THE REFERENCE STRING INDEXING METHOD

H.-J. Schek
IBM Wissenschaftliches Zentrum
Tiergartenstrasse 15
D-6900 Heidelberg

SUMMARY

The motivation for the reference string indexing method may be derived from the intention to retrieve any piece of information by specifying arbitrary p a r t s of it. Common restrictions such as the usage only of a certain set of descriptors or (complete) keywords in document retrieval systems or the specification of only certain (inverted) attribute values for queries in formatted files should be removed without loosing performance necessary for interactive usage.

The solution to be described is essentially based on the realistic assumption that the frequency distribution for the occurrence of character strings with a certain length, or words, or word sequences in textual files, and also for the occurrence of attribute values or value combinations in formatted files is n o t uniform but rather highly hyperbolic or "Zipfian". The same is valid also for the u s a g e of data, expressed as the "80-20"-law. Exploiting this assumption, a (small) set of "reference strings" is generated by a statistical analysis of collected queries or - if not available - by usage estimation with the original data. The inversion to these reference strings with respect to records or record clusters gives the reference string index.

Corresponding to the estimated usage frequency, a search argument may have been made available completely as a reference string or has to be decomposed into shorter reference strings. Therefore, the reference string access is adaptive with the consequence that a routine query may be answered faster than a non-routine one.

The reference string index may be applied as a new adapted index in information retrieval systems as well as in formatted files as single or multi-attribute index. In addition it can be applied for phonetic and general record similarity search.

1. Introduction

The reference string indexing method is related to problems often called partial match retrieval, associative search or multi-attribute retrieval. A directly accessible piece of stored information (record, block or cluster) which is represented as a string of characters shall be retrieved (and optionally subsequently updated) by specifying one or several a r b i t r a r y fragments (or substrings, fragmentary information) which must be contained in the desired record(s).

This provides a more general "accessing by contents" than by preassigned and often artificial keys. So, the main objective of the proposed access method is to support formal search functions summarized as functions for "partial match retrieval" Direct access becomes possible in cases where serial file scanning often has been the only solution until now. Examples for such searches are queries specifying values for not inverted attributes in formatted records, queries with partially specified attribute values, search with phonetic patterns, and nearest neighbour search. In text retrieval systems with automatic indexing, an important application of the reference string access method is the search with f r a g m e n t s of keywords which appear e.g. as c o m p o n e n t s of chemical substances or in c o m p o u n d w o r d s of a natural language. In formatted files the reference strings provide a p a r t i a l inversion either for a single attribute, or for several attributes as a new multi-attribute index, in both cases in adaptation to the data.

The idea of the reference string access method is easily explained by an example: The set of all records which contain somewhere the string 'CHROMAX' is contained in the set of all records which contain somewhere both strings 'MAX' & 'CHROMA' which, in turn is contained in the set with 'AX' & 'CHROM' & 'OMA'.

Obviously, many more decompositions would be possible. However, decompositions of this kind motivate the idea of introducing a set of standard strings which are used in every decomposition. These strings will be called reference strings or shorter

"refstrings" and may or may not be meaningful strings of characters. The only purpose of a refstring is to give a reference into a record where it occurs. So it is assumed that each refstring has an inverted list which means that a list of accession numbers, unique keys, or logical pointers of those records is known which contain somewhere the corresponding refstring as a substring.

The automatic determination of a reasonable set of refstrings is the key problem and the main idea for its solution is to determine substrings which have a high r e f e r e n c e frequency, in other words, which have a high probability to be n e e d e d for a reference into the records. Estimates for these probabilities may be derived from an analysis of the queries used in the past and by extrapolation to future usage. For a first estimation of usage frequencies, letter and word resp. attribute co-occurrence statistics of the original data itself are evaluated to give an initial set of refstrings.

It is important to observe that two facts are exploited essentially in this approach. The distribution of the reference frequency of data as well as the distribution of the occurrence and co-occurrence frequency of letters or syllables in words or words in texts etc. is not uniform but rather hyperbolic. The first is known as "80-20-rule" /KNU73/ and the second as Zipf's distribution /ZI49/. This means that only a small quantity of data is used in most of the queries resp. it means that a small set of different strings covers the whole (textual) file. With respect to the refstring index it means that the size (number of different refstrings) can be kept small even if one prescribes a high mean selectivity.

The reference string indexing method has been developed starting in 1974 with the problem of similarity search in keyword lists including the misspelling case and phonetics /SCHE75/. In the more general context here and in its further extensions it has relations to the following literature: determination of "equifrequent character strings" and application for text searching and data compression /BA74, CLA72/, substring and pattern matching /AHO75, KNU74, MCR74, HA71/, partial match

retrieval and related index or tree (trie) constructions /RI76, BU76, BE75, WO71, LU70/ , term association and statistical thesaurus constructions /STE74, SA68, LUS67/, general indexing and access methods /WE75, WA75, BAY73/. The following is a shortened and revised version of /SCHE77/.

2. The Reference String Index

In order to include the most general case it is assumed that the file P from which data have to be retrieved contains directly accessible records. Each record may be formatted in fields (attribute structure) or may be without format (free format case like in natural language texts) or it may be a mixture between both which means that certain attributes may contain a textstring as attribute value such as a title of a book. Without loss of generality it is further assumed that attribute values appear in their character representation. The reference string index is motivated by the intention to retrieve any record by specifying arbitrary parts of it. One or several strings which must occur in the matching records as substrings - disregarding attribute or word positions for the moment - shall be sufficient for retrieval.

2.1 Refstrings for Substring Matching

Obviously, a solution for the retrieval-by-parts problem consists in the solution of the substring or pattern match problem. However, in the context here methods are not allowed which scan the whole file, even though there are sophisticated methods. The reason is that one should avoid to transfer the whole file from external to internal storage. Therefore, one would like to have a fast method which allows to decide which records contain somewhere a certain substring x, or somewhat weaker, which may contain x with a high probability.

