Good Worst-Case Algorithms for Inserting and Deleting Records in Dense Sequential Files

Dan E Willard [*]
SUNY Albany
and
Consultant, Bell Communications Research

Abstract

Consider a file which arranges records in sequential order, and stores them with possible empty spaces in M consecutive pages of memory We develop an insertion-deletion algorithm which runs in a worst-case time approximately proportional to $\log^2 M$ divided by the page-size when the set of manipulated records has cardinality O(M)

1. Introduction

Let KEY(R) denote the key of the record R, ADD(R) the address of the page containing this record, and S a time-varying set of records stored in M consecutive pages of auxiliary memory Given $d < D$, a **(d,D)-dense representation** of S will be defined as a file satisfying the following three conditions

i) There may be no more than N=dM records in this sequential file (the symbol N is an often-used abbreviation for the product dM in this paper)

ii) No page may contain more than D records

iii) All records in this file will be stored in ascending order, that is, they will satisfy the condition $ADD(R_1) \leq ADD(R_2)$ whenever $KEY(R_1) < KEY(R_2)$

If $d = D$, the dense file concept reduces to the classical notion of a sequential file Wiederhold [Wh77] has noted that such files are very useful when processing several records with nearby key values because most auxiliary memory architectures support the fastest access when retrieving sequences of records with nearby physical addresses For instance, applications with batch processes would benefit from sequential organization The main disadvantage of conventional sequential files is, of course, that they require complete reorganization after the insertion or deletion of a single record This difficulty can be partially alleviated by leaving empty spaces in the sequential file and by using overflow pointers, however, these techniques will not fully solve the dynamic maintenance problem for sequential files because much of the efficiency advantage of these files is lost when records with neighboring key-values are no longer stored close by Overflow mechanisms become especially unmanageable when a large surge of insertions is attempted in a relatively small portion of the sequential file, such bursts tend to overwhelm even the best heuristics because they make impossible the storage of overflow records in areas even near their originally intended locations For these reasons, Wiederhold has concluded that conventional overflow methods are unsuitable for maintaining sequential files in many dynamic environments

In this article, we study an alternative approach that shifts the records among adjacent pages rather than using overflow pointers when space is needed for inserting a record in a sequential file We show how to use this method to achieve a worst-case record insertion-deletion complexity of $O[(\log^2 M)/(D-d)]$ page-accesses in (d,D)-dense files

[*] Supported partially by NSF Grant #DCR 8412447

2. Literature Survey

Let $T_{n,N}$ denote the worst-case number of page accesses that a particular algorithm needs to perform a sequence of n insertion and deletion operations on a data structure which is **initially empty** and which never contains more than N records Then the **amortized** complexity of this algorithm will be defined to be $MAX\{T_{n,N}/n \mid n \geq 1\}$ A few articles have discussed algorithms for inserting and deleting records in data structures similar to (d,D)-dense sequential files The optimization of worst-case insertion-deletion time on dense files has not been discussed in the previous literature, which has instead focused either on expected complexity under a stationary probability distribution or on amortized complexity [Fr79, IKR80, HKW86] have investigated the expected time for updating dense sequential files under a variety of different probability models Melville and Gries [MG78, MG80], Itai, Konheim and Rodeh [IKR80] and Willard [Wi81] have independently proposed several different algorithms for controlling amortized time Our interest in the present paper is to develop an algorithm whose amortized time is the same as [IKR80, MG78, MG80, Wi81]'s implication for (d,D)-dense files, *but which also provides* $O[(\log^2 M)/(D-d)]$ *worst-case time* The present paper has been greatly influenced by technique that Lueker and Willard [WL85] applied to K-fold and augmented trees [BCW-85] has calculated the amortized complexity for data structures which permit record sizes to be variable, but which differ from the other papers on this subject by not insisting that the record addresses satisfy condition (iii) of (d,D)-density

3. Controlling Amortized Time

This section introduces some notation and briefly outlines an algorithm for inserting and deleting records in (d,D)-dense files in amortized time $(\log^2 M)/(D-d)$ Section 4 defines a stronger algorithm that also guarantees worst-case time The main purpose of the present section is to outline the motivation behind the more complex treatment appearing later

In both sections, we make the simplifying assumption that $D-d > 3\lceil \log M \rceil$ Our complexity results will also hold for all other values of D-d, since if D-d is small we can treat a sequence of several consecutive pages as one page and thereby increase the runtime coefficient by only a constant factor The latter topic appears at the end of Section 5

For convenience, this paper always assumes the page addresses in our sequential file are integers between 1 and M Also, we will maintain a special binary tree, called the **calibrator**, whose every node v is associated with two page-addresses, A_v^- and A_v^+, and which stores inside the node v a term N_v, called the **rank-counter,** indicating the number of records whose page address lies in the range $[A_v^-, A_v^+]$ The closed interval $[A_v^-, A_v^+]$ is called RANGE(v), and it is defined as follows

i) The root's range will be the entire file, that is, [1,M]

ii) The left son of an internal node v will have range $[A_v^-, \lfloor (A_v^- + A_v^+)/2 \rfloor]$, and the right son $[\lfloor (A_v^- + A_v^+)/2 \rfloor + 1, A_v^+]$

iii) Each leaf v in the calibrator will have $A_v^- = A_v^+$ (Thus, its range will contain precisely one page)

