International Journal of Computer and Information Sciences, Vol. 3, No. 2, 1974

The Choice of Partial Inversions and Combined Indices

Michael Stonebraker[1]

Received September 1973; revised January 1974

It is often undesirable or impossible to provide redundant indices for all domains of a file existing on a secondary storage device. The problem considered in this paper is the selection of a limited number of indices which best facilitate interaction with a file. A probabilistic model of interaction activity encompassing queries and updates is presented, and a parametric description of the storage medium is assumed. Significant results which are independent of many file and storage characteristics are found concerning the best choice of indices in two cases. The first is the choice of domains to include in a partial inversion. Here it is desired to find the best possible subset of domains for which to provide indices. The second case concerns the choice of combined indices. In this situation the best way of grouping domains is sought in order to provide one index for each group.

1. INTRODUCTION

The problem of choosing the redundant information which best facilitates interactions with a file existing on a secondary storage device of constrained size is often of great concern to designers of information systems. One solution is to provide redundant indices in order to speed requests. This approach can be informally described using both relational terminology[1] and standard definitions as follows.

Suppose a relation (file) consists of a collection of tuples (records) each having zero or more values for each of a set of domains (attributes). An index, then, is a table which associates with each value or range of values for

Research sponsored in part by the General Foods Corporation, New York, New York.

[1] Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley, California.

a given domain a list of tuples having that domain in the required range. If an index is provided for each domain, a totally inverted file results. This is the storage structure used in TDMS.[2]

Two drawbacks of this scheme are the following: First, a large amount of space is consumed by the indices. Second, whenever the relation is updated the redundant information must also be changed, requiring additional overhead. If updates are frequent, it may be undesirable to provide all possible indices, even if space is available. Two approaches which are commonly suggested as solutions to either problem are (a) partial inversions and (b) combined indices.

In either case one would provide fewer indices than domains. In a partial inversion one would choose a subset of domains for which to provide indices. Hence space is saved and there are fewer indices to update. However, retrieval on any nonindexed domain might only be possible by a sequential search of the relation. Alternately, one can group certain domains together and provide one index for each resulting group. The range of this group would be the Cartesian product of the ranges of its members. The number of indices provided would then equal the number of groups. Under certain conditions (such as each domain appearing in one and only one group) space is again saved and there are fewer indices to update.

The combined index approach has been suggested in Ref. 3 along with an implementation scheme providing all possible combined indices. The storage cost of this approach is certain to be unacceptable for all but a very modest number of domains. A methodology for choosing from among these combined indices is presented in Ref. 4. However, only a framework is given from which no conclusions can be drawn. This framework bears some resemblance to the one we suggest in the next section. Additional work on combined indices is reported in Ref. 5. The choice of indices for a partial inversion is discussed in Ref. 6. Again, only a framework is presented which depends in part on trial and error and from which it is difficult to draw conclusions. The approach taken in this paper is to more fully characterize the query set and the storage medium so that analytic results concerning good indexing strategies can be derived.

It is clear that a more general statement of the indexing problem is:

> Choose the subsets of the set of domains for which to provide indices in an "optimal" fashion subject perhaps to the additional constraint that there be no more than a given number of indices.

Because of analytic difficulties, we shall only treat two special cases. First, we shall discuss the choice of a partial inversion with the assumption that no combined indices can be formed. In several important cases we shall be able to find the optimal set of domains for which to provide indices. Second, we

shall consider the choice of combined indices under the assumption that each domain appears in one group. It will be shown that the optimal assignment of domains to groups can be found in certain situations. Moreover, in some instances a simply stated adjacency property must be satisfied by the groups.

The remainder of this paper is organized as follows. In the next section we shall present assumptions made about the relation and the query set. Then in Section 3 we formally define redundant indices and make suppositions concerning the storage medium. In Section 4 we offer results on the choice of partial inversions. Section 5 treats the choice of combined indices with the assumption that the number of possible groups is fixed. Finally, Section 6 summarizes the results and draws conclusions.

2. ASSUMPTIONS CONCERNING THE QUERY SETS

We shall assume that a relation F consists of N tuples $\{O_i\}$, $1 \leqslant i \leqslant N$, each containing a value for m domains, e.g., $O_i = (v_{i1}, \ldots, v_{im})$. Hence v_{ij} is the value possessed by tuple i for domain j. Note that the particular representation of the various domains is immaterial. The distribution of values over the domains is likewise unconstrained. Furthermore, it is assumed that the relation is normalized[1] (i.e., F is a simple formatted file). Also, it is assumed that the entire data base consists of a single relation.[2]

The following assumptions concerning data base activity will be made. First, we shall ignore insertions and deletions and be concerned solely with updates and retrieval. We assume that Q represents the probability that an interaction with the data base is an update, in which case alteration of a single domain of a single tuple is allowed. It is assumed that each domain is equally likely to be altered. The remaining interactions are retrieval requests.

Second, interactions are made from an on-line terminal and are to be satisfied rapidly. (Clearly, batching of interactions gives the storage structure designer a different set of problems than might otherwise exist.)

The third assumption is that there are only finitely many queries[3] which are restricted in the following way. Let R_j denote the range of domain j, i.e., $R_j = \{v_{ij}\}$, $1 \leqslant i \leqslant n$. Let $\mathscr{B}_j = \{b_{1j}, \ldots, b_{sj}\}$ be a partition of R_j, i.e., $b_{ij} \subset R_j$ for all i and $\bigcup_{i=1}^{s} b_{ij} = R_j$ and $b_{ij} \cap b_{lj} = \varnothing$ for all $i \neq l$. Let B_j be the number of elements in $\mathscr{B}_j$. We shall then assume for each domain j that there exists a finite partition $\mathscr{B}_j$ whose members are the only subsets of R_j that can appear in the qualifying portion of a query.

[2] Under certain independence conditions the extension of results presented here to multi-relation data bases appears straightforward.

