PREPRINT. International Journal of Computer Matehmathics, 38:37-48, 1990.

Maintaining α-balanced Trees by Partial Rebuilding

Arne Andersson*
Department of Computer Science
Lund University
Box 118
S-221 00 Lund
Sweden

Abstract

The balance criterion defining the class of α-balanced trees states that the ratio between the shortest and longest paths from a node to a leaf be at least α. We show that a straight-forward use of partial rebuilding for maintenance of α-balanced trees requires an amortized cost of $\Omega(\sqrt{n})$ per update. By slight modifications of the maintenance algorithms the cost can be reduced to $O(\log n)$ for any value of α, $0 < \alpha < 1$.

KEY WORDS α-balanced trees, partial rebuilding, search trees.
CR CATEGORIES: E.1, F.2, I.1.2.

1 Introduction

In his thesis Olivie [9] introduced a class of binary search trees, which he calls *α-balanced trees*, or *αBB-trees*. Let h(v) denote the length for the longest path from a node v to a leaf and let s(v) denote the length of the shortest path. We give a formal definition of α-balanced trees below.

Definition 1 *A binary tree is α-balanced if the following is true for each node v in the tree:*

$$\frac{s(v)}{h(v)} \geq \alpha, \quad h(v) \geq \frac{1}{1-\alpha} \tag{1}$$

$$h(v) - s(v) \leq 1, \quad h(v) < \frac{1}{1-\alpha} \tag{2}$$

*This work was partially supported by a grant from the National Swedish Board for Technical Development.

where α is a constant, $0 < \alpha < 1$.

In other words, the ratio between the shortest and longest paths from a node to a leaf is at least α. An αBB-tree allows efficient element location, since its height is restricted to be

$$\mathrm{h}(T) \leq \left\lceil \frac{1}{\alpha} \log n \right\rceil \tag{3}$$

Olivie gave updating algorithms using local restructuring with good worst case behaviour for $\alpha = 1/2$ (half-balanced trees) [10]. He also gave a general insertion algorithm for the case $0 < \alpha \leq 1/2$ and a deletion algorithm for $\alpha = 1/3$. However, the question whether there are efficient algorithms for all possible values of α was left open. According to Olivie, for $1/2 < \alpha < 1$ the design of efficient algorithms seems to be difficult.

Later, Overmars [12] showed that by using *weak deletions* and *global rebuilding* an αBB-tree may be maintained at an amortized cost of $O(\log n)$ per update for $\alpha \leq 1/2$. Furthermore, a data structure consisting of two αBB-trees may be maintained at $O(\log n)$ cost in the worst case.

As pointed out by Olivie, there is an intersting many-to-one correspondence between α-balanced trees ($\alpha = 1/2$) and symmetric binary B-trees, or SBB-trees [3], such that each SBB-tree corresponds to one $1/2$-balanced tree and each $1/2$-balanced tree corresponds to a number of SBB-trees. In the same way the generalized symmetric binary B-trees, or SBB(k)-trees [2], satisfies the balance criterion of α-balanced trees. Thus, using the algorithms for SBB(k)-trees efficient algorithms for $1/2 \leq \alpha < 1$ can be formulated [11]. However, in this case the tree is actually not viewed as α-balanced, and a tree which satisfies the α-balance criterion may require restructuring in order to satisfy the SBB(k)-criterion.

In this paper we analyze the behaviour of αBB-trees when maintained by *partial rebuilding*. The method of partial rebuilding was introduced by Overmars and van Leeuwen [13] and it is a simple method to maintain balanced tree structures. The idea behind the method is brutal but powerful; each time a given balance criterion is violated at a node v the subtree rooted at v is rebuilt to achieve perfect balance. The simplicity of the method makes it useful in situations where local restructuring operations, such as rotations, does not work. Such a case occurs when maintaining k-d-trees [4]. A scheme for partial rebuilding of weight-balanced trees [8] with logarithmic amortized bounds was first introduced by Overmars and van Leeuwen [12, 13]. They showed that the low amortized cost for maintaining such trees, $O(\log n)$ per update, is due to the large number of updates required to make an initially well-balanced subtree become unbalanced. Other classes of trees maintained by partial rebuilding can be found in [1].

