INCREMENTAL FILE REORGANIZATION SCHEMES

EDWARD OMIECINSKI

Computer Science Department
North Dakota State University
Fargo, North Dakota 58105 USA

ABSTRACT

For many files, reorganization is essential during their lifetime in order to maintain an adequate performance level for users. File reorganization can be defined as the process of changing the physical structure of the file. In this paper we are mainly concerned with changes in the placement of records of a file on pages in secondary storage. We model the problem of file reorganization in terms of a hypergraph and show that this problem is NP-hard. We present two heuristics which can be classified as incremental reorganization schemes. Both algorithms incorporate a heuristic for the traveling salesman problem. The objective of our approach is the minimization of the number of pages swapped in and out of the main memory buffer area during the reorganization process. Synthetic experiments have been performed to compare our heuristics with alternative strategies.

1. INTRODUCTION

We can define file reorganization as the process of changing the physical structure of the file [9,12]. Reorganization may be performed for a variety of reasons such as to improve performance (e.g. reduce retrieval time) and to enhance storage utilization (e.g. compact space). Due to changes in user access patterns and unpredictable insertions and deletions, file reorganization becomes a necessary function.

File reorganization can be classified into three basic approaches [9]: off-line reorganization, incremental reorganization and concurrent reorganization. The first method prohibits user access to the entire file during the reorganization period. The second method is an on-line strategy in which reorganization occurs incrementally. Under this strategy, the part of the file which is being reorganized is locked while user access is permitted to the remainder of the file. The third method is also an on-line approach but where reorganization is done continuously with file usage.

The traditional approach taken for file reorganization is off-line reorganization [9,13]. With this approach, the major effort addresses the problem of determining optimal reorganization points [1,12,14]. Work concerning concurrent reorganization is oriented towards performance modelling [9,10]. For incremental reorganization, the problem of determining optimal reorganization procedures has not been addressed and is the focus of this study.

In this paper, we consider file reorganization as being required due to a need to alter the placement of records of a file on pages of a secondary storage device. An example of this would be to place records which are frequently accessed together on the same page or pages in order to reduce retrieval time, e.g. record clustering [5,15]. A second example would be to move records from overflow pages to primary pages of the file, e.g. hash based files [9].

In section 2 we model the file reorganization problem in terms of a hypergraph and show that this problem is NP-hard. In section 3 we present our two heuristics for incremental file reorganization: STATIC_COST_REORGANIZATION and DYNAMIC_COST_REORGANIZATION. We illustrate one scheme with an example in section 4. In section 5, we present the results of a number of experiments used to evaluate the performance of our reorganization schemes.

2. THE REORGANIZATION MODEL

Our reorganization approach is incremental with file usage, and thus requires locking only a small part of the file, while permitting access to the remainder of the file. We assume that the dominating cost is that incurred by page accesses from secondary storage. We also assume that input and output to the secondary storage device is accomplished by using a single main memory buffer area. Thus our objective function is the minimization of the number of pages swapped in and out of the buffer during the reorganization process.

Our procedure requires two mappings as input: one PG, corresponding to the old (file) state and another NPG, corresponding to the new state. These mappings satisfy:

$$\begin{aligned} &PG: \; R \rightarrow P, \\ &NPG: \; P' \rightarrow 2^{R'} \end{aligned} \qquad (1)$$

where R = set of record indentifiers
P = set of old page numbers
P' = set of new page numbers ($P \subseteq P'$)
$2^{R'}$ = set of subsets of R of size $\leq$ pagesize

To implement the mapping PG we assume the existence of a PAGE-TABLE which associates with every record identifier the page number on which it resides. The use of the PAGE-TABLE introduces another level of indirection between any directory (index) and the data file, but has the advantage that changes to the data file do not affect the directory. Normally, in a tree structured directory [7], the pointers in the leaves represent the actual physical addresses, i.e., page numbers, of the corresponding records. Using the PAGE-TABLE concept, the directory is modified such that these pointers represent record identifiers. In addition, we shall need a LOCK table which contains a 1 bit entry for every page to indicate whether or not the page is currently being reorganized.