A first, unrealistic solution would be to provide an index for each possible substring which occurs in P. The size of such a complete substring index would be prohibitively high and is also not necessary as the following considerations show:

1. Only those substrings which have a high probability to be specified in queries should be included in the substring index called refstring index with the refstrings as entries.

2. If a search argument x is not contained in the refstring index but if x contains substrings y1,y2,...,yn which are refstrings, then every record which does not contain all yi,i=1,...,n must not contain x and can be excluded from processing.

3. In order to support an arbitrary substring search (meaning that at least one decomposition into refstrings is possible) either all single characters, or all character pairs, or all character triples are defined to be refstrings too and are included in the refstring index.

The second principle /HA71/ may introduce false drops: A record containing all substrings yi of x will not necessarily contain x. To express it more precisely: Let be J(x) the set of all record numbers of those records containing x anywhere. Then the following corrolaries hold

C1: If y substring of x then $J(x) \subseteq J(y)$

C2: If x contains y and z as substrings then $J(x) \subseteq J(y) \cap J(z)$

C3: If x contains y and y contains z as a substring then, instead of (2) one has $J(x) \subseteq J(y)$ (intersection with J(z) is redundant).

C1 is easily be proved, C2 and C3 are consequences from C1.

The first and third consideration lead to the introduction of two disjoint sets of refstrings, namely the set of basic refstrings BRS and the set of additional refstrings ARS with the following definitions.

Def. BRS: The set of basic reference strings $BRS=\{brs_1,...,brs_m\}$ contains a 1 1 strings of a certain length k which

occur in P (k=1, or k=2, or k=3).

Def. ARS: The set of additional reference strings $ARS=\{ars_1,\ldots,ars_n\}$ contains certain refstrings with length greater k.

Because of this definition one has for the set of all refstrings RS = BRS ∪ ARS and BRS ∩ ARS = 0. The determination of ARS is described in chapter 3.

The motivation for these definitions will be clear if one applies RS for arbitrary substring searches: Let x be a search string with length between k and m. Then the list of record numbers J(x) which contain x as a substring is desired. For that purpose relate to x the set of all substrings in x and denote it by X. further, determine the refstrings in x

$$AXRS := X \cap ARS, \quad BXRS := X \cap BRS, \quad XRS := AXRS \cup BXRS$$

Because all substrings with lenght k of file P are in BRS one has

A1: If BXRS = $\emptyset$ then x does not occur in P.

If this trivial case is excluded the set $XRS=\{xrs_1,xrs_2,\ldots,xrs_l\}$ has at least one element. In application of (C2) using a 1 1 refstrings one has

$$A2: \quad J(x) \subseteq J'(x) := J(xrs_1) \cap J(xrs_2) \cap \ldots\ldots \cap J(xrs_l)$$

In A2 redundant intersections may occur if one or several xrs_i's are contained in one or several (longer) elements xrs_j. They may be omitted applying (C3). Therefore, the set $MXRS \subseteq XRS$ is introduced

$$MXRS = \{mxrs \mid (mxrs \in XRS) \text{ and } mxrs \text{ is not substring of any other element of } XRS\}$$

Obviously, MXRS contains all longest possible refstrings leading to non-redundant logical operations. One may select other non-redundant subsets SXRS⊂XRS but MXRS has special properties

A3: 1. $J(MXRS) = J(XRS)$
2. $J(MXRS) \subseteq J(SXRS)$

for all subsets $SXRS \subset XRS$. In other words, the set MXRS leads to the smallest possible number of records to be transferred and to the smallest number of false drops. Especially,

A4: If $MXRS = \{x\}$ then $J(MXRS) = J(x)$

In this case the smallest possible number of block transfers is also necessary. No false drops are encountered in this case.

2.2. Refstrings for Partial Match Retrieval

The application of the refstrings for partial match retrieval (PMR) in the sense of /BE75/, also called a query of order s /YA77/, is derived from the substring search described so far.

A query which specifies s attribute values $v_1, v_2, \ldots, v_s$ is regarded as a substring match problem where all s substrings $v_1, \ldots, v_s$ have to occur in one record. So, instead of a single set related to one substring, one defines V_i to be the set of all substrings in v_i, i=1,2...,s and uses as set X the union $V_1 \cup V_2 \ldots \cup V_s$ All other definitions remain unchanged.

It is obvious that this method allows also to specify attribute values itself only partially. This is important in cases where attributes may consist of several keywords like a title of a book or where compound attributes such as chemical formulas occur. The refstring index supports a search with arbitrary attribute components.

A further generalization of PMR beyond the definition in /BE75/ is again especially important for the specification of several equivalent keywords in the free format case. For simplicity of notion this problem is discussed for the case when one attribute value v may have two equivalent values x and y. Therefore, records have to be accessed containing somewhere x or y or both.

In order to avoid the union $J(x) \cup J(y)$ in the practically important case where x and y have common reference strings on defines

$$M.RS = MXRS \cap MYRS$$

Then the number of logical operations is reduced without changing the result by

$$J'(xvy) := J[M.RS] \cap (J[MXRS-M.RS] \cup J[MYRS-M.RS])$$

Note that the number of logical operations corresponds also to the number of secondary data transfers.

A final extension is the similarity search: Again the set MXRS is determined which contains all refstrings without the redundant ones belonging to $X: = V_1 \cup V_2 \ldots \cup V_s$ where V_i again are the sets of all substrings in the given attribute values v_i. But instead of determining the records J[MXRS] which have to contain e v e r y reference string from MXRS, a weaker requirement is set up: All records containing a sufficiently h i g h n u m b e r of reference strings - not necessarily all - are considered as candidate records for a refined similarity inspection. To be more specific, let p(i) be a positive (weight) number, related to each reference string i, then a record j is a candidate if

$$p(i_1)+p(i_2)+\ldots+p(i_l) \geq \hat{p}.$$

The value $\hat{p}$ is a given threshold and the indices i_j ,j=1,2,...,1, belong to those refstrings from MXRS which occur also in the considered record.

