1976

Multi-Dimensional Clustering for Data Base Organizations

J. H. Liou

S. B. Yao

Report Number:
77-213

MULTI-DIMENSIONAL CLUSTERING FOR DATA BASE ORGANIZATIONS

BY

J.H. Liou
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

and

S.B. Yao
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

CSD-TR 213

December 1976

(To appear in Information Systems)

MULTI-DIMENSIONAL CLUSTERING FOR DATA BASE ORGANIZATIONS

J. H. Liou*
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

S. B. Yao
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

ABSTRACT

A new record-clustering scheme is introduced, in which the record address is determined by multiple keys. Associated with this storage scheme is a new type of index called multi-dimensional directory. Those keys which determine the record address are jointly indexed by this directory. A data base structure which combines this new technique and the file inversion technique is analyzed. The costs of retrieval, update and storage space for this data base structure are mathematically formulated. An example illustrates that this new data base structure can be superior to the classical combination of indexed sequential and file inversion techniques.

Keywords: index, directory, multi-key query, inverted file, retrieval, update, file organization, data base.

CR Categories: 3.70, 3.73, 3.74, 4.33

*Supported by the Naval Electronics System Command, Contract N00039-75-C-0034 and ENG 74-17586.

MULTI-DIMENSIONAL CLUSTERING FOR DATA BASE ORGANIZATIONS

1. INTRODUCTION

A conjunctive multi-key query can be answered by utilizing the single key indices in either of the following two methods. In the first method, each single-key index is searched to find a set of record addresses satisfying the atomic condition involving the indexed key. Then intersection is performed on these sets to obtain the set of record addresses which satisfy the conjunctive query. In the second method, one key is selected and its index searched to find the records satisfying the atomic condition involving this key. The content of each qualified record is then checked with the conditions on other keys.

The first method requires many accesses to different index files. And the second method requires many accesses to records which do not satisfy the query. Furthermore, if the file is sorted on any single key (as in the case of an indexed sequential file), the target records of multi-key query will usually be distributed over the entire file space. The accesses to the main file can be as many as the number of target records. For example,

if a file has 60 pages and if 20 target records are randomly distributed over the 64 pages, then the expected number of page accesses is 17.4 [10].

In this paper a new method of record clustering is introduced. Unlike the binary partitioning method [1], the multi-dimensional key space (spanned by the index keys) is systematically divided into small cells using each key exactly once. Records that occur in the same cell are stored in the same page of secondary storage. A multi-dimensional directory (MDD) can be looked up to find the cells (pages) which contain the desired records.

It is then shown that a single key index (SKI) is preferable to MDD for handling a key that has a large number of distinct values. A hybrid data base structure that combines MDD, and SKI techniques is consequently proposed. The hybrid structure has the advantage that it handles most types of queries very efficiently.

For simplicity we make the following assumptions: File F contains a set of fixed length records. Each domain of keys $A_1, A_2, \ldots, A_K$ consists of a set of discrete values. An atomic condition is $A_i = a_i$, where a_i is in the domain of A_i. The most general query condition is defined as $q_\alpha \triangleq \bigwedge_{i \in \alpha} (A_i = a_i)$, where α is a string of integers selected unrepetitively from $\{1,2,\ldots,K\}$. For example $q_i \triangleq (A_i = a_i)$ and $q_{ijK} \triangleq (A_i = a_i) \wedge (A_j = a_j) \wedge (A_K = a_K)$. Since we are only interested in the "query condition" part of a query, throughout this paper the word query is often used to mean query condition.

2. MDD FILE STRUCTURE

In the following a two-dimensional directory on keys A1 and A2 of file F is illustrated. Denote the total number of records in F by $R \cdot N$, where R is the number of records blocked in a page and N is the number of data pages. Each record in F corresponds to a point in the space spanned by A_1 and A_2 (Fig. 1). Let m_1 and m_2 be two integers such that $m_2 = \lceil N/m_1 \rceil$, the smallest integer greater than or equal to N/m_1. In our particular example,

$N = 16$, $R = 4$, and $m_1 = m_2 = 4$. The file is organized by the following procedure:

(a) Divide domain A_1 into m_1 intervals $A_{11}, A_{12}, \ldots, A_{1m_1}$ such that each $A_{1i} \times A_2$ cell contains the same number $(R \cdot \frac{N}{m_1})$ of records. Call the cells obtained the 1st-degree cells. Denote a 1st-degree cell by C_i (Fig. 2).

(b) Divide each 1st-degree cell into m_2 subcells such that each subcell contains the same number $(R \cdot \frac{N}{m_1 m_2} = R)$ of records. Call the subcells the 2nd-degree cells and denote a 2nd-degree cell by C_{ij} (Fig. 3).

(c) Assign a page of secondary storage to each 2nd-degree cell. Store the records that occur in the same 2nd-degree cell into the page assigned to it.

(d) Enter the parameters into the multi-dimensional directory (Fig. 4). Note that the number of parameters equals the number of cells. That is, m_1 parameters are needed for the 1st-degree cells and $m_1 \times m_2$ parameters are needed for the 2nd-degree cells. For example, the parameter for C_2 is G and the parameter for C_{22} is 6. Also, there is a page pointer immediately following each 2nd-degree cell parameter.

It is very straightforward to generalize this procedure to the K-dimensional case. Each (i-1)th-degree cell is divided into m_i ith-degree cells for $I = 1,2,\ldots,K$. (The 0-th-degree cell is defined to be the whole key space.) A data page is assigned to each Kth-degree cell. A jth degree cell is denoted by $C_{i_1 i_2 \ldots i_j}$. $C_{i_1 \ldots i_j i_{j+1}}$ is a subcell of $C_{i_1 \ldots i_j}$, and $C_{i_1 \ldots i_j}$ is a supercell of $C_{i_1 \ldots i_j i_{j+1}}$. A total of $\sum_{j=1}^{K} \left(\prod_{i=1}^{j} m_i \right)$ parameters and $\prod_{I=1}^{K} m_i$ page pointers are stored in the MDD.