[3] This requirement is equivalent to assuming that atoms exist for this relation.[7]

Fourth, we shall assume that the allowable queries conform to the following format:

$$\text{GET}(v_{ij}, j \in W \mid v_{i1} \subset H_1, \ldots, v_{in} \subset H_n) \tag{1}$$

Here W is a subset of the first m positive integers; it is often called the target list, and it indicates which domains of qualifying tuples are desired. Qualifying tuples are those that obey the qualification expression to the right of W. Note that $n \leqslant m$ domains are involved in this expression and that the quantity H_j can either be R_j or a member of $\mathscr{B}_j$ for $j = 1, 2, \ldots, n$. Hence, qualifying tuples are those that simultaneously have all n domains with suitable values. Note also that when $H_j = R_j$ the jth domain in no way restricts qualifying tuples and appears inconsequentially in the query. Domains $n + 1, \ldots, m$ can appear only in the target list. Such a domain might be "telephone number" which would often be desired as output but never used to restrict qualifying tuples.

The following example illustrates this query set. Suppose the relation of interest F has three domains, part #, supplier #, and supply city, and is expressed as

$$F(\text{part \#, supplier \#, supply city})$$

Since each domain is discrete, $\mathscr{B}_j$, $1 \leqslant j \leqslant 3$, can be the set of singletons in R_j. The following are English language equivalents of allowable queries:

GET ALL SUPPLIER #'s THAT SUPPLY PART # 10
GET ALL PART #'s FOR WHICH SUPPLIER # = 5 AND SUPPLY CITY = SAN FRANCISCO

An inadmissible query, where H_3 is not R_3 or a member of $\mathscr{B}_3$, would be

GET ALL PART #'s SUPPLIED FROM A CITY IN CALIFORNIA

This query set is chosen for several reasons:

(a) It contains several frequently used and well understood query sets. For example, retrieval by a primary key, such as domain 1, would require H_1 to be a member of $\mathscr{B}_1$ and $H_i = R_i$ for $2 \leqslant i \leqslant n$. Hence only the first domain would appear in a restrictive manner in any query.

(b) Query sets approximating those allowed by many existing data base management systems[8] can be decomposed into queries of the form (1). For example, suppose that the qualifying expression can contain any Boolean operators and that H_j is relaxed to be a union of partition members of $\mathscr{B}_j$. Such an expression can easily be processed into terms of the form of the qualifying expression in (1) which are OR'd together. Hence the qualifying

tuples for the query are a union of qualifying tuples for retrievals of the form (1).[4]

(c) It is possible to process some queries from a first order predicate calculus[9] into terms of the form (1). For example, the query

FIND ALL SUPPLIERS IN THE SAME CITY AS A SUPPLIER OF PART # Z

can be decomposed into retrievals of the form of Eq. (1). Hence (1) may well be a useful primitive for complex query sets.

The fifth and final assumption made concerning queries is that query activity is adequately modeled by the condition that $H_1, \ldots, H_n$ in Eq. (1) are mutually independent random variables distributed as follows:

$$\text{prob}[H_i = R_i] = 1 - P_i$$
$$\text{prob}[H_i = b] = P_i/B_i \qquad \text{for any} \quad b \in \mathscr{B}_i$$

Hence P_i represents the probability that domain i appears restrictively in a query. If so, any partition member is equally likely to be requested. (Note that $\{P_i\}$, $1 \leqslant i \leqslant n$, can either be estimated or observed from frequency data.)

If information were known to the contrary, another distribution over the partition members could be assumed, although it would complicate the analysis. However, the independence assumption cannot easily be removed.[5]

3. ASSUMPTIONS CONCERNING THE STORAGE ORGANIZATION

We assume that a main area would contain the N tuples which are stored sequentially and that redundant indices would be constructed for augmented performance.

The precise implementation of these indices is now described.
Let X be any subset of the domains of the relation and T^X be the Cartesian product of the ranges of domains in X.

An index for X is a set of lists, one for each member of T^X. Each list would contain the tuples which fall within one member or pointers to all such tuples. Hence, implementing an index for X requires assigning a bucket on a secondary storage device for each member of T^X, where tuples or pointers

[4] For such queries note that the indicated retrieval strategy is not necessarily the best choice.

[5] The reader will note this assumption and the finiteness of the query set are the key conditions that allow mathematically tractable expressions for quantities such as the expected retrieval time.

Table I

	Part #	Supplier #	Supply city
1.	1	2	Los Angeles (L.A.)
2.	2	2	Los Angeles
3.	1	1	San Francisco (S.F.)
4.	3	1	San Francisco
5.	3	3	New York (N.Y.)
6.	3	2	New York

would be stored.[6] In all further analysis we shall assume the latter case to be true. We will also assume that pointers require four bytes.

We assume that access to any record either in the main area or in the indices requires a constant delay C_1,[7] followed by a transfer time of C_2 time units per four bytes of information. Of course, C_1 and C_2 will depend on the device used and on physical placement of information thereon.

A main area and two indices are indicated in the following example.

Consider F(part #, supplier #, supply city) with six tuples (Table I). An index for supplier # would contain three buckets with contents:

bucket {supplier # = 1} = {3, 4}
bucket {supplier # = 2} = {1, 2, 6}
bucket {supplier # = 3} = {5}

An index for X = {part #, supply city} would have nine buckets as follows:

bucket {1, L.A.} = {1}
bucket {1, S,F,} = {3}
bucket {1, N.Y.} = {$\varnothing$}
bucket {2, L.A.} = {2}
bucket {2, S.F.} = {$\varnothing$}
bucket {2, N.Y} = {$\varnothing$}
bucket {3, L.A.} = {$\varnothing$}
bucket {3, S.F.} = {4}
bucket {3, N.Y.} = {5, 6}

Note in the example relation that queries require various amounts of time for obtaining the list of qualifying tuples from the indices. For instance, any query that has domains 1 and 3 appearing restrictively, e.g.,

[6] We shall assume that the bucket for any member of T_X can be accessed without searching any portion of the index by utilizing an address computation technique.

[7] This corresponds to arm motion and latency time of a secondary storage device.

