G

TRITA-NA 7908
C₁

THE ROYAL INSTITUTE OF TECHNOLOGY
STOCKHOLM
SWEDEN

DEPARTMENT OF
NUMERICAL ANALYSIS
AND COMPUTING SCIENCE

CERN LIBRARIES, GENEVA

CM-P00069453

TRITA-NA-7908

Inst för Numerisk analys
KTH
100 44 STOCKHOLM 70

Dept of Numerical Analysis and
Computing Science
The Royal Institute of Technology
S-100 44 STOCKHOLM 70, Sweden

ON SEARCH PERFORMANCE FOR
CONJUNCTIVE QUERIES IN COMPRESSED,
FULLY TRANSPOSED ORDERED FILES

by

Per Svensson

TRITA-NA-7908

Also issued as Technical Report of the Swedish National
Defence Research Institute.

On search performance for conjunctive queries in compressed, fully
transposed ordered files

by Per Svensson

Swedish National Defense Research Institute

Summary:

A new file organization method, providing very high performance for a large class of associative queries, is abstractly defined. The organization may be viewed both as a development of the fully transposed file, and as a generalized trie. Its average search performance is modelled under some simplifying assumptions. Also, the model's predictions are compared with measurement results obtained from a prototype system, and a qualitative agreement is found.

For purposes of comparison, analytical and measured cost curves for fully transposed file search are also given.

1. Introduction

The concept of a transposed file (TF) was introduced by Wiederhold in [1]. In [2], Batory outlined search algorithms for the evaluation of certain kinds of associative queries against transposed files. In the sequel, we will discuss fully TF only. Batory also presented results from a simulation study, where estimates of the performance of his algorithm for a fully TF was compared to corresponding estimates for inverted and multilist file organisation.

The general form of the queries considered in Batory's simulation experiment was

$$(R): \bigvee_{i=1}^q \left(\bigwedge_{j \in J} R_j \in W_{ij} \right), \quad J = \{j_1, \dots, j_k\} \quad (1.1)$$

where k is the number of distinct attributes in the query, W_{ij} non-empty sets of values, and R a "target" variable, taking values in the set of records which constitutes the file. R_j is the value of the j :th attribute in record R .

As cost criterion Batory used the number of disk block accesses. The results were presented as functions of the parameters k and q of (1.1), and the number of records found to satisfy the query.

For our purposes, Batory's comparisons between fully transposed and fully inverted file retrieval performance in the case $q = 1$ is of some interest. These roughly indicate the following break even ratios $p = r/N$, where R is the number of records satisfying the query and N is the total number of records in the file:

k	2	3	4
p	0.065	0.04	0.015

For values of $r/N > p$, transposed file search (using Batory's algorithm) outperforms inverted file search, and vice versa. We reiterate that these are simulation results based on specific and relatively simple performance models, and that the cost function is the number of disk block accesses.

Independently the present author and his colleagues designed and evaluated a prototype file handler [3], using a file structure which can be described as a combination of a fully transposed structure with an elaborate data compression technique. From the point of view of the present paper, the important aspect of the data compression technique is the use, where possible, of run-length encoding for each of the attribute subfiles. Thus, disregarding details of implementation, irrelevant to our present purpose, a sequence of equal values is represented by two items only, the number of elements and their common value. The original reasons for introducing this technique were the following.

In the relational data model, a subset of the attributes of a relation is required to uniquely identify each tuple in the relation. Codd [4] used the term primary key, or key for short, to denote this subset. In many applications the key consists of several attributes, each of which is thus of smaller cardinality than the key set itself. After sorting the rows of a relation table according to a sort key composed of the primary key attributes in some order, it is therefore likely that run length compression will considerably reduce the storage space needed for most of the key attributes. By the use of specially designed access interfaces, the compressed representation may be kept also in high level algorithms, such as associative or multivariable query procedures. In this way, we believed we could improve the performance of these procedures enough to more than outweigh the added complexity of the storage structures and access methods in comparison to a straight-forward design of a fully transposed file handling system.

The performance evaluation of our prototype system Prelat [3] more than confirmed these expectations. We were in fact quite surprised to find that the performance of our search algorithm was almost consistently worse when we forced it to use indexes (fully inverted files) than when we did not. There were several reasons why this could not be a spurious result, e.g. caused by poor implementation of the inverted file search algorithm. The presentation of Batory's results at the Fourth Conference on Very Large Data Bases provided only a partial explanation of this phenomenon.