Making a careful analysis, we show that a straight-forward use of partial rebuilding for maintenance of α-balanced trees requires an amortized cost of $\Omega(\sqrt{n})$. Thus, the cost per update for this class of trees is not competitive to the cost for weight-balanced trees. However, the amortized cost when making only insertions is $O(\log n)$ for any value of α. This fact may be used to apply the method of weak deletions on α-balanced trees, resulting in an amortized cost of $O(\log n)$ per update. Thus, we have shown that there exist efficient algorithms for $1/2 < \alpha < 1$, solving a problem left open by Olivie.

2 Terminology

We assume that a binary search tree contains one element in each internal node and that the leaves are empty. The height of a tree T equals the number of edges on the longest path dow the tree and it is denoted $\mathrm{h}(T)$. By the *weight* of T, denoted $|T|$, we mean the number of leaves in the tree. Observe that n elements are stored in a tree of weight $n+1$. Since we do not distinguish between nodes and subtrees, the subtree v refers to the subtree rooted at the node v in T. The height and weight of v are denoted $\mathrm{h}(v)$ and $|v|$, respectively. The number of insertions and deletions made in the subtree v since the last time v was involved in a rebuilding are denoted $\mathrm{ins}(v)$ and $\mathrm{del}(v)$, respectively, and the total number of updates (deletions and insertions) is denoted $\mathrm{upd}(v)$. In order to measure changes in height, weight, and number of updates in a subtree, we denote the subtree v immediately after the latest time it was rebuilt as v_o. The notation $\mathrm{h}(v_o)$, $|v_o|$ and so forth follows from above.

In our analysis we assume that rebuilding of a subtree v takes linear time, examples of linear algorithms for balancing trees are found in [5, 6, 7, 14].

3 Analysis of Partial Rebuilding

The analysis of α-balanced trees maintained by partial rebuilding is somewhat more complicated than the analysis of weight-balanced trees. Our analysis of the update cost is based on a computation of how many updates are required to make an initially well-balanced tree become out of balance.

In Lemmas 1 we give a relation between the height and weight of a subtree. This relation is used in Lemma 2 to analyze the number of insertions required to increase the height of the tree by a certain amount. In a similar way the number of deletions required to decrease the length of the shortest path is given in Lemmas 3 and 4. Using these results we calculate the number of updates required to make a node unbalanced in Lemma 5. Finally, in Theorem 1 we show that the maintenance cost for α-balanced trees when maintained by partial rebuilding is $\Omega(sqrtn)$ per update.

Lemma 1 *For any possible value of $h(v)$ there is a node v which satisfies the balance criterion of an α-balanced tree such that*

$$|v| \leq \frac{2^{(\mathrm{h}(v)+1)\alpha} - 1}{2^\alpha - 1} \tag{4}$$

Proof: From Definition (1) follows that the smallest value of $\mathrm{s}(v)$ is given by

$$\mathrm{s}(v) = \lceil \alpha \mathrm{h}(v) \rceil \tag{5}$$

Let p denote the smallest child v may have. We have that

$$\mathrm{s}(p) = \mathrm{s}(v) - 1 \tag{6}$$

If p has its smallest weight, then

$$|p| = 2^{\mathrm{s}(p)} = 2^{\mathrm{s}(v)-1} = 2^{\lceil \alpha \mathrm{h}(v) - 1 \rceil} \tag{7}$$

We get the minimal value of $|v|$ by adding the weights of the smallest possible subtrees on a path down to a leaf. If we choose the longest path we get

$$\begin{aligned} |v| &= 1 + \sum_{i=1}^{\mathrm{h}(v)} 2^{\lceil \alpha i - 1 \rceil} \\ &\leq \sum_{i=0}^{\mathrm{h}(v)} 2^{\alpha i} \\ &= \frac{2^{\alpha(\mathrm{h}(v)+1)} - 1}{2^\alpha - 1} \end{aligned} \tag{8}$$