3. Determination of Refstrings

The set of additional refstrings ARS is the better the smaller the number of inverted lists can be kept which have to be accessed and processed for the execution of all queries including update actions. Under limitation of storage and because of update processing it is reasonable to limit the set of refstrings to a

set of such strings which are really needed and which have a high probability to occur in a query. A query which specifies rarely used keywords or attribute values will then be answered by the usage of the basic reference strings (perhaps somewhat slowlier) whereas a common query is answered fast and more directly with the aid of the additional refstrings.

Let be S a file which contains a sample of previous query arguments. If no query arguments are available the original file P or a sufficient large sample of it is used as S. Assume further that a delimiter character separates query arguments, attribute values, or keywords etc. A substring s from S is understood in the following as a substring of S which does not contain the delimiter character.

The algorithm for refstring determination to be described is motivated by the following considerations:

(1) The frequency of a certain substring from S is regarded as an indicator for the frequency of future usage.
(2) If the frequency of a certain substring is either too low then s is not considered worthwile being a reference string. If the frequency is too high, then s is a reference string only if the condition in the next rule (3) is satisfied.

In the low frequency case, establishing an inverted list is unnecessary because it is never or rarely used; in the high frequency case, the substring has a high probability to occur within a longer refstring.

(3) If a certain substring s is contained in a reference string rs as substring then s is considered to be a refstring too only if the frequency of s w i t h o u t the occurrences in rs is high enough.

Therefore, a decision whether a certain string is taken as refstring will not be based on the absolute reference frequency alone. One has to take into account the frequency of indirect references by longer refstrings. If e.g. the string ROM occurs always in connection with CHROM or BROM and these two have been

included in the set of refstrings it is not worthwhile to have also ROM as refstring.

The algorithm needs two main steps: Refstring candidates are determined first for the refstring generation in the second step. The second step is described first.

3.1 Refstring Determination Using Candidates

Assume that sets Q_k of candidates q_k for refstrings with length k are available together with their absolute frequencies $f(q_k)$. Generally f(s) denotes the frequency of a substring s in S, and RS_j denotes the set of refstrings with length j. Assume further that the maximum length of a candidate is denoted by m and that the set of refstrings RSj, j=m, m-1...,k+1 have been determined already.

For the determination of RS_k, take a $q_k \in Q_k$ and relate to it the two sets $RSL_{k+1}(q_k)$ and $RSR_{k+1}(q_k)$ being the sets of those refstrings $rs_{k+1} \in RS_{k+1}$ which have q_k as left resp. right substring. Similarly, relate to q_k the two sets $QL^-_{k+1}(q_k)$ and $QR^-_{k+1}(q_k)$ which contain those candidates $q_{k+1} \in Q_{k+1}$ which have q_k as left resp. right substring and which are n o t element from RS_{k+1}.

With these sets the i n d i r e c t reference frequency $irfl(q_k)$ resp. $irfr(q_k)$ of q_k is defined by

$$irfl(q_k) = \sum_{rs_{k+1} \in RSL_{k+1}} f(rs) + \sum_{q_{k+1} \in QL^-_{k+1}} \max [irfl(q_{k+1}), irfr(q_{k+1})] \quad (1)$$

$$irfr(q_k) = \sum_{rs_{k+1} \in RSR_{k+1}} f(rs) + \sum_{q_{k+1} \in QR^-_{k+1}} \max [irfl(q_{k+1}), irfr(q_{k+1})] \quad (2)$$

The weight $p(q_k)$ of each candidate q_k is defined by

$$p(q_k) := f(q_k) - \max[irfl(q_k), irfr(q_k)] \quad (3)$$

and q_k will be selected to be a refstring if

$$p(q_k) \geq t. \quad (4)$$

The value t is a given threshold. Following the definition, one

proves that $p(\cdot)$ in (3) is nonnegative. p is an estimate for the reference probability of q being referenced alone and not within the context of a longer refstring.

In the definition of the indirect reference frequency (1,2) one recognizes two terms: the candidate q_k may be indirectly referenced by longer refstrings r_{k+1} or by candidates q_{k+1} which themselves are only indirectly referenced by any longer refstring rs_j, $j \geq k+1$. Therefore the indirect reference frequency of a q_k depends of the indirect reference frequencies on some q_{k+1}. So, the definitions (4,5) are complete by the additional one

$$irfl(q_m) = irfr(q_m) = 0$$

which states that the refstring candidates with maximal length m are not directly referenced.

A small example shall illustrate the formulas: let

Q = ABCDE(100,0,0),ABCDF(10,0,0),BABCD(120,0,0)
Q = ABCD(150,100,120),ABCE(100,0,0),BABC(200,120,0),ABCF(30,0,0)
Q = ABC(300, .)

The numbers in parantheses behind each string mean the absolute frequency and the indirect reference frequencies irfl and irfr. The underlined elements are refstrings. For a decision whether q =ABC should be a refstring, one determines

RSL = {ABCE(100,0,0)}
RSR = {BABC(200,120,0)}
QL^- = {ABCD(150,100,120),ABCF(30,0,0)}
QR^- = { }

Corresponding to these sets the indirect reference frequencies for ABC are

irfl = 100+max(100,120)+max(0,0) = 220
irfr = 200+0 = 200,

and the weight is given by

$$p = 300 - \max(220,200) = 80$$

If the given threshold value for the weight is not greater 80, ABC is used as refstring.

3.2 Determination of Refstring Candidates

The method described so far assumes that refstring candidates Q_k, $k \leq m$ are available with their frequencies. A straight forward solution would be to use all possible substrings in S as candidates and to count their frequencies. A better solution is to restrict the counting to those substrings which have a chance of being a candidate. Because of (4) and due to $p(q_k) \leq f(q_k)$ one may exclude all substrings having an absolute frequency $f(q_k) < t$. Further one applies the inequality

$$f(x) \leq \min[f(y), f(z)]$$

for a string x which contains y, length k-1, as left and z, length k-1, as right substring. Based on these simple observations, a construction of Q_k, k=1,2,...,m, is given:

```
      set k:=0 and Q1:=Alphabet;
loop: k:=k+1 until m
      count frequencies of qk ε Qk in S;
      redefine Qk by deleting qk ε Qk with f(qk)<t;
      define Qk+1 : qk+1 ε Qk+1 consists of a qk ε Qk and
                    a q'k ε Qk as left resp. right substring;
      goto loop;
```

This algorithm guarantees that no substring s with length $k \leq m$ and with a frequency $f(s) \geq t$ is deleted. The proof is evident by assuming the contrary.