The region search algorithm works as follows: Assume, for simplicity of presentation, that an interval of acceptable values is specified for each attribute. For the first attribute, the result is a single interval of row numbers. When this interval is intersected with the selection interval for the second attribute, the result consists of one row number interval for each (distinct) value of the first attribute which satisfies the selection criterion. As more attributes are processed, more row number intervals are generated, although some of the intervals will be empty, particularly if the file is sparse and the selectivities are high (i.e., the cardinality of the file is much smaller than that of the Cartesian product of the attribute sets, and the cardinality of the selection intervals is smaller than that of the corresponding attribute sets).

The cost of the algorithm is related to the number of non-empty intervals generated. In the analysis below it is assumed that all relations of a given size (satisfying the relational schema) have the same probability to occur in the database, and an average cost, taken over all such sample relations, is computed.

Chapter 2 of this paper is devoted to the development of a probabilistic model of region search performance in an abstract file structure, intended to capture those basic aspects of the Prelat transposed file design and its corresponding search algorithm, which are not covered by Batory's model. In Chapter 3, we present results from performance measurements with Prelat, using stochastically generated files and queries designed to match, as far as possible, the assumptions of the abstract file model. The results are expressed so as to enable a partial empirical validation of the model. Throughout, like in [3], our cost measure is processor time. Thus, our results can not be directly compared to those of Batory. We give, however, also results from analogous measurements on fully transposed file search where the run length compression has been switched off. These results correspond to what would be obtained by using Batory's method on our test examples and with our system implementation, and may thus be used to judge the performance effects of run length compression. In [3], we give reasons why processor time is the adequate measure for the experimental studies.

The use of this cost measure also simplifies a comparison of our scheme with other results on associative search (Bentley [4], Lee & Wong [5], Rivest [6]).

2. Probabilistic analysis of search performance for complete conjunctive queries in compressed, fully transposed ordered files

2.1 Basic definitions

Definition 2.1: A compressed fully transposed ordered file (CFTOF) $F = (F_1, F_2, \dots, F_d)$ is a d -tuple of "subfiles" F_i of the following structure:

Let V_i , $i=1..d$, be finite sets of objects, called "attribute sets", subject to strict total ordering relations $<_i$. Let $c_i = |V_i|$ (cardinality of V_i). Let, for each i , $S_{ij} \subset V_i$, $j = 1..n_i$ be given ordered subsets of V_i , called "suites", such that $S_{ij} = \{v_{ij}^{(1)}, v_{ij}^{(2)}, \dots, v_{ij}^{(r_{ij})}\}$. Thus, $|S_{ij}| = r_{ij}$.

For $k, \ell \in 1..r_{ij}$:

$$v_{ij}^{(k)}, v_{ij}^{(\ell)} \in V_i, v_{ij}^{(k)} <_i v_{ij}^{(\ell)} \Leftrightarrow k < \ell \quad (2.1)$$

The number of suites n_i satisfies the relation

$$\begin{cases} n_1 = 1 \\ n_i = \sum_{j=1}^{n_{i-1}} r_{i-1,j}, \quad i = 2..d \end{cases} \quad (2.2)$$

Finally, let F_i be the n_i -tuple of suites $(S_{i1}, S_{i2}, \dots, S_{in_i})$, $i = 1..d$.

$$\text{From (2.2), we conclude } n_i \leq \prod_{j=1}^{i-1} c_j, \quad i = 2..d \quad (2.3)$$

Definition 2.1: $S_{i+1,\ell}$ is the "successor" suite to the pair $(S_{ij}, v_{ij}^{(k)})$, $k \in 1..r_{ij}$, $j = 1..n_i$, $i = 1..(d-1)$, iff

$$\ell = k + \sum_{m=1}^{j-1} r_{im} \quad (2.4)$$

(Thus, from (2.2), $\ell \in 1..n_{i+1}$).