The constant 1 is added since addition of the weights of subtrees along a path down a binary tree will leave one of the leaves uncounted. The proof follows from equation (8). □

Lemma 2 *Let T_o be a perfectly balanced binary tree. Then the smallest number of insertions required to increase the height of T_o by Δh is given by*

$$\mathrm{ins}(T) < \frac{2^{\frac{\Delta h+1}{1-\alpha}\alpha} - 1}{2^\alpha - 1} \tag{9}$$

Proof: In order to increase the height of T by Δh using a minimal number of insertions we add nodes to a subtree v to achieve

$$\mathrm{h}(v) = \mathrm{h}(v_o) + \Delta h \tag{10}$$

We choose v as the smallest possible subtree in T with a height which may be increased by Δh without increasing the length of its shortest path. Without loss of generality we assume that v_o is a complete binary tree, which implies that

$$\mathrm{s}(v_o) = \mathrm{h}(v_o) = \mathrm{s}(v) = \lceil \alpha \mathrm{h}(v) \rceil \tag{11}$$

and

$$|v_o| = 2^{\lceil \alpha \mathrm{h}(v) \rceil} \geq 2^{\alpha \mathrm{h}(v)} \tag{12}$$

Together with Lemma 1 this gives that

$$\begin{aligned} \mathrm{ins}(v) &= |v| - |v_o| \\ &\leq \frac{2^{\alpha(\mathrm{h}(v)+1)} - 1}{2^\alpha - 1} - 2^{\alpha \mathrm{h}(v)} \\ &= \frac{\left(2^{\alpha(\mathrm{h}(v)+1)} - 1 - 2^{\alpha(\mathrm{h}(v)+1)} + 2^{\alpha \mathrm{h}(v)}\right)}{2^\alpha - 1} \\ &= \frac{2^{\alpha \mathrm{h}(v)} - 1}{2^\alpha - 1} \end{aligned} \tag{13}$$

Since the weight of v is at a minimum we have that

$$\mathrm{h}(v) - \Delta h = \mathrm{s}(v) = \lceil \alpha \mathrm{h}(v) \rceil < \alpha \mathrm{h}(v) + 1 \tag{14}$$

which implies that

$$\mathrm{h}(v) < \frac{\Delta h + 1}{1 - \alpha} \tag{15}$$

From Eqs. (13) and (15) follows that

$$\mathrm{ins}(v) < \frac{2^{\frac{\Delta h+1}{1-\alpha}\alpha} - 1}{2^\alpha - 1} \tag{16}$$

which completes the proof. $\square$

The number of deletions required to decrease the height of a node is given in Lemmas 3 - 4 below. The computations are similar (but not identical) to the computations used in the proofs of Lemmas 1 - 2 above.

Lemma 3 *Let v be a node in an α-balanced tree. Then it is possible that*

$$|v| > 1 + \sum_{i=1}^{\mathrm{s}(v)} 2^{i/\alpha - 2} = \frac{1}{4} \cdot \frac{2^{(\mathrm{s}(v)+1)/\alpha} - 1}{2^{1/\alpha} - 1} \tag{17}$$

Proof: From Definition (1) follows that the largest possible value of $\mathrm{h}(v)$ is given by

$$\mathrm{h}(v) = \left\lfloor \frac{\mathrm{s}(v)}{\alpha} \right\rfloor > \frac{\mathrm{s}(v)}{\alpha} - 1 \tag{18}$$

Let p denote the largest child v may have. We have that

$$\mathrm{h}(p) = \mathrm{h}(v) - 1 \tag{19}$$

If p is as large as possible then

$$|p| = 2^{\mathrm{h}(p)} = 2^{\mathrm{h}(v)-1} \tag{20}$$

Together with Definition (1) this gives that

$$|p| = 2^{\mathrm{h}(v)-1} > 2^{\lfloor \mathrm{s}(v)/\alpha \rfloor - 2} \tag{21}$$

