An Approximation Algorithm for Secondary Index Selection in Relational Database Physical Design

R. BONANNO
CRAI, Cosenza, Italy

D. MAIO AND P. TIBERIO
CIOC-CNR, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna

The operational efficiency of a relational DBMS is greatly dependent on the effectiveness of the physical design. A problem of considerable interest for the DBA is to provide an efficient set of access structures taking into account both memory constraints and workload characteristics. The paper presents an approximation algorithm which produces a near-optimal solution to the secondary index selection problem for the entire set of DB relations, given that primary access structures have already been chosen.

1. INTRODUCTION

As DBMSs and their usage become complex, the physical design becomes an increasingly difficult operation due to the high number of factors to be considered. For this reason a considerable research effort has been made in developing automatic tools to help the database administrator (DBA) to make decisions and trade-offs. One of the ultimate objectives of a database designer is to provide an optimal set of access path structures in order to minimise the time spent in I/O operations, taking into account maintenance costs, memory constraints and workload characteristics. This problem has to be solved not only on DB installation but also for subsequent tuning, when the workload is expected to change and/or new files may be added to an existing DB. The paper focuses attention on the secondary index selection problem in relational DBMSs.

At loading time most file systems require the creation of a primary access path on a given relation column.[1-3] Secondary indexes are also allowed in order to reduce the access time for some DB operations. In System R a tool named DBDSGN is available which accepts all valid SQL statements in the workload and solves the combined problem concerning the column on which each relation should be ordered and on which other columns' secondary indexes should be created.[4] A methodology for secondary index selection, primarily applicable to small/medium-size DBMSs whose characteristics are more limited than those of System R, has been developed at the University of Bologna within the Informatica (DATAID) project supported by the Italian National Council of Research.[3,5] These methodologies consider the case of DBMSs which use at most one index per relation to retrieve tuples when executing a statement. An overall description of the major steps of the index selection method follows.

Step (1) The specification of DB statistics and workload characteristics.

Step (2) The cost evaluation for each transaction class and for each possible access path.

Step (3) The access path effectiveness comparison, in order to determine those indexes which are obviously useless or providing minor benefits when others are present; in other words, the generation of an efficient set of indexes taking into account the constraints on storage.

Step (1) derives directly from the DB logical design in terms of workload and relation characteristics. At the implementation time, collecting more accurate parameters would require monitoring on an operating database system.

The objective of step (2) is to find, for each relation, a set of indexes which are candidates for the optimal set. A correct solution for this step can be reached by looking at statistical models and evaluation functions, provided precise considerations are made on both the physical implementation of the relations and indexes and the strategies used in solving transactions (in particular, join methods and index maintenance techniques.[6-12]

Step (3), that is the generation procedure of the optimal set, has to select the indexes that ensure the minimal global cost under the constraints imposed by the DBA on the memory. This problem has been shown by Comer[13] to be NP-hard even for simple cost criteria. In[4] the design tool utilises an algorithm which requires an exponential exploration of all possible index subsets if applied without restrictions being introduced under user control. More recently in[14] it was shown that the index selection problem can be solved by solving a properly chosen instance of the Knapsack problem (KP). In the context of a query-processing model which assumes the use of intersection lists Ref. 14 presents, for a single relation, an approximation algorithm which solves the KP in polynomial time. Another recent study concerns Wang's approach to the optimal physical design of multiple relational data bases,[15] in which two algorithms based on the theory of separability are presented. A theory was introduced in[16] as a formal basis for understanding the interrelationships among files: i.e. joins are separated into single-table queries, given a set of join methods that satisfies a certain property called separability. (Note that not all join methods are 'separable', see for instance those used in System R.) A separability-based method for secondary index selection was presented in Ref. 3. Finally it is also worth mentioning that other approaches and formalisations of the index selection problem for files can be found in Refs. 17–22.

The aim of this paper is to describe a secondary index selection method, based on optimal assignment principles, that explores a number of combinations linearly proportional to the cardinality of the candidate indexes set. The method is applied in a multiple environment pro-

vided join methods are separable. Section 2 introduces assumptions on the query-processing model. Definitions and notations are also discussed in the same section. Section 3 formulates the index selection problem in terms of minimisation of an objective function and proposes an approximation algorithm. Section 4 gives an example.

