A Generalization of AVL Trees

Caxton C. Foster
University of Massachusetts

A generalization of AVL trees is proposed in which imbalances up to Δ are permitted, where Δ is a small integer. An experiment is performed to compare these trees with standard AVL trees and with balanced trees on the basis of mean retrieval time, of amount of restructuring expected, and on the worst case of retrieval time. It is shown that, by permitting imbalances of up to five units, the retrieval time is increased a small amount while the amount of restructuring required is decreased by a factor of ten.

A few theoretical results are derived, including the correction of an earlier paper, and are duly compared with the experimental data. Reasonably good correspondence is found.

Key Words and Phrases: AVL trees, balanced trees, information storage and retrieval

CR Categories: 3.7, 3.72, 4.49, 5.31

Author's address: Graduate Research Center, Computer and Information Sciences Department, University of Massachusetts, Amherst, MA 01002.

AVL trees are quite well known [1–9]. They are characterized by the fact that at any node in the tree the two subtrees dependent from that node have maximum heights that, at most, differ by one from each other. They appear to offer a good compromise between the ease of construction of free trees on the one hand and balanced trees on the other.

In this paper, we generalize the concept of AVL trees by permitting the imbalance at nodes of the tree to take on values larger than one while still not permitting unrestricted growth. Since in various applications different ratios of postings to searches will occur, it seemed useful to explore the spectrum of trees with constrained imbalances so that a designer could select the one best fitted to his needs.

We will discuss five aspects of these generalized AVL trees: the very worst retrieval time possible under a constraint; the average weight of the tree containing the very worst retrieval time; the average number of times the tree must be "restructured" per item posted; the number of nodes that must be updated; and the average weight of items in trees constructed according to these rules.

The Construction of an AVL Tree

The method of constructing AVL trees has been explained fully elsewhere [1, 2, 5, 6].

When a new node is added to an AVL tree, we must move back up the tree along the ancestral path of the new node testing each ancestor for one of three possible conditions: (1) we have already reached the root of the tree and therefore terminate the retracing procedure: (2) we reach an ancestor whose shorter subtree was lengthened by the addition of the new node and terminate the retracing because the effects of the new node are thereby "adsorbed"; and (3) we reach an ancestor whose longer subtree was lengthened by the addition of the new node and whose longer subtree now exceeds its shorter subtree by more than the permissible amount (Δ). Here we must rebalance the tree.

It is interesting to note that exactly the same mechanism of restructuring that works for standard AVL trees also applies for larger Δ's. While there might be some temptation to try to improve on this technique for large Δ by "balancing as closely as possible" when a critical node is reached, this is not necessary and, indeed, may not even be desirable. The standard methods of restructuring just "adsorb" the extra length generated by the new node. Further restructur-

ing (as in Figures 1 and 2) may decrease the maximum path length below what it was before the new node was added so that more retracing and possibly more rebalancing would have to be undertaken.

Mintrees

A mintree of height h is that AVL tree of the specified height constructed from the fewest possible number of nodes. It is constructed by taking one item as the root and, on one side of the root, hanging a mintree of height $h - 1$ and, on the other, a mintree of height $h - 1 - \Delta$. Now suppose we have an arbitrary AVL tree of N nodes. Its most remote item can require no more than H probes to retrieve, where H is the height of the longest mintree that can be constructed with N nodes or less. $N(\Delta, h)$ (the minimum number of nodes required to build an AVL tree with maximum imbalance Δ and a height h) is given by the recursion equation

$$N(\Delta, h) = 1 + N(\Delta, h - 1) + N(\Delta, h - 1 - \Delta). \quad (1)$$

Given N, the size of the tree we are dealing with, and Δ (the permitted imbalance), we can readily find the worst possible retrieval time by iterating eq. (1) until its value exceeds N. The value of h for which $N(\Delta, h) \leq N < N(\Delta, h + 1)$ is then the "worst possible" number of probes required to retrieve an item from an AVL tree of this size and imbalance. Figure 3 shows the worst case for trees of 1000 items with various permitted imbalances.

The Average Weight of Mintrees

We define the weight of a tree to be the sum over all levels of the number of elements m_i on the ith level times i: $W = \sum_i m_i i$.