Given the mappings from (1) we can obtain the mapping: $\Delta: P' \rightarrow P$ which is defined as follows:

$$\Delta_{p'} = \{PG(r) \mid r \in NPG(p')\} \text{ where } p' \in P'.$$

Thus Δ gives the set of pages (actually page numbers) which have to be available in a main memory buffer in order to construct a new page. Our reorganization problem can now be modelled as a hypergraph [2], $H=(V,E)$ where the vertices correspond to the current pages, i.e., $V=P$, and the edges correspond to the new pages, i.e., $E=\{\Delta_{p'} \mid p' \in P'\}$. Figure 1a illustrates a simple hypergraph H, with four edges, corresponding to the pages in the new state: $\Delta_1 = \{1,2,3\}$, $\Delta_2 = \{1,2,4,6\}$, $\Delta_3 = \{5,6,7\}$ and $\Delta_4 = \{3,7,8\}$.

Given a hypergraph H, we can obtain its representative graph which we denote as H_R. H_R is the pair (V',E') when $V'=E$ and $E'=\{(\Delta_i,\Delta_j) \mid \Delta_i, \Delta_j \in V' \text{ and } \Delta_i \cap \Delta_j \neq \emptyset\}$. The representative graph H_R of the hypgergraph H in Fig. 1.a is pictured in Fig. 1.b.

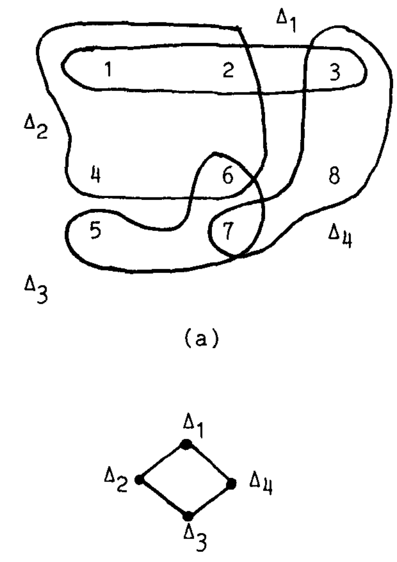


FIGURE 1 HYPERGRAPH H AND ITS REPRESENTATIVE GRAPH

Since we assume that the major costs involved in the reorganization are due to page fetches, we shall associate with every edge (Δ_i,Δ_j) in H_R a cost, $cost(\Delta_i,\Delta_j)$, which reflects the number of page faults occurred in constructing Δ_j, given that the pages relevant to Δ_i are currently in the buffer.

We can formally define the file reorganization problem as follows:

Given, a set of m pages (i.e. the vertices of $H_R: \Delta_1, \Delta_2, \ldots, \Delta_m$) to be constructed for the new file state where the number of distinct pages in the old state that need to be accessed is

$$n = \left| \bigcup_{i=1}^{m} \Delta_i \right| ; \; 1<B<n \text{ where B is the buffer}$$

capacity in pages; for all i, $|\Delta_i| \leq B$; and a cost function $COST(\Delta_i,\Delta_j)$ for each pair of new pages Δ_i, $\Delta_j \in V'$. Determine an ordering in which new pages should be constructed (i.e., a permutation $\langle \pi(1), \pi(2), \ldots, \pi(m) \rangle$) such that

$$\sum_{i=1}^{m-1} COST(\Delta_{\pi(i)}, \Delta_{\pi(i+1)}) \text{ is minimized.}$$

Theorem: The file reorganization problem is NP-hard.

Proof: If we restrict for all i, $|\Delta_i|=B$, then we have eliminated the effect of any page replacement policy. Hence, the only pages that would be contained in the buffer would be those needed by the current $\Delta_{\pi(i)}$. As such, the cost associated with an edge $(\Delta_{\pi(i)}, \Delta_{\pi(i+1)})$ is simply $|\Delta_{\pi(i+1)} - \Delta_{\pi(i)}|$. With this restriction the problem is one of finding a tour or a Hamiltonian path of least cost for H_R, also known as the open traveling salesman problem [4,6] which is a known NP-complete problem.