Throughout this paper, M_v denotes the number of pages in v's range, that is, $M_v = A_v^+ - A_v^- + 1$

An **important notation convention** is that 0 rather than 1 denotes the depth of a tree root (some articles follow the other notation convention) Also, let g(v,r) and p(v) denote the quantities

$$g(v,r) = d + \frac{\text{Depth}(v)+r-1}{\lceil \log M \rceil}(D-d) \tag{3 1}$$

$$p(v) = N_v / M_v \tag{3 2}$$

Define a calibrator tree to be **BALANCE(d,D)** if its every node satisfies the requirement $p(v) \leq g(v,1)$ It is easy to see that if the calibrator tree satisfies BALANCE(d,D) then the sequential file must have (d,D)-density (See Figures 1a and 1b for an example) The significance of the condition BALANCE(d,D) is that it is a useful vehicle for maintaining (d,D)-density All the algorithms discussed in this paper will rely on this technique

Our algorithm for optimizing the worst-case time of individual commands is a modified version of a somewhat simpler algorithm which optimizes only amortized complexity We therefore begin our discussion with a brief review of the latter algorithm, whose closest analog in the previous literature was proposed by Itai, Konheim, and Rodeh [IKR80] (More distantly related algorithms have appeared in [MG78, MG80]) The algorithm in this section is called CONTROL 1, and until the end of section 5 we make the simplifying assumption that $D-d > 3\lceil \log M \rceil$ CONTROL 1 consists of the following steps

A) First, use the calibrator as a binary search tree to find the page-address of the record R that is to be inserted or deleted Perform the insertion or deletion operation commanded, and increment or decrement the rank-counters N_v that should be changed after this operation

B) If step A has caused the calibrator to violate the condition BALANCE(d,D) temporarily then do the following Let v denote the highest node violating this balance condition and f_v the father of v Rearrange the records in the pages descending from f_v so that they are spread with sufficiently equal density in this range to guarantee that every node w descending from f_v satisfies $p(w) \leq p(f_v)+1$

The first step of CONTROL 1 requires CPU time O(log(M)) and typically only two or three page-accesses These costs are quite small and need concern us no further The second step of CONTROL 1, which requires $O(M_{f_v})$ page accesses, can be costly when M_{f_v} is large However, [IKR80] observes that M_{f_v} is usually a small number when CONTROL 1 invokes step B and that the amortized time of step B is $O((\log^2 M)/(D-d))$ in our cost notation

Our goal in this paper is to design a more elaborate algorithm, called CONTROL 2, which converts step B's amortized cost $O((\log^2 M)/(D-d))$ into a strict worst-case time The intuition behind this modification is quite simple Since shifting the entire set of records descending from node f is expensive when M_f is a large number, our stronger algorithm will reduce worst-case costs by employing an evolutionary process that gradually shifts records in f's range over an extended sequence of insertion-deletion commands when this type of rebalancing is necessary Such an evolution redistributes the workload it takes the small number of commands otherwise having excessive runtime and divides their workload over a long enough sequence of commands to assure no individual command violates the time-bound $O((\log^2 M)/(D-d))$ The main challenge will be to design CONTROL 2 to operate correctly when several nodes have activated concurrent evolutionary shift processes that are operating in opposite directions That is, CONTROL 2 must take several precautions to avoid various types of thrashing conditions which could otherwise arise It must all guarantee that the file **continually** satisfy the constraint BALANCE(d,D) at the end of each insertion/deletion command We designed one satisfactory algorithm for performing the densifying task as early as [Wi82], but the presentation in the present paper is much easier to understand

4. The Algorithm CONTROL.2

This chapter defines the new algorithm CONTROL 2, and the next chapter provides an example and an intuitive proof sketch Throughout our discussion, A_v^-, A_v^+, g(v), N_v, M_v, p(v) and RANGE(v) have the same definitions as in section 3, and f_v again denotes v's father Four new definitions used in this section are listed below

i) WARNING(v) This is a flag that normally equals 1 when $p(v) \geq g(v,2/3)$, it equals 0 when $p(v) \leq g(v,1/3)$, and it is allowed to contain either value when $g(v,1/3) < p(v) < g(v,2/3)$ The purpose of WARNING(v) is to signal when a node v's density comes close to violating the upper limit g(v,1) *We shall say that a node v is in a* **warning** state when WARNING(v)=1, and it is in a **non-warning** state when WARNING(v)=0

ii) DIR(v) is a constant that equals 1 when v is the right son of its father, and it equals 0 when it is a left son

iii) DEST(v) and SOURCE(v) are two pointers, called the "destination" and "source" pointers, that lie in the range of v's father, (henceforth denoted as RANGE(f_v)) CONTROL 2 uses these pointers to move records from the page SOURCE(v) to DEST(v) when v is in a warning state (i e WARNING(v)=1) The algorithm guarantees that no records shall ever be stored between these two pages at the time of this source-to-destination record movement (CONTROL 2 is obligated to guarantee this condition because its record movement would otherwise violate part (iii) of Chapter 1's definition of a sequentially ordered density) This record movement is to the left when DIR(v)=1, and it is to the right when DIR(v)=0 That is, DEST(v) < SOURCE(v) in the first case and DEST(v) > SOURCE(v) in the second (CONTROL 2 only employs the pointers DEST(v) and SOURCE(v) when v is in a warning state, and the two pointers are otherwise *undefined*)