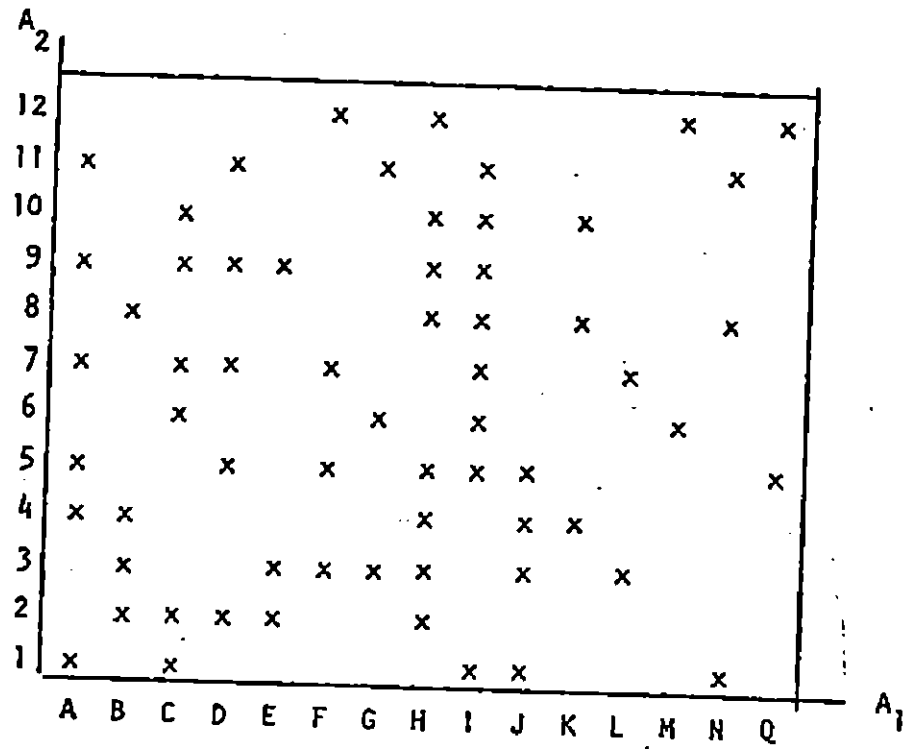


Figure 1

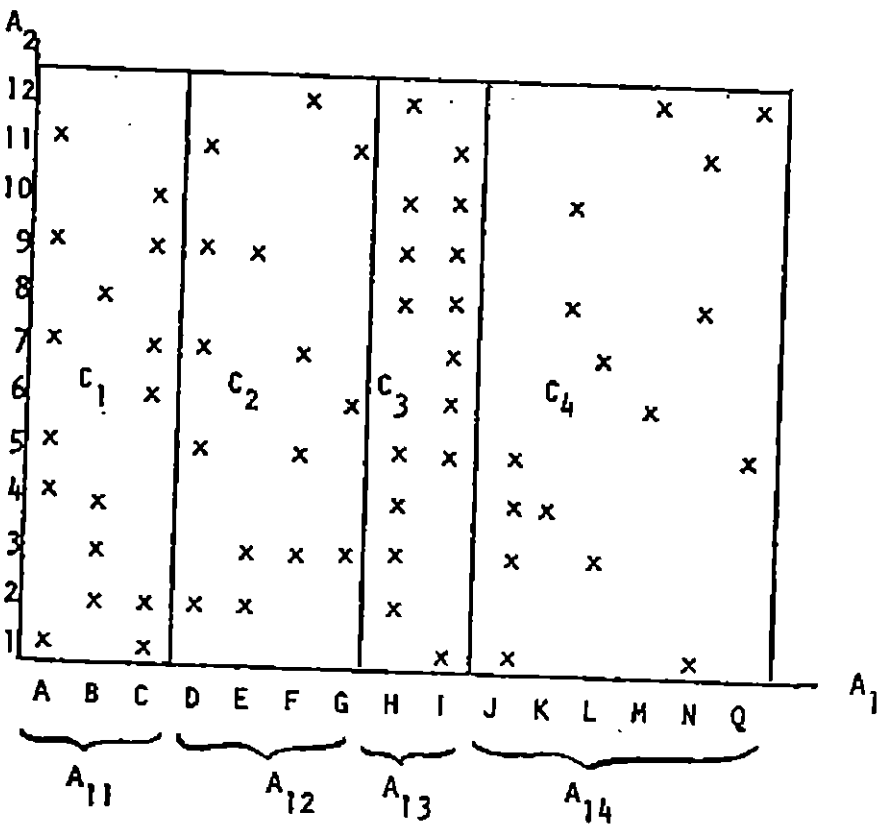


Figure 2

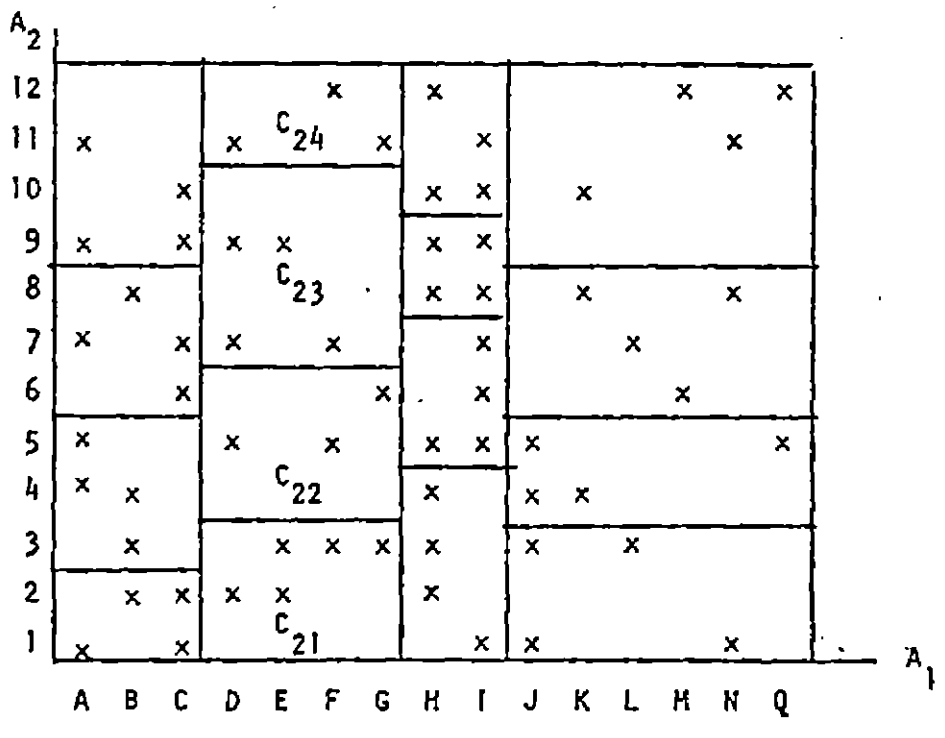


Figure 3

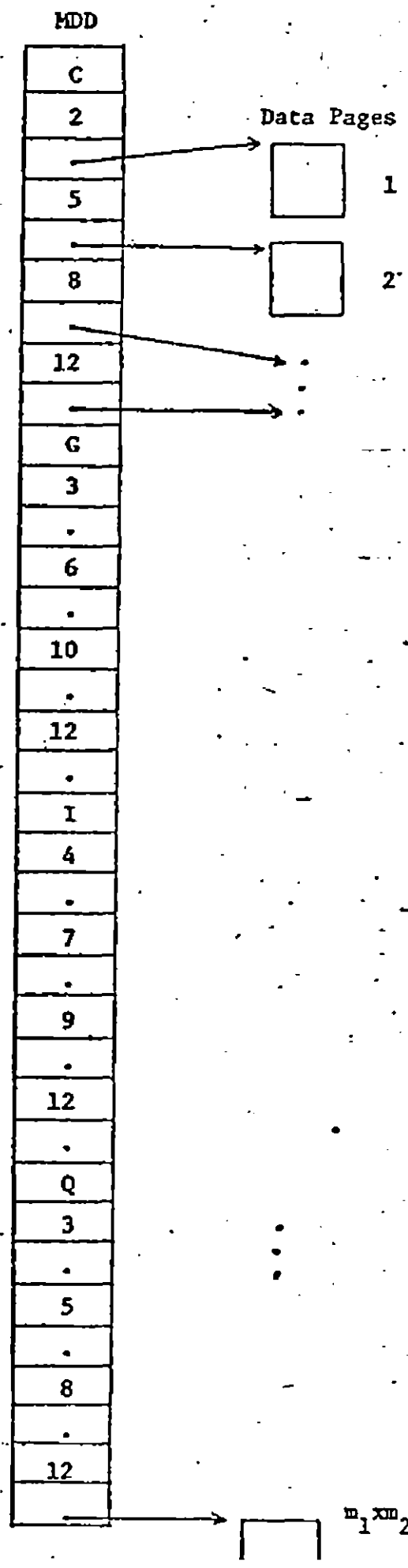


Fig. 4.

Let us discuss the three items below in more detail:

(1) storage of MDD, (2) retrieval efficiency of MDD file structure, and (3) determination of $m_1, m_2, \ldots, m_K$.

(1) Storage of MDD

Let the length of key A_i be L_i and the length of a page pointer be LP. The data structure of an MDD can be described by a PL/1 like data declaration [3]:

```
0  Z0 (m1 instances)
   1  A1 [L1]
   1  Z1 (m2 instances)
      2  A2 [L2]
      2  Z2 (m3 instances)
         :
      (k-1)  AK-1 [LK-1]
      (k-1)  ZK-1 (mK instances)
             K   AK [LK]
             K   PT[LP]
```

Z_i is the heading for a data structure and is not a data item. A_i is a data item whose length is L_i. An instance of Z_i (there are m_{i+1} such instances) is the data structure needed to store all the parameters within an (i+1)th -degree cell, i.e., the parameters for the (i+1)th degree cell, its subcells, sub-subcells, and so on.

The space S(d), required to store the parameters within a d-th-degree cell is

d	S(d)
K	$L_K + LP$
K - 1	$L_{K-1} + m_K (L_K + LP))$
K - 2	$L_{K-2} + m_{K-1} (L_{N-1} + m_K (L_K + LP))$
⋮	⋮
1	$L_1 + \sum_{j_1=1+1}^{K} L_{j_1} (\prod_{j_2=j_1}^{K} m_{j_2}) + LP \cdot \prod_{j_3=1+1}^{K} m_{j_3}$
⋮	⋮
0	$\sum_{j_1=1}^{K} L_{j_1} (\prod_{j_2=j_1}^{K} m_{j_2}) + LP \cdot \prod_{j_3=1}^{K} m_{j_3}$

S(0) is the total space the MDD requires.

(2) Retrieval efficiency of the MDD file structure

We will use the number of page accesses (npa) as a yardstick for the estimation of the performance of a file structure.

In the MDD structure the records are clustered so that all records which have the same value of A_1 are stored in the N/m_i pages. The query types and npa to data pages required are listed below:

q_α	npa(α)
$A_1 = a_1$	N/m_1
$A_2 = a_2$	N/m_2
⋮	⋮
$A_1 = a_1 \wedge A_2 = a_2$	$N/(m_1 m_2)$
⋮	⋮
$A_1 = a_1 \wedge A_2 = a_2 \ldots \wedge A_K = a_K$	1

(3) Determination of m_i

We can set the values of $m_1, m_2, \ldots, m_K$ such that the average number of page accesses per query is minimized. This is achieved by considering the occurrence frequency of each type of query.

The query type, its occurrence frequency and its npa required are listed in the following:

q_α	P_α	$npa(\alpha)$
$A_1 = a_1$	P_1	N/m_1
$A_2 = a_2$	P_2	N/m_2
$\vdots$	$\vdots$	$\vdots$
$A_1 = a_1 \wedge A_2 = a_2$	P_{12}	$N/(m_1 m_2)$
$\vdots$	$\vdots$	$\vdots$
$A_1 = a_1 \wedge A_2 = a_2 \cdots \wedge A_K = a_K$	$P_{12\cdots K}$	1

The occurrence frequency P_α can be obtained by user estimation or system monitoring.

The average npa requires is $\sum_\alpha [P_\alpha \cdot npa(\alpha)] = \sum_\alpha [P_\alpha \cdot \frac{N}{\prod_{i \in \alpha} m_i}]$.

The values of $m_1, m_2, \ldots, m_K$ which minimize $\sum_\alpha [P_\alpha \cdot \frac{N}{\prod_{i \in \alpha} m_i}]$ subject to $\prod_{i=1}^{K} m_i = N$ present analytic difficulties. A reasonable approximation is discussed next.

When P_α is large, it is desirable to reduce $npa(\alpha)$. This can be achieved by increasing m_i, $i \in \alpha$. It is therefore a sound idea to make m_i proportional to $f_i \triangleq \sum_{\alpha \text{ contains } i} P_\alpha$.

$$\left.\begin{array}{l} \frac{m_1}{f_1} = \frac{m_2}{f_2} = \ldots = \frac{m_K}{f_K} \\ \text{---------} \end{array}\right\} \Rightarrow \quad m_i = f_i \left(\frac{N}{\prod_{k} f_k}\right)^{1/K} \qquad \text{(a)}$$

Some domains of a file can be so small that they cannot be partitioned at will. For example, the domain SEX has only two values and can only be divided into at most two intervals. If $_{m}l > |A_i|$ (the number of distinct A_i values in the file) for $i = 1,2,\ldots,K_1$, we have to set $m_i \leftarrow |A_i|$ for $i = 1,2,\ldots,K_1$. The value of m_l, $l = K_1+1,\ldots,K$ also have to be modified as follows:

(i) $m_l \leftarrow m_l \left(\prod_{r=1}^{K_1} \frac{m_r}{|A_r|}\right)^{1/K-K_1}$, $l = K_1+1,\ldots,K$; where m_i and m_r on the right hand side of the arrow are those obtained from expression (a).

And then,

(ii) $m_j \leftarrow |A_j|$, $j = 1,2,\ldots,K_1$

The new values of m_i satisfy (a) $\prod_{l=1}^{K} m_i = N$, and (b) $\frac{m_i}{f_i}$ = constant for $i = K_1+1,\ldots,K$. We will see an example in section 4.

3. HYBRID DATA BASE STRUCTURE

In the last section, it was shown that a query $A_i = a_l$ requires N/m_i npa to the data pages. Let x_i be the average number of records satisfying $A_i = a_i$. (That is, $x_i = (N \cdot R)/|A_i|$.) It is quite possible that $x_i << N/m_i$. Of the N/m_i pages examined, at most x_i pages contain any target records. The false drops (access of pages which do not contain any target records) can be avoided by taking key A_1 out of the MDD and indexing it by an SKI (Single Key Index)*. Fig. 5 shows a file with three keys A_1, A_2 and A_3. A_1 is indexed by an SKI, while A_2 and A_3 are jointly indexed by the MDD.

In the next three sections, the access cost, storage space, and update cost required for a hybrid data base structure are analyzed and mathematically formulated. We can then determine which keys should be indexed by the MDD index and which keys by the SKI indices to obtain the best hybrid data base structure.