2. ASSUMPTIONS AND DEFINITIONS

Several assumptions are made throughout the paper. We assume that the DBMS being considered provides the following characteristics:

(*a*) the indexes are structured as B$^+$-trees and the leaves contain all the key values, each followed by the set of tuple identifiers (TIDs) where the value appears;
(*b*) at most, one index per relation can be used to access tuples in executing a statement;
(*c*) joins are performed according to separable or approximately separable methods:
(*d*) an index cannot be used to access tuples for an update statement when it is currently modified, since this may lead to hitting the same TID more than once in practical systems.[9]

We do not deal with details concerning the first two design phases: (1) the specification of DB statistics and workload characteristics and (2) the cost evaluation for each transaction class and each possible access path. A comprehensive analysis of these two mandatory steps of the physical design is given in Refs. 3, 4, 5, 7, 9 and 15. In the following we suppose, as a result of steps (1) and (2) of the previous section, a set of matrices to have been built which will represent, along with some workload characteristics, the inputs to the secondary index selection algorithm. Here we would repeat that our approach assumes that the choice of the primary keys has already been made in the DB logical design phase.

Hypothesis (c) on the join execution method allows application of the query-separability concept;[16] in order to compare the effectiveness of indexes built on columns belonging to the same table the join itself is considered to be subdivided into distinct, single-table queries. Therefore the workload satisfies the following rules:
all the joins are separated in single-table queries; queries and update statement are grouped in order to characterise a workload for each table independently.
Since the system uses only one index per table, in order to evaluate the effectiveness of a given index on the global workload cost, the methodology requires computation of the cost of each statement where only that index is present in the relative table. This cost is generally expressed as the weighted sum of I/O and CPU operations. Let us introduce the following notations:

T_h a generic table subject to the physical design, h = 1, ..., NT.
Q_h the set of NQ_h statements q_i^h which refer to table T_h. All the Q_h are disjoint sets (joins have already been separated); $Q \equiv \cup_h Q_h$ is denoted as the global workload with NQ cardinality.
$ISET_h$ the set of the NI_h secondary indexes which are candidates to be built on the T_h table. All the $ISET_h$ are disjoint sets: we denote $ISET$ as $\cup_h ISET_h$ with cardinality NI.
$|A^h|$ the access cost matrix relative to table T_h. In the case of a selection statement $q_i^h \in Q_h$ the element A_{ij}^h represents the minimum cost of accessing