A little example explains the generation of candidates. Let

Q_3 = {CNO(100), H2O(500), NOH(150), CH2(50), 2OC(60), CA5(200), HCL(300)}

and t=50. Then, only for the following combinations

Q_4 = {CNOH(100), CH2O(50), H2OC(60)}

the frequencies in F have to be counted. The underlined numbers in Q_4 are only estimates (upper limits) for the final

frequencies. They are applied to improve the hash table generation necessary for counting. The combinations with high estimated numbers are used first when the probability for hash conflicts is low.

3.3 Determination of Refstring Combinations

Since a refstring may not be longer than the substrings in S between delimiters (refstrings do not go beyond attributes or single words), the frequency count of certain refstrings may be still too high. Therefore, the above algorithm is applied again but now one level higher: instead of analysing the co-occurrence of characters within strings in S now the co-occurrence statistics of refstrings within the next larger context are determined. Depending on S and the application this larger context is defined to be one query or one sentence of one document, generally one record in S.

The same technique as for the refstring generation is applied for refstring sequences: First, candidate sequences are determined up to a maximum length and then the refstring sequences are selected out from them using the formula (3). The sequence order and the adjacency condition can be removed depending on the application considered.

The described algorithm exploits the hyperbolic distribution already for the refstring generation. It has only a complexity of O(n) and seems to be suitable particularly for the production of longer refstrings and their combinations. The sort step in /BA75/needs O(n.logn) operations and the method described there seems to be more suitable for a small refstring set with short lengths.

3.4 Quantitative Results, Examples

As mentioned in the introduction the realistic assumption of having hyperbolic-like occurrence or usage statistics is essential for the number of refstrings. The fig. 1 shows the distribution of substrings with length 3 occurring in more than 35000 German words.

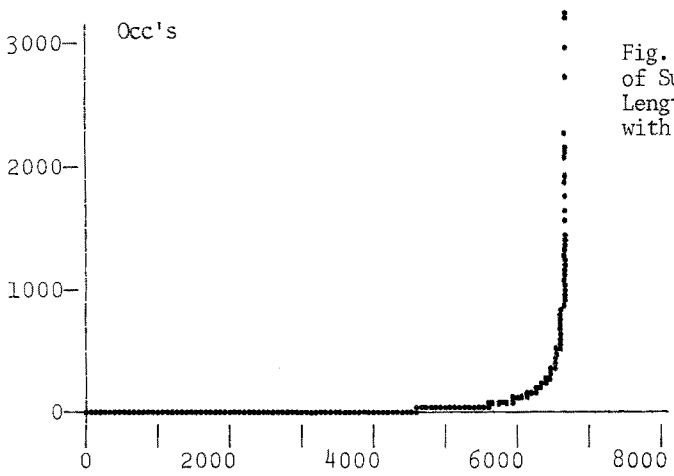

Fig. 1: Distribution of Substrings with Length 3 in a List with 36000 German words

A refstring index to a list of 44000 chemical substances contains only appr. 2000 refstrings with lengths between 3 and 5 and gives a mean selectivity of 0.2 percent (Each refstring occurs in at least 50 and, as a mean value, in 90 substances).

REFSTR	WEIGHT	REFSTR	WEIGHT
METH	50	ETHYL	888
MINE	62	HYDRO	814
AMIN	89	ATION	628
TION	139	77009	566
MINO	61	AMINO	360
LINE	101	YDROX	387
LINO	52	DROXY	378
IONE	108	-METH	336
HYDR	86	IDINE	340
DINO	55	DINE-	69
LING	86	LINE-	64
DING	83	PHENY	345
ROLI	85	HENYL	339
OLIN	101	AMINE	266
PHEN	73	OLINE	207
IDIN	111	THYL-	303
STER	141	ANALO	284
CHEN	72	HALOG	286
NYL-	70	-ACID	283
GENE	100	CHLOR	279
ATIC	82	RIDIN	271
RINE	168	-ANAL	277
RING	90	ALPHA	273
ETHI	75	STERO	172

Tab.1 Sample of Refstrings with Length 4 and 5 Generated with a List of 44000 Chemical Words

REFSTR	WEIGHT	REFSTR	WEIGHT
TERM	572	=3750	162
MINI	450	=SYS7	115
ICPU	338	=SYS3	96
0000	19	=5320	71
SCPU	227	=3740	61
OFDE	197	=3270	48
SWIT	183	=2260	42
OTHR	179	=3420	43
DASD	170	=3600	37
ELAS	166	=S306	37
MEMO	146	=3330	25
0001	23	01000	15
LCPU	112	=3145	27
1000	15	00200	13
2000	11	00500	11
0100	14	00300	15
TAPE	92	02000	13
0002	19	OPABX	23
6000	11	=3135	22
0007	24	00030	13
0200	19	=2740	20
0020	12	=XX34	20
TCUS	33	=2365	13
0030	25	=3168	19

Tab.2 Sample of Refstrings with Length 4 and 5 Generated with the 3000 DP Product Records

Table 1 shows the beginning of the refstring list with length 4 resp. 5 from this application. Table 2 contains the beginning of the refstring list from a DP products application. These refstrings are generated from 3000 formatted records with 4 searchable attributes. In a current larger test, 3000 refstrings have been generated to a formatted file with more than 30000 records and 9 searchable attributes each containing organizational data on IBM customers.

The maximum length of these automatically generated refstrings was m=5. Therefore strings like PHENY and HENYL could not be combined to PHENYL.

4. Follow-on Problems of the Refstring Index

Critical points in the application of the refstring index are

(1) the detection of refstrings in a partial match query
(2) the number of refstrings and their inverted lists to be processed
(3) the number of records in P to be accesssed, especially the ratio between the number of real matches and the number of candidates to be inspected.