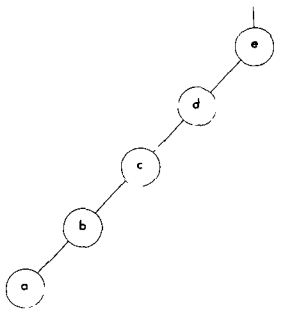
The average weight of an item in a tree is then W/N, where N is the number of elements in the tree. The average weight is equivalent to the mean number of probes required to retrieve an item selected at random from the tree.

Despite the fact that K.C. Tan [7] has shown that intuition is not to be relied on in dealing with AVL trees, I was tempted to believe that mintrees might be the heaviest (largest weight) trees constructable with N nodes. My referee was "kind" enough to demonstrate that while a mintree of 20 nodes had a height of six and a total weight of 76, it was easily possible to construct an AVL tree of 20 nodes with a height of five and a weight of 77.

Nonetheless, we can derive the weight of a mintree analytically, and that is rare enough in the area of AVL trees to be worth reproducing here.

In a mintree constructed as above, each item on both of the subtrees is one unit farther from the starting

Fig. 1. Permitted imbalance of 3 is exceeded at node -e- by the addition of the new node -a-.



point than it was before. Thus we have

$$W(\Delta, h) = 1 + W(\Delta, h - 1) + N(\Delta, h - 1) + W(\Delta, h - 1 - \Delta) + N(\Delta, h - 1 - \Delta). \quad (3)$$

Collecting terms involving the number of elements, we have

$$W(\Delta, h) = N(\Delta, h) + W(\Delta, h - 1) + W(\Delta, h - 1 - \Delta). \quad (4)$$

The Average Weight of Constrained Trees

Although some users will be interested in the worst case discussed above, it is likely that a more useful number is the average number of probes required to retrieve an item from an average tree constructed according to the rules.

As previously noted, analytical results are not easy to come by in this field. Consequently, we turned to simulation and built a number of trees under different constraints and measured their average weights. Table I shows the results of building 2900 independent trees, 100 of each kind. Weights were measured after the last item was entered. Figure 4 shows the average weight of an item from a tree of 1000 items as a function of the permitted imbalance. An examination of the data shows that the increase in weight of an AVL tree over a balanced tree[1] is approximately independent of the size of the tree and roughly equal to 0.28Δ. There *is* a tendency, more pronounced for larger Δ, for the difference to increase with tree size. Since a $\Delta = 5$ is already half the height of a balanced tree of 2000 items, this is not surprising.

The Number of Rebalances

In a balanced tree the number of items that must be moved (on the average) when a new item is added to the tree is a strong function of how densely occupied

[1] For example, the differences between columns 0 and 1 are a less than linear function of N. Doubling N increases the difference by about 5 percent.

Fig. 2. One conceivable way of restructuring the tree of Figure 1, which generates a deficit in height.

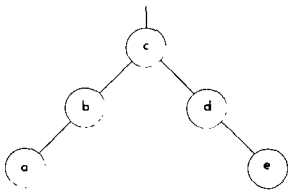


Table I. Average weight of an item selected at random from an AVL tree of size N with permitted imbalance of Δ. [Note that $\Delta = 0$ corresponds to a balanced tree and is included for comparison purposes.]

$N \backslash \Delta$	0	1	2	3	4	5	∞
125	5.96	6.26	6.42	6.63	6.85	7.06	7.92
250	6.96	7.26	7.44	7.66	7.89	8.17	9.38
500	7.96	8.27	8.46	8.70	8.98	9.25	10.79
1000	8.96	9.29	9.49	9.76	10.05	10.37	12.17
2000	9.96	10.31	10.53	10.82	11.13	11.43	—

Fig. 3. The maximum height of an AVL tree with 1000 nodes as a function of the permitted imbalance.

