An Empirical Comparison of B-Trees, Compact B-Trees and Multiway Trees

David M. Arnow
Aaron M. Tenenbaum

Dept. of Computer and Information Science
Brooklyn College of C.U.N.Y.

ABSTRACT

It is well-known that the B-tree data structure yields excellent worst-case search costs and for that reason is widely employed in the organization of external files and in the implementation of data bases. In this paper, we examine general B-trees empirically and compare them with a less restrictive structure, the general multiway tree, and a more restrictive structure, the compact B-tree. We compare search costs, insertion costs, and space costs of these three structures for both small and large orders and indicate their relative utility for large and small data sets. Although there are cases when general multiway trees are more effective than B-trees, this is not the case for most practical situations. Compact B-trees are also shown to degrade rapidly in the presence of insertions and are therefore only useful for static data sets.

0. Introduction

Tree structures of various types are widely used to store data to permit efficient access, insertion and deletion operations. This paper is concerned with the organization of data on secondary storage media. In this case, the choice of viable tree structures is severely restricted because of the great difference in speed of memory-to-memory and device-to-memory operations, as well as because of the problem of space utilization. Because of their excellent worst-case and reasonable average-case access properties, B-trees [2] or their variants have become the tree structure of choice.

In this paper, we examine general B-trees empirically and compare them with a less restrictive structure, the general multiway tree, and a more restrictive structure, the compact B-tree, a storage-efficient B-tree variant. In the next section, the data structures of interest are defined. In Section 2, some analytical work on the efficiency of multiway trees is presented. Section 3 discusses the empirical methods that were used. Section 4 compares various properties of the trees under consideration, and Section 5 takes up the issue of the dynamic behavior of compact B-trees.

Supported by the National Science Foundation under Grant IST 80-21350.

1. Definitions

In this section, the data structures of interest are defined.

B-TREES. A B-tree of order M [2,3,5] is a tree in which no node contains more than M-1 keys; all nodes, excluding the root, contain at least (M-1) **div** 2 keys; all leaves are at the same level; and any non-leaf node with N keys has N+1 children. (We are using "order" as it is used in [7,10] rather than as in [3,5]. We are interested in the use of B-trees as storage/retrieval structures. As such, the nodes of the B-tree contain keys from an ordered set. Each non-leaf node has one fewer key than children, and, ordering the keys in the node $k_1, k_2, \ldots, k_N$, the following conditions must hold: k_1 is greater than all the keys in the first subtree of the node, all the keys in the i-th subtree ($1<i<=N$) are greater than k_{i-1} and less than k_i, and k_N is less than all the keys in the (N+1)-th subtree. An algorithm for inserting keys into B-trees may be found in [2,3,5].

COMPACT B-TREES. A compact B-tree [6,7] is a space-optimal B-tree. Following Rosenberg and Snyder [7], compact B-trees may be characterized as follows. Let the profile of a depth d B-tree of k keys be the (d+2)-tuple

$$(v_0, v_1, \ldots, v_{d+1})$$

where v_i is the number of nodes at the i-th level in the tree and v_{d+1} equals k+1. Note that v_0 is 1 since that is the root level, and v_{d+1} corresponds to the division of the key space into k+1 intervals defined by the k keys in the tree. Rosenberg and Snyder [7] prove that an order M B-tree is compact if and only if for $0<=i<d$, $v_i = \lceil v_{i+1}/M \rceil$. In other words, to construct a compact tree for k keys, maximize the number of nodes at the lowest level $\lceil (k+1)/M \rceil$; then, working up, maximize the nodes at succeding levels. The effect of this is to maximize both branching and key density in the lower levels. In addition to this characterization of space optimal B-trees, Rosenberg and Snyder [7] also prove that, regardless of order, for large data sets, the search-time costs of these trees are nearly optimal.

There is no known efficient algorithm for compactness-preserving insertion into a compact B-tree. Rosenberg and Snyder suggested periodic compaction, perhaps concurrently with periodic backups. Between compactions, insertions and deletions into the structure are to be carried out in the usual B-tree fashion.

M-TREES. A multiway-tree, or M-tree, of order d is a generalization of a binary search tree. Such a tree has three kinds of nodes: leaf nodes, which have no children, semi-leaf nodes with exactly d-1 keys but fewer than d children, and internal nodes with d-1 keys and d children. (Note that a node with exactly d-1 keys and no children is both a leaf and a semi-leaf.) Used as a storage/retrieval structure, the usual ordering property holds among the keys in a node and its subtrees. An example of an order 5 M-tree is given in Figure 1.1.

To insert a key into an M-tree, a search is made for the appropriate leaf or semi-leaf node. In the case of a leaf node with fewer than d-1 keys, the new key is inserted into the leaf node. In the case of a semi-leaf node, a new leaf node is created. The key is stored in the new leaf node, which becomes the child of the semi-leaf node in such a way as to preserve the order condition. (If the parent node had d-1 children before the insertion, then it is no longer a semi-leaf node but an internal node.) Thus, M-tree growth takes place at the leaves, as in conventional binary trees. Figures 1.2a and 1.2b illustrate two successive additions of keys (4 and 83) to an M-tree of order 5 (Figure 1.2a). The search for a node for key 4 yields a semi-leaf node. The creation of a new leaf for 4 causes the semi-leaf node to become an internal node (Figure 1.2b). The search for a node for key 83, yields a leaf node. The insertion of 83 into that leaf, results in the leaf becoming a semi-leaf (while remaining a leaf as well, since it has no descendants) (Figure 1.2b).