The three subroutines employed by the algorithm CONTROL 2 are called SHIFT(v), SELECT(L) and

ACTIVATE(v) These three subroutines are crucial for understanding the mainline procedure of CONTROL 2, and we will therefore discuss them first

CONTROL 2 calls SHIFT(v) only when v is in a warning state, indicating some action must be taken to prevent p(v) from eventually possibly exceeding g(v,1) DEST(v)'s initial value is determined before CONTROL 2 makes this subroutine call, the three steps of this subroutine assign SOURCE(v) a new value, move records from the page SOURCE(v) to DEST(v), and then modify the value of DEST(v) The purpose of the "source-to-destination" record movement is to perform an operation that will *eventually* lower p(v)'s value The formal algorithmic definition of SHIFT(v) appears below, and an example illustrating how CONTROL 2 employs this subroutine appears in Section 5

1) In the respective cases where DIR(v)=1 and 0, define SOURCE(v) to be the least (respectively greatest) address to the right (left) of DEST(v) that contains one or more records

2) Define UP(v) to be the set of nodes x where $DEST(v) \in RANGE(x)$ but $SOURCE(v) \notin RANGE(x)$ Move as many records from the location SOURCE(v) to DEST(v) as is possible until either SOURCE(v) is vacated or some $x \in UP(v)$ has $p(x) \geq g(x,0)$ (Naturally, this record movement should be performed in a manner consistent with the sequential storage order in the (d,D)-dense file Thus, if there is insufficient space in DEST(v) to store all the records from the location SOURCE(v), then priority should be given to moving records from SOURCE(v) to DEST(v) with lower key values when DIR(v)=1, and to moving records with higher key values when DIR(v)=0)

3) Let x^* denote the node of least depth in UP(v) satisfying $p(x^*) \geq g(x^*,0)$ at the end of step 2 If such a node exists then set DEST(v)=

   a) $A_{x^*}^{+} + 1$ (when DIR(v)=1)

   b) $A_{x^*}^{-} - 1$ (when DIR(v)=0)

The mainline of CONTROL 2 will make a subroutine call to SHIFT(v) only when v is in a warning state, i e the flag WARNING(v)=1 Often there shall be several different nodes v that have warning flags raised over them, and the mainline of CONTROL 2 will have to decide which of these several eligible nodes should next be the object of the shift operation described in the previous paragraph Such decisions are made by a subroutine, called SELECT(L), whose argument L is that leaf which was the most recent recipient of the user's record insertion or deletion command SELECT(L) uses the procedure defined below to decide which node v should be the next recipient of a shift operation The example in Chapter 5 will explain how CONTROL 2 uses this subroutine

1) Find the lowest ancestor α of the leaf L such that some proper descendant β of α is in a warning state, i e WARNING(β)=1

2) Let v denote one of the nodes of greatest depth among the descendants of α that are in a warning state SELECT(L) will return this node name when CONTROL 2 calls it

The last subroutine employed by CONTROL 2 is ACTIVATE(w) CONTROL 2 calls ACTIVATE(w) when a non-warning state node w satisfies $p(w) \geq g(w,2/3)$ This subroutine, accordingly, raises w into a warning state, and it performs the corresponding initialization tasks of assigning DEST(w) its starting value and making a "roll-back" change on the DEST(y) pointer of any warning state node y satisfying the double relation $DEST(y) \in RANGE(f_w) \subset RANGE(f_y)$ (This "roll-back" intuitively represents our algorithm's method for preventing fatal thrashes between two warning state nodes whose destination pointers are traversing overlapping ranges) The formal definition of ACTIVATE(w)'s procedure appears below, and the example in Chapter 5 explains how CONTROL 2 uses this subroutine

1) Raise w into a warning state, i e set WARNING(w)=1

2) Set DEST(w) =

   a) $A_{f_w}^{-}$ if DIR(w)=1,

   b) $A_{f_w}^{+}$ if DIR(w)=0

3) Apply the rollback rule 0 below to change DEST(y)'s value when DIR(y)=0, $RANGE(f_y) \supset RANGE(f_w)$ and $A_{f_w}^{-} \leq DEST(y) \leq A_{f_w}^{+} - 1$, and use the rollback rule 1 for the mirror image of this case where DIR(y)=1, $RANGE(f_y) \supset RANGE(f_w)$ and $A_{f_w}^{-} + 1 \leq DEST(y) \leq A_{f_w}^{+}$

   a) **Roll-back Rule 0:** Set DEST(y) = $A_{f_w}^{+}$

   b) **Roll-back Rule 1:** Set DEST(y) = $A_{f_w}^{-}$

The intuition behind the roll-back rules is that some possible future invocations of the subroutine SHIFT(w) may undo the previous record movements of the subroutine