GET ALL SUPPLIERS WHO SUPPLY PART # 1 FROM NEW YORK

requires one access to a bucket in the second index with a delay of C_1 and a transfer time whose expected value is $C_2N/(B_1 * B_3)$. A query with only domain 3 appearing, e.g.,

GET ALL PART #'s SUPPLIED FROM NEW YORK

requires B_1 members to be fetched from the second index each with delay C_1 and expected transfer time $C_2N/(B_1 * B_3)$. If only domain 2 appears restrictively, e.g.,

GET ALL PART #'s SUPPLIED BY SUPPLIER # 3

one member of the first index must be retrieved, requiring an expected time of $C_1 + (C_2N/B_2)$.

The update mechanism can now be discussed more precisely. An update has been assumed to involve the changing of one domain for a one tuple. It will be assumed that the appropriate tuple in the main area can be identified by a tuple identifier such as the tuple number.[8] One such update could be

CHANGE SUPPLIER NUMBER TO 3 FOR TUPLE 3

Performing such a change requires a single tuple to be retrieved from the main area and then stored again. The expected time of this operation in inconsequential to the upcoming analysis. Moreover, should an index exist that includes the domain in question, two buckets in that index must be modified. It is assumed that the two buckets are equally likely to be any buckets in the index.[9] For the update noted above it is clear that tuple 3 must be deleted from the first bucket of the supplier # index and added to the third.

The expected time to perform an update for the domain supplier # can be found as follows. The expected time to access each of the buckets to be changed is $C_1 + (C_2N/B_2)$. Writing the updated record does not, however, require the same expected time. Since C_1 represents both arm motion and latency, writing the revised record only requires C_2N/B_2 plus the portion of C_1 that represents latency. Let C_1 be divided into $C_{\text{arm}} + C_{\text{lat}}$. Writing a record, then, requires

$$C_{\text{lat}} + (C_2N/B_2)$$

[8] An equivalent condition would be to require that a primary key be included in an update request. Note that a more complex model of update could be assumed but only at the expense of more complicated algebra.

[9] Relaxing this assuption to allow any given distribution over the buckets of an index does not affect the validity of any of the subsequent theorems.

The expected time to adjust the index then is

$$2[C_1V + 2C_2N/B_2] \qquad \text{where} \quad V = 1 + C_{\text{lat}}/C_1$$

Having stated our assumptions concerning queries, updates, and indices, we can now discuss the performance criteria, which shall be to minimize the expected interaction time. Remember that interactions are updates and retrievals with probabilities Q and $1 - Q$, respectively. Throughout we shall ignore CPU time by assuming it is either small or overlapped with retrieval. Hence for retrieval we shall be concerned with the time required both to retrieve the contents of suitable redundant buckets in order to obtain pointers to potentially relevant records and to read these records from the main area. For update we shall note the time required to read and then write a tuple in the main area and the time required to update the contents of redundant buckets, if necessary.

These criteria will be applied to two situations, one where storage space is not constrained, and the other where it is. In the first situation we shall seek the unconstrained best choice of indices. In the second case we shall presume that space is available for exactly L such indices and attempt to find those that are optimal. Clearly, each index requires storage for N redundant pointers and the space consumed by interrecord gaps, empty buckets, etc. If N is large and there are a moderate number of buckets, it is reasonable to assume that each index requires an equal amount of space and that there is room for a fixed number of them. Otherwise, the assumption becomes suspect. However, if whole records are stored in the redundant buckets, then this assumption is more reasonable.

4. OPTIMAL PARTIAL INVERSIONS

This section will present results concerning the choice of a partial inversion which minimizes interaction time. First, we derive for any set of indices D in such a partial inversion the expected interaction time $E_D(\tau)$ as

$$E_D(\tau) = Q * E_D{}^U(\tau) + (1 - Q)\, E_D{}^R(\tau)$$

Here $E_D{}^U(\tau)$ and $E_D{}^R(\tau)$ are the expected update and retrieval times, respectively. Since we have assumed that all domains are updated with equal probability and since the main area record must be updated regardless of organization and can consequently be ignored, then

$$E_D{}^U(\tau) = \sum_{i \in D} \frac{2}{m} \left(C_1V + \frac{2C_2N}{B_i} \right)$$

Here $C_1V + (2C_2N/B_i)$ is the expected time of reading, then writing a record in the index for domain i.

The term $E_D{}^R(\tau)$ is composed of three components: the time to access (a) the redundant indices, (b) "false drops" in the main area, and (c) qualifying tuples. We shall treat each component in turn.

(a) *Access to redundant indices.* If domain i appears restrictively in a query, then a single bucket must be accessed from that index.[10] Since each bucket in the index is equally likely to occur, the expected time for this operation is

$$C_1 + (C_2N/B_i)$$

Since domain i appears restrictively in a query with probability P_i, it is clear that the expected time to access redundant indices is

$$\sum_{i \in D} P_i \left(C_1 + \frac{C_2N}{B_i}\right)$$

(b) *Access time to "false drops."* False drops occur because the list of pointers obtained by accessing the appropriate indices may contain spurious elements if unindexed domains are specified in a query. For example, in a relation with three domains, part #, supplier #, and supply city, and an index for each of the first two, the query

GET ALL SUPPLIER #'s FOR WHICH PART # = 5 AND SUPPLY CITY = SAN FRANCISCO

will involve an excessively long list of qualifying tuples from the first index. Each of these must be retrieved from the main area and the number that do not satisfy the query eliminated. These "false drops" involve unproductive retrieval time.

The expected number of "false drops" is shown in the appendix to be

$$N\left[\prod_{i \in D}\left(\frac{P_i}{B_i} + 1 - P_i\right)\right]\left[\prod_{i \in \bar{D}}\left(1 - \frac{P_i}{B_i}\right) - \prod_{i \in \bar{D}}(1 - P_i)\right] \qquad (2)$$

Here $\bar{D}$ is the set of unindexed domains. Each false drop requires an unproductive retrieval time of $C_1 + C_2m$.[11]

[10] Note that a bucket from the index for domain i always is retrieved if the domain appears restrictively in a query. The complications of an adaptive scheme are discussed in Section 6.

[11] The assumption is made that a tuple requires $4m$ bytes of storage. The results to be presented, however, are valid for any tuple length.