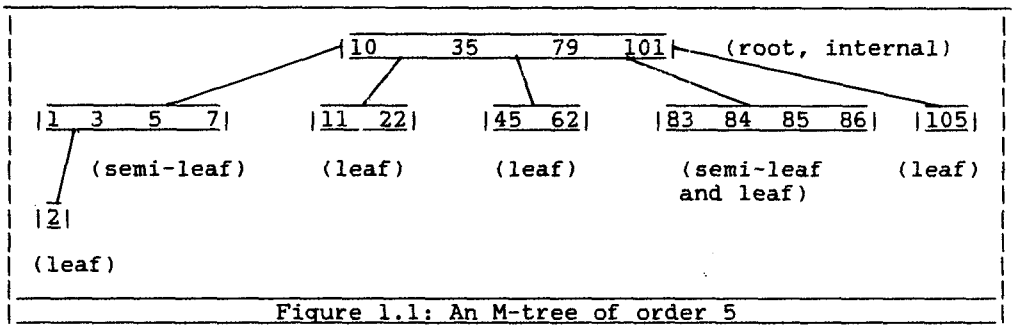

Figure 1.1: An M-tree of order 5

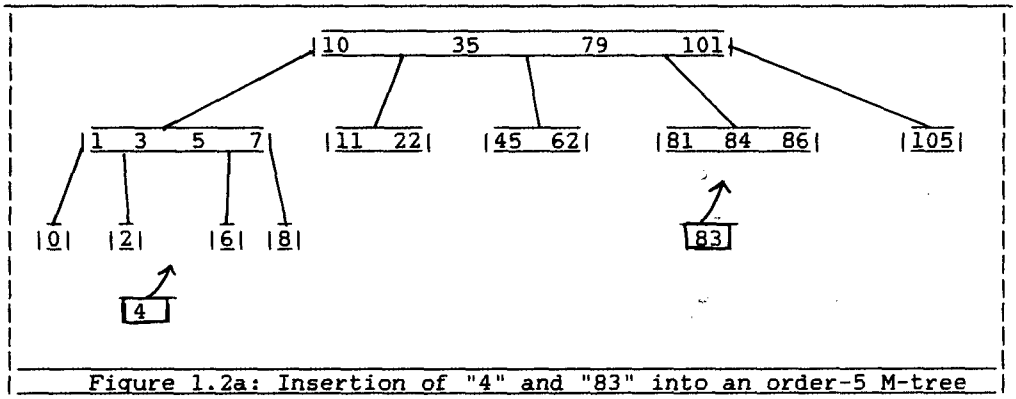

Figure 1.2a: Insertion of "4" and "83" into an order-5 M-tree

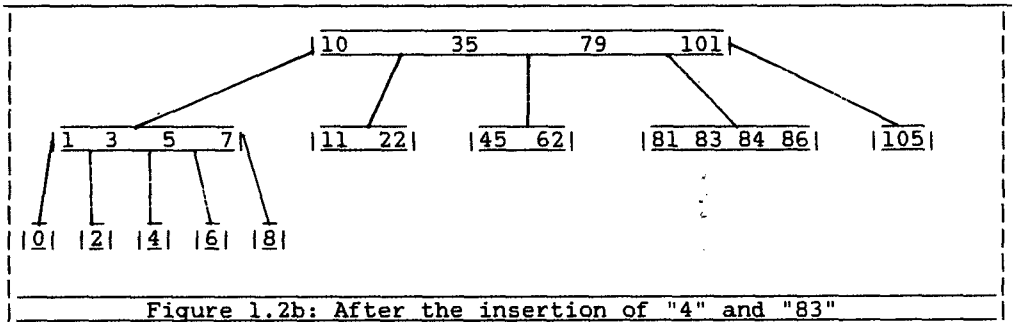

Figure 1.2b: After the insertion of "4" and "83"

Choosing among the above information structures involves making a number of trade-offs, including space-time and average-case vs. worst case. In general, the papers and textbooks on the subject make little or no mention of M-trees and emphasize the time costs of storage and retrieval operations, rather than storage costs. For example, Comer [3] writes that "the balancing mechanism [of B-trees] uses extra storage to lower the balancing costs (presumably, secondary storage is inexpensive compared to retrieval time)." In the original article of Bayer and McCreight [2] proposing B-trees, the emphasis was on finding a balancing scheme such as that employed in AVL trees [1,4] but which would be appropriate for secondary storage. (The AVL balancing mechanism is inappropriate for secondary storage because, unlike that of the B-tree, it does not restrict the changes it makes in the tree structure to a single path from the root to the point of insertion.) Although Bayer and McCreight do refer to the storage utilization lower bound of 50% as a benefit, it appears that they were considering hashing and related indexing schemes as "competition". There is no mention of M-trees in that paper or its references (for example, [8]). The reader receives the impression that B-trees are preferable to M-trees because they provide faster access time, albeit at an increased storage cost. In fact, as this paper shows, this is not the case at all.

2. An Analysis of M-Trees

One can generalize the analysis of search cost for binary trees [9] to M-trees of order d as follows. Assume that there are n>d-1 distinct keys with values of 1 through n. Assume that the first d-1 keys to arrive are the set

$$\{i_1, i_2, \ldots, i_{d-2}, i_{d-1}\}$$

and that, without loss of generality, $i_j < i_{j+1}$ for j=1...d-2. For convenience, define

$i_0=0$ and $i_d=n+1$.

After the remaining n-d+1 keys arrive, the tree has the representation shown in Figure 2.1.