SHIFT(y), and step 3 of ACTIVATE(w) corrects for this anticipated problem by rolling back DEST(y) into the furthermost position that could have been affected by this conflict (This puts DEST(y) in a position to correct any damage done in the future by SHIFT(w)) These points will become clearer as we describe the mainline of CONTROL 2 in the next several paragraphs

Figure 2 illustrates the procedure employed by the mainline of CONTROL 2 The argument Z of this subroutine consists of an insertion or deletion command, and J represents an integer that should be assigned a value greater than $\Omega((\log^2 M)/(D-d))$, for CONTROL 2 to properly manipulate a BALANCE(d,D) file occupying M pages Until the end of section 5, we also require $D-d > 3 \lceil \log M \rceil$ The significance of these two lower bounds will be explained later

Figure 2 formally defines the four steps of the algorithm CONTROL 2 Its first step is similar to the analog for CONTROL 1 It simply inserts or deletes the record indicated by the user's command Z and then accordingly changes the N_w counters in the calibration tree The second and third steps of CONTROL 2 check to see whether any of the changes in N_w have caused p(w) either to fall beneath g(w,1/3) or rise above g(w,2/3), CONTROL 2's response to the first change is to lower w into a non-warning state if it was not previously there, similarly if w was previously in a non-warning state, CONTROL 2's response to the second change consists of calling ACTIVATE(w) to raise w into a warning state The fourth step of CONTROL 2 is the aspect of this procedure which guarantees the BALANCE(d,D) condition (that is, the requirement that all nodes v satisfy $p(v) \leq g(v,1)$) This step consists of J repetitions of a cycle that first calls SELECT(L) to choose a node v that should have its density decreased, then calls SHIFT(v) to perform an operation whose repetition will *ultimately* cause a decrease in p(v)'s value, and finally checks to see whether any node should have its warning flag lowered because the previous step decreased its density sufficiently (See Figure 2 for more details)

It is easy to verify that all records in the time varying set S are stored in sequential order under the algorithm CONTROL 2 The non-trivial aspect is to show that if $D-d > 3 \lceil \log M \rceil$ and if $J > \Omega\{ \log^2 M/(D-d)\}$ then CONTROL 2 will also guarantee that every calibration tree node v will satisfy the condition BALANCE(d,D) Our interest in this theorem arises for two reasons The first is that the time cost of the algorithm CONTROL 2(Z,J) can be approximated as being proportional to J in a quite realistic cost model that counts only auxiliary page accesses Our theorem shall thus imply that worst-case time $O((\log^2 M)/(D-d))$ is sufficient for CONTROL 2 to guarantee the conditions BALANCE(d,D) and (d,D)-density when $(D-d) > 3 \lceil \log M \rceil$ The second interesting point is that our algorithm and complexity model generalize to files not necessarily satisfying the constraint $(D-d) > 3 \lceil \log M \rceil$, using one further idea outlined at the end of section 5

The intuition behind CONTROL 2's good performance is that if $J > \Omega\{(\log^2 M)/(D-d)\}$ then the repeated applications of J SHIFT operations in step 4 prevents p(v) from ever exceeding g(v,1) because the aggregate effect of several shifts lowers p(v) to a safe value satisfying $p(v) < g(v,1/3)$ before such a violation can occur In order to appreciate the significance of CONTROL 2, it must be remembered that the retrieval of a "stream" of records with **consecutive** key values will be faster in a sequential file than in a B-tree (because the latter entails much disk arm movement when consecutive records are not stored in adjacent locations) Update costs are probably somewhat higher under CONTROL 2 than under B-tree algorithms, but the advantage of storing records in sequential order will make CONTROL 2 desirable in those applications where frequent stream retrieval requests make the reduced disk-arm movement a significant savings Note that CONTROL 2, unlike B-trees, is programmed to access consecutive pages in one fell swoop during update operations Typically J should $\cong$ 18

Some readers may wonder how a procedure as complicated as CONTROL 2 was conceived The answer is that CONTROL 2 is a more elaborate version of the simpler algorithm CONTROL 1, which attained $O((\log^2 M)/(D-d))$ **amortized** time A general rule of thumb is that many amortized time controlling procedures can be transformed into worst-case controlling procedures with the same complexity, (for instance, see [WL85]) The results outlined in this paper should interest the data base designer both because of their potential practical applications and because the general techniques may be relevant to other types of problems

5. Main Analysis And An Example

This chapter provides an example and some lemmas which explain the intuition behind the procedure CONTROL 2

Henceforth, the term **measurable time instance** refers to a moment when CONTROL 2 has just completed executing one of the steps of 1,2,3,4a,4b, or 4c and it is about to commence executing the next of these six steps We will call a measurable moment **type-i** iff it immediately follows the execution of step i The terms **flag-stable** refers to a measurable moment of type 3,4a or 4c, and the term **flag-unstable** refers to the remaining measurable moments of types 1,2 or 4b The term p(x,t) refers to the value of p(x) at the time t Flag-stable measurable moments are so named because they satisfy the following proposition