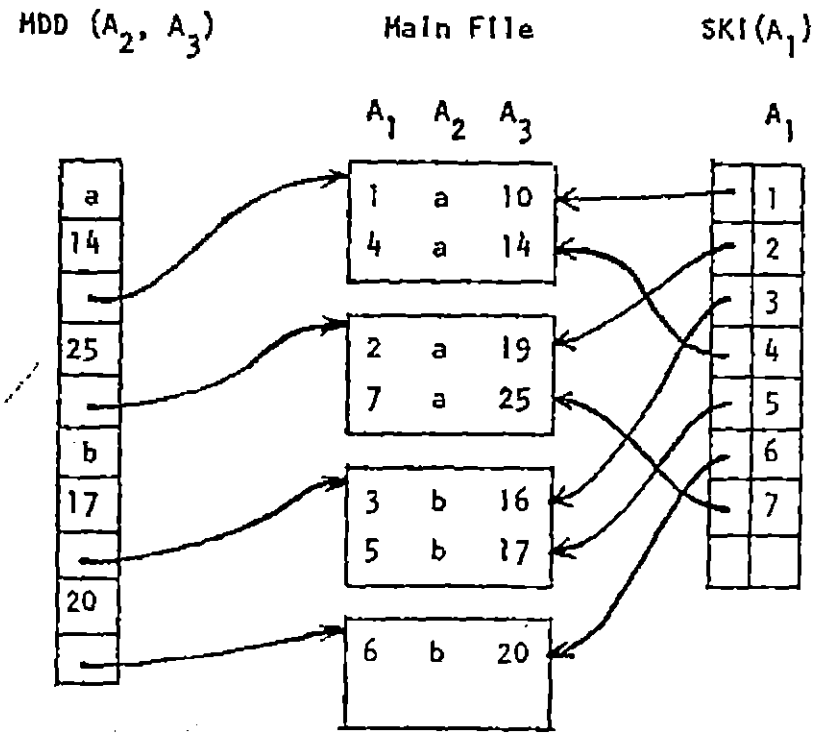
---

*'Single Key Index' instead of 'Secondary Key Index' is used because it is also used to index the primary key and because the word 'single' is to be stressed.

(a) A file sequenced on A_1

A_1	A_2	A_3
1	a	10
2	a	19
3	b	16
4	a	14
5	b	17
6	a	20
7	a	25

(b) The same file indexed by the MDD and an SKI



4. RETRIEVAL PERFORMANCE OF THE HYBRID DATA BASE STRUCTURE

Let X_1 contain the keys indexed by MDD and X_2 contain the keys indexed by SKI indices. Let $q_\alpha = q_{\alpha_1} \wedge q_{\alpha_2}$ where α_1 contains only keys in X_1 and α_2 contains only keys in X_2. To answer a query q_α, there are three alternatives: (1) The MDD is searched to find the records satisfying q_{α_1}, whose content are then checked to see if q_{α_2} is satisfied. (2) The SKI indices are used to find the records satisfying q_{α_2}, whose contents are then checked to see if q_{α_1} is satisfied. (3) Page addresses obtained from searching the MDD (using q_{α_1}) and record addresses obtained from searching the SKI indices (using q_{α_2}) are intersected; the resultant record addresses are then used to find the records satisfying q_α. For each query the alternative which requires the smaller npa is used. The average npa for an arbitrary query is

$$\sum_{\alpha} P_\alpha \min \{n_1(\alpha), n_2(\alpha), n_3(\alpha)\},$$

where $n_i(\alpha)$ is the npa required to answer query q_α when the i-th alternative is used.

$$n_1(\alpha) = s_1 + t_1$$

$$n_2(\alpha) = s_2 + t_2$$

$$n_3(\alpha) = s_1 + s_2 + t_3$$

where s_1 (s_2) is the npa for searching the MDD (SKI indices) and t_1, t_2 and t_3 are the npa for searching the data pages. Although we do not explicitly indicate it, s_i and t_i are all functions of α. When α_1 is null, (i.e. no key in α is indexed by the MDD) $n_1(\alpha) = 0 + N$. Similarly, when α_2 is null $n_2(\alpha) = 0 + N$.

In other cases, s_1, t_1, s_2, t_2 and t_3 can be evaluated as follows:

(1) s_1: npa to MDD

The space required for the MDD is S(0) bytes, or $\lceil S(0)/PAGESIZE \rceil$ pages. For the present we assume the entire MDD is searched.

$s_1 = \lceil S(0)/PAGESIZE \rceil$

(2) t_1: npa to primary pages when the MDD is used.

$$t_1 = \max \{N, \prod_{i=1}^{K} m_i\}^{*} / \prod_{j \in \alpha_1} m_j$$

*Note that it is possible that $\prod_{i=1}^{K} m_i < N$. For example if only A_j is indexed by the MDD and $|A_j| = 2$. Then $\prod_{i=1}^{K} m_i = m_j = 2$.

(3) s_2: npa to SKI indices

We follow Cardenas' model [2] for the search of SKI indices (fig. 6). The key-value pages store all (i, a_i) pairs for all A_i indexed by the SKI, where $a_i \in A_i$. The key-value pages are sorted and a master index determines which key-value page to go given $A_j = a_j$. In this key-value page the entry (j, a_j) can be found which in turn points to an accession page. This accession page contains the R#'s of all records which have $A_j = a_j$. The master index is assumed to require only one page. To find the records which satisfy q_{α_2}, i.e., $\prod_{i \in \alpha_2} (A_i = a_i)$, the master index is retrieved to the main memory once and for each key in α_2 a key-value page and an accession page are accessed. The total npa is therefore $1 + 2|\alpha_2|$

$$s_2 = 1 + 2|\alpha_2|$$

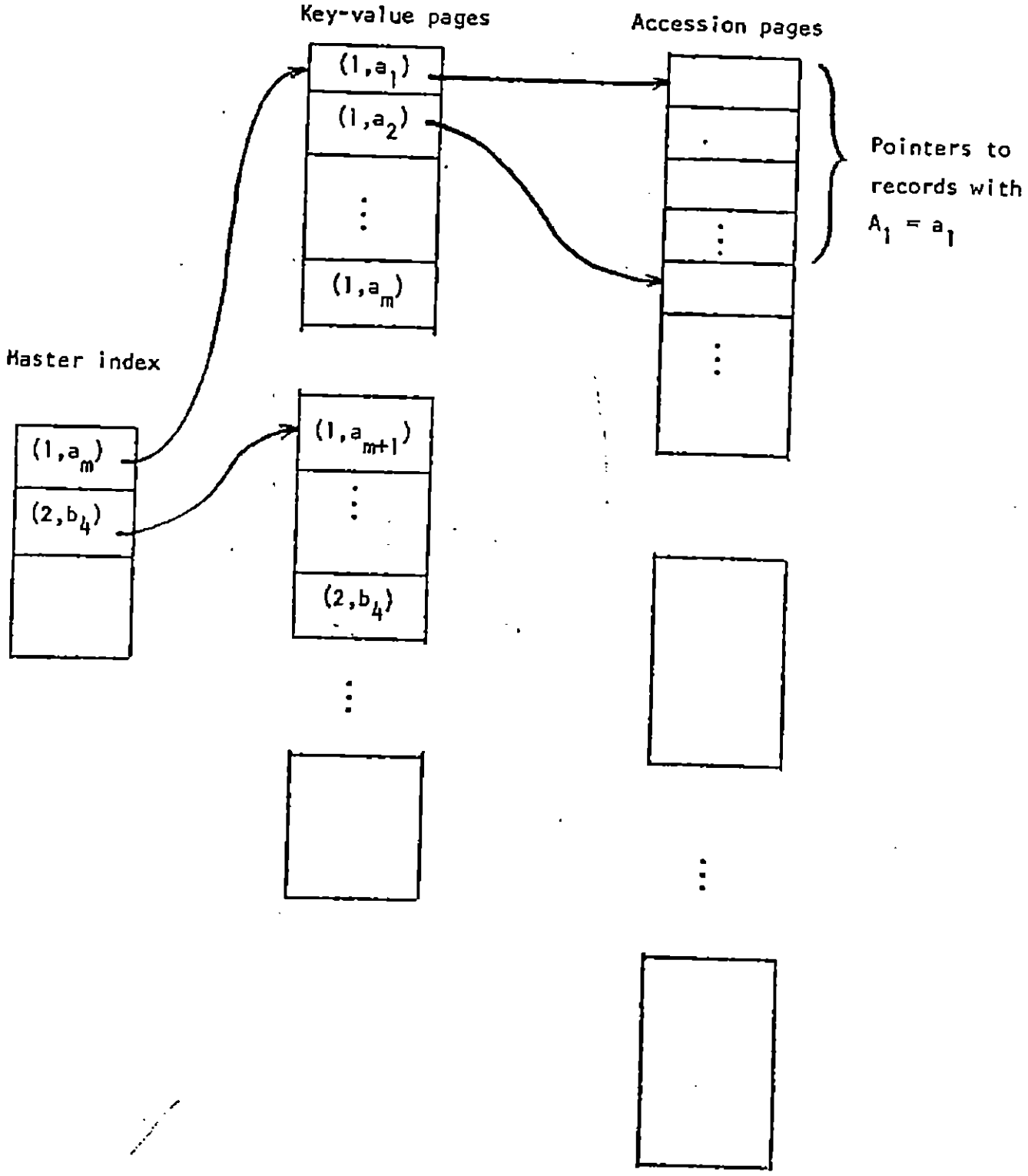

Key-value pages
Accession pages
(1,a1)
(1,a2)
(1,am)
Pointers to records with A1 = a1
Master index
(1,am)
(2,b4)
(1,am+1)
(2,b4)

Figure 6

(4) t_2: npa to primary pages when the SKI indices are used.

The average number of records satisfying $A_i = a_i$ is $\frac{NR}{|A_i|}$. (If a range query is considered, a factor greater than 1 should be multiplied to it.) The average number of records satisfying q_{α_2} is $\frac{(NR)}{\prod_{i\in\alpha_2} |A_i|}$. Therefore the intersection of pointers found in the SKI indices has the estimated size $\frac{(NR)}{\prod_{i\in\alpha_2} |A_i|}$. Because records are not clustered by the keys in α_2, the only reasonable assumption of the target record distribution is that it is random. It has been shown in [10] that when Y target records are randomly distributed over N pages the expected npa for retrieving these Y records can be approximated by $N(1- (1- \frac{1}{N})^Y)$. Therefore,

$$t_2 = N(1- (1- \frac{1}{N})^Y),$$

where

$$Y = \frac{N \cdot R}{\prod_{i\in\alpha_2} |A_i|}$$

(5) t_3: npa to primary pages when both MDD and SKI indices are used.

The t_2 pages obtained in (4), each of which contains at least one record satisfying q_{α_2}, are distributed randomly over the entire file space. The information obtained from the MDD further reduced this number by $1/\prod_{i\in\alpha_1} m_i$. Therefore,

$$t_3 = t_2/\prod_{i\in\alpha_1} m_i$$

In summary, we have:

$$n_1(\alpha) = \begin{cases} 0 + N, \text{ if } \alpha_1 \text{ is null} \\ \lceil S(0)/PAGESIZE \rceil + \max\left(N, \prod_{i=1}^{K} m_i\right) / \prod_{j \in \alpha_1} m_j, \text{ otherwise} \end{cases}$$

$$n_2(\alpha) = \begin{cases} 0 + N, \text{ if } \alpha_2 \text{ is null} \\ 1 + 2|\alpha_2| + N(1-(1-1/N)^{Y}), \text{ otherwise.} \end{cases}$$

$$n_3(\alpha) = \begin{cases} n_1(\alpha), \text{ if } \alpha_2 \text{ is null} \\ n_2(\alpha), \text{ if } \alpha_1 \text{ is null} \\ s_1 + s_2 + \dfrac{t_2}{\prod_{i \in \alpha_1} m_i}, \text{ if both } \alpha_1 \text{ and } \alpha_2 \text{ are not null.} \end{cases}$$

Example

An example is given next to illustrate the above analysis. The same example will be used in the next two sections. Assume a file has the following parameter values:

N•R = 10000 : total number of records

R = 20 : number of records that can be accomodated on a single page of secondary storage.