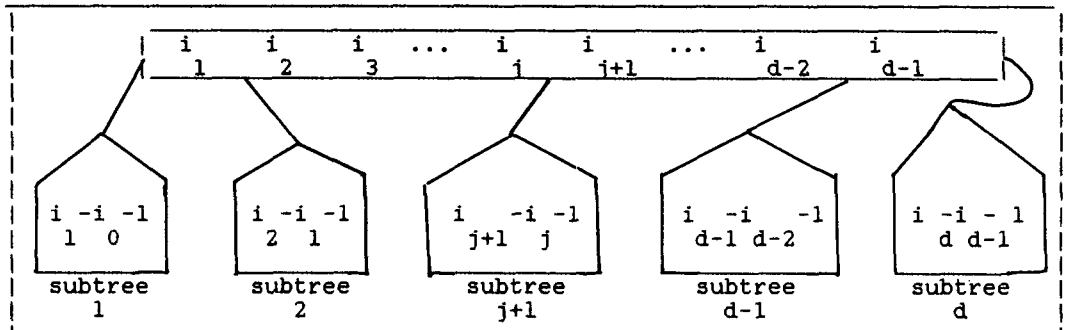

Figure 2.1: M-tree of order d after the arrival of n keys

The expressions in the subtrees indicate the number of keys in each subtree. Let a_n be the average search path length for any key in any order-d M-tree of n keys. Let

$$a_n^{[i_1..i_{d-1}]}$$

be the average search path length for any key in any n-key order-d M-tree, assuming that $\{i_1, i_2, \ldots, i_{d-2}, i_{d-1}\}$ is the set of keys in the root. Then,

$$a_n^{[i_1,\ldots,i_{d-1}]} = (1)\frac{d-1}{n} + \sum_{j=1}^{d} \left[\frac{i_j - i_{j-1} - 1}{n} \right] (1 + a_{i_j - i_{j-1} - 1})$$

By averaging over all possible $\{i_1..i_{d-1}\}$, (C(n,d-1) in number), a_n is found:

$$a_n = \frac{1}{C(n,d-1)} \sum_{i_1=i_0+1}^{n-d+2} \cdots \sum_{i_k=i_{k-1}+1}^{n-d+k+1} \cdots \sum_{i_{d-1}=i_{d-2}+1}^{n} \left[\frac{d-1}{n} + \frac{1}{n} \sum_{j=1}^{d} (i_j - i_{j-1} - 1)(1 + a_{i_j - i_{j-1} - 1}) \right]$$

There is a one-to-one correspondence between the subtrees shown in Figure 2.1 and the terms of the innermost sum (over j from 1 to d). The contribution of each term to the total sum equals the contribution tha the corresponding subtree to the average search path length of the entire set of trees. Since no subtree is favored, each subtree and therefore each term makes the same contribution to the total. Thus, we can let the j=1 term replace all the other terms by weighting it with d, the number of terms in the j-sum. Thus,

$$a_n = \frac{1}{nC(n,d-1)} \sum_{i_1=1}^{n-d+2} \cdots \sum_{i_k=i_{k-1}+1}^{n-d+k+1} \cdots \sum_{i_{d-1}=i_{d-2}+1}^{n} \left[d-1 + d(i_1-1)(1+a_{i_1-1}) \right]$$

since $i_0=0$. Since the summand in the this equation is independent of all indices save i_1, the equation can be rewritten as:

$$a_n = \frac{1}{nC(n,d-1)} \sum_{m=1}^{n-d+2} \left[d-1 + d(m-1)(1+a_{m-1}) \right] C(n-m,d-2)$$

since there are C(n-m,d-2) terms in the outer sum of the previous equation. With some rearranging, and using the fact that

$$\sum_{m=1}^{n-d+2} (d-1+d(m-1))C(n-m,d-2) = (d-1)\sum_{m=0}^{n-d-1} C(n-m-1,d-2) + d\sum_{m=0}^{n-d+1} mC(n-m-1,d-2)$$

$$= (d-1)C(n,d-1) + dC(n,d) = nC(n,d-1),$$

this can be rewritten as

$$a_n = 1 + \frac{d}{nC(n,d-1)} \sum_{m=1}^{n-d+1} mC(n-1-m,d-2)a_m.$$

Although we have been unable to obtain a closed form, this relation is useful in providing numerical results for M-trees, as well as for judging the correctness of simulation results for M-trees and B-trees.

Using the same techniques as were used above, the following relations for the average number of nodes and the average number of leaves for M-trees may also be derived:

$$N_n = 1 + \frac{d}{C(n,d-1)} \sum_{m=1}^{n-d+1} C(n-1-m,d-2)N_m \quad \text{(Number of Nodes)}$$

$$L_n = \frac{d}{C(n,d-1)} \sum_{m=1}^{n-d+1} C(n-1-m,d-2)L_m \quad \text{(Number of leaves)}$$

Values derived from these expressions were the same as the empirically derived quantities presented in subsequent sections.

3. Simulation Methodology

General B-trees and even M-trees are exceptionally difficult objects to describe analytically. Since our interests are pragmatic as well as analytic, and as a first step towards a better theoretical understanding of these objects, the results in this paper are primarily empirical. Simulations were carried out by constructing random (in Yao's [10] sense) trees by repeated insertion of keys generated by a non-repeating, uniform, pseudo-random number generator. This method was used to construct both general B-trees and M-trees. Compact B-trees, on the other hand, were constructed by sorting the set of keys used to generate a B-tree or M-tree and building the (unique) compact B-tree which is most skewed to the left, using the algorithm suggested by Theorem 2.2 in [7].