We should note that if the buffer capacity B is equal to n then every tour would yield a minimum cost of n since no page would be swapped from the buffer and fetched back at a later time.

At this time, we will define the cost function in more detail. Actually, we utilize two separate cost functions, each associated with a different reorganization scheme. The first cost function which is associated with the edges of H_R is naive in the sense that it only considers the pages brought into the buffer by the most recently constructed page. This cost function is defined as

$$COST(\Delta_i,\Delta_j) = |\Delta_j - \Delta_i| \;/\; |\Delta_j|.$$

The cost is expressed such as to give preference to an edge (Δ_i,Δ_j) where $|\Delta_i \cap \Delta_j| \;/\; |\Delta_j|$ is closest to 1. The second cost function is defined as follows:

$$COST(\Delta_i\Delta_j) = |\Delta_j - BUFFER| \;/\; |\Delta_j|$$

where BUFFER = set of page numbers corresponding to pages currently in the buffer.

The latter cost function takes into account the entire contents of the buffer. Thus, the cost associated with an edge (Δ_i,Δ_j) is the ratio of the number of pages needed to construct Δ_j which are not currently in the buffer to the total number of pages needed for Δ_j.

3. HEURISTIC REORGANIZATION SCHEMES

Since the file reorganization problem is NP-hard we will focus our attention on efficient heuristics. One assumption which applies to both reorganization schemes is that the number of pages needed to construct any given new page will not exceed the buffer capacity. If this assumption does not hold we will not only have to determine an order in which to construct new pages but also an order for bringing the pages contained in Δ_i into the buffer. An outline of our first reorganization algorithm is shown below.

Algorithm: STATIC_COST REORGANIZATION

Input: $H_R = (V',E)$ with $|V'|=m$ and file F
Output: A permutation of V', i.e., $\Delta_{\pi(1)}, \Delta_{\pi(2)}, \ldots$ and reorganized file F'
Step 1: Determine a tour of H_R based on the cost function
$COST(\Delta_i,\Delta_j) = |\Delta_j - \Delta_i| \;/\; |\Delta_j|$
using a greedy heuristic;
Step 2: For $\ell \leftarrow 1$ to m do
- 2.1 Determine pages to be swapped from the buffer using a K-lookahead buffer paging policy UNLOCK pages swapped out;
- 2.2 Bring in pages needed by $\Delta_{\pi(\ell)}$ not currently in buffer LOCK pages brought in;
- 2.3 Rearrange records on buffer pages until all records which make up the current page $\Delta_{\pi(\ell)}$, i.e. $NPG(\Delta_{\pi(\ell)})$ contained on a single page in the buffer; modify PAGE-TABLE;
- 2.4 Write page $\Delta_{\pi(\ell)}$ to disk;

End.

We see that the STATIC_COST_REORGANIZATION algorithm employs a K-lookahead buffer page replacement strategy. With K-lookahead buffering, we examine the next K new pages to be constructed. If pages necessary to construct the next K new pages are in the buffer then we want to retain those pages if possible by giving priority to the pages for $\Delta_{\pi(\ell+1)}, \Delta_{\pi(\ell+2)}, \ldots, \Delta_{\pi(\ell+k)}$ in this order. When K=m-1, we have the optimal strategy that selects for replacement that page which will not be referenced for the longest time in the future [8].

To determine a solution for the traveling salesman problem, the STATIC_COST_REORGANIZATION algorithm uses the Greedy heuristic [6]. Starting at a given vertex of H_R, the Greedy algorithm constructs the tour by choosing the next edge of minimum cost until all vertices are contained in the tour. Once the complete tour has been determined, the reorganization is performed by constructing one new page at a time for the given order.

Our second algorithm, DYNAMIC_COST_REORGANIZATION, alternates between finding the next edge in the tour and doing the reorganization for the vertex reached by this edge, i.e., constructing a new page. In order to determine the next edge in the tour we again employ a greedy approach which chooses that unvisited edge which has minimum cost.