Definition 2.2: A "path" in F is a d -tuple of index pairs $P = ((j_1, k_1), (j_2, k_2), \dots, (j_d, k_d))$ such that for $i = 1..(d-1)$, $S_{i+1, j_{i+1}}$ is the successor suite of $(S_{ij_i}, v_{ij_i}^{(k_i)})$, $k_i \in 1..r_{ij_i}$, $j_i \in 1..n_i$; also, $k_d \in 1..r_{dj_d}$.

Definition 2.3:

A "record" in F is a d -tuple of objects $R = (v^{(1)}, v^{(2)}, \dots, v^{(d)})$, such that for all $i \in 1..(d-1)$ and some $j = j_i$, $v^{(i)} \in S_{ij}$, and $S_{i+1, j_{i+1}}$ is the successor of $(S_{ij}, v^{(i)})$. Further, $v^{(d)} \in S_{d, j_d}$.

The following facts follow immediately from the definitions:

- (1) there is a 1-1 correspondence between records and paths; thus, any record R in F may be identified by its corresponding path $P(R)$.
- (2) the records in F are distinct and lexicographically ordered; thus, they may be identified by ordinal number.
- (3) through every element of every suite in F passes at least one path (i.e., each index pair (j_i, k_i) corresponding to $(S_{ij_i}, v_{ij_i}^{(k_i)})$ in F , belongs to at least one path).

A CFTOF may be interpreted as a generalized trie [7], where the branching factor (maximum number of descendants) varies with the level of the trie. As mentioned in the Introduction, however, the definitions above admit an interpretation (implementable on a computer) which is different from the "classical" list structure: a CFTOF may be obtained by sorting a set of unique records (d -tuples of attribute values), applying run length compression to each attribute and storing the resulting "sub-files" separately. In this process, one has to maintain, for each subfile, the original "address interval" (interval of ordinal numbers in the sorted set of records) for each object in each subfile.

Thus, the concept of a CFTOF may be viewed as a development of the idea of a fully transposed file, applicable to those attributes which form the primary key.

Definition 2.4:

An "s-dominator" of a path $P = ((j_1, k_1), \dots, (j_d, k_d))$ in the CFTOF F is the s-tuple $P_s = P_s(P) = ((j_1, k_1), (j_2, k_2), \dots, (j_s, k_s)), s \in 1..d$

It follows that the s first components $(v^{(1)}, v^{(2)}, \dots, v^{(s)})$ of a record R are uniquely determined by $P_s(P(R))$. Below, we will use the notation $v_s(P_s(P))$ to emphasize this correspondence.

Let $\text{Succ}(P_s)$ be the set of successors of P_s , i.e., the set of $(s+1)$ -tuples $P_{s+1} = (P_s, (j_{s+1}, k_{s+1}))$, each element of which dominates a path in F . Let Succ^* be the transitive closure of Succ , i.e. $\text{Succ}^*(P_s)$ is the set of paths dominated by their common s-dominator P_s . Define $\text{Succ}^*(P_d) = \{P_d\}$. Then,

$$\begin{cases} \text{Succ}^*(P_s) = \sum_{P_{s+1} \in \text{Succ}(P_s)} |\text{Succ}^*(P_{s+1})|, & s = 1, 2, \dots, d-1. \\ \text{Succ}^*(P_d) = 1 \end{cases} \quad (2.5)$$

Since $|\text{Succ}(P_s)| \leq c_{s+1}$, we have

$$|\text{Succ}^*(P_s)| \leq \prod_{i=s+1}^d c_i, \quad s = 1..(d-1) \quad (2.6)$$

2.2 Conjunctive and region query algorithm

Definition 2.5:

A conjunctive query is a predicate of the form

$$(R): \bigwedge_{i \in I} R_i \in W_i, \quad I = \{i_1, \dots, i_k\} \quad (2.7)$$

where R is a record variable in the file F . If $W_i \supseteq V_i$, the set of values of the i :th attribute, then the conjunct $R_i \in W_i$ is trivially satisfied and may be excluded from the query. We call the query complete when $\{i_1, i_2, \dots, i_k\} = 1..d$ and $W_i \not\supseteq V_i, i \in 1..d$.

Definition 2.6:

A simple region query is the special case $W_i = [a_i, b_i], i \in \{i_1, \dots, i_k\}$, of (2.7).