QUANTITIES OF INTEREST. The quantities that we have investigated are the time costs for the insertion and retrieval of records and the storage utilization of the entire file structure. The time cost of a search or an insertion is considered to be the number of i/o operations involved, that is, the number of nodes accessed. It is assumed that, except for a very limited buffer capacity, all the nodes of the file structure reside in secondary storage. Thus, the time cost of a retrieval of a record from either a B-tree, compact B-tree, or M-tree is the number of node-visits required to find the node in which the record resides or to verify its absence from the structure. The time cost of an insertion into an M-tree is the number of reads required to find the appropriate node for the insertion, plus an additional write to update that node. If the latter was a semi-leaf, then yet another write operation is counted, since insertions into semi-leaves require the allocation and writing of a new leaf node. The time cost of an insertion into a B-tree or compact B-tree is the number of reads required to find the correct leaf node for insertion, plus the writing out of that leaf, plus twice the number of splits resulting from the insertion. This is because it is assumed that the buffer capacity is sufficient to hold all the nodes in a path in the tree (obviating additional reads) and because each split necessitates the writing of an additional sibling node and the parent of the split node (two writes per split). (Note that, strictly speaking, one cannot "insert" into a compact B-tree using the above method, since such an insertion is likely to destroy compactness. We speak of insertion into a compact B-tree as a non-compactness-preserving operation.) As described above, the compact B-trees in the simulation were skewed to the left, that is, the compact B-trees were constructed in such a way that the full nodes in a given level were placed to the left of those that were not full. Because of this, the insertion cost measurements for these trees would be higher than those for the best compact B-trees. However, an informal analysis and some measurements show that this bias is negligible insofar as the basic results of this paper are concerned.

The storage utilization for any of the types of trees is $k/((d-1)N)$ where k is the number of keys present in the tree, N the number of nodes in the tree, and d the order of the tree (i.e. d-1 is the maximum number of keys that can be possibly stored in a single node).

For each distinct tree in a sample set of randomly generated trees of a particular type, order, and file size, the storage utilization, average retrieval cost and average insertion cost were computed. The average retrieval cost for one tree is the time cost for searching averaged over all k keys in the tree. (Unsuccessful searches are not considered.) The average insertion cost for one tree is the time cost for inserting a key averaged over all k+1 intervals defined by the k keys present in the tree. Each of these single-tree quantites were averaged over the entire sample set that was generated for trees of a particular type, order and file size.

MEASUREMENT. To measure the average search cost for a particular tree generated by the simulation, a sum of the path lengths from the root to each key was computed and divided by the number of keys. To measure the storage utilization, a simple procedure for counting nodes and keys was employed, in conjunction with the expression for storage utilization above. Finally, to measure the expected insertion cost for a particular tree generated by the simulation, the costs of insertion into each of the k+1 intervals in the key-space (in a k-key tree) were averaged.

VALIDITY. The validity of the simulations was verified by computing quantities that could be calculated analytically. For example, the storage utilization of B-trees was compared with the results of Yao [10], the storage utilization of compact 2-3 trees was compared with the work of Rosenberg and Snyder [7], and the storage utilization and search costs of M-trees were compared with the analytical results in Section 2. The differences between analytically obtained results and those obtained through simulation were (absolutely) small and within the statistical error of the simulation. Furthermore, these comparisons showed that a sample size of 32 was sufficient to attain the desired accuracy. Along with these precautions, the usual statistical concerns regarding adequate pseudo-random number generation were verified.

4. Empirical Results

In this section, we compare the observed properties of M-trees, B-trees, and compact B-trees.

SEARCH TIME. Table 4.1 shows the higher search time cost of B-trees, expressed as a percentage difference relative to M-trees. Figure 4.1 plots search time cost versus file size for B-trees, M-trees and Compact B-trees of order 9. In Table 4.1, the large variations in sign and absolute value are not due to statistical fluctuations associated with the empirical work, but rather are due to the nature of the B-trees and M-trees themselves. On the whole, M-trees are lower in search time cost than B-trees, sometimes by as much as 25%. On the other hand, as can also be seen from the Table 4.1, there are situations where M-trees do worse in this regard, though apparently not by more than 5 or 6%. In fact, as one considers larger and larger file sizes for a particular order, a cyclical behaviour is observed in the relative difference between B-tree and M-tree search time cost. For example, consider order 13, the fourth column of entries in Table 4.1. For 500 keys, the B-tree is at a 4% disadvantage with respect to the M-tree. As keys are added, however, the B-tree gains first a 2% advantage (at 750 keys), and then a 6% advantage (at 1000) keys. This is due to the fact that over extended periods of key insertions, B-trees will not grow in height (one of their chief advantages over M-trees). At 2000 keys, the B-tree advantage in search time over M-tree has disappeared-- and the B-tree is now at a 14% disadvantage. This is because at some point between 1000 and 2000 keys the B-trees' root split and there was an increase in height. Following this, the M-tree advantage shrinks to 8%, 4%, 2% (3000, 4000, 5000 keys resp.) until once again the B-tree gains a slight advantage (at 6000 keys). This advantage persists until height growth (between 9000 and 10000 keys), when the difference becomes 11% in favor of the M-trees. This type of behaviour may be observed for all orders (Table 4.1).

Table 4.1
Per Cent Higher Search Time Cost
Of B-trees Over M-trees

Keys	Order 3	5	9	13	17	21	41
500	-2.0	5.8	19.1	4.2	15.1	24.2	0.8
750	3.2	-1.3	11.5	-2.1	8.2	15.9	-3.7
1000	-0.8	-0.7	6.6	-6.4	3.9	11.2	-7.4
2000	1.0	3.3	-3.6	14.3	-5.5	1.8	23.7
3000	1.4	-2.4	14.5	8.3	20.4	-3.2	16.3
4000	2.4	2.8	11.1	4.5	16.1	-6.8	12.3
5000	-0.1	7.5	8.0	1.7	13.0	16.8	9.5
6000	-0.4	5.1	5.6	-0.4	10.6	_____	_____
7000	1.0	3.2	3.8	-2.2	8.7	_____	_____
8000	1.9	1.6	2.2	-3.7	7.1	_____	_____
9000	3.0	0.2	0.8	-3.5	5.7	_____	_____
10000	1.7	-1.1	-0.4	11.3	4.5	_____	_____

A negative value indicates B-tree superiority.

