argument may be any sequence of characters not containing a space or newline character.

Keyword arguments as structure names or ddnames

The argument of a keyword will frequently be a reference to a CAMIR file. If the argument is prefixed with the # character, it gives the ddname of the file. For example

```
    EXPAND FROM #QS TO #QS2 WITH #MST;
```

QS and MST are the ddnames of two files on secondary storage. QS2 is the ddname of a new output file on secondary storage. A distinction is made in CAMIR between large and small files. The document and inverted document files are classed as large files. When reference is made to a large file by its ddname, the character # may be omitted. Small files constitute almost everything else, and after reading a small file by using the argument #QS, say, it will be in core as a structure with name QS. Output may also go either to an external file, using the # character, or to an internal file with the # character omitted. The command COPY, which has the syntax:

```
    COPY FROM/(C) TO/(C);
```

can be used for transferring structures between internal and secondary storage, and for making additional copies of structures. Thus

```
    COPY FROM #QS TO A;
```

reads the file from ddname QS and makes a copy of it as structure A. At the end of the operation, the file is also left in core as structure QS, so that in fact two copies of the file are created as structures. Equally,

```
    COPY FROM QS TO #OUT;
```

outputs the structure QS as a file with ddname OUT. The structure QS is not itself changed.

Unwanted structures may be deleted by the DEL command, which is followed by a structure list. e.g.

```
    DEL C QS QS2 MST ... ;
```

The current structure

Many of the keyword arguments default to the value C. C can be thought of as a current structure, which will be utilised unless some other structure is explicitly named.

BMATCH (Best match)

Syntax:
```
BMATCH FROM/(C) DOCS/K(C) TO/(C) CUTOFF/KI(10)
        DWEIGHTS/S QWEIGHTS/S;
```

The FROM structure is a file of queries (here and elsewhere, 'queries' refers to any record in the general form / q t_1 w_1 t_2 w_2 ... /) and DOCS supplies the source of the document file. Each query and document gives rise to a match value. This will be one of the following:

a) The number of common terms between the query and document (if DWEIGHTS and QWEIGHTS are both uset).

b) The sum of the query weights for the common terms (if QWEIGHTS only is set).

c) The sum of the document weights for the common terms (if DWEIGHTS only is set).

d) The sum of the product of document weight and query weight for the commom terms (if DWEIGHT and QWEIGHT are both set).

CUTOFF gives the maximum number of documents to be included per query in the resulting TO structure. The TO structure itself consists of records of the form

/ q d_1 m_1 d_2 m_2 ... d_k m_k /

This gives the set of documents d_1 ... d_k which give the highest match values for the query q. They are arranged in decreasing m_i order, or, when the m_i values are the same, in increasing d_i order.

BWEIGHTD (Beta weights for documents)

Syntax:
```
BWEIGHTD FROM/A TO/A WITH/AK;
```

The WITH file contains the l, m, h values for each term in the collection, derived from Harter's analysis of term frequencies. WITH will have been produced by the HARTER command (q.v.). FROM is the document file, and TO will be the same document file with Harter's beta weights added in.

If term t has parameters l, m, h in the Harter model, and its frequency in a particular document is k, its beta weight will be

$$\frac{s_1}{s_1 + s_2} + \frac{l - m}{\mathrm{sqrt}(l + m)} \qquad \text{(i.e. } P(d \text{ in } I|k) + z)$$

where $s_1 = h\, e^{-l}\, l^k$

$s_2 = (1-h)\, e^{-m}\, m^k$

See p 284 of [3].

The beta weights are expressed in fixed point form by multiplying by 1000 and taking the integer part.

CHOOSE

```
Syntax:  CHOOSE FROM/(C) TO/(C) WITH/K(RELS) COMPLEMENT/S
                NULLTONULL;
```

This is typically useful when the FROM structure is a set of document--match values, derived from BMATCH, and the WITH structure is a set of relevance assessments. The TO structure can then be a reduced version of the FROM structure, containing only relevant documents.