(c) *Access to qualifying tuples.* The time to access qualifying tuples is a constant regardless of the storage organization and hence will be ignored in the sequel.

Combining all terms above, one obtains the following expression for $E_D(\tau)$:

$$E_D(\tau) = Q\left[\sum_{i\in D}\frac{2}{m}\left(C_1V + \frac{2C_2N}{B_i}\right)\right] + (1-Q)\left[\sum_{i\in D}P_i\left(C_1 + \frac{C_2N}{B_i}\right)\right]$$
$$+ (1-Q)(C_1 + C_2m)\,N\left[\prod_{i\in D}\left(\frac{P_i}{B_i} + 1 - P_i\right)\right]$$
$$\times\left[\prod_{i\in\bar{D}}\left(1 - \frac{P_i}{B_i}\right) - \prod_{i\in\bar{D}}(1-P_i)\right] \tag{3}$$

The problem can now be stated as follows:

> Choose the indexed domains such that (3) is minimized, subject perhaps to the additional constraint that there be no more than L indices.

The following theorems point out the obtained results. The proofs of all results in this paper appear in the appendix.

Theorem 1. In the case where

$$P_1 = P_2 = \cdots = P_n \qquad \text{but} \quad B_1 > B_2 > \cdots > B_n$$

one should index the first i domains, stopping only when inclusion of the next domain does not reduce (3) or when it violates a constraint on the number of indices.

In the case where domains are equally likely to appear in queries one should do the obvious thing: Index the domains with the largest range since they give the greater resolution. One should stop when indexing the next domain ceases to yield improvement. This point depends on the characteristics of the device C_1 and C_2, on the update frequency Q, on the nature of the domains already indexed $\{(P_i, B_i)\}$, and on the size of the relation N. The exact nature of the relationship is indicated in the appendix. An analogous result is the following.

Theorem 2. In the case where

$$B_1 = B_2 = \cdots = B_n \qquad \text{but} \quad P_1 > P_2 > \cdots > P_n$$

one should index the first i domains, stopping only when inclusion of the next domain does not reduce interaction time or when it violates a constraint on the number of indices.

This result suggests the conclusion that frequently appearing domains should be indexed if each domain offers the same resolution. Again, the above stopping condition depends on query and storage parameters.

One additional result is a direct consequence of Theorems 1 and 2.

Theorem 3. If $\{(P_i, B_i)\}$, $1 \leqslant i \leqslant n$, has the property that $P_i = P$ or $B_i = B$, then there exist a P^* and a B^* such that an optimal set of indexed domains includes those domains j with $B_j > B^*$ or $P_j > P^*$.

If $\{(P_i, B_i)\}$ are allowed to be arbitrary, it appears that no simple algorithm will yield an optimal set of indexed domains for all values of C_1, C_2, Q, and N.

However, we can offer a solution in the cases where $\{P_i, B_i)\}$ are required to lie on the curve defined by $P_i * B_i = 1$. Figure 1 illustrates in P–B space the allowed locus of such pairs. Also labeled are the points corresponding to $B_i = 2, 3, 4, 5, 10, 20$, and 40. If $B_i = 1$, then the trivial case arises of a partition with one element in it. Hence $P_i = 1$ will be excluded. If the domains are ordered by decreasing probability of appearance, then the best set of indices must contain adjacent domains, as follows.

Theorem 4. If $P_1 \geqslant P_2 \geqslant \cdots \geqslant P_n$ and $P_i * B_i = 1$, $P_i \neq 1$, for all i, then the best selection of indices must include the domains between any two which it contains.

If the seven labeled points in Fig. 1 denote the domains of one relation, then Theorem 4 ensures that *a*, *b*, *d*, and *e* can *not* be the best domains to index, since *c* is not included. Alternately, *c*, *d*, *e*, and *f* might well be the best choice. Figure 2 diagrams the results of the previous two theorems. It illustrates three sets of domain in P–B space denoted by X, Y, Z. The X and Y

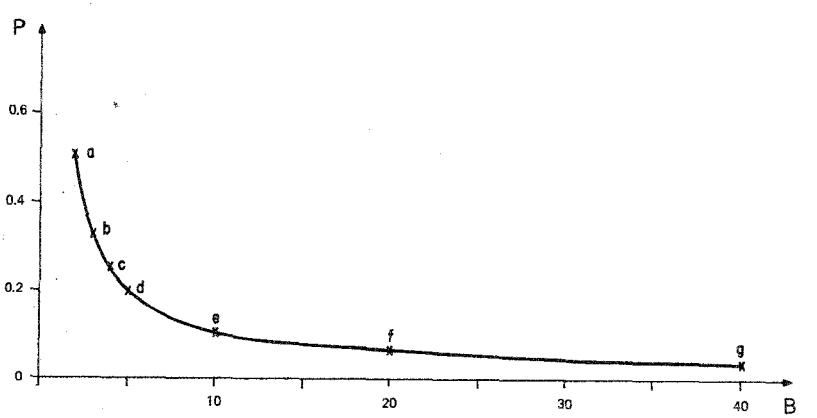

Fig. 1. The curve $P * B = 1$.

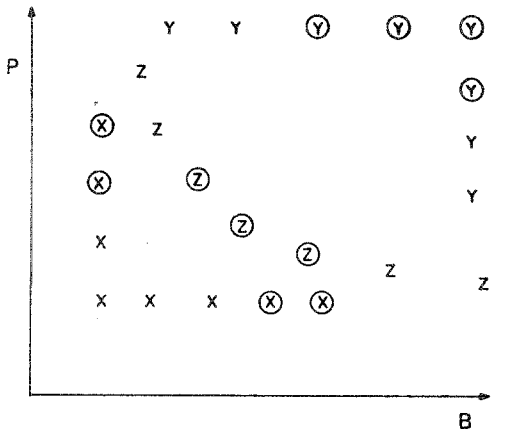

Fig. 2. Three sets of domains in P–B space.

sets satisfy the conditions of Theorem 3, while the Z set is assumed to satisfy Theorem 4. By Theorem 3, the best choice of indices for the X set might be those that are circled. Similarly, a best set of Y indices might be the one indicated. Moreover, Theorem 4 assures that the best set of Z indices has the form shown in the figure. Note that a fundamental change in indexing has occurred in moving from the X set to the Z set (i.e., from indexing distant domains to neighboring ones). The Z and Y sets, however, have the same pattern. It is suspected that domains lying along arcs with a smaller curvature than that of $P_i * B_i = 1$ might require this property of indexing neighboring domains. This supposition appears very difficult to demonstrate.