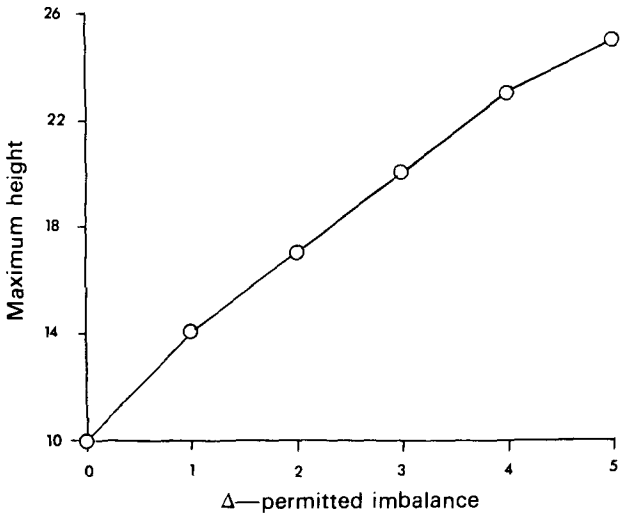
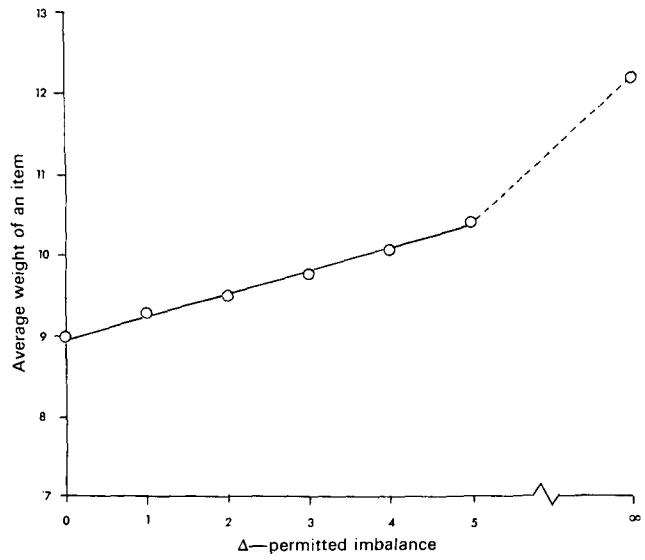


Table II. The average number of restructurings necessary per item added to build an AVL tree of size N with permitted imbalance of Δ.

$N \backslash \Delta$	1	2	3	4	5
125	.458	.201	.123	.072	.044
250	.458	.203	.120	.071	.042
500	.464	.209	.120	.074	.045
1000	.469	.209	.120	.073	.045
2000	.469	.208	.121	.074	.047

Fig. 4. The average weight of an item as a function of the permitted imbalance for trees with 1000 nodes.



the bottommost layer of the tree happens to be. If that layer is almost full, then as much as all the items in the tree may have to be moved to add one new item. In an AVL tree, at most three items have to be moved when an excessive imbalance is discovered. Equally likely only two items must be moved. Moreover, it is not always necessary to so restructure the tree. For $\Delta = 1$, we can derive a theoretical estimate of how often we should expect to have to restructure the tree.

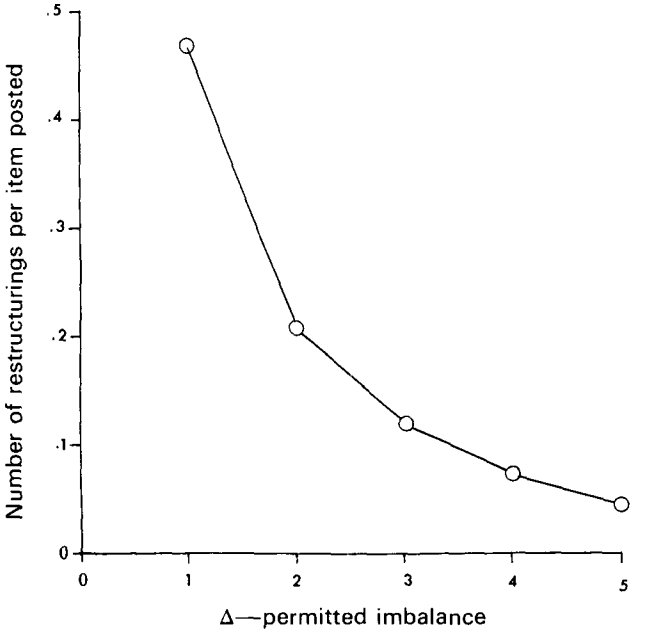
As we climb up the tree retracing the path to the newly posted item, we, with very high probability, eventually come to a node that was already unbalanced to the left or to the right. We have no reason for favoring one possibility over the other in a tree built of random numbers, so we assume them equal. This is called the critical node. It would seem reasonable to assume further that the direction from which we approach this critical node is independent of the direction in which the critical node is imbalanced. If this is the case, then with equal probability the posting of the new item has lengthened the longer leg of the critical node or the shorter leg. If the former, the node becomes excessively unbalanced and a restructuring must take place; if the latter, an unbalanced node is converted to a balanced node and the retracing procedure can terminate with the extra length adsorbed. This argument says that restructurings should occur 0.5 of the time after posting a new item, but an examination of column 1 of Table II shows that in our experiment they occurred only 0.46 to 0.47 of the time.