Starting with the last point first, one observes that it contains two separate problems: the overall number of accesses of P regardless of matches or mismatches and the mismatch-ratio (false drops). Both critical points are improved by the following.

4.1 Clustering by Refstrings

If one assumes that the refstrings are parts of records with a suitably high reference frequency - and with this objective they have been generated - and that a PMR-action qualifies more than one record, then it is desirable that all qualifying record candidates are collected in one or a few blocks to be transferred from external to internal storage. Since access is organized by refstrings, blocks have to be defined by common refstrings, in other words, by common pieces of record contents. This is

usually called clustering. The following approach generalizes the remarks in /SCHE75/:

Let be MXRS the set of the longest non-redundant refstring as related to a string x and p(.) a positive weight function defined for refstrings (e.g. proportional to reference frequency). A positive valued function ri(x) called "reference importance" is defined for each string x by

$$ri(x) = \sum_{rs \in MXRS} p(rs)$$

If one applies this function on P, records can be sorted in descending order of this "importance". Roughly spoken, the first record contains the longest and most often referenced reference strings. A proximity measure between two strings $x \in P$ and $y \in P$ may be defined by

$$prox(x,y) = \sum_{rs \in M.RS} p(rs)$$

MXRS resp. MYRS belongs to x and y resp. and M.RS contains refstrings common to x and y. Obviously, prox is symmetric:

$$prox(x,y) = prox(y,x)$$

and satisfies the inequality

$$prox(x,y) \leq \min(ri(x), ri(y)).$$

These functions will be used to define clusters:

let $0<\alpha\leq 1$; k=0; next:=1

loop: set $\hat{r}:=r_{next}$; $\hat{ri}=ri(\hat{r})$; k:=k+1;

(r_{next} is the record in P being not yet element of a cluster)

define next cluster $C_k:=\{r_j \in P \mid prox(\hat{r},r_j) \geq \alpha \hat{ri}\}$;

redefine F by deleting records $r_j \in C_k$ from P;

Goto loop;

One observes that this algorithm defines clusters with pairwise disjoint elements. Each record is in exactly one cluster and each cluster contains at least one record because $prox(\hat{r},\hat{r}) = \hat{ri} \geq \alpha \hat{ri}$ for each α. The factor α is responsible for the

size of the cluster. The number of clusters decreases for $\alpha \to 0$.

Such an algorithm has a complexity of $O(n^2)$ whereas sorting with respect to the ri-function needs only O(nlogn). Experiments in /NU76/ show that conventional blocking after the ri-sort compares favorable with the original clustering regarding the number of data transfers.

More important for the access method using the refstring index is the fact that the inversion to be computed for the refstrings is related to the clusters not to the records. Therefore, the maximum number of inverted list entries is reduced to the number of clusters which can be an order of magnitude smaller than the number of records. The advantage of this method is that the secondary data overhead regarding storage size and processing time is reduced. Several matching records can be found in a single cluster. The disadvantage is the higher number of false drops In fact, a cluster candidate contains all specified refstrings but not necessarily within single records.

4.1.1 A Worst Case Simulation

To quantify the above remarks a worst case test was simulated in the following way: The file P was a list of 32000 different German words. Each word is a record. Clustering was introduced by using every NC words (NC=1,3,5,10) as one cluster after having sorted P alphabetically. Search arguments were simulated by selecting 500 fragments at random from the interior of the words. The refstring index contained only letter pairs as (basic) refstrings. No additional longer refstrings were allowed which again simulates the worst case behaviour of the access method with respect to the number of data accesses and false drops.

In the following table 3 the number of words in each cluster is NC=10. N-ACCESS is the number of primary data (cluster) accesses, and N-MATCH is the umber of found matching words in these clusters.

INPUT STRING (RANDOM)	N-ACCESS (CLUSTER)	N-MATCH (WORDS)
AUPTVERB	1	2
SOZIALET	4	4
EINTRAGU	5	2
INTERPRE	5	3
NERSCHAF	48	9
STRIENAT	4	2
STENSPAR	13	1
HEINKEHR	2	7
GROSSKRE	1	1
PERSONEL	12	5
NZUNGSAU	4	1
RWALTUNG	34	96
HANDELSP	6	4

Table 3: Extract of the Worst Case Test with Random Fragments, Length 8 in a Word Component Search Application

The table 4 shows the distribution of the access rate, defined as the number of cluster accesses divided by the total number of clusters. One recognizes that in 55 percent of the 500 queries with length-6-fragments and NC=3, the access rate is lower or equal 0.1 percent corresponding to 10 clusters or less.

ACCESS RATE INTERVAL (PERCENT)	NC=1 LENGTH					NC=3 LENGTH					NC=10 LENGTH				
	4	5	6	7	8	4	5	6	7	8	4	5	6	7	8
0-0.1	30	55	70	83	87	16	37	55	70	79	4	14	24	34	45
0.1-1	49	39	27	15	12	45	47	38	27	19	25	39	45	47	43
1-5	20	6	3	2	1	32	16	7	3	2	33	32	24	16	12
>5	1	0	0	0	0	7	0	0	0	0	38	15	7	3	0

Tab.4 Access Rate Distribution in the Worst Case Test

The match-access ratio is defined as the quotient between the number of matching words and the number of cluster accesses which are necessary to answer a fragment search. In table 5 the distribution of this quotient is shown for the 500 search fragments. The match access ratio can be greater 1 for NC greater 1.