5. OPTIMAL COMBINED INDICES

In this section we shall present results on the best groupings of domains to form combined indices. The assumption will be made throughout that each domain is to appear in one and only one index and that L or less indices can appear. Before proceeding, however, a concept of adjacency must be formalized and some notation introduced.

Let the domains of F be ordered by decreasing value of P, the probability of the domain appearing restrictively in a query. Hence domain 1 is the most likely one to appear in a query and domain n the least likely. A set of s combined indices $D_1, \ldots, D_s$ will be denoted by listing the domains in each index. Thus {14, 2, 3} would denote three indices, the first containing domains 1 and 4, the second domain 2, and the third domain 3. A set of combined indices will be said to have the *adjacency property* if each D_i contains contiguous domains in the ordering. While the above set of three combined indices does not have the adjacency property, the set {12, 3, 4} does.

First, we shall consider the case where $B_1 = B_2 = \cdots = B_n$ and $P_1 > P_2 > \cdots > P_n$, in which case all domains are divided into an equal number of buckets. The theorem concerning combined indices in this situation is the following.

Theorem 5. In the case that

$$B_1 = B_2 = \cdots = B_n \quad \text{and} \quad P_1 > P_2 > \cdots > P_n$$

interaction time is minimized by choosing a set of combined indices with the adjacency property as long as $n - L < 2$.

Hence, as long as the number of domains exceeds by two or less the number of allowed indices, the adjacency property must hold for the optimal set of combined indices. It appears that most practical cases satisfy this condition. It is not known whether Theorem 5 can be extended to hold regardless of the values of n and L.

For example, if a relation has five domains and three allowable indices, there are 25 possible sets of combined indices. However, only six have the adjacency property. Hence the search for the best solution has been considerably narrowed.

We now consider a second case where $P_1 = P_2 = \cdots = P_n$ but $B_1 \neq B_2 \neq \cdots \neq B_n$. Here domains are equally likely to appear, yet the number of buckets in a domain range can vary. The following two theorems indicate that this situation is not characterized by any sort of adjacency property.

Theorem 6. If $n = 3$, $L = 2$, $P_1 = P_2 = P_3 = P$, and $B_1 > B_2 > B_3$, then {1, 23} is the best choice of domains to index.

Here one should group the two domains with a smaller number of buckets. On the other hand, the following theorem indicates an alternate conclusion.

Theorem 7. If $n = 4$, $L = 2$, $P_1 = P_2 = P_3 = P_4 = P$, and

$$B_1 > B_2 > B_3 > B_4$$

and two domains are to be in each index, then {14, 23} is the best choice.

Here one should group the domains with the largest and smallest number of buckets into one index.

It is an open question as to which characteristics are displayed by the best combined indices in this situation. Likewise, when both P's and B's can be unequal, the best choice is unknown.

6. CONCLUSIONS

This paper has been concerned with the best selection of combined indices and partial inversions under the condition that R_i be refinable by a finite partition $\mathscr{B}_i$. The results we have derived assume a search strategy whereby all appropriate indices are accessed and the resulting lists of pointers are intersected. Suppose one of the indexed domains is a primary key or a candidate key,[1] in which case B_i may well be equal to N and each redundant bucket in this index would contain only one pointer.[12] In this case there will be at most a single tuple which satisfies a query in which domain i appears in a restrictive fashion. The search strategy specified in this paper is sure to be inefficient in this case. Alternately, one would access the index for domain i and then the one possible qualifying record in the main area. In such cases the assumed search strategy yields retrieval times that are too conservative and the analysis above should be viewed with caution. Modifying the analysis to allow an alternate search strategy for queries that specify a candidate key is a straightforward extension. However, a search strategy that is adaptive would be far more difficult to consider. In such a search strategy the order of indices investigated and the number retrieved could vary, depending on the size of an intermediate target list and perhaps on knowledge about the distribution of the N elements over the domain ranges.

Also, relations for which it is unreasonable to assume that $H_1, \ldots, H_n$ are independent random variables cannot be usefully analyzed by this scheme.

In cases where the assumptions apply, however, significant answers have been obtained as to the best choice of storage structures. In the situation where a partial inversion is sought the best choice has been obtained when all domains are equally likely to appear in a query, when each domain has the same number of members in the partition of its range, when each domain has either a given probability of occurrence or a given number of partition members, and when $P_i * B_i = 1$ for all i. It has been conjectured that the adjacency property possessed by the best set of indexed domains in the latter case may be true for many other sets of domains. This represents an interesting avenue of additional research. Also, it is evident that results that generalize Theorem 3 are not difficult to obtain.

In the case where combined indices are sought the best choice must obey a certain adjacency property if domains have an equal number of partition members. This statement has been demonstrated when the number of allowable indices is within two of the number of domains. For the general case its truth is conjectured but remains an open question. In other situations preliminary results indicate various conclusions. Insight into the best choice of combined indices in the general case remains elusive.

[12] A hashing function might be used to directly address the buckets in such an index.

No attempt has been made to find the best storage structure when domains may be unindexed or appear in one or more combined indices. Extensions along these lines might also be investigated.

APPENDIX

First Eq. (2) is derived, then the seven theorems are proved.

Derivation of Eq. (2). If D is the set of domains in a partial inversion then access to relevant indices creates a target list of records whose expected length is

$$Y = N^* \prod_{i \in D} \left(P_i \frac{1}{B_i} + 1 - P_i \right)$$

Here domain i appears with probability P_i and restricts the target list by $(1/B_i)$. Otherwise, the list is not restricted. The expected number of false drops is then

$$Y^* \left[\prod_{i \in \bar{D}} \left(P_i \frac{B_i - 1}{B_i} + 1 - P_i \right) - \prod_{i \in \bar{D}} (1 - P_i) \right]$$

Here $\bar{D}$ is the set of unindexed domains. In this equation if an unindexed domain i appears in a query, than a proportion $(B_i - 1)/B_i$ of the elements of the target list are expected to be false drops. If domain i does not appear, then there will be no resulting false drops. Hence the term inside the square brackets is the expected proportion of false drops. Equation (2) easily results from the above equation.