More generally, FROM is a structure whose nth record is

$/ \; q \; d_1 \; m_1 \; d_2 \; m_2 \; \ldots \; d_k \; m_k \; /$

and RELS is a structure whose nth record is

$/ \; q \; D_1 \; x_1 \; D_2 \; x_2 \; \ldots \; D_l \; x_l \; /$

The D_i must be in increasing order, the x_i are dummy. With COMPLEMENT and NULLTONULL both unset, the nth record of TO will be

$/ \; q \; d_1' \; m_1' \; \ldots \; /$

where every d_i m_i for which d_i is not also a D_j has been removed.

If COMPLEMENT is set, every d_i m_i for which d_i is also a D_j is removed.

If NULLTONULL is set, the nth record of TO will be / q / whenever the nth record of RELS is / q /. This is for certain evaluation purposes where one wishes to kill off queries which have no relevant documents.

COPY

Syntax: COPY FROM/(C) TO/(C);

This has already been descibed. A copy is made of the FROM structure and it is written to the TO structure.

DEBUG

Syntax: DEBUG;

This causes a global debugging flag to be switched on. Subsequent commands may test for this flag and produce monitoring information if it is set.

DEL (Deletion of structures)

Syntax: DEL <list of zero or more structure names>;

The given structures are deleted from core, thereby making more space available to the system.

EVAL (produces a P-R graph from the lin-file)

Syntax: EVAL FROM/(C) TO/(SYSOUT);

FROM is a structure called a lin-file, which will have been produced by MAKELIN (q.v.). TO gives the ddname of the text file to which the precision-recall graph derived from the lin-file is directed. By default it goes to SYSOUT, the output log stream for CAMIR.

The precision-recall values are worked out by producing precision values for each query with a non-null list of rels at standard recall positions (0.1, 0.2, ... 1.0) and then averaging. This is fully described on pages 194-195 of [4] and in chapter 7 of [5].

The "graph" in fact simply consists of a listing of the pairs of R-P values.

EX (examine structures)

Syntax: `EX;`

The structures currently in the system are printed out by name, together with the number of words of core that they consume.

EXPAND (query expansion via an MST of terms)

```
Syntax:  EXPAND FROM/(C) TO/(C) WITH/AK
                A/IK B/IK C/IK PRINT/S;
```

FROM is a structure representing a set of queries. The resulting TO structure is the same set of queries, but with additional terms given by the term classification of WITH.

WITH consists of a single record, as follows

$$
\begin{array}{l}
/\ 1 \\
\quad t_1\ u_1\ m_1 \\
\quad t_2\ u_2\ m_2 \\
\quad \ldots \\
\quad t_k\ u_k\ m_k\ /
\end{array}
$$

The record id-word, 1, is of no significance here, but it is important not to omit it. The record provides a term classification. Term t_i is linked to term u_i, and m_i gives the strength of the link. Frequently this record will correspond to an MST of terms, in which case two adjacent nodes in the tree, t and u, which are linked together with strength m, will give rise to two triplets in the record, namely

```
t u m
u t m
```

The initial order of the triplets in the record is irrelevant - they are reordered after input by increasing t, and when the t is constant by decreasing m, and when the m is constant by increasing u. m must be in integer form.

A, B and C are used to control the expansion process. Each query term t will have a class of terms [u] associated with it, where t u occurs as the first two terms of a trio in the WITH structure. In other words [u] is the class of terms to which t is linked. The first C of these terms with the highest m values are taken, or the whole set [u] if it contains fewer than C terms.

This is done for each term t in the query, and the result is a class of terms obtained by collecting all of these u-terms together. This class, U, contains each u-term and its corresponding m value. If two members have the same u, e.g. u m_1 and u m_2, the pair with the smaller m is removed. Then the class is arranged by decreasing m, and all but the first AQ+B terms are discarded, where Q is the size of the original query.

Thus there are two controls in operation. When expanding t, only the strongest C links are taken. Then the query is expanded only by the linear factor AQ+B of the original size.

C is given as an integer, A and B as percentages (i.e. multiplied by 100) e.g.

```
    ... A 50 B 125 C 2;
```

By default A, B and C are infinite. If A is set, B defaults to zero, and if B is set A defaults to zero.