Table 4.2 compares the search time cost of M-trees and compact B-trees and shows the superiority of low order compact B-trees in this area. As can be seen from the Figure 4.1 and Table 4.2, compact B-trees have search time costs substantially lower than that of either B-trees or M-trees for low orders. This advantage disappears at high orders, where the cyclical behaviour that characterizes the B-tree/M-tree difference reappears, although more favorably for the compact B-tree.

Table 4.2
Per Cent Higher Search Time Cost
Of M-trees Over Compact B-trees

Keys \ Order	3	5	9	13	17	21
500	15.7	14.5	10.3	-5.5	-16.2	-25.0
1000	11.6	3.4	-8.1	5.2	-4.9	-12.0
2000	20.5	13.0	2.1	14.1	4.6	-2.5
3000	13.4	17.7	7.3	-9.4	9.5	2.5
4000	16.7	4.1	10.7	-5.5	12.7	____
5000	19.1	6.7	13.1	-2.7	-13.8	____
6000	20.9	8.8	15.0	-0.5	-11.4	____
7000	12.0	10.5	-5.0	1.3	-9.5	____
8000	13.4	12.0	-3.4	2.8	-7.9	____
9000	14.6	13.2	-2.0	4.1	-6.5	____
10000	15.7	14.2	-0.8	5.2	-5.2	____

A negative value indicates M-tree superiority.

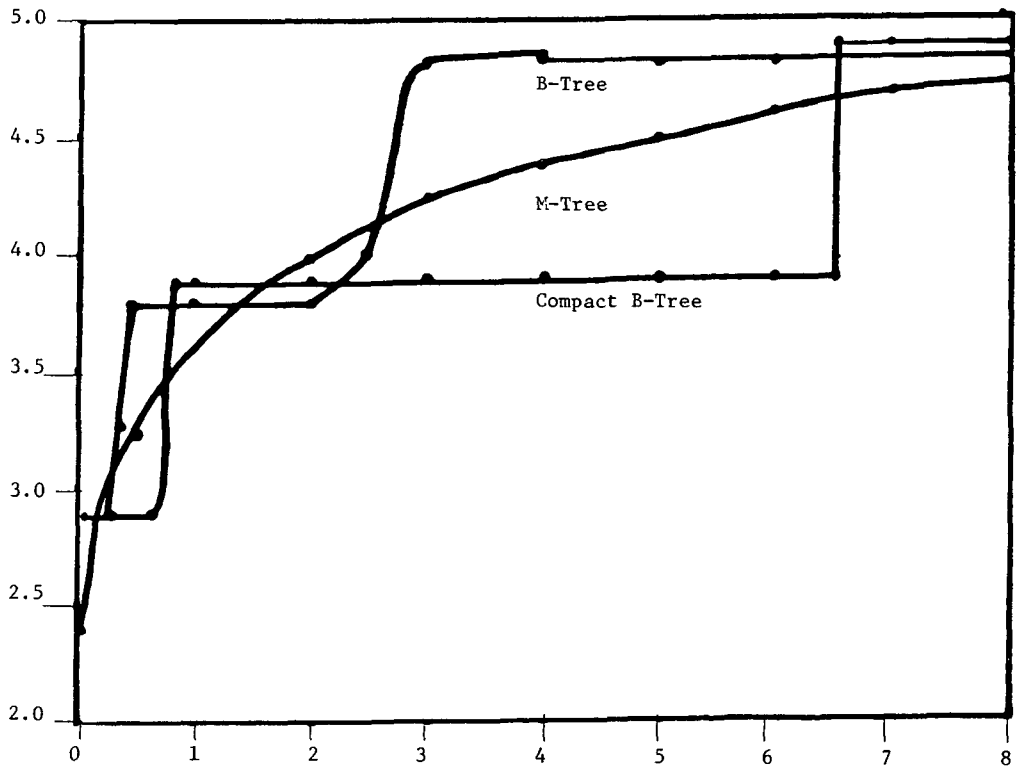

FIGURE 4.1: Average Search Time Cost Vs. File Size (in thousands of keys) At Order 9.

STORAGE. Table 4.3 presents the storage utilizations of B-trees and M-trees of several orders and file sizes. Figure 4.2 plots the storage utilization of B-trees, compact B-trees and M-trees as a function of order, for a file of 10000 keys. Storage utilization for B-trees and compact B-trees shows no dependence on order; for B-trees, it ranges from 67 to 71%, for compact B-trees from 92 to nearly 100%, independent of order. Storage utilization for M-trees on the other hand, shows a strong dependence on order. It is not widely realized that for order 3, M-trees are superior (in storage only, not in search cost) to B-trees, and even for order 5 they are competitive, although they are never superior to compact B-trees.) Thereafter, the storage utilization plummets for higher order M-trees. Figure 4.2 illustrates dramatic decline relative to B-tree storage utilization.

Table 4.3
Storage Utilization Of B-Trees And M-Trees

Keys \ Order	3	5	9	13	17	21
100	0.68	0.67	0.66	0.70	0.67	0.64
	0.83	0.64	0.46	0.39	0.34	0.28
200	0.67	0.68	0.68	0.67	0.70	0.69
	0.84	0.65	0.45	0.35	0.32	0.32
500	0.67	0.68	0.68	0.68	0.68	0.68
	0.83	0.64	0.46	0.36	0.29	0.25
1000	0.67	0.68	0.68	0.68	0.68	0.69
	0.83	0.64	0.46	0.37	0.31	0.26
2000	0.67	0.68	0.68	0.68	0.68	0.69
	0.83	0.64	0.46	0.36	0.31	0.27
5000	0.67	0.67	0.68	0.68	0.69	0.68
	0.83	0.64	0.46	0.36	0.30	0.27

Lower entries are storage utilizations for M-trees.
Compact B-tree utilizations are all near 0.99