Theorem 1. In the case where

$$P_1 = P_2 = \cdots = P_n \qquad \text{but} \quad B_1 > B_2 > \cdots > B_n$$

one should index the first i domains, stopping only when inclusion of the next domain does not reduce (3) or when it violates a constraint on the number of indices.

Proof. If D and $\bar{D}$ are the sets of indexed and unindexed domains, respectively, then (3) represents the expected interaction time. The change in this time obtained by adding domain j to D is

$$\begin{aligned} \Delta_j(\tau) &= E_{D+j}(\tau) - E_D(\tau) \\ &= C_1 \left[\frac{2QV}{m} + (1 - Q) P_j \right] + \frac{C_2 N}{B_j} \left[\frac{4Q}{m} + (1 - Q) P_j \right] \\ &\quad + \theta \left[\frac{(2P_j/B_j) - P_j}{1 - P_j/B_j} \right] - \phi \left[\frac{P_j}{B_j(1 - P_j)} \right] \end{aligned} \tag{4}$$

where

$$\theta = (1 - Q)\, N(C_1 + C_2 m) \prod_{l \in D} \left(\frac{P_l}{B_l} + 1 - P_l\right) \prod_{k \in \bar{D}} \left(1 - \frac{P_k}{B_k}\right)$$

$$\phi = (1 - Q)\, N(C_1 + C_2 m) \prod_{l \in D} \left(\frac{P_l}{B_l} + 1 - P_l\right) \prod_{k \in \bar{D}} (1 - P_k)$$

For two unindexed domains i and j we know $P_i = P_j = P$ and hence

$$\begin{aligned}
\Delta_j(\tau) - \Delta_i(\tau) &= \left[\frac{4Q}{m} + (1 - Q)\, P\right]\left(\frac{C_2 N}{B_j} - \frac{C_2 N}{B_i}\right) \\
&\quad + \theta\left[\frac{(2P/B_j) - P}{1 - (P/B_j)} - \frac{(2P/B_i) - P}{1 - (P/B_i)}\right] \\
&\quad - \phi\left[\frac{P}{B_j(1 - P)} - \frac{P}{B_i(1 - P)}\right] \\
&= \left(\frac{1}{B_j} - \frac{1}{B_i}\right)\left\{\left[\frac{4Q}{m} + (1 - Q)\, P\right] C_2 N \right. \\
&\quad \left. + \frac{\theta(2P - P^2)}{(1 - P/B_i)(1 - P/B_j)} - \frac{\phi(P - P^2)}{(1 - P)(1 - P)}\right\}
\end{aligned}$$

Since

$$\frac{\theta}{(1 - P/B_i)(1 - P/B_j)} \geqslant \frac{\phi}{(1 - P)(1 - P)}$$

$$\Delta_j(\tau) - \Delta_i(\tau) < 0 \qquad \text{if} \quad B_j > B_i$$

Hence one obtains greatest improvement incrementally by selecting the domain with largest range.

Suppose that a set D^* of indexed domains has been chosen and that a domain j exists in $\bar{D}^*$ such that $B_j > B_i$ for some domain $i \in D^*$. Temporarily delete i from D^*. Only two possibilities can occur: (1) both i and j cannot be added with improvement. In this case the performance has been improved by deleting a domain. (2) Domain j offers performance advantages over domain i. In this case interaction time is reduced by adding j to D^* and deleting i. After a sequence of such steps, during each of which interaction time is reduced, a set of indexed domains results, each having a larger number of elements in its associated partition than any nonindexed domain. It is possible that interaction can be improved by adding more domains, but the above argument can be repeated to demonstrate the theorem. ∎

Theorem 2. In the case where

$$P_1 > P_2 > \cdots > P_n \qquad \text{but} \quad B_1 = B_2 = \cdots = B_n$$

one should index the first i domains, stopping only when inclusion of the next domain does not reduce interaction time or when it violates a constraint on the number of indices.

Proof. If D is the set of indexed domains and $\bar{D}$ is the set of those that are unindexed, then we may again consider

$$\begin{aligned}\Delta_j(\tau) - \Delta_i(\tau) = &\left(C_1 + \frac{C_2 N}{B}\right)(1 - Q)(P_j - P_i) \\ &+ \theta\left[\frac{(2P_j/B) - P_j}{1 - (P_j/B)} - \frac{(2P_i/B) - P_i}{1 - (P_i/B)}\right] \\ &- \phi\left[\frac{P_j}{B(1 - P_j)} - \frac{P_i}{B(1 - P_i)}\right]\end{aligned}$$

i.e.,

$$\begin{aligned}\frac{\Delta_j(\tau) - \Delta_i(\tau)}{P_j - P_i} = &\left(C_1 + \frac{C_2 N}{B}\right)(1 - Q) + \theta\left\{\frac{-1 + (2/B)}{[1 - (P_j/B)][1 - (P_i/B)]}\right\} \\ &+ \phi\left[\frac{-1}{B(1 - P_i)(1 - P_j)}\right] \qquad (5)\end{aligned}$$

If $\Delta_j(\tau) < 0$, then subtracting $(4 - 2V)\, QC_2N/mB$ from (4) and dividing by P_j yields

$$\begin{aligned}&\left[\frac{2QV}{mP_i} + (1 - Q)\right]\left(C_1 + \frac{C_2 N}{B}\right) \\ &\quad + \frac{\theta}{[1 - (P_j/B)][1 - (P_i/B)]}\left(-1 + \frac{2}{B}\right)\left(1 - \frac{P_i}{B}\right) \\ &\quad + \frac{\phi(-1 + P_i)}{(1 - P_i)\, B(1 - P_j)} < 0 \qquad (6)\end{aligned}$$

Since all terms in (6) are larger than the corresponding terms in (5), it is clear that

$$\Delta_j(\tau) < 0 \Rightarrow \{[\Delta_j(\tau) - \Delta_i(\tau)]/(P_j - P_i)\} < 0 \qquad (7)$$

A conclusion of (7) is that greatest improvement results if the most likely domain is indexed. A similar argument to that used in the previous theorem now yields the desired result. ∎

Theorem 3 is a straightforward extension of the foregoing theorems. Therefore, details will not be presented.