N = 500 : minimal number of primary pages the file requires

K = 4 : number of search keys.

$|A_1| = 10000$
$|A_2| = 2000$
$|A_3| = 100$
$|A_4| = 2$
: sizes of domains A_1, A_2, A_3, and A_4

L = L1 = L2 = L3 = L4 = 8:

Key length in bytes

Lp = 2 : page-pointer length in bytes

PAGESIZE = 2048 : page size in bytes

Also assume the probability distribution of different queries is:

α	P_α
1	.58
2	.12
3	.09
4	.03
23	.06
24	.06
34	.03
234	.03

Because key A_1 is an identifier of the records, $P_{1x} = 0$ for all non-null string x. In the following the hybrid file structure with $X_1 = \{A_1, A_3, A_4\}$ and $X_2 = \{A_2\}$ is used for illustration.

(1) determine f_i

$f_1 = 0.58$

$f_3 = 0.12 + 0.06 + 0.06 + 0.03 = 0.27$

$f_4 = 0.03 + 0.06 + 0.03 + 0.03 = 0.15$

(2) determine m_i

$$\left.\begin{array}{l} \frac{m_1}{f_1} = \frac{m_3}{f_3} = \frac{m_4}{f_4} \\ m_1 m_3 m_4 = N \end{array}\right\} \Rightarrow \left\{\begin{array}{l} m_1 = 16.1 \\ m_3 = 7.5 \\ m_4 = 4.2 \end{array}\right.$$

Because $m_4 = 4.2 > |A_4| = 2$, we have to set

$$\left.\begin{array}{l} m_1 \leftarrow m_1 \left(\frac{m_4}{|A_4|}\right)^{1/2} \\ m_3 \leftarrow m_3 \left(\frac{m_4}{|A_4|}\right)^{1/2} \\ m_4 \leftarrow |A_4| \end{array}\right\} \Rightarrow \left\{\begin{array}{l} m_1 = 26.2 \\ m_3 = 9.5 \\ m_4 = 2.0 \end{array}\right.$$

Making m_1, m_3, m_4 integral subject to $m_1 m_3 m_4 \geq 400$, it is obtained that

$$m_1 = 25,\ m_3 = 10,\ m_4 = 2$$

(3) determine s_1 (number of pages the MDD requires)

$$L \cdot (m_4 + m_4 m_3 + m_4 m_3 m_1) + Lp \cdot m_4 \cdot m_3 \cdot m_1$$

$$= 8(2 + 20 + 500) + 2 \cdot 500$$

$$= 5176$$

$$s_1 = \lceil 5176/PAGESIZE \rceil = \lceil 5176/2048 \rceil = 3$$

(4) For each α find $n_1(\alpha)$ and $n_2(\alpha)$

(i) $\alpha = 1$ ($\alpha_1 = 1$, $\alpha_2 = null$)

(a) $t_1 = N/m_1 = 20$

$n_1 = s_1 + t_1 = 3+20$

(b) $n_2 = 0+N = 500$

(c) $n_3 = n_1 = 3+20$

(ii) $\alpha = 2$ ($\alpha_1 = null$, $\alpha_2 = 2$)

(a) $n_1 = 0+N = 500$

(b) $s_2 = 1+2|\alpha_2| = 3$

$$Y = \frac{N \cdot R}{|A_2|} = \frac{10000}{2000} = 5$$

$$t_2 = N(1 - (1-1/N)^Y) \simeq 5$$

$n_2 = s_2 + t_2 = 3+5$

(c) $n_3 = n_2 = 3+5$

(III) $\alpha = 234$ $(\alpha_1 = 34, \alpha_2 = 2)$

(a) $t_1 = N/m_3m_4 = 25$

$n_1 = s_1 + t_1 = 3+25$

(b) $s_2 = 1+2|\alpha_2| = 3$

$Y = N\ R/|A_2| = 5$

$t_2 = N(1-(1-\frac{1}{N})^Y) \quad 5$

$n_2 = s_2 + t_2 = 3+5$

(c) $n_3 = s_1+s_2+t_2/\prod_{i\in\alpha_1} m_i$

$= 3+3+5/10\cdot 2$

$= 3+3+.25$

The n_1, n_2 and n_3 values of all other α can be obtained in the same way (Fig. 7).

(5) Calculate $\sum_{\alpha} P_\alpha \min(n_1(\alpha), n_2(\alpha), n_3(\alpha))$

See Fig. 7

The npa for some other hybrid data base structures is shown in fig. 8. To make it neater, for each α only the smallest of n_1, n_2 and n_3 is shown. The structure with X_1 = null is the case when the record address is not determined by the value of any key and every key is indexed by an SKI.

In the structure with $X_1 = \{A_1\}$ (and therefore $X_2 = \{A_2A_3A_4\}$) the MDD indexes only a single key, A_1. But the MDD is still different from the SKI indices of A_2, A_3 and A_4 because

(a) the file is sorted on A_1, and

(b) N values of A_1 (one for each data page) are entered in the MDD while $|A_i|$ values are entered into the index of A_i, $i = 2,3,4$.

As a matter of fact, this structure corresponds to the popular classical structure described as, "the file is indexed sequential on the primary key A_1 and inverted on the secondary keys A_2, A_3 and A_4".

$X_1 = \{A_1A_3A_4\}$

$m_1 = 25$, $m_2 =$, $m_3 = 10$, $m_4 = 2$

α	$n_1(\alpha)$	$n_2(\alpha)$	$n_3(\alpha)$	$P_\alpha \min\{n_1(\alpha), n_2(\alpha)\}$
1	3 + 200	0 + 500	3 + 20	13.34
2	0 + 500	3 + 5	3 + 5	.96
3	3 + 50	0 + 500	3 + 50	4.77
4	3 + 250	0 + 500	3 + 250	7.59
23	3 + 50	3 + 5	3 + 3 + 0.5	.39
24	3 + 250	3 + 5	3 + 3 + 2.5	.48
34	3 + 25	0 + 500	3 + 3 + 250	.87
234	3 + 25	3 + 5	3 + 3 + .25	.20

$$\sum_{\alpha} P_{\alpha} \min(n_1(\alpha), n_2(\alpha)) = 28.57$$

(1) $X_1 = \{A_1, A_2, A_3, A_4\}$

$m_1 = 11,\ m_2 = 6,\ m_3 = 4,\ m_4 = 2$

α	$n_1(\alpha)$	$n_2(\alpha)$	$n_3(\alpha)$	$P_\alpha \min\{n_1(\alpha), n_2(\alpha)\}$
1	3 + 48			29.58
2	3 + 88			10.92
3	3 + 132			12.15
4	3 + 264			8.01
23	3 + 22			1.50
24	3 + 44			2.82
34	3 + 66			2.07
234	3 + 1			.12
				67.17

(2) $X_1 = \{A_1, A_2, A_3\}$

$m_1 = 15,\ m_2 = 7,\ m_3 = 5,\ m_4 =$

α	$n_1(\alpha)$	$n_2(\alpha)$	$n_3(\alpha)$	$P_\alpha \min\{n_1(\alpha), n_2(\alpha)\}$
1	3 + 35			22.04
2	3 + 75			9.36
3	3 + 105			9.72
4		0 + 500		15.00
23	3 + 15			1.08
24	3 + 75			4.68
34	3 + 105			3.24
234	3 + 15			.54
				65.66

Figure 8

(3) $X_1 = \{A_3, A_4\}$

$m_1 =$, $m_2 =$, $m_3 = 100$, $m_4 = 2$

α	$n_1(\alpha)$	$n_2(\alpha)$	$n_3(\alpha)$	$P_\alpha \min\{n_1(\alpha), n_2(\alpha)\}$
1		3 + 1		2.32
2		3 + 5		.96
3	3 + 6			.81
4	3 + 250			7.58
23	3 + 6			.54
24		3 + 5		.48
34	3 + 3			.18
234	3 + 3			.18
				12.58

(4) $X_1 = \{A_1\}$

$m_1 = 500$, $m_2 =$, $m_3 =$, $m_4 =$

α	$n_1(\alpha)$	$n_2(\alpha)$	$n_3(\alpha)$	$P_\alpha \min\{n_1(\alpha), n_2(\alpha)\}$
1	3 + 1			2.32
2		3 + 5		.96
3		3 + 91		8.46
4		0 + 500		15.00
23		5 + 1		.36
24		5 + 2.5		.45
34		5 + 48		1.59
234		7 + 1		.24
				29.38

Figure 8 (continued)

(5) $X_1 = \{A_4\}$

$m_1 =$, $m_2 =$, $m_3 =$, $m_4 = 2.$

α	$n_1(\alpha)$	$n_2(\alpha)$	$n_3(\alpha)$	$P_\alpha \min\{n_1(\alpha), n_2(\alpha)\}$
1		3 + 1		2.32
2		3 + 5		.96
3		3 + 91		8.46
4	3 + 250			7.59
23		5 + 1		.36
24		5 + 2.5		.45
34		5 + 48		1.59
234		5 + 1		.18
				21.91

(6) $X_1 =$ null

$m_1 =$, $m_2 =$, $m_3 =$, $m_4 =$

α	$n_1(\alpha)$	$n_2(\alpha)$	$n_3(\alpha)$	$P_\alpha \min\{n_1(\alpha), n_2(\alpha)\}$
1		3 + 1		2.32
2		3 + 5		.96
3		3 + 91		8.46
4		0 + 500		15.00
23		5 + 1		.36
24		5 + 2.5		.45
34		5 + 48		1.59
234		7 + 1		.24
				29.38

Figure 8 (end)

The structure which requires minimal npa turns out to be the one with $X_1 = \{A_3, A_4\}$ (and $X_2 = \{A_1, A_2\}$).

Figure 9 shows how the npa varies with different probability distributions, where distribution P is the one used in fig. 8. The column under distribution P in fig. 9(b) is directly taken from the results of fig. 8. In fig. 9(a) from left to right the probability of $\alpha = 1$ increases while the probabilities of all other α's decrease. For distribution Q the performance of the classical structure ranks 11th among all hybrid file structures. For distributions P and R it ranks 9th and 7th respectively. Its improvement with increasing probability of query on the primary key A_1 is obvious.