MATCH ACCESS RATIO	NC=1 LENGTH					NC=3 LENGTH					NC=5 LENGTH					NC=10 LENGTH				
	4	5	6	7	8	4	5	6	7	8	4	5	6	7	8	4	5	6	7	8
> 1.	22	40	60	76	82	48	53	62	76	80	42	54	60	68	76	40	41	50	56	59
.75-1.	36	28	18	12	10	14	10	10	4	6	18	8	8	4	4	14	10	6	4	4
.5-.75	20	16	9	8	8	14	14	10	10	10	14	13	9	8	8	14	14	12	12	10
.25-.5	12	8	9	4	0	14	12	12	6	2	14	12	11	12	4	16	16	14	14	12
0-.25	10	8	4	0	0	10	11	6	4	2	12	13	12	8	8	16	19	18	14	15

Tab.5 Match Access Ratio in the Worst Case Test

4.2 Decomposition of a PMR-Query into its Refstrings

This problem does not exist in classical inverted file approaches for unformatted and formatted DB-systems. Either a keyword inversion or inversion for certain attributes has been provided for or not. In the more general refstring inversion, the attribute values or keywords specified have to be inspected to determine which refstring inverted lists are applicable and which are suited best. For the determination whether a certain substring within an argument of a query is a refstring, a special hashing technique has been applied based on hashing by division and chaining /MA75/. The modification applied needs three bytes for each refstring with length 3 and four bytes for each refstring with length 4 or 5 including the link field and a chain flag. So, hash tables for more than 1000 refstrings with length 4 or 5 may be stored in one page of 4k bytes. Details are described in a separate note /SCHE77/.

Sequential application of the hash function on necessary string positions within the specified query arguments may therefore lead directly to the set MXRS. In order to reduce the number of refstrings further, the string positions which have to be tested may be selected in such a way that also highly correlated refstrings, (not only completely redundant ones) are avoided. If e.g. in the search argument ATOMIC the refstrings ATOM,TOMI,OMIC are contained, one could omit TOMI and use only the intersection of the lists between ATOM and OMIC because of the

three-characters overlapping TOMI.

In general, instead of testing each position, a number of s<k (e.g. s=k-1) characters is skipped after having found a refstring with length k.

The following examples show how a query in a component search application is transformed into refstrings. They belong to the list with 44000 chemical terms.

Query	Refstrings
AMINO ACID	AMINO ACID
BETA HYDROXYLASE	HYDRO BETA OXY YLA ASE
BETA-HYDROXY LASE	BETA- -HYDR LASE ROX XY
HYDROXY PROPYL METHYL	HYDRO PROPY METHY OXY YL
DICHLOR ACETAT	CHLOR ACETA DIC TAT
HALOGEN	HALO OGEN
CHLOROFORM	CHLOR FORM RO OF
SULFONAMID	AMID SUL FON LF NA
IONISATION	ATION IONI ISA
GEN-MUTATION	ATION GEN N-M UTA MU
FUZZY	ZZ UZ FU ZY

4.3 Processing of Inverted Lists

For certain basic refstrings the number of occurrences may be very high, (e.g. the strings ER, EN, NE occur very often in German language). In a question for all clusters, containing ER and EN and VE simultaneously, no longer, additional refstrings may be used. Therefore, intersection of these three inverted lists is necessary. Considerations due to Haerder /WE75/ with respect to storage and processing of inverted lists and own experiments resulted in an implementation of uncompressed bit lists a n d index lists. As a rule of thumb - gained experimentally by timings - an inverted list is implemented as index list if the frequency of the corresponding refstring is less than 0.2 to 0.5 % of the number of clusters. This is the same order of magnitude as commonly used for a decision whether access should be direct using the inverted list or sequential neglecting the access path. Haerder gives an upper limit of 1-5 % for direct access.

According to the splitting into bit and index lists, the

intersection between inverted lists is executed

1. as an operation between two bit lists or
2. as testing bits at given index positions in case of intersection between a bit and an index list
3. as an intersection between two index lists which is solved by locally changing one index list into a bit list and subsequent application of method 2.

The advantage of this approach is that the lists may be unordered - an important point because of update actions. Union is performed similarily but instead of testing bits at the index positions, one bit is set in the bit list which now contains the result of the union.

Corresponding to these possibilities for execution, complexity bounds may be found easily. The remarks are restricted to the intersection case. Assume that k refstrings $rs_1, \ldots, rs_k$ have been found. Assume further that rs_i are sorted in ascending order of their frequencies denoted by m_i. So, rs_1 is the most selective refstring. If one denotes with ε the elementary operation to test or to set a bit at a certain position then the total number t_ε of elementary operations is between

$$m_1 \leq t_\varepsilon \leq (k-1)m_1 + \sum_{i=2}^{k} m_i$$

From timing results one finds that one elementary operation ε needs 22 microseconds CPU time on an IBM /370-145. This means that the intersection of 11 refstrings with occurrence frequencies $m_1 = 100$ needs less than 44 milliseconds CPU time on this model confirming that CPU time for list processing is not a problem.

A critical point, however, is the number k of lists to be transferred from external to internal storage. One should note that the main advantage for additional longer refstrings is the reduction of the number of inverted lists to be processed. A further reduction is obtained by the elimination of refstrings from MXRS which are estimated to be highly correlated. Now, as a third possibility for the reduction of k during execution, let

q1 resp. qc be the cost to read and process an inverted list resp. one cluster and denote by $rc(\hat{k})$ the number of cluster candidates after the processing of the first $\hat{k}$ list ($\hat{k} < k$). Assume further that an estimate β for the reduction of $rc(\hat{k}+1) = \beta \cdot rc(\hat{k})$ is known. Then inverted list processing may be terminated after $\hat{k}$ if

$$rc(\hat{k}) \leq \frac{q1}{qc(1-\beta)}$$

This formula is valid even for the optimistic estimate $\beta = 0$ (next intersection would lead to an empty list) or for the pessimistic $\beta = 1$ (next reduction does not reduce rc).

5. Applications

The access method by the refstring index is non hierarchical and not influenced by a special sequence of the clusters to be accessed. Several different applications are obtained by a special interpretation of the clusters and the related refstrings.

5.1 Refstring Indexing in Non-Formatted Data

Documentation systems with automatic indexing such as the STAIRS system generate a dictionary which contains each document string between delimiters apart from strings contained in a "negative list". In the application of such a system for a language with compositа as for the German language but also for medical diagnosis texts or for descriptions of chemical substances etc., the problem arises that important components are contained in the interior of the dictionary strings which means that a query specifying only components of a dictionary entry may not be answered directly with the aid of the prepared inversion. A common solution to this wellknown problem is either to add c e r t a i n components to the dictionary or to establish a compound word relation within a thesaurus.