Once a new vertex, say Δ_j, has been reached in the greedy phase, we have to construct this new page in a single page of the buffer, say the first one for the sake of simplicity. It could be that the first page of the buffer corresponds to physical page ℓ, and consequently Δ_j will be stored in physical location ℓ since we are using only the storage space available originally. We shall also see that records can be moved around the other pages in the buffer in order to determine which pages are to be swapped if necessary. Hence our H_R graph is a dynamic one, i.e., the numbers of the Δ_i's for the successive vertices in the tour may change.

We proceed now to give an outline of our second reorganization algorithm:

Algorithm: **DYNAMIC_COST_REORGANIZATION**
Input: $H_R = (V',E')$ with $|V'|=m$ and file F
Output: A permutation of V', i.e., $\Delta_{\pi(1)},\Delta_{\pi(2)},\ldots,\Delta_{\pi(m)}$ and reorganized file F'
Notation: BUFFER = the set of page numbers of the pages currently residing in the buffer $\overline{\text{BUFFER}}$ = the set of records contained in the pages currently residing in the buffer BUF = the buffer capacity in pages

While tour of H_R is not complete do

1.1 Determine next edge of tour based on cost function $COST(\Delta_i,\Delta_j)=|\Delta_j-\text{BUFFER}|/|\Delta_j|$ using a greedy heuristic; let $\Delta_{\pi(\ell)}$ be the new vertex reached by this edge.

1.2 Determine pages to be swapped from the buffer using fewest-records buffer paging policy;
UNLOCK pages swapped out;

1.3 Bring in pages needed by $\Delta_{\pi(\ell)}$ not currently in the buffer;
LOCK pages brought in;

1.4 Rearrange records on buffer pages until all records which make up the current page $\Delta_{\pi(\ell)}$, i.e. $NPG(\Delta_{\pi(\ell)})$, are contained on a single page in the buffer;
modify PAGE_TABLE;

1.5 Write page $\Delta_{\pi(\ell)}$ to secondary storage;
1.6 (Consolidate)
Rearrange records for pages in the buffer such that records which belong to the same new page are grouped together as follows:

a) For each $k \in P'$ such that (Δ_k not in tour yet) and

$(NPG(k) \cap \overline{\text{BUFFER}} \neq \emptyset)$ do

$S_k = \{r \mid r \in NPG(k) \text{ and } P(r) \in \text{BUFFER}\}$;

$S_0 = \overline{\text{BUFFER}} - \bigcup S_k$

b) Order above sets (excluding S_0) by non-increasing size to obtain $S_{\theta(1)},S_{\theta(2)},\ldots,S_{\theta(n)}$

c) Allocate sets in order $S_{\theta(1)},\ldots,S_{\theta(n)},S_0$ to buffer pages, $1,2,\ldots,$BUF and modify PAGE_TABLE.

The buffer paging policy employed by our reorganization procedure chooses the candidate pages for swapping to be the pages in the buffer which contain the fewest number of records needed by new pages. As a result of the consolidation procedure (Step 1.6) all that the fewest-records buffering algorithm has to do is to select the pages in reverse order of consolidation, i.e., page[BUF], page [BUF-1], ... where page[x] stands for the page in position x in the buffer.

As shown in [5], the two dominant time costs in running the STATIC_COST and DYNAMIC_COST reorganization algorithms are incurred by the greedy algorithm and the buffer paging strategy. The greedy algorithm requires time proportional to the square of the vertices visited, (i.e., the number of new pages to be constructed). The total worst case time complexity for the STATIC_COST reorganization is $\theta(m^2+m*BUF*Pagesize)$ where m = number of new pages to construct and BUF = buffer capacity in pages. The worst case time complexity for the DYNAMIC_COST reorganization is $\theta(m^2*Pagesize + m*BUF*Pagesize \log_2 BUF*Pagesize)$.

4. EXAMPLE

For the sake of brevity, we will illustrate only the DYNAMIC_COST_REORGANIZATION scheme by way of the following example. The hypergraph and associated representative graph for this example is pictured in Figure 2.