We get the maximal value of $|v|$ by adding the weights of the largest possible subtrees on a path down to a leaf. If we choose the shortest path we get

$$|v| > 1 + \sum_{i=1}^{\mathrm{s}(v)} 2^{i/\alpha - 2} = \frac{1}{4} \cdot \frac{2^{(\mathrm{s}(v)+1)/\alpha} - 1}{2^{1/\alpha} - 1} + \frac{3}{4} \tag{22}$$

which completes the proof. $\square$

Lemma 4 *Let T_o be a perfectly balanced binary tree. Then the smallest number of deletions to decrease the shortest path of T_o by Δs is given by*

$$\mathrm{del}(T) < \frac{(3 \cdot 2^{1/\alpha} - 4) \cdot 2^{\frac{\Delta s + 1}{1-\alpha}}}{(2^{1/\alpha} - 1) \cdot 4} + O(1) \tag{23}$$

Proof: In order to decrease the shortest path of T by Δs using a minimal number of deletions we remove nodes from a subtree v to achieve

$$\mathrm{s}(v) = \mathrm{s}(v_o) - \Delta s \tag{24}$$

We choose v to be the smallest possible subtree in T in which the shortest path may be decreased by Δs without decreasing its height. Without loss of generality we assume that v is a complete binary tree, which implies that

$$\mathrm{h}(v_o) = \mathrm{s}(v_o) = \mathrm{h}(v) = \left\lfloor \frac{\mathrm{s}(v)}{\alpha} \right\rfloor \tag{25}$$

and

$$|v_o| = 2^{\lfloor \mathrm{s}(v)/\alpha \rfloor} \leq 2^{\mathrm{s}(v)/\alpha} \tag{26}$$

This together with Lemma 3 gives that

$$\begin{aligned}
\mathrm{del}(v) &= |v_o| - |v| \\
&\leq 2^{\mathrm{s}(v)/\alpha} - \frac{1}{4} \cdot \frac{2^{(\mathrm{s}(v)+1)/\alpha} - 1}{2^{1/\alpha} - 1} - \frac{1}{4} \\
&< \frac{4 \cdot 2^{(\mathrm{s}(v)+1)/\alpha} - 4 \cdot 2^{\mathrm{s}(v)/\alpha} - 2^{(\mathrm{s}(v)+1)/\alpha} + 1}{(2^{1/\alpha} - 1) \cdot 4} \\
&= \frac{3 \cdot 2^{(\mathrm{s}(v)+1)/\alpha} - 4 \cdot 2^{\mathrm{s}(v)/\alpha}}{(2^{1/\alpha} - 1) \cdot 4} + \frac{1}{2^{1/\alpha} - 1} \\
&= \frac{(3 \cdot 2^{1/\alpha} - 4) \cdot 2^{\mathrm{s}(v)/\alpha}}{(2^{1/\alpha} - 1) \cdot 4} + \frac{1}{2^{1/\alpha} - 1}
\end{aligned} \tag{27}$$

Since v has maximal weight we have that

$$\mathrm{s}(v) + \Delta s = \mathrm{h}(v) = \left\lfloor \frac{\mathrm{s}(v)}{\alpha} \right\rfloor > \frac{\mathrm{s}(v)}{\alpha} - 1 \tag{28}$$

which implies that

$$\mathrm{s}(v) < \frac{\Delta s + 1}{1 - \alpha} \alpha \tag{29}$$

From Eqs. (27) and (29) follows

$$\begin{aligned}
\mathrm{del}(v) &< \frac{(3 \cdot 2^{1/\alpha} - 4) \cdot 2^{\mathrm{s}(v)/\alpha}}{(2^{1/\alpha} - 1) \cdot 4} + \frac{1}{2^{1/\alpha} - 1} \\
&< \frac{(3 \cdot 2^{1/\alpha} - 4) \cdot 2^{\frac{\Delta s + 1}{1 - \alpha}}}{(2^{1/\alpha} - 1) \cdot 4} + \frac{1}{2^{1/\alpha} - 1}
\end{aligned} \tag{30}$$

which completes the proof. □

Lemma 5 *In order to violate the balance criterion at a node v in an αBB-tree, $O(\sqrt{|v|})$ updates are sufficient.*