To find the records R that satisfy (2.7), one first determines the set of paths (or "record identifiers") in F , which corresponds to such records. The operation of actually retrieving these records may or may not follow as a second step (in a fully transposed file design, often only a subset of the attributes need actually be retrieved).

Since the cost of the second step is independent of the search algorithm used, since complete retrieval is not always needed, and since the cost of the first step very often dominates that of the second, in the sequel we always disregard the second step.

To execute a given complete conjunctive query against a given CFTOF, the following "algorithm" may be used (P_s is the s -dominator of a path P in F):

Algorithm 2.1

```

     $T := \{P \text{ in } F\};$ 
  for  $i := 1$  to  $d$  do
     $T := T - \{P \mid v_i(P_i(P)) \notin W_i\};$ 

```

In an implementation of this algorithm, the set T may be represented by a list of record number intervals. Thus, for example, $\{P \text{ in } F\}$ may be represented by $[1, N]$, where $N = \sum_{j=1}^d r_{dj}$ is the total number of records in F .

2.3 Performance analysis of Algorithm 2.1Postulate 2.1 (File generation):

Let N records be randomly drawn without replacement from the sample space $V = V_1 \times V_2 \dots \times V_d$. Let F be the CFTOF which corresponds to the ensuing set of records.

Proposition 2.1:

Let $F = (f_1, f_2, \dots, f_d)$ be a random CFTOF, generated as described in Postulate 2.1. Denote the average cardinality of a suite in f_i by E_i .

Then, using the notation of Section 2.1, and letting $K_i = \prod_{s=1}^i c_s$, $i \in 1..d$,

$$E_i = c_i \left(1 - \left[\frac{K_d - K_d/K_i}{N} \right] \right), \quad i = 1..d \quad (2.8)$$

Proof: Let $U(V)$ be the "universal" CFTOF corresponding to the sample space $V = V_1 \times V_2 \times \dots \times V_d$, i.e., that CFTOF which contains all $c_1 c_2 \dots c_d$ records in V . Then F is some substructure of U such that every path in F corresponds to one in U .

Any given i -dominator Q_i in U partitions the sample space into the set of records corresponding to $\text{Succ}^*(Q_i)$ and its complement. Clearly, $|\text{Succ}^*(Q_i)| = K_d/K_i$, $i \in 1..d$.

If $i < d$, each one of the c_{i+1} elements of $\text{Succ}(Q_i)$ dominates a disjoint subset of $\text{Succ}^*(Q_i)$, each of cardinality $|\text{Succ}^*(Q_i)|/c_{i+1} = K_d/K_{i+1} = |\text{Succ}^*(Q_{i+1})|$. In this way, we have for any $Q_i, i \in 1..(d-1)$ defined the following partition of V :

- . the set $\text{Succ}^*(Q_i)$, partitioned into $\text{Succ}^*(Q_{i+1}^{(1)})$, ..., $\text{Succ}^*(Q_{i+1}^{(c_{i+1})})$
- . its complement $\overline{\text{Succ}^*(Q_i)}$

The probability that a suite on the level $i+1$, $i \in 1..(d-1)$, of F has cardinality m , $m = 0, 1, \dots, c_{i+1}$, equals the probability that exactly m of the sets $\text{Succ}^*(Q_{i+1}^{(1)})$, ..., $\text{Succ}^*(Q_{i+1}^{(c_{i+1})})$ contain a record in F . Let Q_0 be defined by $\text{Succ}(Q_0) = \{Q_1 | Q_1 \text{ in } U\}$ and let $|\text{Succ}^*(Q_0)| = K_d$. Then the discussion above holds also when $i = 0$.

From Lemma 2.1, below, we therefore get $E_i = e(c_i, N, K_d/K_i, K_d)$, $i \in 1..d$.

Lemma 2.1 Let U be a set of K distinguishable objects. Let U contain c disjoint subsets $S_1, S_2, \dots, S_c$, each of cardinality k , $c \cdot k \leq N$. Select a random sample of size N from U without replacement. Let the expected number of subsets S_i represented in the sample be $e(c, N, k, K)$.

Then

$$e(c, N, k, K) = c (1 - \binom{K-k}{N} / \binom{K}{N}) \quad (2.9)$$

Proof Let A_i be a random variable, such that $A_i = 1$ if the subset S_i is represented in the sample and $A_i = 0$ otherwise. Let $\text{Prob}(A_i=1) = p_i \equiv p$.