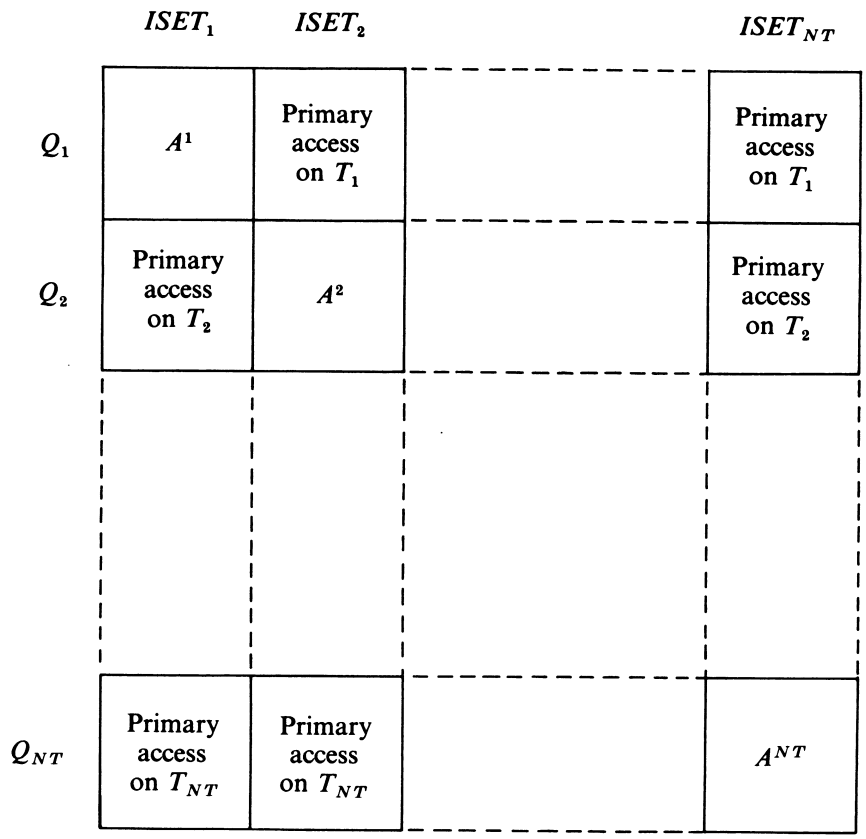

Figure 1. The access cost global matrix.

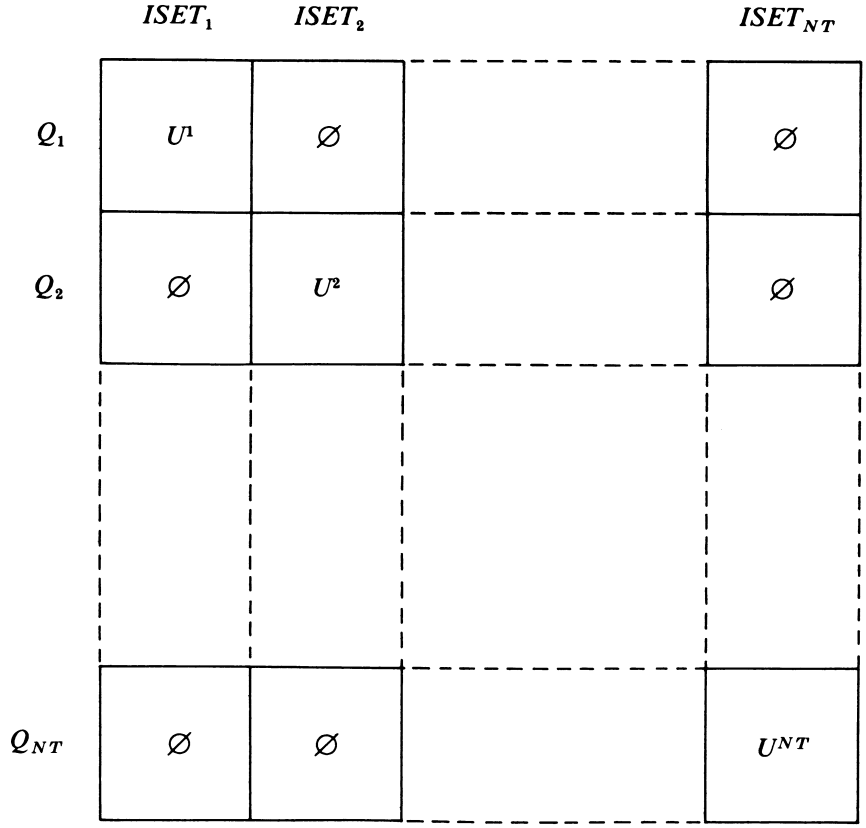

Figure 2. The maintenance cost global matrix.

tuples when only the secondary index $I_j^h \in ISET_h$ is available, in addition to the primary access path and, possibly, the sequential relation scan when allowed. In the case of a maintenance statement q_i^h, A_{ij}^h also includes the maintenance cost of the primary access and relation.

$|U^h|$ the secondary index maintenance cost matrix relative to table T_h. Each U_{ij}^h represents the cost of updating the index $I_j^h \in ISET_h$ due to the $q_i^h \in Q_h$ statement.

$\overline{MEM^h}$ the storage cost vector for each table T_h; each MEM_j^h element represents the secondary storage pages required by the index $I_j^h \in ISET_h$.