GETRECS (forming a subfile)

Syntax: GETRECS FROM/K(DOCS) TO/ WITH/AK;

The WITH structure contains a single record of the form

$$/ \; 1 \; id_1 \; id_2 \; \ldots \; id_k \; /$$

The id-entries give a list of id-words which identify a set of records in the FROM file. The resulting TO file contains just this set of records. The list id_1 ... id_k must have the same order as their corresponding records in the FROM file. The FROM and TO files are large files - i.e. not representable by in-core structures.

GETTERMS (term statistics from a feedback set)

Syntax: GETTERMS FROM/(C) TO/(C) DOCS/K(DOCS);

The FROM structure is a file of documents (typically known relevant documents) in which the id-word may be interpreted as a query number. Thus the nth record of FROM has the form

$$/ \; q \; d_1 \; w_1 \; d_2 \; w_2 \; \ldots \; d_R \; w_R \; /$$

The w_i are dummy. DOCS supplies the source of the document file. The output structure TO contains information about the terms in the documents d_i, and will be called the t-info file. From a t-info file feedback weights may be computed (see NEWQS). The nth record of TO has the form

$$/\ q\ R\ t_1\ r_1\ t_2\ r_2\ \ldots\ t_m\ r_m\ /$$

$t_1 \ldots t_m$ are the terms in the documents $d_1 \ldots d_k$ and r_i is the number of these documents in which t_i appears. The t_i are in increasing order.

<u>HARTER</u> (computes l, m, h of the Harter term model)

Syntax: HARTER FROM/A TO/A;

The FROM file is the inverted documents file, in which the weight w_i attached to each term t_i is the term's within-document frequency. In the Harter model, the probability that a term occurs k times in a document is given by

$$P(k) = h \frac{e^{-l}\ l^k}{k!} + (1-h) \frac{e^{-m}\ m^k}{k!}$$

The parameters l, m and h are computed for each term using the data giving its distribution in the inverted document file, and with Harter's estimation method (see JASIS July-August 1975 p 202). The output in the TO structure consists of the four records

$$/\ 1\ h_1\ h_2\ \ldots\ h_T\ /$$
$$/\ 2\ l_1\ l_2\ \ldots\ l_T\ /$$
$$/\ 3\ m_1\ m_2\ \ldots\ m_T\ /$$
$$/\ 4\ n_1\ n_2\ \ldots\ n_T\ /$$

h_t l_t, m_t are the values of pi, l and m for term t. The total number of terms in the collection is T. n_t is the number of documents in which term t occurs, a statistic which is useful in later weight estimation.

The values in the first three records are given as 32-bit floating point numbers. So the command WRITE, which expects fixed point integers everywhere, cannot be used with this particular structure. Instead HSTATS is used.

If DEBUG has been obeyed, the various stages of the evaluation process are printed out.

HSTATS (prints out the Harter parameters)

Syntax: `HSTATS FROM/ TO/(SYSOUT);`

This prints out on dd TO the structure FROM, which is the output from HARTER. The result is in five columns: i, l, m, h, z. i is the term number. z is (l-m)/sqrt(l+m). Each of l, m, h and z are printed out as the integer part of the original value multiplied by 100,000.

HWEIGHT (term weighting based on the Harter model)

Syntax: `HWEIGHT FROM/(C) TO/(C) WITH/AK MODE/AIK;`

FROM is a set of input queries, and TO a set of output queries, identical except for the new weights. The weights are expressed as integers after some suitable scaling. The WITH structure is the output from HARTER (q.v.). MODE supplies an integer which determines the weighting scheme. The weight is a function of l, m, h and n for each term.

The mode-function correspondence has been altered continuously to run various experiments. At present we have the following:

mode	function	
1	log(D/n)	where D is the document count
2	1+log(D/n)	
5	h=0 -> 0, m=0 -> 10, log(l/m)	
6	1/sqrt(n)	
7	$(1+\log(D/n))^2$	
8	(l-m)/sqrt(l+m) = z	
9	1	
10	z(1+log(D/n))	where z is as in 8.
etc.		