EXAMPLE: REORGANIZATION PROBLEM

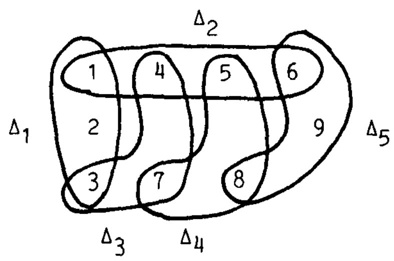
	PG		
1	1	19	5
2	1	20	5
3	1	21	6
4	1	22	6
5	2	23	6
6	2	24	6
7	2	25	7
8	2	26	7
9	3	27	7
10	3	28	7
11	3	29	8
12	3	30	8
13	4	31	8
14	4	32	8
15	4	33	9
16	4	34	9
17	5	35	9
18	5	36	9

NPG

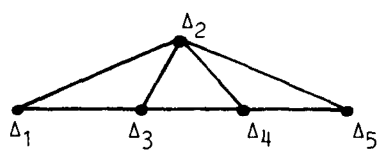
1	1	2	5	10
2	3	13	17	21
3	9	14	15	25
4	18	19	26	29
5	22	23	30	33

Buffer Size = 4 Pages

FIGURE 2: Hypgergraph and representative graph for example



Hypergraph H



Representative graph H_R

Since the buffer is initially empty, the cost of choosing any of the vertices as a starting point of the tour is the same. We choose the vertex Δ_j where $|\Delta_j|$ is the smallest as is done with the static cost reorganization scheme. In this example, where more than one vertex satisfies this criterion, we choose the one whose subscript is smallest. We choose Δ_1 to start with and bring pages 1, 2 and 3 into the buffer. The buffer contents before (on left) and after (on right) record rearranging is shown below. At this point we write page 1 back to secondary storage.

PAGE #	BUFFER (BEFORE)				BUFFER (AFTER)			
1	1	2	3	4	1	2	5	10
2	5	6	7	8	3	6	7	8
3	9	10	11	12	9	4	11	12

The number of page accesses is four (i.e. three for reading and one for writing). Next, we rearrange records on pages 2 and 3 using the consolidation procedure, Step 1.6. The result is the following:

2	3	9	7	8
3	6	4	11	12

$S_2 = \{3\}$ $S_3 = \{9\}$

$S_0 = \{7,8,6,4,11,12\}$

To determine the next edge of the tour we apply the Greedy algorithm to the cost function:

COST $(\Delta_1,\Delta_2) = 3/4$ COST $(\Delta_1,\Delta_3) = 2/3$
COST $(\Delta_1,\Delta_4) = 3/3$ COST $(\Delta_1,\Delta_5) = 3/3$

The edge of least cost is (Δ_1,Δ_3) so we bring in pages 4 and 7. The buffer contents before and after rearranging records and after consolidation appears below.

	BUFFER before rearranging				BUFFER after rearranging			
2	3	9	7	8	14	9	15	25
3	6	4	11	12	6	4	11	12
4	13	14	15	16	13	3	7	16
7	25	26	27	28	8	26	27	28

	BUFFER after consolidation			
3	13	3	26	16
4	6	4	11	12
7	8	7	27	28

This new page, Δ_3, requires three page accesses. Now we determine the next edge of the tour.

$COST(\Delta_3,\Delta_2) = 2/3$

$COST(\Delta_3,\Delta_4) = 2/3$

$COST(\Delta_3,\Delta_5) = 3/3$

We choose (Δ_3,Δ_2) for the next edge and bring pages 5 and 6 into the buffer which necessitates writing page 7 back to secondary storage. Again, we show the buffer before and after rearranging records for the new page and after consolidation.

BUFFER (before rearranging)

5	17	18	19	20
6	21	22	23	24
4	13	3	26	16
3	6	4	11	12

BUFFER (after rearranging)

5	17	21	13	3
6	18	22	23	24
4	19	20	26	16
3	6	4	11	12

BUFFER (after consolidation)

6	18	19	26	24
4	22	23	20	16
3.	6	4	11	12

Constructing Δ_2 generates four page accesses. We then determine the next edge of the tour using our cost function.

$COST(\Delta_2,\Delta_4) = 1/2$

$COST(\Delta_2,\Delta_5) = 2/3$

We construct Δ_4 next by bringing page 8 into the buffer. The buffer snapshot follows.