$\bar{P}^h$ the primary access cost vector for table T_h; each P_i^h element represents the minimum between the cost of the primary access path on T_h and the cost of the sequential relation scan (if allowed) paid to access tuples in solving the statement q_i^h. Where q_i^h is a maintenance statement then P_i^h includes the primary access maintenance cost and the relation maintenance cost.

In the former definitions we have implicitly supposed that index maintenance costs are constant, no matter the access path chosen. This assumption is commonly made in most of the papers on index selection, even if it is an approximation as shown in Ref. 9. We have kept the above hypothesis, since it simplifies the optimal set generation algorithm, and the approximations introduced are in most cases of the same level as those commonly made in estimating costs.

A remark is perhaps in order. Each statement q_i that the DBA considers to be relevant for the design must be included in the workload Q. The choice of relevant statements is carried out by analysing the user's requirements, which are derived from questionnaires and/or available measurements. For each q_i the DBA analysis of the workload must provide a weight w_i that is a function of:

the frequency of execution over the period of time t during which the workload is not expected to change substantially;

distribution of its execution during t;

particular needs for response time.

In order to find a mathematical formulation of the index selection problem let us introduce the following further definitions.

$|GA|$ the global matrix obtained as in Fig. 1.

Let $q_i \in Q$ and $I_j \in ISET$,

$|GA|$ is defined as follows:

for all $i, j, i', j' / q_i \equiv q_i^h$ and $I_j \equiv I_j^h$: $GA_{ij} = A_{i'j'}^h$

for all $i, j, i', j' / q_i \equiv q_i^h$ and $I_j \not\equiv I_{j'}^h$: $GA_{ij} = P_{i'}^h$

Analogously we define the global matrix $|GU|$ obtained as in Fig. 2, where

for all $i, j, i', j' / q_i \equiv q_i^h$ and $I_j \equiv I_{j'}^h$

$GU_{ij} = U_{i'j'}^h$

for all $i, j, i', j' / q_i \not\equiv q_i^h$ and $I_j \not\equiv I_{j'}^h$

$GU_{ij} = \emptyset$

Since the cost matrices for the multi-table problem were unified, in the following section r is denoted, for simplicity of exposition, as the statement identifier, $r = 1, \ldots, NQ$. Furthermore, we denote s as the index identifier, $s = 1, \ldots, NI$.

3. THE INDEX SELECTION PROBLEM

On the basis of the definitions introduced in the previous section, the index selection problem can now be formulated as follows.

Given the set *ISET* of the secondary indexes, find the subset $BSET \subseteq ISET$ such that the global cost is minimal

$$C = \sum_r w_r * [\min \{GA_{rs}/s \in BSET\} + \sum_{s \in BSET} GU_{rs}]$$

C can be rewritten as:

$$C = \sum_r \min \{w_r * GA_{rs}/s \in BSET\} + \sum_{s \in BSET} \sum_r w_r * GU_{rs}$$

Denoting with $GA_{rs} = w_r * GA_{rs}$ for all r, s

$$GU'_s = \sum_r w_r * GU_{rs} \quad \text{for all } s$$

we have

$$C = \sum_r \min \{GA'_{rs}/s \in BSET\} + \sum_{s \in BSET} GU'_s \qquad (1)$$

The problem is then approached by the minimisation of the following objective function:

$$z = \sum_r \sum_s GA'_{rs} * x_{rs} + \sum_s GU'_s * y_s$$

subject to

$$\sum_{s \in BSET} MEM_S \leqslant M$$

and;

$$\sum_s x_{rs} = 1 \quad \text{for} \quad r = 1, \ldots, NQ$$
$$x_{rs} \leqslant_1^0 \quad \text{for} \quad r = 1, \ldots, NQ; \quad s = 1, \ldots, NI$$
$$y_s \leqslant_1^0 \quad \text{for} \quad s = 1, \ldots, NI$$

where:

$x_{rs} = 1$ means that we are considering the statement r as solved using the index s or the primary access path;

$x_{rs} = 0$ means that the index s is not used for r;

$y_s = 1$ if $s \in BSET$:

$y_s = 0$ if $s \in BSET$:

M is the total memory constraint for the secondary indexes (recall that for all $s \in ISET$ we have $MEM_s \leqslant M$).