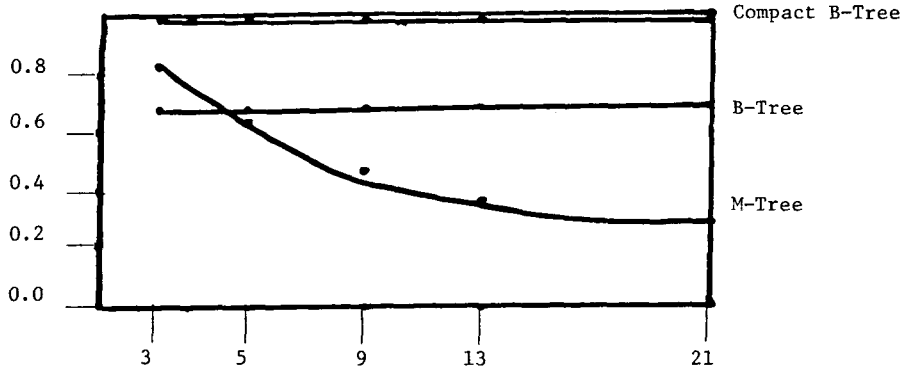

FIGURE 4.2: Storage Utilization Vs. Order

INSERTIONS. The cost of insertions for M-trees is consistently lower than that for B-trees. Figure 4.3 and Table 4.4 illustrate this. By comparing Table 4.1 with Table 4.4, it may be seen that as much as half the difference and most of the variability of the difference is due to the advantage that M-trees have in search-time over B-trees. The difference that cannot be accounted for by search time is more or less constant over the orders and file sizes considered. That component results from the splitting costs of B-tree insertions. Splitting is a much more frequent event when inserting into compact B-trees. The cost of insertions for compact B-trees may be as much as twice that of B-trees and M-trees for sufficiently large file sizes. This may be seen in Table 4.5 which compares insertion cost between B-trees and compact B-trees.

Table 4.4
Per Cent Higher Insertion Time Cost
Of B-trees Over M-trees

Keys \ Order	3	5	9	13	17	21
500	24.3	30.0	42.2	34.1	43.8	48.6
750	28.3	22.2	35.4	24.5	35.8	41.8
1000	23.3	22.7	30.4	19.3	30.4	38.2
2000	22.5	24.2	19.4	36.5	18.9	27.8
5000	19.5	25.6	27.1	22.5	____	38.4

Positive values indicate M-tree superiority.

Table 4.5
Per Cent Higher Insertion Time Cost
Of Compact B-trees Over B-trees

Keys \ Order	3	5	9	21	41	101
500	41.9	30.9	7.9	28.9	0.8	11.8
1000	44.2	42.0	34.6	27.0	0.3	5.5
2000	45.6	36.2	43.6	27.2	28.3	0.0
5000	44.9	42.9	40.4	16.0	44.7	0.8
10000	45.9	42.3	45.4	38.8	44.4	0.0

Positive values indicate superiority of B-trees over Compact B-trees

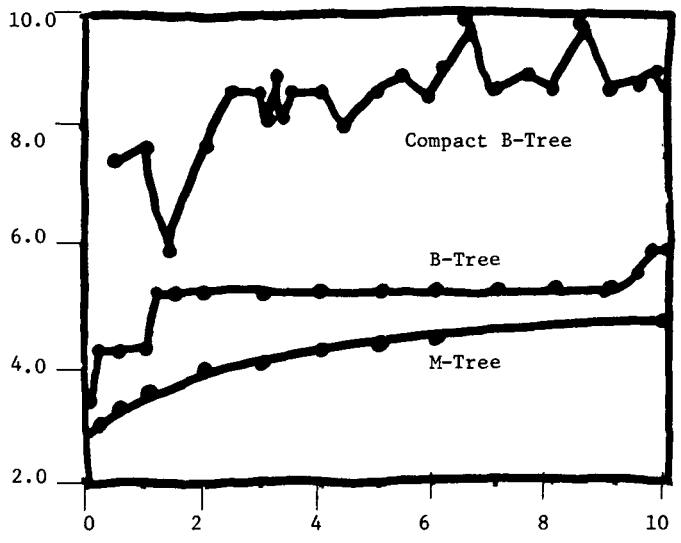

FIGURE 4.3: Average Insertion Cost Vs. File Size (in thousands of keys) For Order 13

5. Degradation of Compactness

Considering only the properties discussed in the above section, for higher orders, it is preferable to use compact B-trees provided that either the number of searches is substantially greater than the number of insertions or storage considerations outweigh concerns of speed, AND the compaction process can be undertaken in conjunction with the backup operation at a convenient time. This is because M-trees are hopelessly storage inefficient at high orders, and compact B-trees use significantly less storage than B-trees. Furthermore, compact B-trees provide somewhat faster retrievals and decidedly slower insertions than do B-trees.

Unfortunately, compact B-trees are advantageous only in the context of data bases which are very non-volatile. Figure 5.2 plots the decline in storage utilization as new keys are inserted, without recompaction, into compact B-trees of various orders, with an initial size of 10000 keys. The decline is smallest, though not insignificant, for the lower order trees; for high order, such as 41, after a 0.5% increase in file size, more than half the space-utilization advantage of the compact B-tree over an ordinary B-tree is lost. After a 1.4% increase, there is no advantage, and by 2%, the once-compact tree is at a decided disadvantage. Thereafter, its storage utilization plunges to near pessimal levels. Rosenberg and Snyder [7] prove analytically that for order 3, the advantage persists until at least 2.5% additional insertions have been made. Our empirical results indicate that for low orders, the advantage persists for quite a while longer.

Interestingly, the insertion inefficiency (relative to B-trees) of compact trees persists after its space advantage has disappeared. Table 5.1, which displays a number of properties of order 41 compact B-trees as insertions are made, demonstrates this. For example, for an order 41 compact B-tree with an initial size of 5000 keys, after 2% growth, with storage utilization fallen to 64%, the insertion cost is still 20% greater than that of a random B-tree (and 62% greater than that of an M-tree).