Theorem 4. If $P_1 > P_2 > \cdots > P_n$ and $P_i * B_i = 1$, $P_i \neq 1$, for all i, then the best selection of indices must contain all domains between any two which it contains.

Proof. Consider three unindexed domains i, j, and k such that $P_i > P_j > P_k$. Suppose D is the set of indexed domains. Consider two quantities

$$\Delta_1 = E_{D+ij}(\tau) - E_{D+ik}(\tau) \qquad \text{and} \qquad \Delta_2 = E_{D+jk}(\tau) - E_{D+ik}(\tau)$$

It can be shown that

$$\Delta_1 = (P_j - P_k)(K_1 + K_2\psi_1 + K_3\psi_2 + K_4\psi_3)$$
$$\Delta_2 = (P_j - P_i)(K_1 + K_2\psi_1' + K_3\psi_2' + K_4\psi_3')$$

where

$$K_1 = C_1(1 - Q) + (4QC_2N/m), \qquad K_2 = C_2N(1 - Q)$$
$$K_3 = \frac{\theta}{(1 - P_i^2)(1 - P_j^2)(1 - P_k^2)}, \qquad K_4 = \frac{\phi}{(1 - P_i)(1 - P_j)(1 - P_k)}$$
$$\psi_1 = P_j + P_k, \qquad \psi_1' = P_j + P_i$$
$$\psi_2 = (P_i^2 + 1 - P_i)(2P_j + 2P_k - P_jP_k - 1)$$
$$\psi_2' = (P_k^2 + 1 - P_k)(2P_j + 2P_i - P_iP_j - 1)$$
$$\psi_3 = (P_i^2 + 1 - P_i)(P_jP_k - P_j - P_k)$$
$$\psi_3' = (P_k^2 + 1 - P_k)(P_iP_j - P_i - P_j)$$

Since $P_i \leqslant 1/2$, it can also be shown that

$$\psi_1' > \psi_1, \qquad \psi_2' > \psi_2, \qquad \text{and} \qquad \psi_3 > \psi_3'$$

and that

$$\psi_2' - \psi_2 \geqslant \psi_3 - \psi_3'$$

Since $K_3 \geqslant K_4$, we then know that $K_1 + K_2\psi_1' + K_3\psi_2' + K_4\psi_3' > K_1 + K_2\psi_1 + K_3\psi_2 + K_4\psi_3$. Suppose $\Delta_2 > 0$. Then $K_1 + K_2\psi_1' + K_3\psi_2' + K_4\psi_3' < 0$ and $K_1 + K_2\psi_1 + K_3\psi_2 + K_4\psi_3 < 0$. Thus $\Delta_1 < 0$ and $\min[\Delta_1, \Delta_2] < 0$. Therefore $D + ik$ is never preferred to both $D + ij$ and $D + jk$. A similar argument to that used in Theorem 1 can now be used to justify the adjacent nature of the best set of domains. ∎

Theorem 5 will be demonstrated in steps and two intermediate results will be presented first. Since L combined indices will involve a constant expected update time regardless of their composition, we can be concerned solely with the expected retrieval time. Also, there will be no false drops to deal with.

It is evident that the adjacency property is true for all possible sets of indices for $n = 2$. We turn now to the case where $n = 3$ and show that $\{13, 2\}$ is never a preferred set of combined indices.

Theorem A. In the case where $n = 3$, $L = 2$, $B_1 = B_2 = B_3 = B$ and $P_1 \geqslant P_2 \geqslant P_3$, $\{13, 2\}$ is never preferred over both $\{12, 3\}$ and $\{1, 23\}$.

Proof. Let $\{E_i(\tau)\}$, $1 \leqslant i \leqslant 3$, represent the expected retrieval time for $\{i, jk\}$, where $j \neq k \neq i$. It can be demonstrated that

$$E_i(\tau) = P_i[C_1 + (C_2N/B)] + P_jP_k[C_1 + (C_2N/B^2)] + [P_j(1 - P_k) + P_k(1 - P_j)]\, B[C_1 + (C_2N/B^2)]$$

Let $M_{ij} = E_i(\tau) - E_j(\tau)$ and let k be the third domain. By algebraic manipulation it can be demonstrated that $M_{ij} = (P_i - P_j)\{C_1(1 - B) + P_k(2B - 1)[C_1 + (C_2N/B^2)]\}$. If $M_{12} > 0$, it follows easily that M_{23} and M_{13} are also positive. Hence $\{12, 3\}$ is the best choice. If $M_{12} \leqslant 0$, then $\{13, 2\}$ is not preferred to $\{1, 23\}$. ∎

We can now demonstrate the result for $n = 4$.

Theorem B. For $n = 4$ and $L \in \{2, 3\}$ the best set of indices must have the adjacency property.

Proof. There are several cases to be considered separately. Treat $L = 2$ and the case where D_1 and D_2 each contains two domains. Let i, j, k, l be the elements of 1, 2, 3, 4 and $E_{ij,kl}(\tau)$ be the expected retrieval time for $\{ij, kl\}$. It can be demonstrated that

$$E_{ij,kl}(\tau) = (C_1 + C_2N/B^2)(P_iP_j + P_kP_l) + B(C_1 + C_2N/B^2) \times [P_i(1 - P_j) + P_j(1 - P_i) + P_k(1 - P_l) + P_l(1 - P_k)]$$

Let

$$M_1 = E_{12,34}(\tau) - E_{13,24}(\tau), \qquad M_2 = E_{12,34}(\tau) - E_{14,23}(\tau)$$

A lengthy manipulation shows that

$$M_1 = -(P_2 - P_3)(P_1 - P_4)(2B - 1)(C_1 + C_2N/B^2)$$
$$M_2 = -(P_2 - P_4)(P_1 - P_3)(2B - 1)(C_1 + C_2N/B^2)$$

Clearly, $\{12, 34\}$ is the best choice.