Let $\bar{V}$ be the vector which, in the rth entry, stores the identifier of the access path used by the statement r. The search for an initial solution *BSET* provides for initialisation of V: initially V_r contains the identifier of the index (secondary or primary) whose access GA'_{rs} is minimum for the statement r. Note that the initial *BSET* does not necessarily coincide with *ISET*.

Let *TEST* be the set of secondary indexes which, at a given step of the generation algorithm, have already been tested for a given group of statements. Initially it is the empty set.

The approximation algorithm SIS (secondary index selection) is essentially composed of two parts: the first ($p1$) finds an optimal solution without memory constraint, the second ($p2$) operates when the unconstrained solution does not fit in the available memory. Since the initial solution was found without considering the update costs, part $p1$ tries to remove from *BSET* the indexes with higher update cost. In this phase indexes can simply be removed from *BSET* or others substituted that were not considered in the initialisation.

All the indexes are tested in order to minimise the objective function. Part $p2$ takes the constraint on the memory into account by performing substitutions in *BSET*. More precisely, we remove from *BSET* the index whose memory requirement (weighted by a function $F(MEM_k)$ defined by the user) multiplied by its global cost, is maximum. Then, we replace it with another index whose ratio between the global cost increase and the memory gain is minimum. We repeat substitutions until the indexes in *BSET* satisfy the memory limit. SIS is reported as follows. We suppose that the initial *BSET* has been already chosen. Furthermore, we denote P'_r as $P_r * w_r$.

SIS: Secondary index selection

$p1$: (* find *BSET* without constraints on the storage*)
while ($\exists\, s \in BSET/s \in TEST$) **do**
begin
$p1.1$:(* find the candidate s to be removed from *BSET**)
find $s \in BSET/GU'_s = \max \{GU'_j/j \in BSET, j \notin TEST\}$:
$p1.2$:(* evaluate the effectiveness of the index s for the group of statements which use s*)
for all k $\in$ *ISET* **do**
begin
$SOM_k := \sum_{r/V_r = s} GA'_{rk}$;
(* discover if the maintenance cost has to be considered*)
if ($k \notin BSET$ *or* $k = s$) **and**
($\exists\, GA'_{rk}/V_r = s \Rightarrow GA'_{rk} \neq P'_r$)
then $SOM_k := SOM_k + GU'_k$
end;
(* search the index k which is more useful than s*)
find $kmin/SOM_{kmin} = \min \{SOM_k/k \in ISET\}$:
(* update the vector V*)
for all $r/V_r = s$ **do** $V_r := kmin$:
$p1.3$:(* update the solution *)
$BSET := \{k \in ISET/\exists V_r = k\}$
if $kmin \in TEST$ **then** $TEST := TEST\ kmin$;
(* update the set of the indexes already tested*)
$TEST := TEST \cup s$
end:
(* check the validity of the solution *)
for all r
do begin $j := V_r$;
if $GA'_{rj} = P'_r$ **then** $V_r := 0$:
end:
$BSET := \{k/\exists r$ such that $V_r = k\}$
(* end $p1$*)
$p2$: (* take into account the constraints on the memory*)
while $\sum_{s \in BSET} MEM_s > M$ **do**
begin
$p2.1$: **for all** $k \in BSET$
do $PROD_k = (\sum_{r/V_r = k} GA'_{rk} + GU'_k) * F(MEM_k)$;
(* $F(MEM)$ is a function defined by the user *);