Fact 5.1. If t is a flag-stable moment and x is a calibration tree-node then

a) $p(x,t) \leq g(x,1/3)$ implies that WARNING(x)=0 at the time t, i e that x is in a non-warning state, and

b) if $p(x,t) \geq g(x,2/3)$ for a non-root node x then WARNING(x)=1 at this time, i e that x is in a warning state

The proof of Fact 5 1 is an immediate consequence of the algorithmic definition of CONTROL 2, and it is omitted It should be emphasized that Facts 5 1A and 5 1B do not hold for **flag-unstable** moments For instance, suppose an insertion in step 1 causes p(x) to increase from an initial value less than g(x,1/3) to a final value greater than g(x,2/3) Then WARNING(x) will not be set equal to 1 until the CONTROL 2's step 3, implying that the type-1 and type-2 moments following this event will violate the condition (b) by having $p(x) > g(x,2/3)$ and WARNING(x)=0

Throughout this paper, the symbols DEST(v,t) and SOURCE(v,t) denote the positions of v's destination and source pointers at the time t Also, $N_v(t)$ denotes the value of N_v at this time, and $SET(A^-,A^+,t)$ denotes the set of records whose address lies in the closed interval $[A^-,A^+]$ at the time t The symbol SET(v,t) is an abbreviation for $SET(A_v^-,A_v^+,t)$ The last three definitions imply that the cardinality of SET(v,t) equals $N_v(t)$ Sets of the form $SET(A^-,A^+,t)$ are called **timesets.**

Example 5 2. We will now illustrate an example where CONTROL 2 inserts and deletes records in a sequential file consisting of 8 pages whose density parameters are D=18 and d=9 The calibration tree for this file appears in Figure 3 The symbols L_1,L_2 L_8 denotes its leaves, and v_1,v_2 v_7 denotes its internal nodes Throughout our example, it is assumed that j is the page in the sequential file that corresponds to the leaf L_j, that is the leaf L_j satisfies the equality $A_{L_j}^- = A_{L_j}^+ = j$

In our example, t_i denotes a flag-stable measurable moment, and $N_{L_j}(t_i)$ indicates the number of records that CONTROL 2 stores in the leaf-page L_j at the time t_i We will make frequent references to the table in Figure 4, whose (i,j)-th entry indicates the value of $N_{L_j}(t_i)$ Our example assumes that CONTROL 2's parameter J=3 and that Z_1 and Z_2 are two insertion commands given to this algorithm

Let t_0 denote the measurable moment just before the command Z_1 is given The first row in Figure 4 indicates the distribution of records at this time The row indicates that all calibration tree nodes satisfy $p(x,t_0) < g(x,2/3)$, and it is therefore legitimate (i e consistent with Fact 5 1) to assume that all calibration tree nodes are in a non-warning state at the time t_0 when our example begins

Suppose Z_1 is a command to insert a record into the page 8 Then step 1 of CONTROL 2 will increment the values of each of N_{L_8} N_{v_7}, N_{v_3} and N_{v_1} As this change causes $p(L_8) \geq g(L_8,2/3)$ and $p(v_3) \geq g(v,2/3)$, step 3 of CONTROL 2 will raise L_8 and v_3 into *warning* states and assign $DEST(L_8)$ and $DEST(v_3)$ the initial values of 7 and 1, respectively Our example has t_1 denote the flag-stable moment at the end of step 3 when these actions are completed

Since J=3 in our example, CONTROL 2 next executes three iterations of step 4 The first execution of step 4a notices that L_8 has depth greater than v_3, and therefore SELECT returns the vertex L_8 The procedure $SHIFT(L_8)$ in step 4b will then

1) set $SOURCE(L_8) = 8$

2) move precisely 6 records from page 8 to 7,

Since the second action lowers $p(L_8)$ to a value under $g(L_8,1/3)$, step 4c of the procedure CONTROL 2 will change L_8 into a non-warning state In our example, t_2 denotes the moment after these actions are completed

The second execution of step 4 will occur between the times t_2 and t_3 At the time t_2, only v_3 is in a warning state Therefore SELECT will return v_3, and step 4b will consequently execute $SHIFT(v_3)$ The first part of this procedure sets $SOURCE(v_3) = 2$, its second part actually performs no record movements (because $p(L_1)$ already exceeded $g(L_1,0)$ at the time when SHIFT was called), the third part of SHIFT(v_3) then sets DEST(v_3) = 2 CONTROL 2 therefore performs no record movements between the times t_2 and t_3, but it assigns DEST(v) a new value during this period

The third execution of CONTROL 2 is the same as the second, except that it attempts to move records between the pages 4 and 2, rather than 2 and 1 The latter movement is successful, and Figure 4 indicates the resulting state of the sequential file at the time t_4

Since J=3 in our example, CONTROL 2 will have completed J executions of step 4 at the end of the time t_4 The instance t_4 thus denotes the time when CONTROL 2 has completed execution of the command Z_1