The above estimation was based on the assumption that the access cost of every page is the same, which is essentially correct with the prospective electronic disks. But with the currently widely used magnetic direct-access-storage-devices (DASD), the access time to a consecutive page is much less than to a random page. Let us assume the access cost to a consecutive page is only one tenth of that to a random page. (Here the parameter "one tenth" is arbitrarily chosen. It varies significantly from one system to another). Then the access cost to n consecutive pages is

$$1 + (n-1)/10,$$

where the first term accounts for the cost of randomly accessing the first page and the second term accounts for the cost of sequentially accessing the remaining n-1 pages. The sequential access to consecutive pages occurs when contiguous data pages are searched. With this modification fig. 9 is changed into fig. 9'.

(a) Probability distributions

α	Q_α	P_α	R_α
1	.3	.58	.86
2	.2	.12	.04
3	.15	.09	.03
4	.05	.03	.01
23	.1	.06	.02
24	.1	.06	.02
34	.05	.03	.01
234	.05	.03	.01

(b) npa for different probability distributions and different hybrid file structures

file structure	average npa required		
X_1	Q	P	R
$\{A_1,A_2,A_3,A_4\}$	80.57	67.17	39.07
$\{A_1,A_2,A_3\}$	79.90	65.66	33.83
$\{A_1,A_2,A_4\}$	55.95	44.62	25.24
$\{A_1,A_3,A_4\}$	33.45	28.57	18.05
$\{A_2,A_3,A_4\}$	29.45	19.27	9.17
$\{A_1,A_2\}$	59.75	44.63	22.54
$\{A_1,A_3\}$	40.90	31.78	21.03
$\{A_1,A_4\}$	35.90	23.19	10.40
$\{A_2,A_3\}$	35.45	24.79	10.93
$\{A_2,A_4\}$	34.25	22.15	10.05
$\{A_3,A_4\}$	17.50	12.58	6.90
$\{A_1\}$	46.30	29.38	12.46
$\{A_2\}$	43.85	27.91	11.97
$\{A_3\}$	30.65	19.99	9.33
$\{A_4\}$	33.90	21.85	10.37

Figure 9

5. STORAGE SPACE FOR THE HYBRID DATA BASE STRUCTURE

(i) As it has been shown in section 3 the MDD requires s_1 pages

$$s_1 = \lceil S(0)/PAGESIZE \rceil$$

(ii) Again we follow Cardenas' scheme for the SKI indices. Let the length of a pointer be LR. Then the SKI indices require

$$\lceil (|X_2| \cdot N \cdot R \cdot LR)/PAGESIZE \rceil \quad \text{accession pages}$$

$$\lceil [\sum_{A_i \in X_2} |A_i| \ (L_i + LR)]/PAGESIZE \rceil \quad \text{key-value pages}$$

1 master index page.

(iii) The number of pages for the data file is:

$$\max \left(\prod_{A_i \in X_1} m_i, N \right)$$

Let LR = 3 bytes. The storage space required for each hybrid data base structure is shown in fig. 10. It is represented by the summation of three terms for (i), (ii), and (iii) above. The storage space depends on the particular integral values chosen for m_i and it also depends on which keys are indexed by the SKI. For $X_1 = \{A_3, A_4\}$ the indices of A_1 and A_2 requires 76 pages, while for $X_1 = \{A_1\}$ the indices of A_2, A_3 and A_4 require only 28 pages. This is because A_1 has much more distinct values than A_3 and A_4 together.

file structure	average npa required		
X_1	Q	P	R
$\{A_1,A_2,A_3,A_4\}$	71.52	66.12	34.02
$\{A_1,A_2,A_3\}$	52.35	52.21	27.55
$\{A_1,A_2,A_4\}$	44.30	37.81	21.27
$\{A_1,A_3,A_4\}$	22.03	16.24	14.17
$\{A_2,A_3,A_4\}$	18.16	12.50	6.57
$\{A_1,A_2\}$	35.67	31.16	17.29
$\{A_1,A_3\}$	17.04	25.58	15.82
$\{A_1,A_4\}$	23.61	17.07	7.46
$\{A_2,A_3\}$	13.65	11.68	6.33
$\{A_2,A_4\}$	23.15	15.42	7.72
$\{A_3,A_4\}$	6.30	6.33	6.33
$\{A_1\}$	22.41	14.99	10.91
$\{A_2\}$	22.25	14.98	7.49
$\{A_3\}$	8.90	6.94	4.86
$\{A_4\}$	22.75	15.18	8.13
Null	23.85	15.91	7.97

Figure 9'(b)

X_1	Storage Space
$\{A_1,A_2,A_3,A_4\}$	4 + 0 + 560 = 564
$\{A_1,A_2,A_3\}$	3 + 6 + 504 = 513
$\{A_1,A_2,A_4\}$	3 + 6 + 520 = 529
$\{A_1,A_3,A_4\}$	3 + 17 + 510 = 530
$\{A_2,A_3,A_4\}$	3 + 60 + 504 = 567
$\{A_1,A_2\}$	3 + 11 + 504 = 518
$\{A_1,A_3\}$	3 + 22 + 504 = 529
$\{A_1,A_4\}$	3 + 23 + 500 = 526
$\{A_2,A_3\}$	3 + 22 + 500 = 525
$\{A_2,A_4\}$	3 + 66 + 500 = 569
$\{A_3,A_4\}$	3 + 76 + 500 = 579
$\{A_1\}$	3 + 28 + 500 = 531
$\{A_2\}$	3 + 71 + 500 = 574
$\{A_3\}$	3 + 81 + 500 = 584
$\{A_4\}$	3 + 82 + 500 = 585
Null	0 + 87 + 500 = 587

Figure 10

6. UPDATE COSTS OF THE HYBRID DATA BASE STRUCTURE

In the above we have been concerned with the access efficiency to records in a data base. We now consider the case when a data base is subject to frequent updates.

In the comparison of update costs of different hybrid data base structures we make the following assumptions:

(1) In order to update the SKI indices efficiently one B-tree is used to store all the SKI indices.

(2) The MDD is static, i.e., not altered until the next file reorganization.

(3) The ISAM technique [4] is used to handle the updating of the data pages.

The ISAM technique (slightly modified here) for handling updates is: reserve some free space in each data page at the creation of a file; store extra records in its overflow page (shared by many data pages) when a data page becomes full; and reorganize the file when the overflow pages become highly utilized.

There are three types of update: (1) insert a record, (2) delete a record, and (3) modify a record. Since the retrieval cost has been accounted for in section 4, the update cost evaluated in this section is the cost after the target page has been retrieved. The target page is the data page the new records belongs to (in case of an insert) or the data page which contains the record to be deleted (in case of a delete) or modified (in case of a modify). In the sequel we analyse the cost of each update type for a general hybrid data base structure. The actual figures of the update costs for the file structures $X_1 = \{A_1\}$ and $X_1 = \{A_3A_4\}$ are calculated.

(A) Insert a record:

Figure 11 shows the procedure for inserting a record after the target page has been retrieved.

Step (1) requires 1 npa

(1)' requires 2 npa

If ε is the probability that the data page is full, then the npa for step (1) and (1)' is $1 + \varepsilon$

Step (2) is further analysed in fig. 12.

Step (a) requires 1 npa

Step (b) requires 1 npa

Step (c) and (d) require 1 and 2 npa respectively if no overflow occurs. When overflow occurs a page split is necessary. The frequency of page split in a B-tree is dependent on the page capacity γ (number of values per page). In [6] (P. 476) Knuth shows that if no insertion occurs the average time we need to split a page is less than $\frac{1}{(\gamma/2 - 1)}$ split per insertion. Each overflow requires in extra one read and one write accesses to the new page, i.e., a total of two extra pages. Also assume the probability that a_1 is a new value to be δ. Then the total npa for step (2) is

$$\underbrace{1}_{(a)} + |X_2|[\underbrace{1}_{(b)} + \underbrace{\delta \cdot (1 + \frac{2}{\gamma_1/2 - 1})}_{(c)} + \underbrace{(2 + \frac{2}{\gamma_1/2 - 1})}_{(d)}],$$

where γ_1 is the capacity of the key-value page and γ_2 is the capacity of the accession page.

The total insert cost is then:

$$1 + \varepsilon + 1 + |X_2|\ [1 + \delta(1 + \frac{2}{\gamma_1/2 - 1}) + (2 + \frac{2}{\gamma_2/2 - 1})]$$

$$\simeq 1 + \varepsilon + 1 + |X_2|\ (1 + \delta + 2)$$

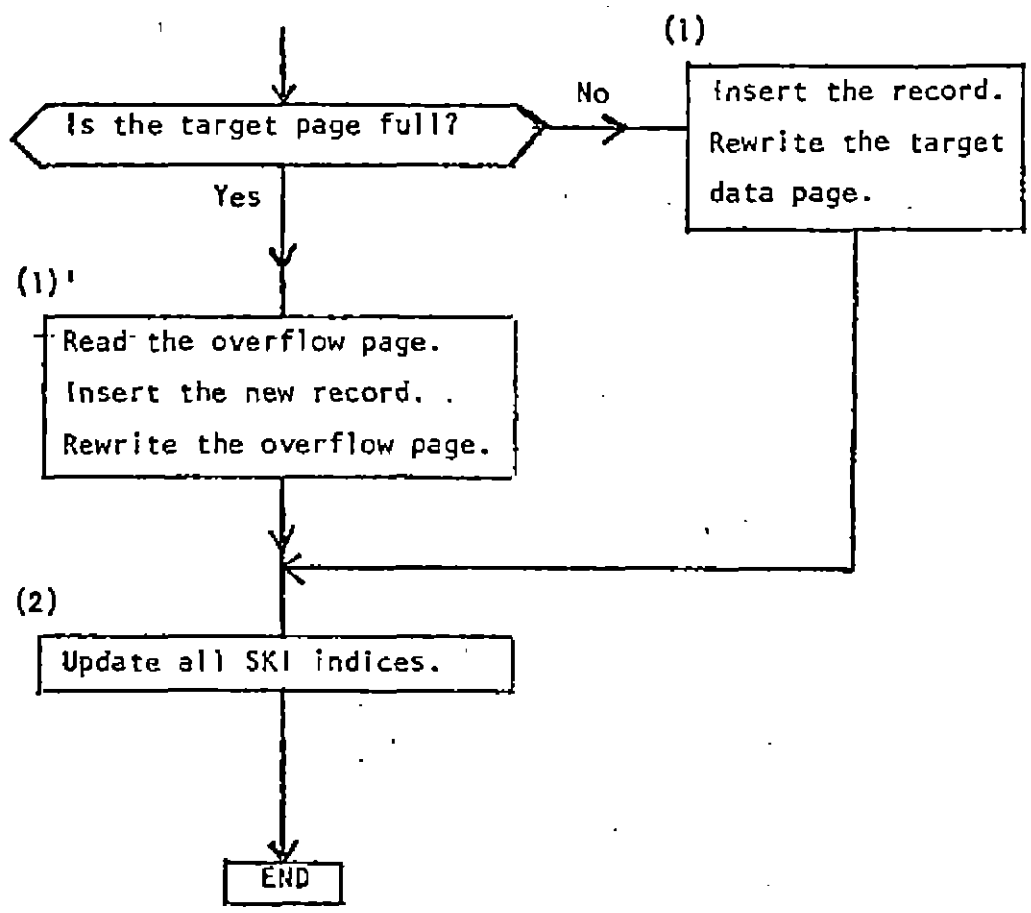

(1)
Is the target page full?
No
Insert the record.
Rewrite the target
data page.
Yes
(1)'
Read the overflow page.
Insert the new record.
Rewrite the overflow page.
(2)
Update all SKI indices.
END