MAKELIN (making the lin-file)

Syntax:
```
MAKELIN FROM/(C) MATCH/K(MATCH) DOCS/K(DOCS) TO/(C)
        DWEIGHTS/S QWEIGHTS/S EXCLUDE/K;
```

MAKELIN is used in conjunction with RMATCH. RMATCH produces as output a match file, which is used as the argument of MATCH in MAKELIN. To be meaningful, the queries in the FROM structure, the document file of DOCS, and the settings of DWEIGHTS and QWEIGHTS must be the same as in the corresponding RMATCH command. A record from the TO structure has the form

/ q d_1 p_1 d_2 p_2 ... d_n p_n /

d_1 ... d_n are the documents associated with query q. If the document-match pairs for every document in the collection are collected for query q, they may be ordered by decreasing match value, and, when the matches are the same, by increasing document number. Then p_i gives the position down the list of document d_i in this ordering.

If EXCLUDE is set, it determines a file whose nth record has the form

/ q D_1 w_1 D_2 w_2 ... D_k w_k /

The w_i are dummy. D_1 to D_k specify a list of files which are to be regarded as absent from the system in the construction of the lin-record for q. This option is useful in residual evaluations of retrieval performance.

NEWQS (feedback query weighting)

Syntax: NEWQS FROM/(C) TO/(C) WITH/AK TTOT/AK WEIGHT/K;

The structure FROM supplies a list of queries, and the resulting structure TO is identical except for the new weights (cf. HWEIGHT). The WITH structure is a t-info file produced by GETTERMS (q.v.). TTOT is a structure composed of one record:

/ 1
n_1 n_2 ... n_T /

and n_t is the number of documents in which term t occurs. (Cf the fourth record generated by HARTER, which could be used for TTOT here.) WEIGHT supplies the appropriate weighting scheme. The possible values are:

WEIGHT MILLER4

The weight for t_i is

$$\log \frac{(r_i + 0.5)(D - n_i - R + r_i + 0.5)}{(n_i - r_i + 0.5)(R - r_i + 0.5)}$$

where R, r_i are from the t-info record, and D is the collection size.

WEIGHT EIQ

The weight is

$$\log \frac{N_{00}\,D}{N_{0.}\,N_{.0}} + \log \frac{N_{11}\,D}{N_{1.}\,N_{.1}} - \log \frac{N_{01}\,D}{N_{0.}\,N_{.1}} - \log \frac{N_{10}\,D}{N_{1.}\,N_{.0}}$$

where $N_{11} = r_i$
$N_{01} = R - r_i$
$N_{10} = n_i - r_i$
$N_{00} = D - n_i - R + r_i$

and $N_{0.} = N_{00} + N_{01}$ etc.

The weights are scaled and put into integer form.

PARS (setting up the collection parameters)

Syntax:
```
PARS DCOUNT/IK TCOUNT/IK;
```

This sets up the two parameters D, the number of documents in the system, and T, the number of terms in the system. It should always be used prior to any sequence of CAMIR commands. DCOUNT can be readjusted for the benefit of weight calculation, e.g.

```
PARS DCOUNT 2500;
NEWQS ... ;
PARS DCOUNT 11429;
```

RMATCH

Syntax:
```
RMATCH FROM/(C) DOCS/K(DOCS) RELS/K(RELS) TO/(C)
       DWEIGHTS/S QWEIGHTS/S;
```

The FROM structure is a set of queries, the RELS a set of relevance assessments. DOCS gives the source of the document file, and DWEIGHTS, QWEIGHTS have the same interpretation as in BMATCH. The resulting TO structure is a copy of the RELS structure with records of the form

$/ \; q \; d_1 \; m_1 \; d_2 \; m_2 \; \ldots \; d_n \; m_n \; /$

where m_i is the match value between query q and document d_i.