The most reasonable explanation for this result,

of course, is that in climbing back up the tree we have occasionally reached the root and hence terminated before reaching a critical node. If this is the case, we should see an increase in the number of restructurings as the tree gets larger and the root farther away. Indeed, this tendency is present in the data, although a bit small to erect complicated theories on.

Much more important than these minor deviations from theory is the fact that the number of restructurings required drops dramatically as the permitted imbalance increases. Figure 5 shows this for trees of size 1000. To a first approximation, each increase of Δ by one cuts the number of restructurings required by a factor of two.

Fig. 5. The number of restructurings per item posted for trees with 1000 nodes.



Climbing Back up the Tree

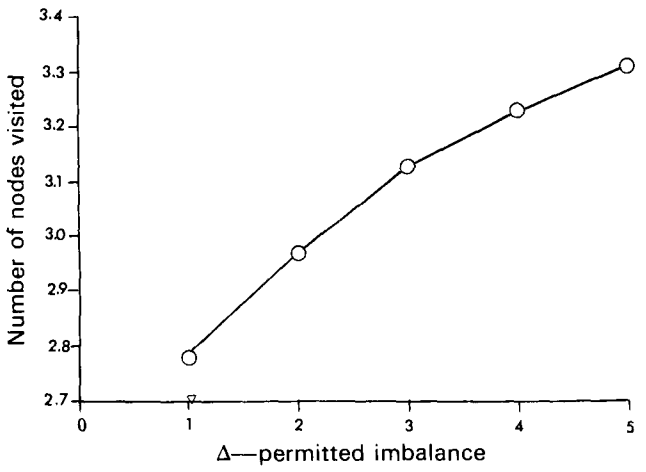
After adding an item to a tree, we must examine its ancestors in turn (father, grandfather, etc.) until we reach a critical node or reach the root. For standard AVL trees, it has been suggested that the names of the nodes visited and the direction taken on the way down be kept on a push-down list and that this list be used to find the ancestors rather than doubly linking all the nodes of the tree together. This idea can be immediately applied to generalized AVL trees. For standard AVL trees, the balance factor $(-1, 0, +1)$ is stored at each node and is used to decide the fate of the climbing process. One is able to do this because, after restructuring, the balance factors of the restructured nodes are easy to compute. But for generalized AVL trees it is easier to keep a "height" associated with each node. The height may be computed as one plus the larger of the height of its left subtree and its right subtree.

$$H(X) = 1 + \max\{H(L(X)), \quad H(R(X))\}. \tag{5}$$

Since H is a relatively small integer, only a few bits are necessary to hold it; and it can quickly be computed for each ancestor and each node affected by a restructuring.

Part of the overhead involved in keeping a tree AVL is concerned with climbing back up the tree. Consequently, in our experiment we kept track of how many ancestors were visited, including the critical node, in the retracing process. These observations are shown in Table III.

Fig. 6. Number of nodes visited (including the critical node) after posting a new item for trees of size 1000.



It is possible to derive the expected number of nodes visited for standard AVL trees, and an attempt was made in my earlier paper [2]. But Knuth [6] has pointed out an error in the derivation. The following argument is his.

Let P be the probability (assumed independent) that a node in a standard AVL tree is balanced. The probability of visiting exactly k nodes *before* reaching the critical node is

$$P^k(1 - P), \tag{6}$$

Table III. The average number of nodes visited in updating an AVL tree of size N with permitted imbalance of Δ after adding a new item to the tree.