Let Z_2 denote the next command given to CONTROL 2, and suppose this command is an order to insert a record into the page 1 The latter causes $p(L_1) \geq g(L_1,2/3)$, and step 3 of CONTROL 2 will therefore call the subroutine ACTIVATE(L_1) This subroutine raises L_1 into a warning state and sets DEST(L_1) = 2 and DEST(v_3) = 1 The third action is due to ACTIVATE's roll-back rule 1, and it is the first occasion in our example where a roll-back rule is employed In our example, t_5 denotes the measurable moment following the execution of ACTIVATE(L_1)

The remainder of the command Z_2 is similar to the command Z_1 It consists of three executions of step 4, whose effect on the sequential file is indicated by the three rows of Figure 4 for the times t_6,t_7 and t_8 The first iteration of step 4b calls the subroutine SHIFT(L_1) to move thirteen records from the page 1 to 2 (SHIFT(L_1) stops the record movement after the thirteenth record transfer because $p(L_2) \geq g(L_2,0)$ at this time) Step 4c of CONTROL 2 will then notice that $p(L_1,t_6) \leq g(L_1,1/3)$, and it will accordingly lower L_1 into a non-warning state The main action in the second execution of step 4 consists of a subroutine call to SHIFT(v_3) that moves eleven records from page 2 to 1 (SHIFT(v_3) halts the record transfer after the eleventh record movement because $p(L_1) \geq g(L_1,0)$ at that time) A second action of SHIFT(v_3) consists of setting DEST(v_3) = 2 at the end of this procedure The third execution of step 4 makes another subroutine call to SHIFT(v_3), whose effect is to move five records from page 5 to 2 (record movements stop after the fifth record transfer because $p(v_4) \geq g(v_4,0)$ at this time) At the end of this iteration, step 4c lowers v_3 into a non-warning state on account of the fact that $p(v_3) \leq g(v_3,1/3)$ At the end of this example, all nodes in the calibration tree have returned to a non-warning state, and the eighth row in Figure 4 indicates the record distribution

The algorithm CONTROL 2 is intended for applications where the File F always has cardinality less than dM and where J, d and D satisfy the inequalities (5 1) and (5 2) below

$$(D-d) > 3 \lceil \log M \rceil \qquad (5\ 1)$$

$$J > \Omega\{ \lceil (\log^2 M) \rceil \ / \ (D-d)\} \qquad (5\ 2)$$

Theorem 5 5 shall state that CONTROL 2 guarantees that the sequential file will satisfy (d,D)-density at the end of each insertion and deletion command when the file was properly initialized and when the first sentence of this paragraph holds At the end of this chapter, we will explain how the constraint (5 1) may be dropped with a slightly more elaborate algorithm The second constraint (5 2) specifies the number of page accesses that CONTROL 2 must invoke It tells us, simply, that the time $O(\log^2 M/(D-d))$ is sufficient to maintain (d,D)-density because such magnitudes satisfy (5 2)'s inequality

It is important that J, d and D be assigned the values recommended in the previous paragraph because otherwise CONTROL 2 could cause some pages to eventually contain more than D records See the bottom paragraph on page 11 for a summary of the types of applications where CONTROL 2 outperforms a B-tree and a summary of its intuition

The proofs in our full-length paper [Wi85] are rather long, and we will give only an intuitive over-view in this conference paper We begin with two preliminary lemmas

Lemma 5.3. Suppose that d and D satisfy equation (5 1) and that some insertion command of CONTROL 2 causes $p(v) > g(v,1)$ Let t denote the last flag-stable moment immediately before this command, and let t^* denote the last flag-stable moment before t when $p(v,t^*) < g(v,2/3)$ Then CONTROL 2 must have executed at least $\lfloor M_v(D-d) \ / \ (3 \lceil \log M \rceil \ ) \rfloor$ insertion commands whose first step inserted a record into RANGE(v) between the times t^* and t

Proof. An inspection of the procedure CONTROL 2 reveals that the only aspect of this procedure that can increase $p(v)$ when it is exceeding $g(v,2/3)$ is step 1. Furthermore, an individual invocation of step 1 can increment $p(v)$ by no more than amount of precisely $1/M_v$. Since between the moment t^* and the first measurable moment after t, the quantity $p(v)$ must have increased by an amount of at least $g(v,1) - g(v,2/3) = (D-d) / (3 \lceil \log M \rceil)$, it follows that at least $\lfloor M_v(D-d) / (3 \lceil \log M \rceil) \rfloor$ invocations of step 1 must have occurred between the times t^* and t. Q.E.D.

Corollary 5.4 Say a call to the subroutine SHIFT **is related to v** iff this call occurs when v is in a warning state and at the same time step 1 of CONTROL 2 has inserted a record into RANGE(v). Then the preceding Lemma implies CONTROL 2 must have executed at least $J \cdot \lfloor M_v(D-d) / (3 \lceil \log M \rceil) \rfloor$ invocations of SHIFT that are related to v between the times t^* and t. Q.E.D.

Proof. Lemma 5.3 indicates that there are at least $\lfloor M_v(D-d) / (3 \lceil \log M \rceil) \rfloor$ occasions between the times t^* and t when step 1 of CONTROL 2 has inserted a record into RANGE(v). Since each such occasion is followed by J invocations of SHIFT related to v, there must be a total of at least $J \cdot \lfloor M_v(D-d) / (3 \lceil \log M \rceil) \rfloor$ related SHIFT operations occurring between the times t^* and t. Q.E.D.