If file growth continues for a sufficiently long period, even if no recompaction takes place, the storage utilization will rise. Furthermore, it will eventually again exceed the storage utilization of a random B-tree, and then, once again, fall below that of the B-tree. As more keys are added, the storage utilization shows a dampened oscillation about the average storage utilization for random B-trees. This is seen in Figure 5.1 and in Table 5.1. This is also seen in Table 5.2, which, displays the peaks and troughs of storage utilization, along with other properties, as keys are added to an originally compact order-21 B-tree of 5000 keys. From Table 5.2, it may be seen that in this instance there are 2 troughs before the storage utilization stabilizes around the value 0.69, after the addition of nearly 10000 additional keys.

Table 5.1
Changes In Properties During Insertions
To 5000-Key, Order 41 Compact B-tree

Tree Type	% Additions To Tree	Storage Utilization	Search Cost	Insertion Cost
RANDOM B-TREE	0.0	0.680	2.964	4.061
COMPACT TREE	0.0	0.992	2.975	7.329
COMPACT+	0.2	0.904	2.973	5.823
COMPACT+	0.4	0.854	2.971	5.684
COMPACT+	0.6	0.810	2.970	5.548
COMPACT+	0.8	0.775	2.968	5.425
COMPACT+	1.0	0.744	2.967	5.313
COMPACT+	1.2	0.716	2.966	5.195
COMPACT+	1.4	0.691	2.964	5.083
COMPACT+	1.6	0.672	2.963	4.995
COMPACT+	1.8	0.658	2.963	4.918
COMPACT+	2.0	0.644	2.962	4.842
COMPACT+	3.0	0.601	2.959	4.583
COMPACT+	4.0	0.575	2.957	4.394
COMPACT+	5.0	0.558	2.956	4.254
COMPACT+	6.0	0.548	2.955	4.157
COMPACT+	8.0	0.546	2.955	4.082
COMPACT+	10.0	0.551	2.955	4.050
COMPACT+	20.0	0.592	2.958	4.000
COMPACT+	30.0	0.641	2.961	4.000
COMPACT+	40.0	0.690	2.964	4.004
COMPACT+	50.0	0.733	2.966	4.015
COMPACT+	60.0	0.761	2.968	4.053
COMPACT+	70.0	0.759	2.967	4.070
COMPACT+	80.0	0.748	2.967	4.109
COMPACT+	90.0	0.722	2.966	4.136
COMPACT+	100.0	0.690	2.964	4.125

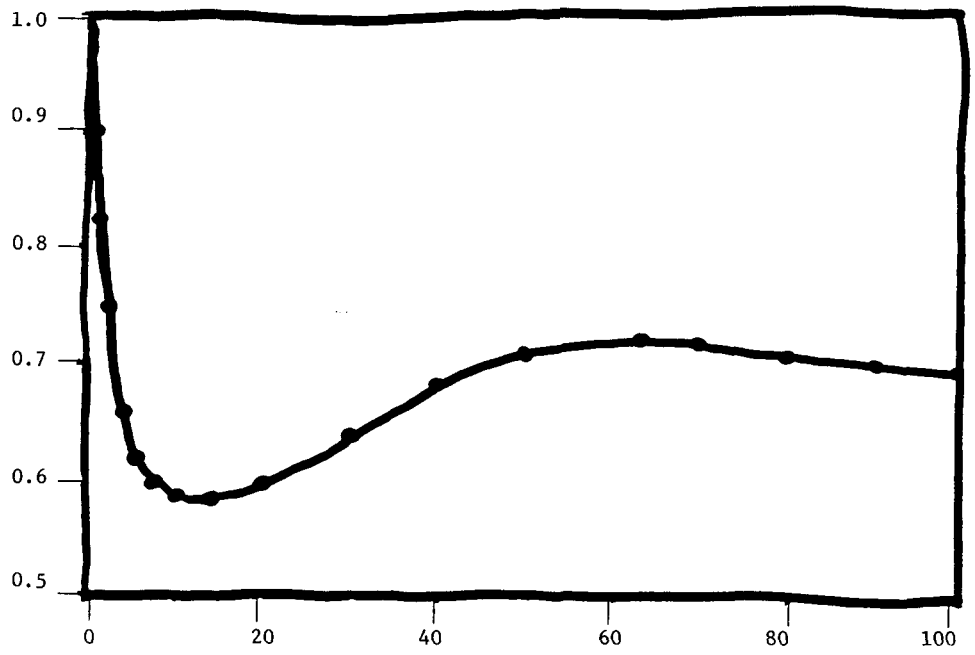

FIGURE 5.1: Decline In Storage Utilization Due To Insertions Into A Compact B-tree of Order 21 (Storage Utilization Vs. Percent Growth)

Table 5.2
Peaks And Troughs In Storage Utilization During Course Of Insertion Of Keys Into Compact Order-21 B-tree of 5000 Keys

Tree Type	% Additions To Tree	Storage Utilization	Search Cost	Insertion Cost
COMPACT B-TREE	0.0	0.992	2.950	5.856
COMPACT+	13.0	0.583	3.915	5.168
COMPACT+	59.0	0.726	3.932	5.120
COMPACT+	119.0	0.679	3.927	5.162
COMPACT+	171.0	0.689	3.928	5.140
COMPACT+	191.0	0.688	3.928	5.146