(* find the index candidate to be removed from *BSET**)
find $kmax \in BSET/PROD_{kmax} = max \{PROD_s/s \in BSET\}$;
(*find the set of indexes which can substitute $kmax$*)
$SSET := ISET\text{-}kmax\text{-}\{k \in ISET\text{-}BSET/MEM_k \geqq MEM_{kmax}\}$
(*as in the $p1.2$ step*)

for all $k \in SSET \cup kmax$ **do**

begin $SOM_k := \sum_{r/V_r = kmax} GA'_{rk}$;
if ($k \notin BSET$ or $k = kmax$) **and**
($\exists\, GA'_{rk}/V_r = kmax \Rightarrow GA'_{rk} \neq P'_r$)
then $SOM_k := SOM_k + G\,U'_{rk}$
end:
(* find the index *kmin* which substitutes *kmax* *)
find $kmin \in SSET$ such that the ratio is the minimum:

$$\left[\frac{SOM_k - SOM_{kmax}}{min\ \{(\sum_{s \in BSET} MEM_s\text{-}M),\ (MEM_{kmax}\text{-}NEWMEM_k)\}}\right]$$

where $NEWMEM_k = \varnothing$ if $k \in BSET$
$NEWMEM_k = MEM_k$ if $k \notin BSET$;
(* is the ratio between the drawback introduced by k and the maximum memory gain (not greater than the memory overflow)*)

*p*2.2: (* update V and BSET
for all $r/V_r = \varnothing$ and GA'_r, $kmin \neq P'_r$
do $V_r := k_{min}$;
for all $r/V_r = k_{max}$
do if GA'_r, $k_{min} \neq P'_r$
then $V_r := kmin$ **else** $V_r := \varnothing$;
$BSET := \{k/\exists r$ such that $V_r = k\}$
end (* *p*2 *)

4. AN EXAMPLE

In order to show the application of the algorithm SIS and to explain the meaning of its steps, we have built up a simple example. Let us suppose we have to select indexes for two relations. The following tables report the cost matrices as defined in section 2.

relation T_1

$|A^1|$:

	I^1_2	I^1_2
q^1_1	13	62
q^1_2	18	14
q^1_3	40	40

$|U^1|$:

	I^1_2	I^1_2
q^1_3	0	10
q^1_2	0	0
q^1_3	10	10

$\overline{P^1}$:

q^1_1	62
q^1_2	40
q^1_3	40

$\overline{MEM^1}$:

I^1_1	I^1_2
10	5

relation T_2

$|A^2|$:

	I^2_1	I^2_2	I^2_3
q^2_1	17	67	27
q^2_2	11	6	21
q^2_3	40	40	15

$|U^2|$:

	I^2_1	I^2_2	I^2_3
q^2_1	0	20	0
q^2_2	0	0	0
q^2_3	8	8	5

$\overline{P^2}$:

q^2_1	67
q^2_2	25
q^2_3	40

$\overline{MEM^2}$:

I^2_1	I^2_2	I^2_3
10	8	6

we obtain the following global matrices:

$|GA|$:

	1	2	3	4	5
1	13	62	62	62	62
2	18	14	40	40	40
3	40	40	40	40	40
4	67	67	17	67	27
5	25	25	11	6	21
6	40	40	40	40	15

$|GU|$:

	1	2	3	4	5
1	0	10	0	0	0
2	0	0	0	0	0
3	10	10	0	0	0
4	0	0	0	20	0
5	0	0	0	0	0
6	0	0	8	8	5

$\overline{P}$:

1	62
2	40
3	40
4	67
5	25
6	40

MEM :

1	2	3	4	5
10	5	10	8	6

Let us suppose weights are given as:

$\overline{W}$:

1	100
2	50
3	20
4	100
5	200
6	50

By applying the transformation formula (1) of section 3 we obtain:

$|GA'|$:

	1	2	3	4	5
1	1300	6200	6200	6200	6200
2	900	700	2000	2000	2000
3	800	800	800	800	800
4	6700	6700	1700	6700	2700
5	5000	5000	2200	1200	4200
6	2000	2000	2000	2000	750

$\overline{P'}$:

1	6200
2	2000
3	800
4	6700
5	5000
6	2000

$\overline{GU'}$:

1	2	3	4	5
200	1200	400	2400	250

$\overline{MEM}$:

1	2	3	4	5
10	5	10	8	6

We suppose a memory constraint $M = 20$, and $F(MEM_k) = MEM_k$.