Theorem 5.5: Let F denote a (d,D)-dense file whose records are initially distributed with a uniform density over the address space. Suppose d,D and J satisfy equations (5.1) and (5.2), and the algorithm CONTROL 2 is employed to perform insertions and deletions on a file F whose cardinality never exceeds $N = dM$. Then this file will satisfy the bound BALANCE(d,D) at the end of each insertion and deletion command performed by CONTROL 2.

The formal proof of Theorem 5.5 appears in [Wi85], and it is too lengthy to present within the space limits indicated in SIGMOD's call for papers. However, we can explain the intuition behind Theorem 5.5 by sketching how a violation of the BALANCE(d,D) condition would imply a contradiction.

The combination of Corollary 5.4 and equation (5.2) imply that v can not violate the condition BALANCE(d,D) without the execution of at least $6 M_v \log M$ SHIFT operations that are related to v occurring between the times t^* and t. Our formal proof in [Wi85] examines the implications of such a large number of SHIFT operations, it concludes this sequence must necessarily lower $p(v)$'s value, at some moment between t^* and t, to a quantity strictly less than $g(v,2/3)$. The key aspect of the last sentence is that it contradicts Lemma 5.3's definition of t^*, that is, Lemma 5.3 defined t^* to be the last flag-stable moment when $p(v,t^*) < g(v,2/3)$ before v violates the constraint BALANCE(d,D), but the previous sentence has noted that $p(v)$ must fall beneath $g(v,2/3)$ at some later moment, between the times t^* and t. This contradiction arose because the first sentence of this paragraph assumed v violated the condition BALANCE(d,D), this contradiction shows such a violation can not actually occur, and it thereby verifies Theorem 5.5's claim. (See the unabridged version of our paper [Wi85] for the added details of a formal proof.) Q.E.D.

If we take $J \geq \Omega[(\log^2 M) / (D-d)]$ then CONTROL 2's worst-case time limit satisfies $O((\log^2 M) / (D,d))$. Since every BALANCE(d,D) file also satisfies the constraint (d,D)-dense, Theorem 5.5 thus implies:

Corollary 5.6. Let F denote a (d,D)-dense sequential file whose records are initially distributed with a uniform density throughout this file. Suppose $D-d > 3 \lceil \log M \rceil$. Then the algorithm CONTROL 2 can maintain the condition (d,D)-density by executing no more than $O((\log^2 M) / (D-d))$ page-shift operations per insertion and deletion command.

Finally, we observe that an algorithm analogous to CONTROL 2 can also efficiently maintain (d,D)-density in the same time $O((\log^2 M) / (D-d))$ when the inequality $(D-d) > 3 \lceil \log M \rceil$ **does not** hold. The related algorithm is easiest to describe, if we let K denote the least integer such that

$$K(D-d) > 3 \lceil \log M \rceil \qquad (5.3)$$

Define the i-th **macro-block** in the sequential file to be those page addresses P that satisfy the equality

$$\lceil P / K \rceil = i \qquad (5.4)$$

Our algorithm for the alternate case where $D-d \leq 3 \lceil \log M \rceil$ will be the same as CONTROL 2 except that it will shift records between macro-blocks relative to a $(d^\#,D^\#)$-dense constraint where $D^\# = KD$ and $d^\# = Kd$, rather than shift records between normal sized pages. Since macro-blocks are K times as large as the normal sized pages, one must of course consider macro-block shift operations to be K times as costly as shifting records between standard sized pages By Corollary 5 6, the revised algorithm's cost is therefore $O((\log^2 M) / (K(D-d)))$, in macro-block operations, a quantity which translates into time $O((\log^2 M) / (D-d))$ when measured in terms of normal size page operations (The intuitive reason for the final cost of our procedure to be the same for the two cases where (D-d) is and is not greater than $3 \lceil \log M \rceil$ is that the cost from translating macro-pages to unit-sized pages is less than the dominant cost given in Theorem 5 5) We have thus informally proven the following theorem

Theorem 5.7. For each $d<D$, it is possible to perform insertions and deletions in worst-case time $O(\log^2 M / (D-d))$ in (d,D)-dense sequential files

Our full-length paper [Wi85] proves that $J \cong 90 \lceil \log^2 M \rceil / (D-d)$ is one adequate value for the J parameter in Figure 2 and in Equation (5 2) The proof of this fact in [Wi85] is approximately 40 pages long, and a more elaborate proof can in fact reduce Theorem 5 5's J-parameter by at least one order of magnitude (and probably by 1 1/2 magnitudes)

The update algorithms in Theorem 5 5 thru 5 7 are intended for applications where streams of records with consecutive key values are frequently accessed Although B-trees may have a smaller update cost than CONTROL 2, they are less desirable in an environment where many stream retrieval requests occur because of the increased latency delay arising when the disk draws consecutive keys from non-adjacent memory locations Incidentally, the asymptote $O(\log^2 M/(D-d))$ definitely over-estimates CONTROL 2's real cost because CONTROL 2, unlike a B-tree procedure, can be programmed to access adjacent pages during its update task [Wi82] describes a somewhat more sophisticated version of CONTROL 2 that has a better coefficient, and Hofri-Konheim-Willard [HKW86] show that an expected time O(1) is possible under similar procedures