Now consider the case where $L = 2$ and D_1 will have three domains and D_2 only one. Hence

$$\begin{aligned} E_{ijk,l}(\tau) = {} & P_iP_jP_k(C_1 + C_2N/B^3) \\ & + B[P_iP_j(1 - P_k) + P_iP_k(1 - P_j) + P_jP_k(1 - P_i)] \\ & \times (C_1 + C_2N/B^3) \\ & + B^2[P_i(1 - P_j)(1 - P_k) + P_j(1 - P_i)(1 - P_k) \\ & + P_k(1 - P_i)(1 - P_j)](C_1 + C_2N/B^3) \\ & + P_l(C_1 + C_2N/B) \end{aligned}$$

In this case let

$$M_{ij} = E_{jkl,i}(\tau) - E_{ikl,j}(\tau)$$

A lengthy algebraic computation demonstrates that for $i < j$

$$\begin{aligned} M_{ij} = {} & \{-P_kP_l[(A_0 + C_1)(1 - 3B + 3B^2)] \\ & + (P_k + P_l)[(A_0 + C_1)(2B^2 - B)] - C_1(B^2 - 1)\}(P_i - P_j) \end{aligned}$$

Here $A_0 = C_2N/B^3$.

For any B as P_k and P_l vary it is easy to ascertain that M_{ij} has a single connected region of positive sign and one of negative sign separated by the curve $M_{ij} = 0$ and that along this curve $dP_k/dP_l < 0$. Hence the regions have the composition indicated in Fig. 3.

Clearly, if M_{14} is in region I, then so are M_{24} and M_{34}. Hence {123, 4} is the best choice. Alternately, if M_{14} is in region II, then so are M_{13} and M_{12}. Thus {1, 234} is the best choice and this portion of the theorem is demonstrated.

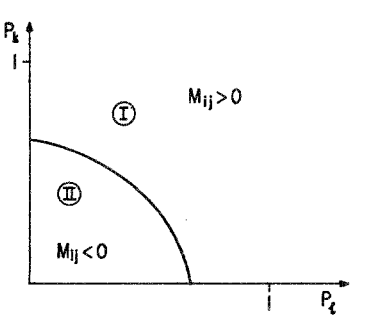

Fig. 3. Two regions of interest.

Finally consider the case where $L = 3$. With three indices and four domains any two may be grouped together and the allowable combinations are:

1. {12, 3, 4}, 2. {13, 2, 4}, 3. {23, 1, 4}
4. {14, 2, 3}, 5. {24, 1, 3}, 6. {34, 1, 2}

Consider the first three alternatives:

$$E_{12,3,4}(\tau) = P_4(C_1 + C_2N/B) + E_{12,3}(\tau)$$
$$E_{13,2,4}(\tau) = P_4(C_1 + C_2N/B) + E_{13,2}(\tau)$$
$$E_{23,1,4}(\tau) = P_4(C_1 + C_2N/B) + E_{23,1}(\tau)$$

Since $E_{13,2}(\tau) \geqslant \min\{E_{12,3}(\tau), E_{23,1}(\tau)\}$, then

$$E_{13,2,4}(\tau) \geqslant \min\{E_{12,3,4}(\tau), E_{23,1,4}(\tau)\}$$

In a similar fashion alternatives 4 and 5 are never preferred and the theorem is proven. ∎

It is now a simple matter to extend the above results using the technique of the latter part of Theorem B to demonstrate the following alternative form of Theorem 5, which we state without proof.

Theorem 5. If $n - L \leqslant 2$, the best choice of indices must have the adjacency property regardless of the value of n.

Theorem 6. If $n = 3$, $L = 2$, $P_1 = P_2 = P_3 = P$, and $B_1 \geqslant B_2 \geqslant B_3$, then {1, 23} is the best choice of indices.

Proof. Let i, j, k be the elements of {1, 2, 3}. It is easily shown that

$$E_{i,jk}(\tau) - E_{j,ik}(\tau) = (B_j - B_i)\left[(P - P^2)\, C_1 + \frac{P^2 C_2 N}{B_i B_j} \frac{B_k - 1}{B_k}\right]$$

Clearly {1, 23} is superior to any other possibility. ∎

Theorem 7. If $n = 4$, $L = 2$, $P_1 = P_2 = P_3 = P_4 = P$, and

$$B_1 \geqslant B_2 \geqslant B_3 \geqslant B_4$$

and two domains are to be in each index, then {14, 23} is the best choice.

Proof. Let i, j, k, l be the elements of {1, 2, 3, 4}. It is easily shown that

$$E_{ij,kl}(\tau) - E_{ik,jl}(\tau) = \frac{P^2 C_2 N}{B_i B_j B_k B_l} (B_k - B_j)(B_l - B_i)$$

Clearly, {14, 23} is the best choice. ∎

REFERENCES

1. E. F. Codd, "A relational model of data for large shared data banks," *CACM* **13**(6) (1970).
2. R. Blier and A. Vorhaus, "File organization in the SDC time shared data management (TDMS) system," in *Proc. 1968 IFIP Congress.*
3. V. Y. Lum, "Multiattribute retrieval with combined indices," *CACM* **13**(11) (1970).
4. F. Palermo, "A quantitative approach to the selection of secondary indexes," *Formatted File Organization Techniques* (IBM Research, San Jose, March 1970).
5. J. Mullin, "Retrieval–update speed tradeoffs using combined indices," *CACM* **14**(12) (1971).
6. V. Y. Lum and H. Ling, "An optimization problem on the selection of secondary keys," in *Proc. ACM Nat. Conf., 1971.*
7. E. Wong and T. Chiang, "Canonical structure in attribute based file organization," *CACM* **14**(9) (1971).
8. CODASYL Systems Committee, "Feature analysis of generalized data base management systems," May 1971.
9. E. F. Codd, "A data base sublanguage founded on the relational calculus," in *Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control,* San Diego.

Printed in Belgium