BSET initially is built by choosing for each query the identifier of the secondary index whose access cost is minimum. Thus we have

BSET = {1, 2, 3, 4, 5} TEST = {∅}

$\bar{V}$ =

1	1
2	2
3	2
4	3
5	4
6	5

(1) The application of the step $p1.1$ of SIS leads to:

$s = 4$

After $p1.2$, $kmin = 3$ and $\bar{V}$ is updated as:

$\bar{V}$ =

1	1
2	2
3	2
4	3
5	3
6	5

Step $p1.3$ updates *BSET* and *TEST* as follows

BSET = {1, 2, 3, 5} TEST = {4}

(2) The new candidate to be removed from *BSET* is $s = 2$. Applying $p1.2$ and $p1.3$ we have $kmin = 1$.

BSET = {1, 3, 5}
TEST = {2, 4}

$\bar{V}$ =

1	1
2	1
3	1
4	3
5	3
6	5

(3) Now $s = 3$

$kmin = 3$ BSET = {1, 3, 5} TEST = {2, 3, 4}

Note that this iteration does not change *BSET* and consequently $\bar{V}$.

(4) Now $s = 5$

$kmin = 5$ *BSET* = {1, 3, 5} *TEST* = {2, 3, 4, 5}

(5) As a result of the last iteration of $p1$:

$s = 1$ $kmin = 1$ *BSET* = {1, 3, 5}
TEST = {1, 2, 3, 4, 5}

$\bar{V}$ =

1	1
2	1
3	1
4	3
5	3
6	5

Note that it could happen in other cases that an index, previously removed from *BSET* for some group of statements, may again enter into *BSET* for a larger group. This leads, as a consequence, to increasing the number of iterations at most to $NI+NQ$.

The next step to be performed consists in checking the validity of the solution obtained without constraints on the memory. In fact, the third query in reality does not use the secondary index 1 but the primary access. Thus we update $\bar{V}$ as follows:

$\bar{V}$ =

1	1
2	1
3	∅
4	3
5	3
6	5

The last step $p2$ takes into account the constraints on the memory by performing substitutions in *BSET*.

The *kmin* index is chosen among those belonging to the following two sets:

{*BSET-kmax*} with no restriction;

{*ISET-BSET*} but considering only those indexes whose memory requirement is less than *kmax* memory occupation.

Referring to our example, as a result of $p2$ first iteration we have:

$kmax = 3$, $kmin = 5$

BSET = {1, 5}

$\bar{V}$ =

1	1
2	1
3	0
4	5
5	5
6	5

This solution satisfies the memory constraints and has a global cost of 11100. The number of combinations examined is 7.

In order to validate SIS's results we have implemented an exhaustively searching algorithm. We found for the previous example the same result by performing (2^5-1) explorations. Tests on SIS have been performed to try and measure the deviations of the heuristic solution from the optimal one. In most situations SIS produced the optimal solution. Because physical design is such a complex

problem, finding the mathematical worst-case bound on the deviations from the optimal solutions produced by heuristic algorithms is virtually impossible.[15] Intuitively the introduced approximation is due to the fact that SIS does not consider the possibility of replacing an index with a group of indexes. Let us refer to the following example, in which we have to select indexes for a single relation. We assume unitary weights for all queries.

$|GA'|$:

1	110	210	180
2	200	300	190
3	300	240	205
4	60	55	50
5	120	110	325
6	300	10	180

1	0	40	0
2	0	30	0
3	80	0	0
4	0	0	0
5	0	0	70
6	0	0	0

$\bar{P}$:

1	210
2	300
3	300
4	100
5	325
6	400

$\overline{MEM}$:

10	8	12

We suppose a memory constraint $M = 23$.

The optimal solution is $BSET = \{1, 2\}$ with cost = 875. SIS's solution is $BSET = \{2, 3\}$ with cost = 885. In fact when, during the execution of SIS, we reach the solution $BSET = \{2, 3\}$ the situation of the vector V is:

1	3
2	3
3	3
4	3
5	2
6	2

Then we stop the execution because no index can be replaced completely by another. On the other hand, we could obtain the optimal solution {1, 2} only if we substituted the index 3 with the couple of indexes {1, 2} for the first four queries.