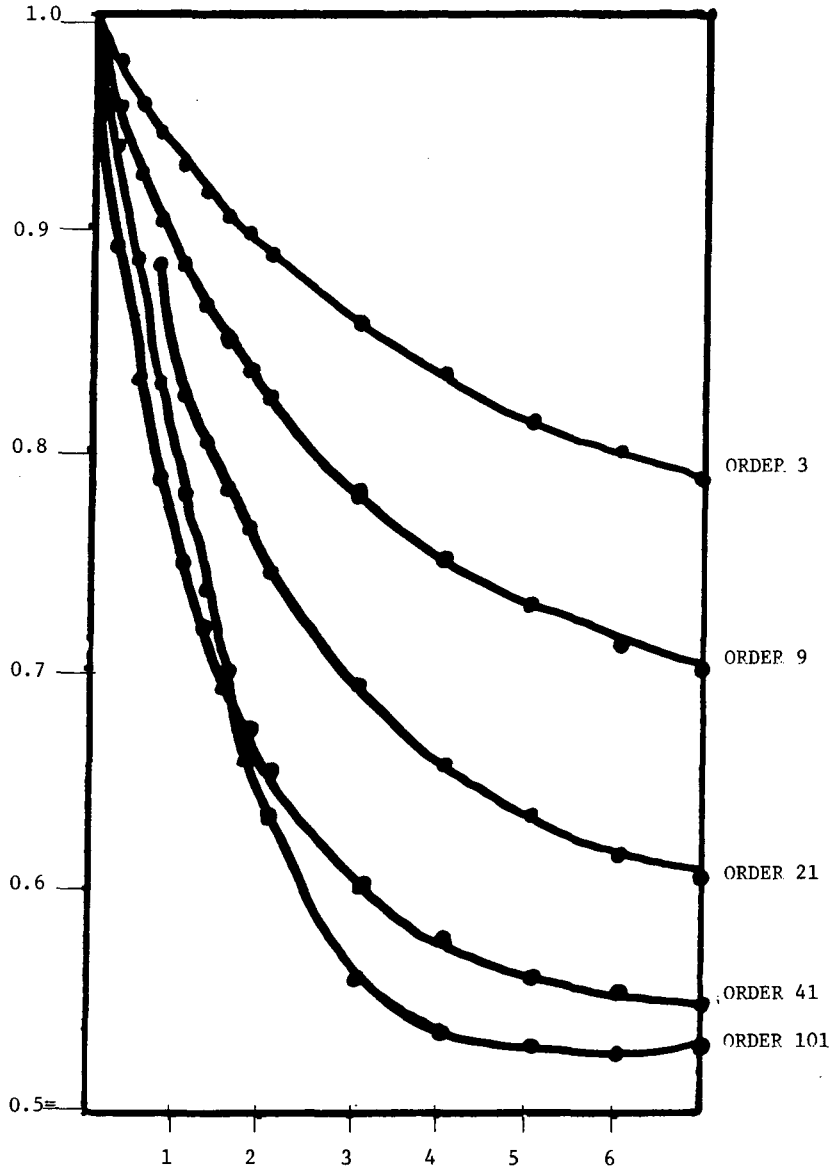

FIGURE 5.2: Storage Utilization Vs. Percent Growth In Initially Compact B-Trees Of Initial Size 10000 Keys.

6. Conclusions

The above results may be interpreted in several ways. If we ignore the demand for high order imposed by practical considerations in data base implementation, we note that M-trees may be preferable to their B-tree counterparts at low orders. In particular, an M-tree of order 3 is preferable to a 2-3 tree (a B-tree of order 3). (Order 3 M-trees outperform 2-3 trees in storage and insertion, and are roughly the same in search costs.) Although compact B-trees are most robust at order 3, it is at this order that these structures provide the least storage advantage over M-trees (99% vs. 83% storage utilization). Furthermore, the relative search cost advantage disappears quite quickly as insertions are made into the compact 2-3 tree.

On the other hand, we may take into account the requirement for high order when these structures are used in secondary storage media. At high orders (21,41,101), we see that M-trees are unusable because of their disasterous storage utilization. It is also clear that unless the structure is very non-volatile, the use of compact B-trees provides no genuine improvement in storage utilization. Even if the structure is growing very slowly, the use of compact B-trees doubles insertion cost and at high order provides little search cost advantage over that of a random B-tree.

We conclude that compact B-trees should only be used with great caution, that unless worst-case concerns override average case performance, M-trees are preferable to 2,3-trees and order 5 B-trees, and that otherwise, of the structures considered here, ordinary B-trees with no compaction are the preferred structures. In addition, we find the strong average case performance of low order M-trees intriguing, and intend to explore the possibility of employing some type of balancing scheme to extend this type of performance to higher orders.

References

1. Adelson-Velskii, G.M., Landis, E.M.: An information organization algorithm. DANSSSR, 146, (1962) 263-266.

2. Bayer, R. and McCreight, E. Organization and maintenance of large ordered indexes. Acta Informatica 1,3 (1972), 173-189.

3. Comer, D. The ubiquitous B-tree. Computing Surveys 11,2 (1978), 121-138

4. Foster, C.C.: Information storage and retrieval using AVL trees. Proc. ACM 20th National Conf. (1965) 192-205.

5. Knuth, D.E. The Art of Computer Programming, Vol. 3. Addison-Wesley, Reading, Mass. (1973)

6. Rosenberg, A.L. and Snyder, L. Minimal comparison 2,3 trees. SIAM J. Computing 7,4 (1978), 465-480

7. Rosenberg, A.L. and Snyder, L. Time- and space-optimality in B-trees. ACM Transactions on Database Systems 6,1 (1981), 174-193.

8. Sussenguth, E.H., Jr.: The use of tree structures for processing files. Comm. ACM 6,5 (1963)

9. Wirth, N. Algorithms + Data Structures = Programs, Prentice-Hall, Englewood Cliffs, N.J. (1976)

10. Yao, A.C. On random 2,3 trees. Acta Informatica 9,3 (1978), 159-170.