BUFFER (before rearranging)

8	29	30	31	32
6	18	19	26	24
4	22	23	20	16
3	6	4	11	12

BUFFER (after rearranging)

8	29	18	19	26
6	30	31	32.	24
4	22	23	20	16
3	6	4	11	12

BUFFER (after consolidation)

6	22	23	30	16
4	24	20	31	32
3	6	4	11	12

Building new page Δ_4 required 2 page accesses. The last new page to be constructed is Δ_5. This requires bringing page 9 into the buffer. The buffer snapshots appear next.

BUFFER (before rearranging)

6	22	23	30	16
4	24	20	31	32
3	6	4	11	12
9	33	34	35	36

BUFFER (after rearranging)

6	22	23	30	33
4	24	20	31	32
3	6	4	11	12
9	16	34	35	36

The last new page needed two page accesses (i.e. one for reading and one for writing) and an additional three page accesses to write the remaining pages from the buffer to secondary storage. This yields a total of 18 page accesses using the DYNAMIC_COST_REORGANIZATION algorithm.

5. EXPERIMENTAL RESULTS

This section presents the results of a number of synthetic experiments. The objective of the experiments is to compare the STATIC_COST and DYNAMIC_COST reorganization schemes with various alternative strategies. For the other strategies we will use two different ordering schemes as well as two different page replacement policies. We also modify the buffer size to see what effect the buffer capacity has on the various reorganization strategies.

For these experiments, we randomly generate records for 25 pages where the page size is 10 records. The record identifiers are in the range from 1 to 1000 and the buffer size is initially 10 pages. These 25 pages represent the new pages that must be constructed. The original state of the file (100 pages) is represented by the following formula:

set of record identifiers on page

$$p = \bigcup_{i=1}^{10} \{10 * (P-1) + i\}.$$

In the first set of experiments, shown in Table 1, we are interested in the effect the ordering scheme has on the STATIC_COST and DYNAMIC_COST reorganization schemes. We have two versions of each scheme differing only in the order in which new pages are constructed. The version denoted as LINEAR simply constructs the new pages in numerically increasing order (i.e., $\Delta_1, \Delta_2, \Delta_3, \ldots$) and the ORDERED version constructs pages in the order determined by the greedy heuristic. The two rightmost columns of the table represents reorganization strategies that use a LINEAR ordering and either an LRU (least recently used) [8] or ARBITRARY (random) page replacement policy.

Table 2 shows the average number of page accesses and the percentage of page accesses less than the LINEAR ordering scheme using the ARBITRARY page replacement policy. From Table 2, we see that the versions of the STATIC_COST and DYNAMIC_COST schemes utilizing their ordering heuristic are superior to the associated schemes using a linear ordering. However, the DYNAMIC_COST scheme with a linear ordering produces a greater savings than either STATIC_COST version. This demonstrates the importance of the buffer management scheme for the DYNAMIC_COST strategy. Another observation is that the LRU page replacement policy which is used in operating systems does not fair much better than the arbitrary page replacement policy. This has also been shown in other studies concerning database systems [3,11].

In the second set of experiments, Table 3, we want to observe, for the STATIC_COST scheme, whether a complete lookahead (i.e. 24-lookahead) yields an increased savings over the 1-lookahead. In addition, we want to see what improvement can be gained by using the greedy heuristic as in the DYNAMIC_COST algorithm to determine the ordering with the LRU and arbitrary page replacement strategies. Table 4 summarizes the results of these experiments. As we see, using the 24-lookahead gives us essentially no improvement. This is as anticipated for these experiments. Since the 10 records for a new page are randomly generated, we would expect the 10 records to reside on close to ten different pages. Hence, with the 24-lookahead scheme as well as with the 1-lookahead scheme it would be possible to keep only a few if any pages in the buffer beyond those needed for the current new page. By using the greedy algorithm for determining the order we see that the LRU and arbitrary schemes are just as good as the STATIC_COST method. Although, the DYNAMIC_COST ordered scheme is the best showing approximately a 41% savings.