The disadvantage of such a method is that the components to be introduced have to be defined m a n u a l l y . They are

application-dependant and necessarily are not complete if new documents containing new terms are introduced. Furthermore, one knows from programs performing a compound word decomposition /SCH077, IZ77/ that a high number (7000-15000) components has to be maintained for the German language. This number of "reference" components even will be increased if documents are used containing also chemical elements or technical compound (artificial) words or if documents in several languages have to be processed.

In this situation it seems to be a good solution to replace the reference components by automatically generated reference strings. They have the advantage to allow the specification of a r b i t r a r y search fragments independantly of a language or of the occurrence of artificial terms.

A further problem in systems with automatic indexing is the high number of dictionary entries (500000 is a usual size). It leads to the question whether all these words are needed at all. In /HE74/ and /GE76/ it is pointed out that a saturation of dictionary entries may not be observed. A high number of very short inverted lists has to be administrated without being ever used. Since each word may appear also in different (flexion) forms such as 'atom', 'atoms', 'atomic', 'atomar', etc., a user being aware of this fact would specify his query in the above example as 'atom**' in order to include all documents containing the keyword atom with at most two additional characters at the end. In order two answer this example question, 4 accesses to the 4 related inverted lists and their union have to be performed.

On the other hand, the dictionary contains also few words which are very frequent and which occur also frequently in a sequence with another frequent word. A typical example may be the sequence 'pattern recognition'. If this sequence is specified in a query, two inverted lists have to be transferred and their intersection has to be computed. A solution to this problem by using refstrings and refstring combinations as document index is currently under investigation. Experiments known from the literature (e.g. /SA68, LUS67/) show that statistical term association give reasonable results, especially for the

application in a floating dictionary /STE 74/. The refstring index applied on documents is a generalization.

The following two examples from a terminal session are taken from experimental programs (PL/1 and ASSEMBLER) for a word component search running under VM-CMS on a /370-145. Virt. Time is the CPU time in milliseconds. The first example is taken from a word component search through a list with 32000 German words (NC=1).

```
SEARCH COMPONENT(S)      VERSICHERUNG RENTE GESETZ

VIRT. TIME     311

FOUND WORDS FOR    VERSICHERUNG RENTE GESETZ

KNAPPSCHAFTSRENTENVERSICHERUNGSGESETZES
RENTENVERSICHERUNGS-NEUREGELUNGSGESETZ
RENTENVERSICHERUNGS-NEUREGELUNGSGESETZE
RENTENVERSICHERUNGS-NEUREGELUNGSGESETZEN
RENTENVERSICHERUNGSAENDERUNGSGESETZ
RENTENVERSICHERUNGSGESETZ
RENTENVERSICHERUNGSNEUREGELUNGSGESETZE
```

In the second example the same programs search through a list with 120000 mixed language words. One cluster contains 5 words (NC=5)

```
SEARCH COMPONENT(S)     PHOSPH  SULF

VIRT. TIME   349

FOUND WORDS FOR      PHOSPH   SULF

BIS(TRIARYLPHOSPHORANYLIDENE)SULFAMIDES
BIS(TRICHLOROPHOSPHORANYLIDENE)SULFAMIDE
SULFIDPHOSPHORE
SULFIDPHOSPHOREN
TRICHLOROPHOSPHORANYLIDENE)SULFAMIC
ZINKSULFID-PHOSPHOREN
ZINKSULFIDPHOSPHORE
```

5.2 Refstring Index as Multi-Attribute Index in Formatted Data

Because attributes in formatted files may also consist of natural language words or personal names, addresses etc., the possibility for a search with components of attributes or arbitrary masking seem also relevant here: A refstring index can be provided for certain attributes to allow partially specified attribute values. In addition, the refstring index method may support queries which

specify more than one attribute value. The usual approach for processing such queries is to prepare inverted lists for each of those "important" attributes or to use combined indexes in the sense of /LU 70/. A new proposal by the refstring inversion is obtained in the following two ways:

1. If one attribute is more important than all others and often further attributes together with the dominant one are specified, than an improvement over the single attribute inversion is obtained if additional refstrings from other attributes are selected which occur often in combination with frequent refstrings of the dominant attribute. Compared with the combined index appraoch, this proposals combines only c e r t a i n attribute v a l u e s instead of whole complete attributes.

2. For several applications it seems to be a reasonable solution to regard the whole records or a subset of "searchable attributes" as "document" for a refstring generation. Therefore, one s i n g l e refstring index is valid for s e v e r a l record attributes. This means that for the computation of the direct record access the attribute information is neglected. Here, again only attribute value combiantions as refstring sequences will occur and not complete combined attributes.

This idea has been applied on a file with 3000 records containing information on DP-products distributed over 8 attributes. Four of them have been assigned to be searchable. Quantitative results are summarized in table 6. They are obtained by a usage simulation in the following two ways:

1. A record out from the 3000 is selected at random and two values of its four searchable attributes are selected again at random to give the arguments of a partial match query. This procedure is iterated to get 100 partial match queries. Obviously, these queries simulate a worst case usage because each attribute value has the same probability to be specified in such a query and therefore do not correspond to the prepared index.

2. The records are sorted corresponding to the function ri, (see 4.1). Then, the two most frequent attribute values are selected in every record, starting with the first record in the new sequence. These two values are considered as arguments of a partial match query. The procedure is iterated until 100 different partial match queries are found. In this case the usage simulation corresponds to the prepared index and is the favorable case .

The following table shows the results of these simulations with respect to secondary data accesses (IOS), primary data accesses (IOP), number of secondary data accesses if only pairs as refstrings are used (IOSB) and the related number of primary data accesses (IOPB). MATCH is the number of found matching records. If the number of candidate records is less than NCM during the processing of the query, the secondary data accesses are stopped and the primary data are accessed. Values of 1 and 10 for NCM are used in the tests.