Then, $X = \sum_{i=1}^c A_i$ is the number of subsets represented in a sample.

$$E(X) = \sum_{i=1}^c E(A_i) = c p$$

But $p = 1 - \text{Prob}(S_1 \text{ is not represented})$

$$= 1 - \binom{K-k}{N} / \binom{K}{N}$$

Thus, $e(c, N, k, K) = c (1 - \binom{K-k}{N} / \binom{K}{N})$

Postulate 2.2 (Cost measure):

To compute the set $T = \{P \mid v_i(P_i(P)) \notin W_i\}$, we assume that we have to scan those suites of f_i , through which some path in the current set T passes. Specifically, we postulate that the cost of searching any one suite S_{ij} is proportional to its cardinality r_{ij} (by taking advantage of the fact that the suites are ordered, the scanning of the suite may often be stopped before its end is reached; rather little is gained in this way, however, and we henceforth disregard this possibility). We neglect the cost of other necessary operations, such as "housekeeping" for the list representing the set T .

Definition 2.7

Let the "permissivity" π_i of the clause $R_i \in W_i$ in (2.7) be defined by

$$\pi_i = |W_i \cap V_i| / |V_i| \quad (2.10)$$

It follows that $\frac{1}{c_i} \leq \pi_i \leq 1$.

Batory [2] uses the term selectivity for a related, although not identical, entity. Since this usage does not give the intuitive meaning to the phrase "high selectivity", we prefer to define the "selectivity" σ_i by the equation $\sigma_i = 1 - \pi_i$.

Postulate 2.3 (Query generation):

We assume that the query to be answered by executing Algorithm 2.1 is characterized by a "permissivity vector" $\underline{\pi} = (\pi_1, \pi_2, \dots, \pi_d)$, where $\frac{1}{c_i} \leq \pi_i < 1$. Thus, the query is complete. (Any query for which $|W_i \cap V_i| / |V_i| = \pi_i$, $i = 1..d$, has the same average execution cost when average is taken over all possible F 's with given N .)

Obviously, the average cardinality of a suite, and thus the average cost of scanning it depends only on its level in F . Denote this average by E_i , $i = 1..d$.

Proposition 2.2

Denote by C_i the average cost of searching any subset of F , whose records are dominated by some common $(i-1)$ -dominator P_{i-1} . Let γ be a proportionality constant. Then, the following recursive relation holds:

$$\begin{cases} C_d = \gamma E_d \\ C_{i-1} = \gamma E_{i-1} + \pi_{i-1} \cdot E_{i-1} \cdot C_i, \quad i = d, d-1, \dots, 2 \end{cases} \quad (2.11)$$

Thus,

$$C_1 = E_1 \left(1 + \sum_{k=1}^{d-1} \prod_{\ell=1}^k (\pi_\ell E_{\ell+1}) \right) \quad (2.12)$$

To derive (2.11), observe that E_{i-1} is the average number of successors of an $(i-2)$ -dominator. Further, $\pi_{i-1} E_{i-1}$ is the average number of second order successors (i.e., successors of successors) of an $(i-2)$ -dominator that need to be searched.

Thus, by combining (2.11) or (2.12) and (2.8), an expression for the average search cost of a given complete conjunctive query has been obtained. The extension to non-complete queries is straightforward, but since the expressions become rather complicated, we leave them out.

3. A comparison between theoretical and measured performance for range queries

As mentioned in the Introduction, the development of the theoretical model of Chapter 2 was motivated by measurements of search performance with our prototype system Prelat [3]. It thus seemed natural to try to relate the predictions of the model to prototype timings. The prototype system was therefore augmented with a random file generator, a query generator, and a procedure which could calculate the modelled cost for each file and query combination. After some preliminary tests, it was decided to produce complete results for the following example:

Test example 3.1

File generation

Seven files, each with four attributes, were generated. In each case, the attribute cardinalities was 10. The file generator worked as follows: a file was generated one record at a time by drawing four rectangularly distributed pseudorandom integers in the interval $[1,10]$ and then repeating this process a given number of times. After sorting the file, multiple records were removed. The resulting files may be considered ordered random samples, drawn without replacement, from the space $U = [1..10] \times [1..10] \times [1..10] \times [1..10]$.