Figure 11

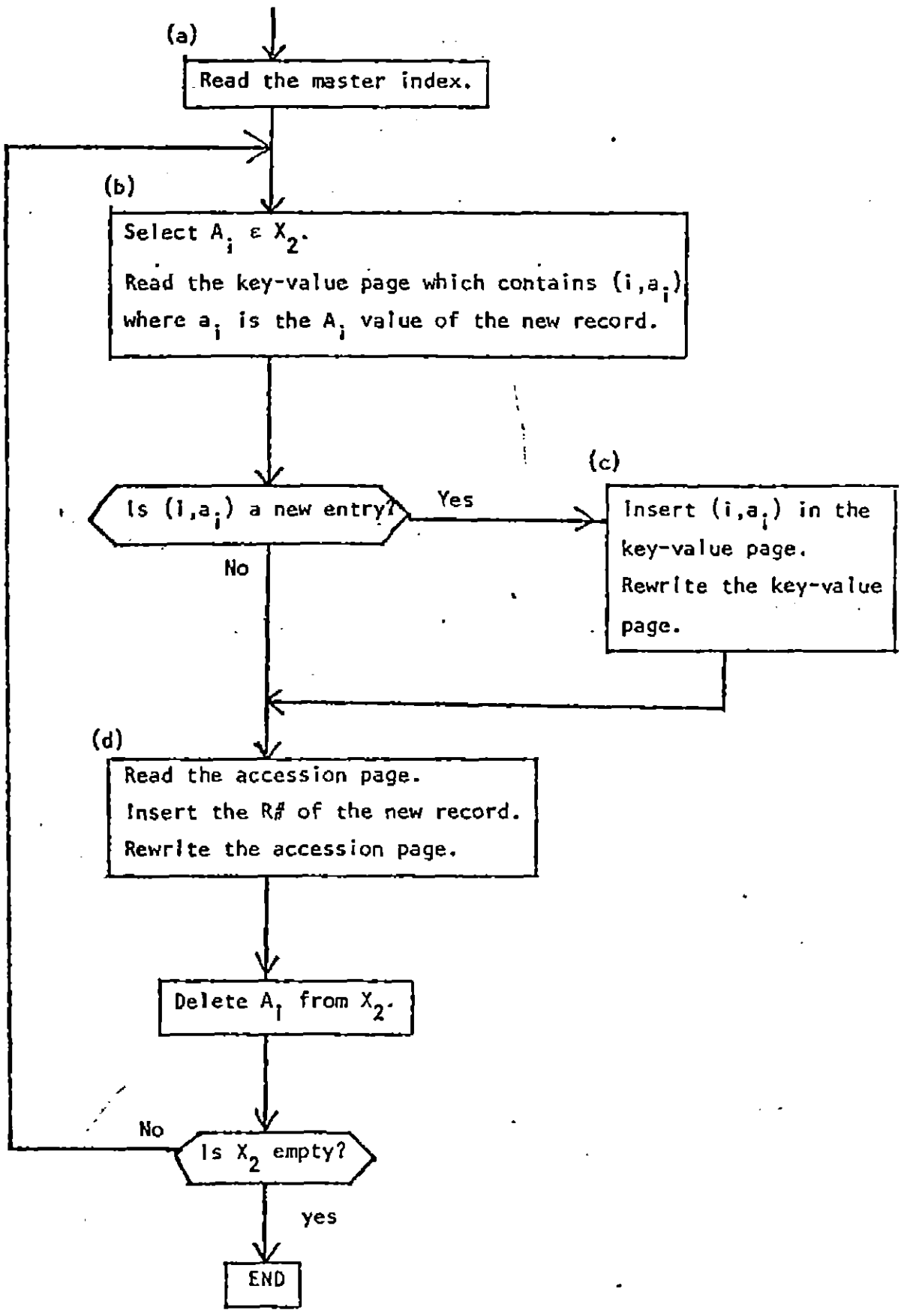

(a)
Read the master index.
(b)
Select $A_i \in X_2$.
Read the key-value page which contains (i,a_i) where a_i is the A_i value of the new record.
Is (i,a_i) a new entry?
Yes
(c)
Insert (i,a_i) in the key-value page.
Rewrite the key-value page.
No
(d)
Read the accession page.
Insert the R# of the new record.
Rewrite the accession page.
Delete A_1 from X_2.
No
Is X_2 empty?
yes
END

Figure 12

For our particular example the values of γ_1 and γ_2 are:

$$\gamma_1 = \frac{\text{PAGESIZE}}{\lfloor (L+LR) \rfloor} = \frac{2048}{\lfloor 11 \rfloor} = 186$$

$$\gamma_2 = \frac{\text{PAGESIZE}}{\lfloor LR \rfloor} = \frac{2048}{\lfloor 3 \rfloor} = 682$$

The overflow terms can be ignored. If ϵ and δ are both assumed to be 0.1, then the total npa is

$$2.1 + |X_2| \times 3.1$$

For $X_1 = \{A_1\}$ it is $2.1 + 3 \times 3.1 = 11.4$

For $X_1 = \{A_3A_4\}$ it is $2.1 + 2 \times 3.1 = 8.3$

(B) Delete a record

Figure 13 shows the procedure for deleting a record after the target page has been retrieved.

Step (1) requires 1 npa

(1)' requires 2 npa

Because only very small portion of the total records are stored in the overflow pages, the average npa for (1) and (1)' is about 1.

Step (2) is further analysed in figure 14.

We use the strategy that no key value is deleted from the key-value pages until the next file reorganization.

Step (a) requires 1 npa

Step (b) requires 2 npa

Step (c) requires 1 npa if no page merge occurs. For similar reasons as in the analysis of insertion, the extra accesses for page merge can be ignored.

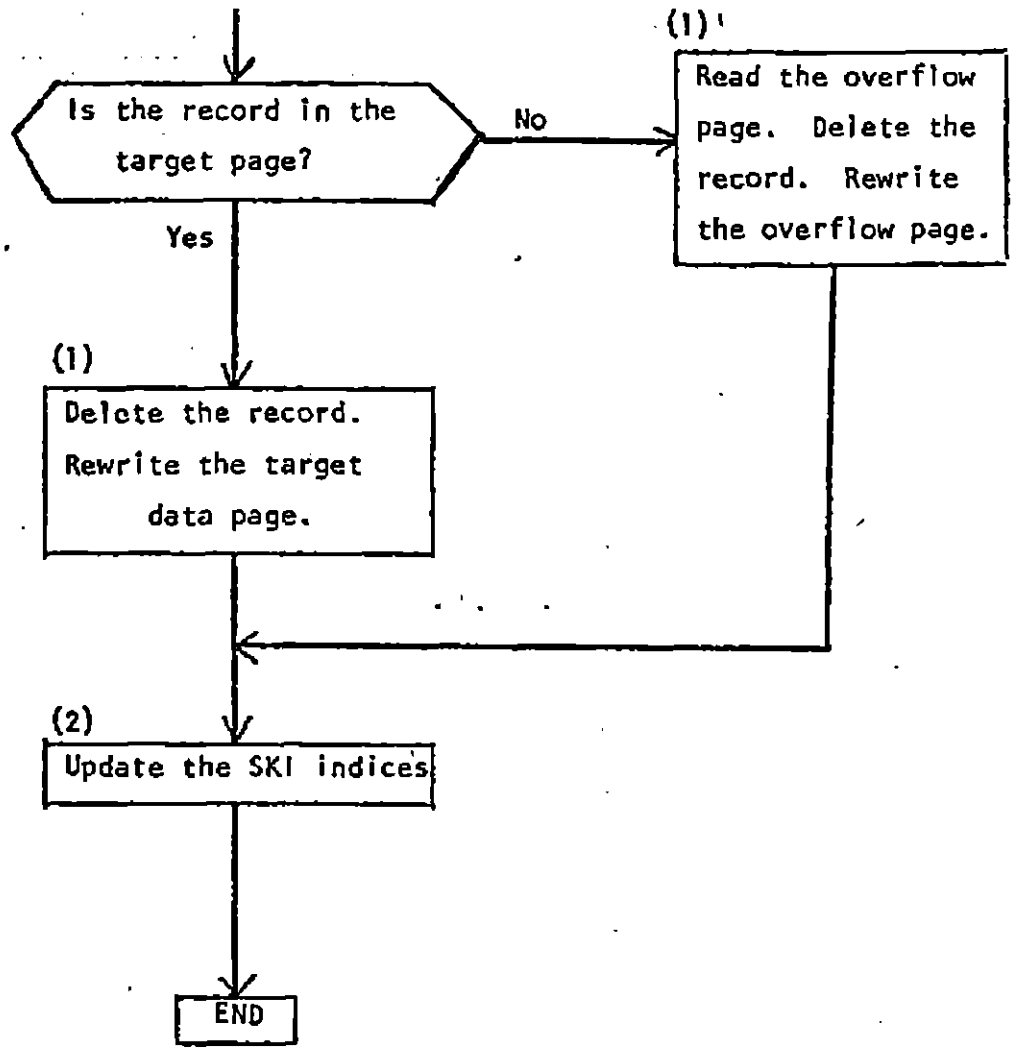

(1)'
Is the record in the target page?
No
Read the overflow page. Delete the record. Rewrite the overflow page.
Yes
(1)
Delete the record. Rewrite the target data page.
(2)
Update the SKI indices
END