Tables 5 and 6 show the results of the third set of experiments. For these experiments the buffer capacity is increased by 50%. Once again, the DYNAMIC_COST ordered scheme provides the largest savings, about 45%. However, since the buffer capacity is extended, the 24-lookahead STATIC_COST scheme now shows a marked improvement over the 1-lookahead scheme and the dynamic ordered LRU and arbitrary schemes. Again, the improvement of the 24-lookahead scheme is due to the fact that we can keep more pages in the buffer (i.e., pages which will be referenced in the future).

Tables 7 and 8 show the results of the fourth set of experiments. For these experiments the buffer capacity is increased 100% over the initial buffer size. The DYNAMIC_COST ordered scheme is still the best approach with approximately a 47% savings. Again the 24-lookahead scheme is showing an improvement due to the increased buffer size. The performance of the other schemes remains about the same.

STATIC		DYNAMIC		LINEAR	
LINEAR	ORDERED	LINEAR	ORDERED	LRU	ARB
426	364	296	226	436	438
420	376	304	270	426	424
434	372	286	260	434	434
388	352	280	230	398	398
436	360	288	242	446	448
426	374	268	242	434	432
418	366	296	240	424	424
418	368	282	242	424	424
426	382	290	270	434	434
424	358	320	238	434	436

TABLE 1. Comparison of Reorganization Schemes with Buffer Size = 10
(In Page Accesses)

	AVERAGE PAGE ACCESSES	% LESS THAN LINEAR-ARB
STATIC-LINEAR	421.6	1.77
STATIC-ORDERED	367.2	14.45
DYNAMIC-LINEAR	291.0	32.20
DYNAMIC-ORDERED	246.0	42.68
LINEAR-LRU	429.0	0.04
LINEAR-ARB	429.2	0.00

TABLE 2. SUMMARIZED DATA FROM TABLE 1

STATIC-ORDERED		DYNAMIC ORDERED	LINEAR		DYNAMIC-ORDER	
1-LOOK	24-LOOK		LRU	ARB	LRU	ARB
360	360	246	444	446	358	356
350	350	264	422	416	348	356
354	352	234	438	438	362	358
356	356	264	440	440	358	358
370	366	244	438	440	360	360
350	350	254	436	436	358	360
378	378	254	432	432	370	372
378	376	282	438	438	360	360
380	378	272	436	436	384	384
370	368	252	440	442	364	364

TABLE 3. COMPARISON OF REORGANIZATION SCHEMES WITH BUFFER SIZE = 10
(In Page Access)

	AVERAGE PAGE ACCESSES	% LESS THAN LINEAR-ARB
1 - LOOK	364.6	16.45
24 - LOOK	363.4	16.73
DYNAMIC-ORDERED	256.6	41.20
LINEAR-LRU	436.4	0.00
LINEAR-ARB	436.4	0.00
DYNAMIC-LRU	352.2	19.29
DYNAMIC-ARB	362.8	16.87

TABLE 4. SUMMARIZED DATA FROM TABLE 3

STATIC-ORDERED		DYNAMIC ORDERED	LINEAR		DYNAMIC-ORDER	
1-LOOK	24-LOOK		LRU	ARB	LRU	ARB
326	282	238	426	408	332	334
324	280	226	384	388	316	334
312	284	208	410	414	330	330
332	290	220	410	408	348	354
350	302	232	412	410	332	344
318	284	228	414	412	326	350
348	312	216	412	412	350	356
336	294	226	414	418	334	354
344	308	218	406	416	342	358
332	298	226	434	418	342	340

TABLE 5. COMPARISON OF REORGANIZATION SCHEMES WITH BUFFER SIZE = 15 (In Page Accesses)

	AVERAGE PAGE ACCESSES	% LESS THAN LINEAR-ARB
1-LOOK	332.2	19.05
24-LOOK	293.4	28.51
DYNAMIC-ORDERED	223.8	45.47
LINEAR-LRU	412.2	-0.44
LINEAR-ARB	410.4	0.00
DYNAMIC-LRU	335.2	18.32
DYNAMIC-ARB	345.4	15.84