	NCM	IOS	IOP	IOSB	IOPB	MATCH
worst case	1	2.8	10.2	5.7	10.2	10.1
favor.case	1	2.5	10.7	5.7	11.9	10.5
worst case	10	2.3	11.6	5.2	10.9	10.1
favor.case	10	2.0	11.4	5.0	12.4	10.5

Table 6: Refstring Index Applied as Multi-Attribute Index

The main influence is the reduction of secondary IOs due to longer refstrings, even in the worst case simulation. The optimal value for the number of secondary IOs is 2.0 which is obtained in the usual approach where all attributes are completely inverted. This value can be kept smaller only if combined indexes or combined refstrings are used.

Acknowledgement

I want to thank Barbara Ruhbach, Rainer Nussbaum and Hans Peter von Reth students at the Universities of Heidelberg and Mannheim for their great assistance in implementing the programs and evaluating the experiments.

References

AHO75 A. V. Aho, Margret J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, Comm. ACM (1875), Vol. 18, No. 6, pp. 333-340

AHO74 A. V. Aho, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, Reading, (Mass.) 1974

BA75 J. J. Barton, S. E. Creasy, M. R. Lynch, M. J. Snell, An Information-Theoretic Approach to Text Searching in Direct Access Systems, Comm. ACM (1974), Vol. 17, No. 6, pp. 345-350

BAY73 R. Bayer, E. McCreight, Organization and Maintenance of Large Ordered Indexes, Acta Informatica 1 (1972), pp. 173-189

BE75 J. L. Bentley, Multidimensional Binary Search Trees Used for Associative Searching, Comm. ACM (1975), Vol. 18, No. 9, pp. 509-517

BU76 W. A. Burkhard, Hashing and Trie Algorithms for Partial Match Retrieval, ACM Transactions on Data Base Systems, (1976), Vol. 1, No. 2, PP. 175-187

CLA72 A. C. Clare, E. M. Cook, M. F. Lynch, The Identification of Variable-Length, Equifrequent Character Strings in a Natural Language Data Base, Computer Journal Vol. 15, No. 3, pp. 259-262

GE76 F. Gebhardt, Wortstatistiken an groesseren Textsammlungen, Nachrichten f. Dokumentation, 2-1977, Hrsg. von der Deutschen Gesellschaft f. Dokumentation e.V., Seite 53-58

HA71 M. C. Harrison, Implementation of the Substring Test by Hashing, Comm. ACM (1971), Vol. 14, No. 12, pp. 777-779

HE74 R. Henzler, Quantitative Beziehungen zwischen Textlaengen und Wortschatz, Hrg. Zentralstelle fuer maschinelle Dokumentation, Frankfurt, Nr. ZMD-A-23, Beuth- Verlag, Frankfurt, 1974

IZ77 H. Izbicki, Composita Program, Documentation Draft, IBM Laboratory Vienna, March 1977

KNU73 D. E. Knuth, The Art of Computer Programming, Sorting and Searching, Addison-Wesley Publishing Company, Reading, (Mass.) 1973

KNU74 D. E. Knuth et al, Fast Pattern Matching in Strings, Technical Report No. STAN-CA-74-440, 1974

LUS67 G. Lustig, A New Class of Association factors, in Mechanized Information Storage, Retrieval and Dissemination, (ed. K. Samuelson), Proeedings of the FID-IFIP Conf., Rome, 1967, North-Holland Publ. Comp. Amsterdam 1968.

LU70 V. Y. Lum, Multi-attribute Retrieval with Combined Indexes, Comm. ACM, (1970), Vol. 13, No. 11, pp. 66-665

MAU75 W. D. Maurer, T.G. Lewis, Hash Table Methods, Computing Surveys, (1975), Vol. 7, No. 1, pp. 6-19

MCR74 E. M. McCreight, A Space-Economical Suffix Tree Construction Algorithm, JACM (1976), Vol. 23, No. 2, pp. 262-272

NU76 R. Nussbaum, Diskussion verschiedener Aehnlichkeitsanordnungen in grossen Wortlisten, Diplomarbeit Universitaet Mannheim, Institut f. Wirtschaftsinformatik, 1977.

SA68 G. Salton, Automatic Information Organization and

Retrieval, Mc Graw-Hill, New York, 1968

SCHE75 H.-J. Schek, Tolerating Fuzziness in Keywords by Similarity Searches, IBM Scientific Center, Heidelberg (1975), Technical Report TR 75.11.010 contained in Kybernetes 6 (1977) Special Issue on Fuzzy Systems

SCHE77 H.-J. Schek, The Reference String Access Method and Partial Match Retrieval, IBM Scientific Center Technical Report TR77.12.009.

SCHO77 G. Schott, Automatische Kompositazerlegung mit einem Minimalwoerterbuch, Vortrag bei der Fruehjahrstagung GMDS-GI, Giessen, April 1977

STE74 I. Steinacker, Indexing and Automatic Significance Analysis, Journal of the American Society for Information Science, (1974), Vol. 25, No. 4, pp. 237-241

WA73 R. E. Wagner, Indexing Design Considerations, IBM Systems Journal, (1973), No. 4, pp. 351-367

WE75 H. Wedekind, T. Haerder, Datenbanksysteme II, Reihe Informatik)18, Bibliographisches Institut Mannheim/Wien/Zürich, B.I.-Wissenschaftsverlag 1976

WO71 E. Wong, T. C. Chiang, Canonical Structure in Attribute Based File Organization, Comm. ACM, (1971), Vol. 14, No. 9, pp. 593-597

YA77 S. Yamamoto, S. Tazawa, K. Ushio, H. Ikeda: Design of a Balanced Multiple-Valued File Organization Scheme with the Least Redundancy, Proc. of the Very Large Data Base Conf., Tokio, Oct. 1977, p.230.

ZI49 G. Zipf, Human Behaviour and the Principle of Least Effort, Addison-Wesley, Cambridge, Mass. 1949.