The following file sizes resulted:

file no.	1	2	3	4	5	6	7
cardinality	947	1780	2589	3253	3942	4511	5057

Query generation

Against each file, several simple region queries of varying permissibilities were executed. For all four attributes, the permitted regions were generated as random subintervals of $[1,10]$ of length 10π , where $\pi = 0.1(0.1)0.9$.

Results

The model cost (using $\gamma = 1$ in (2.9)) for each of the seven test files as function of π is displayed in fig 3.1, and the measured search times

in fig 3.2 (obviously, a larger file corresponds to a longer search time). As can be seen from the diagrams, the correspondence between the modelled and actual performance is far from perfect. After having studied the program's performance characteristics, using detailed time measurements and execution profiles, I concluded that this discrepancy is due to basic design principles of the prototype and thus could not be removed without a major effort. Thus, there may well be some room for improving the practical implementation of a CFTOF that was used in Prelat. On the other hand, it seems clear that our implementation embodies the main advantages of the CFTOF concept. To demonstrate this, we have plotted in figs 3.3 and 3.4, respectively, the "normalized" theoretical cost and measured search times for our method (solid lines), and the method described by Batory [2] and others (dashed line). The way our example is designed, nothing is to be gained by choosing the search order of the four attributes. The normalization in fig 3.3 simply consists of dividing the cost estimate by the file cardinality, whereas in fig 3.4, the search times have been divided by the time required to sequentially read the last attribute. The model corresponding to an uncompressed fully transposed file is trivial and its derivation is left as an exercise for the interested reader (hint: the cost value in fig 3.3 corresponding to $\Pi = 0.1$ is 1.111).

The measurement results for uncompressed files in fig 3.4 were obtained by simply removing the run length compression option in our data compression optimizer, leaving all other features of the prototype system intact. Thus, we do not claim that no faster implementation of the uncompressed file design may exist. We do claim, however, that no matter how such a method is implemented, its performance will never approach that of CFTOF search for small values of Π and sufficiently large files - again, if a significant part of the query refers to the primary key.

4. Conclusions

A new file organization method, claimed to provide very high performance for a large class of associative searches, was described. The new organization may be viewed both as a development of the fully transposed file, and as a generalized trie.

Its average search performance characteristics was modelled under some simplifying assumptions.

Also, the model's predictions were compared with measurement results obtained with a prototype system, Prelat [3]. The agreement is qualitative rather than quantitative. Thus, there might exist implementation solutions with even better search performance.

Finally, we presented for purposes of comparison analytical and measured cost curves for fully transposed file search.

A problem which was not addressed in the paper is how to order the attributes of the primary key, so as to obtain the best search performance. Since this optimal order is in general a function of the selectivity vector, the problem is poorly defined in the sense that the optimal file structure depends on the query characteristics, which in my opinion should not be assumed to be known a priori.

5. Acknowledgements

The prototype system, Prelat, was built by a project group, consisting of Stefan Arnborg, Inga-Lill Bratteby-Ribbing, Gunnar Holm, and the author. Without the devotion and outstanding competence of my colleagues, the system could never have been realized.

Also, I am indebted to Stefan Arnborg for having had access to his unpublished lecture notes on Database theory. His derivation of Rivest's result on partial match query performance in trie structures has considerably influenced this paper.

6. References

- 1 Wiederhold, G.: Data Base Design.
Mc Graw-Hill, New York 1977.
- 2 Batory, D.S.: On searching transposed files.
To appear in ACM Trans. on Database Systems.
- 3 Svensson, P.: Performance evaluation of a prototype data base handler for technical and scientific data processing.
FOA report C-20281-D8, Swedish National Defense Research Institute, December 1978.
- 4 Bentley, J.L.: Multidimensional binary search trees used for associative searching.
Comm. ACM vol. 18, 509-517 (1975).
- 5 Lee, D.T. & Wong, C.K.: Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees.
Acta Informatica vol. 9, 23-29 (1977).
- 6 Rivest, R.L.: Partial-match retrieval algorithms.
SIAM J. Comput. vol. 5, 19-50 (1976).
- 7 Knuth, D.E.: The art of computer programming, vol. 3, Sorting and Searching.
Addison-Wesley, New York 1973.