TABLE 6. SUMMARIZED DATA FROM TABLE 5

STATIC-ORDERED		DYNAMIC ORDERED	LINEAR		DYNAMIC-ORDER	
1-LOOK	24-LOOK		LRU	ARB	LRU	ARB
304	250	208	382	370	316	320
308	244	198	356	362	292	306
298	256	200	378	390	312	318
320	262	204	380	382	306	332
332	266	202	382	394	314	324
306	252	200	392	396	306	324
336	280	198	388	380	330	332
320	264	202	382	384	304	320
336	282	202	384	382	330	342
318	262	200	412	396	298	320

STEP 7. COMPARISON OF REORGANIZATION SCHEMES WITH BUFFER SIZE = 20 (In Page Accesses)

	AVERAGE PAGE ACCESS	% LESS THAN LINEAR-ARB
1-LOOK	317.8	17.15
24-LOOK	261.8	31.75
DYNAMIC-ORDERED	201.4	47.50
LINEAR-LRU	383.7	-0.03
LINEAR-ARB	383.6	0.00
DYNAMIC-LRU	310.8	18.98
DYNAMIC-ARB	323.8	15.59

TABLE 8. SUMMARIZED DATA FROM TABLE 7

6. CONCLUSION

The file reorganization problem has been modelléd in terms of a hypergraph and was shown to be NP-hard. Two heuristics have been introduced which include specific buffer paging strategies. Experiments were performed to compare a number of alternative strategies featuring different orderings and page replacement schemes. The DYNAMIC_COST reorganization algorithm is clearly the superior approach in the experiments. The STATIC_COST algorithm with complete lookahead is good if the buffer capacity is large. The linear LRU and linear arbitrary schemes produce essentially the same results which are poor. The dynamic LRU and dynamic arbitrary schemes produce approximately the same savings and in the case where the buffer capacity is small they are as good as the STATIC_COST algorithm.

REFERENCES

[1] Batory, D.S., "Optimal File Designs and Reorganization Points", ACM Transactions on Database Systems, Vol. 7, No. 1, (1982), pp. 60-81.

[2] Berge, C., Graphs and Hypergraphs, North Holland, New York, (1973).

[3] Brice, R.S. and Sherman, S.W., "An Extension of a Database Manager In A Virtual Memory System Using Partially Locked Virtual Buffers", ACM Transactions on Database Systems, Vol. 2, No. 2, (1977), pp. 196-207.

[4] Garey, M. and Johnson, D., Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, (1979).

[5] Omiecinski, E., Algorithms for Record Clustering and File Reorganization, Ph.D. Dissertation, Northwestern University, Department of Electrical Engineering and Computer Science, (1984).

[6] Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M., "An Analysis of Several Heuristics for the Traveling Salesman Problem", SIAM Journal of Computing, Vol. 6, No. 3, (1977), pp. 563-581.

[7] Scheuermann, P. and Ouksel, M., "Multidimensional B-trees for Associative Searching in Database Systems", Information Systems, Vol. 7, No. 2, (1982), pp. 123-137.

[8] Shaw, A.C., The Logical Design of Operating Systems, Prentice-Hall, Englewood Cliffs, N.J., (1974).

[9] Sockut, G.H. and Goldberg, R.P., "Database Reorganization - Principles and Practice", ACM Computing Surveys, Vol. 11, No. 4, (1979), pp. 371-395.

[10] Soderlund, L., "Concurrent Database Reorganization - Assessment of a Powerful Technique through Modelling," Proceedings of VLDB, (1981), pp. 499-509.

[11] Stonebraker, M., "Operating System Support for Database Management," Communications of the ACM, Vol. 24, No. 7, (1981), pp. 412-418.

[12] Teory, T.J. and Fry, J.P., Design of Database Structures, Prentice-Hall, Englewood Cliffs, N.J., (1982).

[13] Wiederhold, G., Database Design, McGraw-Hill, New York, (1977).

[14] Yao, S.B., Das, K.S., and Teory, T.J., "A Dynamic Database Reorganization Algorithm," ACM Transactions on Database Systems, Vol. 1, No. 2, (1976), pp. 159-174.

[15] Yu, C.T. and Suen, C., "Adaptive Record Clustering", Technical Report, Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, (1983).