Figure 13

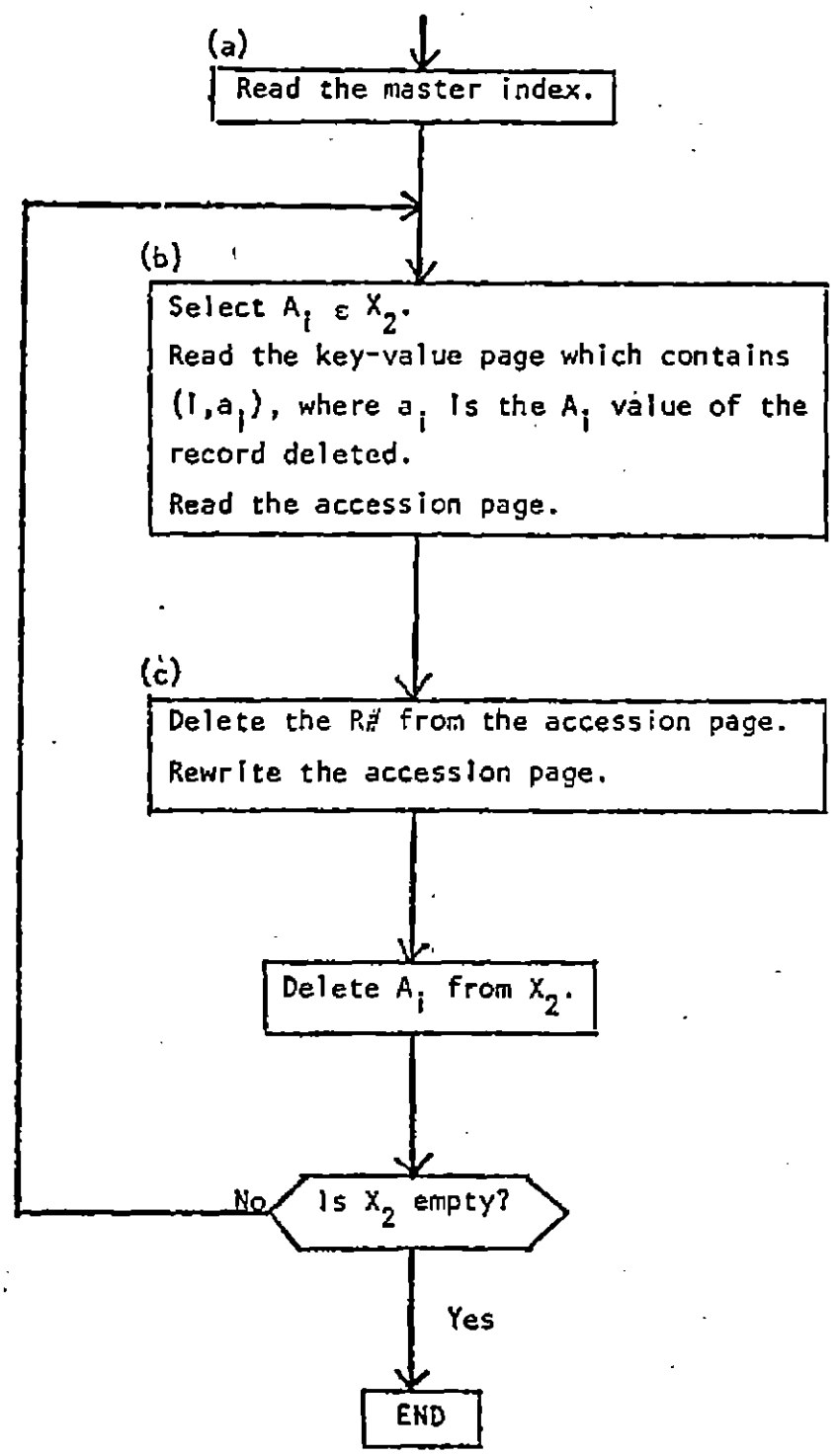

(a)
Read the master index.
(b)
Select $A_i \in X_2$.
Read the key-value page which contains (I,a_I), where a_i is the A_i value of the record deleted.
Read the accession page.
(c)
Delete the R# from the accession page.
Rewrite the accession page.
Delete A_i from X_2.
No
Is X_2 empty?
Yes
END

Figure 14

The total npa for a record deletion is:

$$\underbrace{1}_{(1)} + \underbrace{1 + |X_2|\ (2 + 1)}_{(2)}$$

For $X_1 = \{A_1\}$ it is $1 + 1 + 3 \times 3 = 11$

For $X_1 = \{A_3A_4\}$ it is $1 + 1 + 2 \times 3 = 8$

(C) Modify a record

Figure 15 shows the procedure for modifying a record after the target page has been retrieved.

Step (1) requires 1 npa

The first substep (indicated by a '*') is assumed to be counted in the retrieval cost.

Let f_i be the probability that key A_i has been modified. Assume the new record does not belong to the old page (the target page) if any of the keys indexed by the MDD has been modified. The probability that step (1) is skipped is the probability that every key in the MDD has not been modified:

$$\prod_{A_i \in X_1} (1-f_i)$$

Step (2) requires 1 npa

Step (3): Let A_i be a key indexed by an SKI. If the value of A_i has been modified, for the SKI of A_i the modification is exactly equivalent to the deletion of a record followed by the Insertion of a new record. The npa required to update the SKI of A_i is

$$\underbrace{1}_{\text{(a)}} + \underbrace{1 + 6 + 2}_{\text{(b) (c) (d)}} + \underbrace{1 + 2}_{\text{(a) (b)}} + \underbrace{1 - 1}_{\text{(c)-(a)}}$$

fig. 12 fig. 14 assume the master index is read only once

$$= 1 + (6 + 6)$$

read the master index; deletion and insertion

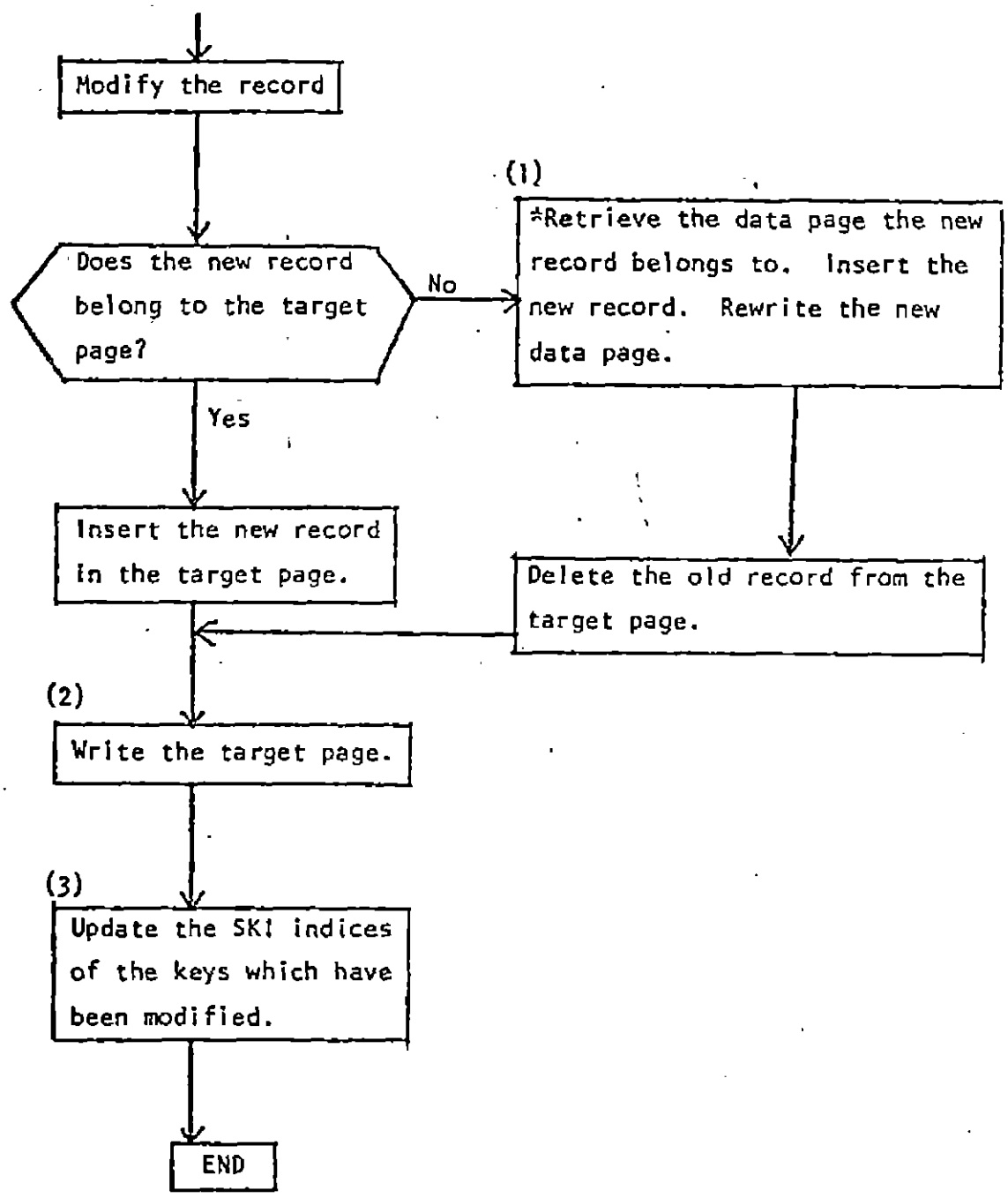

Modify the record
(1)
Does the new record belong to the target page?
No
*Retrieve the data page the new record belongs to. Insert the new record. Rewrite the new data page.
Yes
Insert the new record in the target page.
Delete the old record from the target page.
(2)
Write the target page.
(3)
Update the SKI indices of the keys which have been modified.
END

Figure 15

The total npa for step (3) is

$$1 + (6 + \delta) \sum_{A_1 \epsilon X_2} f_1$$

The total npa for modifying a record is

$$\underbrace{1 - \prod_{A_1 \epsilon X_2} (1 - f_1)}_{(1)} + \underbrace{1}_{(2)} + \underbrace{1 + (6 + \delta) \sum_{A_j \epsilon X_2} f_j}_{(3)}$$

Let $f_1 = 0.01$, $f_2 = 0.3$, $f_3 = 0.2$, $f_4 = 0.1$. (This assumes that the primary key, A_1, is seldom modified. Also, at each update the probability that none of the search keys is modified is $(1-f_1)(1-f_2)(1-f_3)(1-f_4) = 0.56$.)

For $X_1 = \{A_1\}$ the total npa is

$$1-(1-0.01) + 1 + 1 + (6 + 0.1)(0.3 + 0.2 + 0.1) = 5.67$$

For $X_1 = \{A_3A_4\}$ the total npa is

$$1-(1-0.2)(1-0.1) + 1 + 1 + (6 + 0.1)(0.01 + 0.3) = 4.17$$

Figure 16 summarizes the above analysis. It shows that the costs of insert and delete are linearly increasing with the number of keys indexed by the SKI. To see how the cost of modify varies with the increasing number of keys indexed by the SKI, let us examine the change of cost when key A_d is moved from X_1 into X_2. The npa is increased by (denoting the new X_1, X_2 by X_1', X_2')

$$[(6+\delta) \sum_{A_j \epsilon X_2'} f_j - \prod_{A_i \epsilon X_1'} (1-f_i)] - [(6+\delta) \sum_{A_j \epsilon X_2} f_j - \prod_{A_i \epsilon X_1} (1-f_i)]$$

$$= (6+\delta)f_d - [1-(1-f_d)] \prod_{A_i \epsilon X_1'} (1-f_i) = f_d(6+\delta - \prod_{A_i \epsilon X_1'} (1-f_i)),$$

A quantity between $(5+\delta)f_d$ and $(6+\delta)f_d$. This again indicates a higher cost for more keys indexed by the SKI. The data base structure with $X_1 = \{A_3A_4\}$ has been shown to have a better retrieval performance than the classical structure with $X_1 = \{A_1\}$. Because it has fewer keys indexed by the SKI, its update performance is also better as indicated in figure 16.