$N\backslash\Delta$	1	2	3	4	5
125	2.66	2.84	3.00	3.08	3.12
250	2.72	2.91	3.06	3.15	3.23
500	2.76	2.96	3.11	3.21	3.27
1000	2.79	2.98	3.13	3.23	3.31
2000	2.80	2.99	3.14	3.25	3.33

and the expected value of k is given by

$$\hat{k} = P/(1 - P). \tag{7}$$

Then the number of nodes visited *including* the critical node is

$$V = 1 + \hat{k} = 1/(1 - P). \tag{8}$$

To find the value of P, we follow Knuth.

The new item just added will be balanced. The retracing procedure will unbalance $\hat{k}$ nodes that were previously balanced. With probability one half, we exit after creating a balanced node out of an unbalanced (in the opposite direction to the approach) node; and with probability one half, we exit after restructuring, which generates two balanced nodes. Expressing this as an equation, we have

$$P = 1 - \hat{k} + \tfrac{1}{2}(1) + \tfrac{1}{2}(2). \tag{9}$$

Substituting for $\hat{k}$ the value of eq. (7) and solving for P, we obtain

$$P = (9 - \sqrt{41})/4 \simeq .65. \tag{10}$$

From this we have that

$$V = 2.86. \tag{11}$$

From Table III we again find the experimental results to be somewhat different than the theoretical. The lowness of these results could possibly be explained by the nonzero probability of backing up to the root. On the other hand, taking the smallest observed value (2.66), we would, by the inverse of the above argument, predict a value of $P = .625$, not at all far from the theoretical number. Figure 6 shows the observed results for trees of 1000 items.

Conclusions

We have looked at a generalization of the idea of AVL trees, which permits imbalances of up to Δ before requiring a restructuring of the tree. We have found that, by allowing Δ to be as large as four, we increase the number of probes required to retrieve an item by one, but we decrease the number of restructurings from one every other item to only one for every 13.7 items posted. Approximately one half an additional item must be examined during the retracing procedure.

If we expect to build a tree once and then use it for years and years, it is obvious that balanced trees with their minimal retrieval time are the best bet. If we expect to build a tree and seldom, if ever, use it, then free trees with minimum overhead are the best solution. Generalized AVL trees offer a system designer several alternate choices which will allow him to trade off access time for ease of construction.

Our simulation programs were written in FORTRAN for a CDC-3600. Since it seemed as if the time required by this program to carry out the simulation would reflect more the skill of the programmer and the compiler writer than the true structure of the AVL trees, we have reported nodes visited and rebalances rather than times.

A reader interested in converting these to actual times need only write and calibrate a few kernels for his own machine.

Received June 1972; revised January 1973

References

1. Adel'son-Vel'skiy, G.M., and Landis, Ye.M. An algorithm for the organization of information. *Doklady Akad. Nauk USSR Moscos 16*, No. 2 (1962), 263–266. Also available in translation as U.S. Dept. of Commerce OTS, JPRS 17,137, Washington, D.C., and as NASA Document N63-11777.
2. Foster, C.C. Information storage and retrieval using AVL trees. Proc. ACM 20th Nat. Conf. 1965, pp. 192–205.
3. Foster, C.C. *A study of AVL trees.* GER-12158, Goodyear Aerospace Corp., Akron, Ohio, Apr. 1965.
4. Martin, W.A., and Ness, D.N. Optimizing binary trees grown with a sorting algorithm. *Comm. ACM 15*, 2 (Feb. 1972), 88–93.
5. Stone, H., *Introduction to Computer Organization and Data Structures*, McGraw-Hill, New York, 1972.
6. Knuth, D. *The Art of Computer Programming, vol. 3, Fundamental Algorithms.* Sec. 6.2.3, Addison-Wesley, Reading, Mass. (in press).
7. Tan, K.C. On Foster's information storage and retrieval using AVL trees. *Comm. ACM 15*, 9 (Sept. 1972), 843.
8. Reingold, E.M. Notes on AVL Trees. Rep. #441, Dep. of Comput. Sci., U. of Illinois, Urbana, 1971.
9. Knott, G.D. A balanced tree storage and retrieval algorithm. Proc. Symp. on Inf. Stor. and Retri., U. of Maryland, College Park, Md., Apr. 1971, pp. 175–196.