Another deviation introduced by SIS is caused by the fact that the necessary constraint is taken into account only after a first approximation solution has been found.

CONCLUSIONS

We have presented an approximation algorithm which produces a near-optimal solution to the secondary index selection problem in multifile relation DB. The complexity of the exploration is linear.

Future work in this direction will try to extend the complexity to the square of the cardinality of the indexes in order to avoid the approximation described above. This last algorithm is currently under development and will be published in a future paper.

REFERENCES

1. M. M. Astrahan *et al.*, System R, a relational approach to database management. *ACM TODS* **1** (2) (1976).
2. M. Stonebraker E. Wong and P. Kreps, The design and implementation of INGRES. *ACM TODS* **1** (3) (1976).
3. F. Bonfatti, D. Maio and P. Tiberio, A separability-based method for secondary index selection in physical data base design. In *Methodology and Tools for Data Base Design*, edited S. Ceri. Amsterdam, North-Holland (1983).
4. M. Schkolnick and P. Tiberio, Considerations in developing a design tool for a relational DBMS. *Proceedings of the IEEE COMPSAC Conference, Chicago, Nov. 1979*; and in *Data Base Management in the 1980's*, IEEE catalogue no. EHO-181-8 (1981).
5. F. Bonfatti, D. Maio and P. Tiberio, Metodologie di progetto fisico in DBMS relazionali. *Rivista di Informatica*, **8**, (2) (1983).
6. M. M. Astrahan, W. Kim and M. Schkolnick, Evaluation of the System R access path selection mechanism. *Proceedings of the IFIP Conference Tokyo/Melbourne, Oct. 1980.*
7. F. Bonfatti, D. Maio and P. Tiberio, Alcune considerazioni sul calcolo dei costi di esecuzione delle interrogazioni in una base di dati relazionale. *Proceedings of the AICA Conference, Pavia, Sept. 1981.*
8. S. B. Yao, Approximating block accesses in database organizations, *Communications of the ACM* **20**, (4) (1977).
9. M. Schkolnick and P. Tiberio, *A Note on Estimating the Maintenance Cost in a Relational Database.* IBM Research Report RJ 3327, San Jose Ca. (1981). To appear on *ACM TODS.*
10. E. Wong and K. Youssefi, Decomposition: a strategy for query processing. *ACM TODS* **1** (3) (1976).
11. K. Y. Whang, G. Wiederhold and D. Sagalowitz, Estimating block accesses in database organizations: a closed non-iterative formula. *Communication of the ACM* **26**, (11) (1983).
12. T. Y. Cheung, A statistical model for estimating the number of records in a relational database. *Information Processing Letters* **15** (3) (1982).
13. D. Comer, The difficulty of optimum index selection, *ACM TODS* **3** (4) (1978).
14. M. Y. L. Ip, V. V. Raghavan and L. V. Saxton, An approximation algorithm for the index selection problem. Proceedings of the *IEEE COMPSAC* Conference, Chicago, (1981).
15. K. Y. Whang, *Physical Design Algorithms for Multiple Relational Databases.* Internal Report, Stanford University (1982).
16. K. Y. Whang, G. Wiederhold and D. Sagalowicz, Separability: an approach to physical database design. *Proceedings of the International Conference on Very Large Databases, Cannes, France.* (1981).

17. W. F. King, *On the Selection of Indices for a File*. IBM Research Report RJ 1641, San Jose Ca. (1974).
18. J. G. Kollias, A heuristic approach for determining the optimal degree of file inversion. *Information Systems* **4** (1979).
19. A. Putkonen, On the selection of access path in inverted database organization. *Information Systems* **4** (1979).
20. M. Schkolnick, The optimal selection of secondary indices for files. *Information Systems* **1** (1975).
21. M. Stonebraker, The choice of partial inversions and combined indices. *International Journal of Computer and Information Sciences* **3** (2) (1974).
22. H. D. Anderson and P. B. Berra, Minimum cost selection of secondary indexes for formatted files. *ACM TODS* **2** (1) (1977).