7. MULTIPLE-LEVEL MDD

We have assumed that when the MDD is used the whole directory is retrieved and searched. This is impractical when the MDD is large. The exhaustive search of an index can be avoided by creating an index for the index. This idea can also be applied to the MDD [7]. A two-dimensional and two-level MDD is shown in figure 17. The first level MDD requires eight pages (C_{11} to C_{33}). With the second-level MDD available, a search of the second-level MDD tells which pages of the first-level MDD should be searched. Then exhaustive search of the whole MDD is not necessary.

Also, in figure 6 we assumed the master index to need only one page. A general master index may need more than one page and possibly more than one level.

We have purposely avoided these unnecessary complications to make our analysis neater.

8. DISCUSSIONS

In examining the status of the hybrid data base structure in relation with the classical structures, it is very useful to distinguish two types of information system transactions.

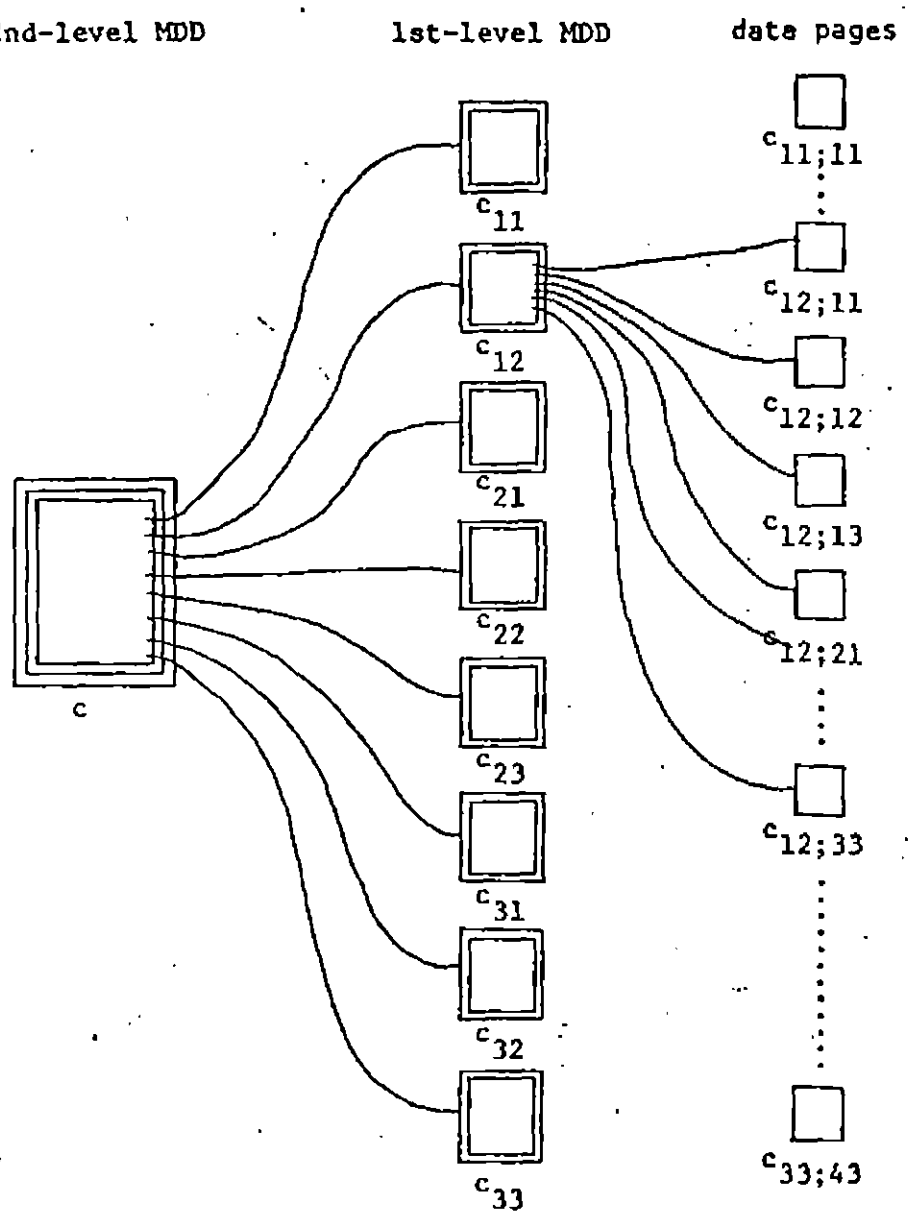
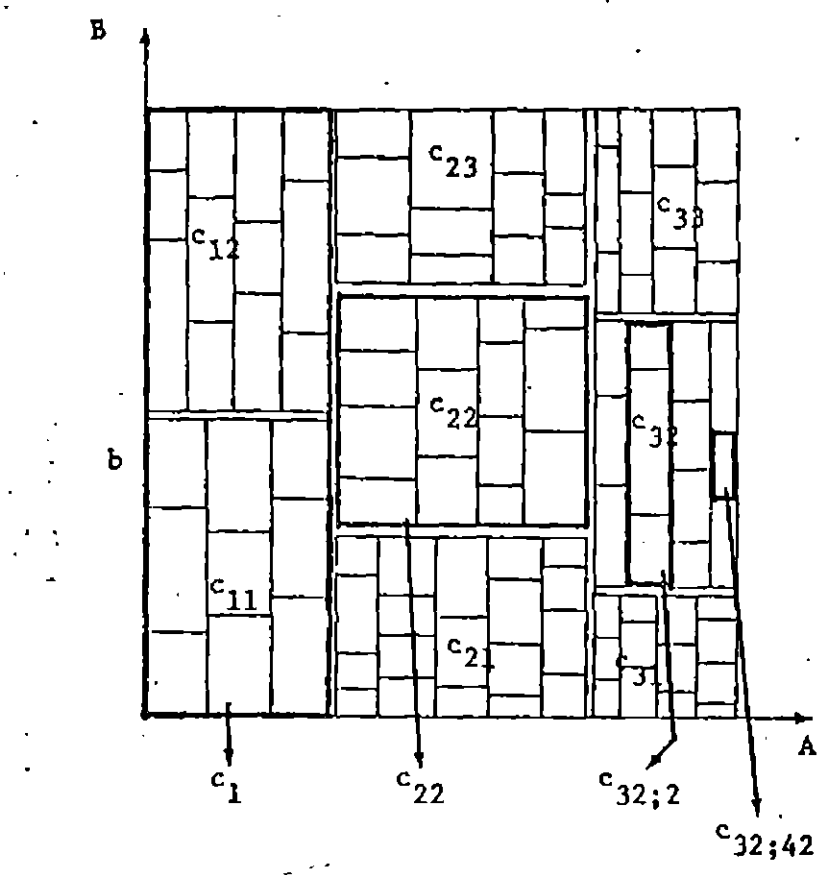

2nd-level MDD
1st-level MDD
data pages
c
c11
c12
c21
c22
c23
c31
c32
c33
c11;11
c12;11
c12;12
c12;13
c12;21
c12;33
c33;43

Figure 17


B
b
A
c_{23}
c_{33}
c_{12}
c_{22}
c_{32}
c_{11}
c_{21}
c_{31}
c_1
c_{22}
$c_{32;2}$
$c_{32;42}$

1. Operational transactions: They are basic transactions of an information system, involving mainly the maintenance of a file. Examples are retrieval, insertion, deletion and modification of records, mostly via the primary key.
2. Executive transactions: They are more complex transactions, involving the extraction of global information from a data base. An example is "Give me the number 1970 Dodge cars registered in Lafayette, Indiana."

The operational transactions can be well handled by the indexed sequential structure. However, the indexed sequential structure alone is not efficient for the executive type of transactions because for each transaction a scan of the whole data base is necessary. This problem can be alleviated by using the SKI to index the nonprimary keys. But this immediately increases the cost of an operational transaction. When a record is updated, the SKI indices must also be updated. The update of the SKI indices is very costly as we have seen in section 6.

A simple information system which handles the operational transaction may not have to handle the executive transactions. But a system built to handle the executive transactions must be able to handle the basic operational transactions. Indexing too many keys of a file may result in a high cost for operational transactions. On the other hand, indexing very few keys may result in a high cost for the executive transactions. Many efforts have been made on the models for the optimal selection of keys to the indexed (by the SKI) [5, 9]. These models are all based on the indexed sequential structure with each nonprimary keys either not indexed or indexed by an SKI index. This classical

structure has been shown to be a member of a class of hybrid structures, each of which is a different combination of the MDD index and SKI indices. Very probably there exists another member of this class which has a lower cost for executive transactions as well as a lower cost for operational transactions. In our example, the structure with each of A_1 and A_2 indexed by an SKI and A_3 and A_4 jointly indexed by the MDD is such a structure. It has a lower executive cost (section 4) because the MDD is constructed with the frequency of retrieval on nonprimary keys considered. It has a lower operational cost (section 6) because only two keys instead of three keys are indexed by the SKI.

It should be pointed out that the probability distribution among different query types must be known at creation in order to decide which keys should be indexed by the MDD. Once a key is indexed by the MDD it remains there until the next re-organization. On the other hand, if all nonprimary keys are indexed by the SKI indices, any SKI index can be dynamically created or deleted to meet the requirements of a varying environment.

REFERENCES

1. Bentley, J. L., "Multidimensional Binary Search Trees Used for Associative Searching", Comm ACM, V. 18, No. 9 (Sept. 1975), 509-516.

2. Cardenas, A. F., "Analysis and Performance of Inverted Data Base Structures," Comm ACM, V. 18, No. 5 (May 1975), 253-263.

3. Hoffman, S. A., "Data Structure that Generalizes Rectangular Arrays," AFIPS Proceedings Joint Computer Conference, 1962, 325-333.

4. IBM, "Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods," IBM Student Text, C20-1649-2.

5. King, W. F., "On the Selection of Indices for a File." IBM Research Report RJ-1341, San Jose, 1974.

6. Knuth, D., The Art of Computer Programming, Vol. 3, Section 6.5, Addison Wesley, 1973.

7. Liou, J. H., "Multi-dimensional Directory for Retrieval on Secondary Keys," ERL-M503, Department of Electrical Engineering and Computer Science, University of California, at Berkeley, February, 1975.

8. Lum, V. Y., "Multi-attribute Retrieval with Combined Indexes," Comm ACM V. 13, No. 11 (Nov. 1970), 660-665.

9. Schkolnick,M., "The Optimal Selection of Secondary Indices for Files", Information Systems, V. 1, 141-146.

10. Yao, S. B., "Approximating Block Accesses in Data Base Organizations", to appear Comm. ACM.