WEIGHT

Syntax: WEIGHT FROM/(C) TO/(C) WITH/AK MULT/S;

The structure FROM is a set of queries, TO has the same shape as FROM with weights taken from WITH. WITH is a single record in the form

```
/ 1
  w1 w2 ... wT /
```

w_i is then the weight for term i. If MULT is set, the w_i weight is multiplied into the term weight already in the FROM structure to get the new term weight.

WEIGHTD

Syntax: WEIGHTD FROM/A TO /A WITH/AK MULT/S;

FROM refers to a document file. TO is a corresponding output file with new weights taken from WITH. WEIGHTD does for the document set exactly what WEIGHT does for the smaller query set. The distinction is that the document file cannot be an in-core structure.

WRITE

Syntax: WRITE FROM/(C) TO/(SYSOUT);

The FROM structure is written to the dd TO in "traditional" form.

Example of the use of CAMIR

As an example, suppose that we want to get a listing of the 10 documents closest to each relevant document of query 37 in the NPL test collection, closeness being measured by adding up the i.d.f. weights for the common terms. To pull out the reldoc numbers for query 37, we do the following:

```
SET MFP1
INPUT
1 37 /
/*
C .IR:BIN TO &S
```

&S now contains a suitable WITH file for the command GETRECS. We run this command using the following piece of Phoenix:

```
    MFP1.IMODS:L(%S=200¦0 %D='$40K250G$' %LIB=MFP1.IMODS
                 FLIB=SYS1.FORTLIB
                 SYSPRINT=%M/N  SYSIN=%H!
    /VBS
    U=&U/N QS=.VAS.XRELS S=&S)
    GETRECS TO #U FROM #QS WITH #S;
    !
```

and this can equally be done by the command sequence:

```
    SET P IRCOM  'GETRECS TO #U FROM #QS WITH #S
    SET P IRENV  'U=&U/N QS=.VAS.XRELS S=&S
    C .IR:DO
```

Similarly, to pull out and print the reldocs themselves, we do:

```
    SET P IRCOM  'GETRECS TO #R WITH #LIST
    SET P IRENV  'R .VAS.Q37DOCS/N LIST &U
    C .IR:DO
    C .IR:TXT FROM .VAS.Q37DOCS TO &A
    PRINT &A
```

- where DOCS has been set up by a JCL line:

```
    //DOCS DD UNIT=TAPE9,...
```

PARS has not been obeyed here, since the GETRECS clearly does not require the TCOUNT or DCOUNT values. Getting the set of closest 10 documents to each of the reldocs can now be done by

```
    //DOCS DD DSN=VAS.XDOCS.ABF,LABEL=2,VOL=SER=MFP106,UNIT=TAPE9
    SET MFP1.VAS
    MFP1.IMODS:L( %D='$35K250G$' %S=200¦0 %LIB=MFP1.IMODS
        FLIB=SYS1.FORTLIB
        HWEIGHTS=.XHARTER
        QS=.Q37DOCS
        MATCH=.Q37MATCH/N/VBS
        SYSPRINT=%M/N SYSIN %H!
    )
    PARS DCOUNT 11429 TCOUNT 7491;
    HWEIGHT FROM #QS WITH #HWEIGHTS MODE 2;
    DEL QS HWEIGHTS;
    BMATCH TO #MATCH CUTOFF 10 QWEIGHTS;
    !
    C MFP1.IR:TXT FROM .Q37DOCS TO &A
    PRINT &A
```

REFERENCES

1. SPARCK JONES, K. and BATES, R.G., Research on automatic indexing 1974-1976, British Library Research and Development Report 5464. Computer Laboratory, Cambridge, 1977.

2. RICHARDS, M. et al., TRIPOS - a portable operating system for mini-computers. Computer Laboratory, Cambridge, 1979.

3. HARTER, S.P., A probabilistic approach to automatic keyword indexing. Journal of the ASIS, 26, 1975.

4. HARPER, D.J. and VAN RIJSBERGEN, C.J., An evaluation of feedback in document retrieval using co-occurrence data. Journal of Documentation, 34, 1978.

5. VAN RIJSBERGEN, C.J., Information retrieval. Second edition. Butterworths, London, 1979.