Figure 1A

3	2	1	2

Figure 1B

The number of records in 4 pages of a dense file (Figure 1a) and its accompanying calibrator (Figure 1b) In this example, d=2 and D=3, and the number inside the node v is its density p(v)

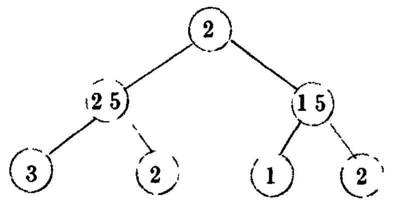

Figure 2 The procedure of CONTROL 2(Z,J)

<u>Algorithm</u> CONTROL 2(Z,J)

```
1) Use the calibrator as a binary search tree to calculate the address of the record specified by the command Z If Z is a deletion command then remove the relevant record from the sequential file and accordingly decrement the rank counters $N_v$ in the calibration tree that require change If Z is an insertion command then add this record to the sequential file and increment the rank counters $N_v$ in the calibration tree that require change

2) If step 1 caused any node in a warning state to satisfy $p(x) \leq g(x,1/3)$ then lower x into a non-warning state, i e set WARNING(x)=0

3) If step 1 caused $p(w) \geq g(w,2/3)$ for a nonroot node w in a non-warning state then call ACTIVATE(w) (to essentially raise w into a warning state)

4) Let L denote the leaf-address of the record R that was inserted or deleted by step 1 Perform J iterations of the following cycle of 3 commands,
    a) Set $v \leftarrow$ SELECT(L)
    b) Do SHIFT(v)
    c) If step b caused any node in a warning state to satisfy $p(x) \leq g(x,1/3)$ then lower x into a non-warning state, i e set WARNING(x)=0

End of algorithm
```

Figure 3

The calibration tree for the 8-page file discussed in Example 5 2

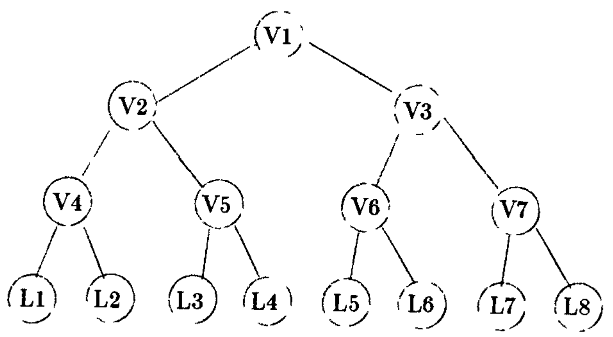

Figure 4

The changes in the record distribution over time for Example 5 2

Leaf / Time	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
t_0	16	1	0	1	9	9	9	16
t_1	16	1	0	1	9	9	9	17
t_2	16	1	0	1	9	9	15	11
t_3	16	1	0	1	9	9	15	11
t_4	16	2	0	0	9	9	15	11
t_5	17	2	0	0	9	9	15	11
t_6	4	15	0	0	9	9	15	11
t_7	15	4	0	0	9	9	15	11
t_8	15	9	0	0	4	9	15	11

References

[BCW-85] B S Baker, E G Coffman, Jr , and D E Willard, "A Dynamic Storage Allocation Algorithm Designed for Badly Fragmented Memory," *JACM* 32 2 (1985) pp 327-343

[Fr-79] W R Franklin, "Padded Lists Set Operations in Expected θ(log log N) Time," *Inf Proc Letters*, 9 4 (1979), pp 161-166

[HKW-86] M Hofri, A Konheim and D Willard, "Padded Lists Revisited", forthcoming report

[IKR-80] A Itai, A G Konheim, and M Rodeh, "A Spare Table Implementation of Priority Queues," *Proceedings of ICALP-1981,* LNCS 115, pp 417-431

[MG-78] R Melville and D Gries, "Sorting and Searching Using Controlled Density Arrays," Technical Report 78-362, Computer Science Department, Cornell University, 1978, see also [MG-80]

[MG-80] R Melville and D Gries, "Controlled Density Sorting," *Inf Proc Letters,* 10 4 (1980), pp 169-172

[Wh-77] G Wiederhold, *Database Design,* McGraw-Hill, New York, 1977

[Wi-81] D E Willard, "Inserting and Deleting Records in Blocked Sequential Files," Bell Labs Tech Report TM81-45193-5, 1981

[Wi-82] D E Willard, "Maintaining Dense Sequential Files in a Dynamic Environment," Bell Labs Tech Mem 45413-821230 2, 1982

[Wi-85] D E Willard, "A Density Control Algorithm for doing insertions and deletions in a sequentially ordered file in good and worst case time" SUNY Albany Technical Report 85-14, 1985

[WL-85] D E Willard and G Lueker, "A Transformation for Adding Range Restriction Capability to Data Structures," *JACM* 32 3 (1985) pp 597-618