Proof: Without loss of generality we may assume that v_o is a complete binary tree. We make insertions to change the shape of v in a way that

$$\Delta\mathrm{h}(v) = \frac{1-\alpha}{2\alpha}\mathrm{h}(v_o) \tag{31}$$

and deletions to achieve

$$\Delta\mathrm{s}(v) = \frac{1-\alpha}{2}\mathrm{h}(v_o) + 1 \tag{32}$$

These updates cause a violation of the balance criterion at v since

$$\begin{aligned}\frac{\mathrm{s}(v)}{\mathrm{h}(v)} &= \frac{\mathrm{h}(v_o) - \Delta\mathrm{s}(v)}{\mathrm{h}(v_o) + \Delta(\mathrm{h}(v)}\\ &= \frac{\mathrm{h}(v_o) - \frac{1-\alpha}{2}\mathrm{h}(v_o)}{\mathrm{h}(v_o) + \frac{1-\alpha}{2\alpha}\mathrm{h}(v_o)}\\ &= \frac{(1+\alpha)\mathrm{h}(v_o) - 2}{(1+\alpha)\mathrm{h}(v_o)}\alpha\\ &< \alpha\end{aligned} \tag{33}$$

The number of updates to achieve this violation is given by

$$\begin{aligned}\mathrm{upd}(v) &= \mathrm{ins}(v) + \mathrm{del}(v)\\ &< \frac{2^{\frac{\Delta\mathrm{h}(v)+1}{1-\alpha}\alpha} - 1}{2^\alpha - 1} + \frac{(3\cdot 2^{1/\alpha} - 4)\cdot 2^{\frac{\Delta\mathrm{s}(v)+1}{1-\alpha}}}{(2^{1/\alpha} - 1)\cdot 4} + O(1)\\ &= \frac{2^{\frac{\frac{1-\alpha}{2\alpha}\mathrm{h}(v_o)+1}{1-\alpha}\alpha} - 1}{2^\alpha - 1} + \frac{(3\cdot 2^{1/\alpha} - 4)\cdot 2^{\frac{\frac{1-\alpha}{2}\mathrm{h}(v_o)+1+1}{1-\alpha}}}{(2^{1/\alpha} - 1)\cdot 4} + O(1)\\ &= \frac{2^{(\mathrm{h}(v_o)+1)/2} - 1}{2^\alpha - 1} + \frac{(3\cdot 2^{1/\alpha} - 4)\cdot 2^{(\mathrm{h}(v_o)+2)/2}}{(2^{1/\alpha} - 1)\cdot 4} + O(1)\\ &= O(2^{\mathrm{h}(v_o)/2}) + O(2^{\mathrm{h}(v_o)/2}) + O(1)\\ &= O(\sqrt{|v_o|})\\ &= O(\sqrt{|v|})\end{aligned} \tag{34}$$

which completes the proof. □

Theorem 1 *Maintaining an α-balanced tree by partial rebuilding results in an amortized cost of $\Omega(\sqrt{n})$ per update.*

Proof: From Lemma 5 follows that we can force a rebuilding of the entire tree after every $\Theta(\sqrt{n})$th update. This gives an amortized cost of $\Omega(\sqrt{n})$ per update, which completes the proof. $\square$

Thus, we conclude that a straight-forward implementation of partial rebuilding does not lead to efficient maintenance algorithms for α-balanced trees.

4 Weak Deletions

The result of Theorem 1 does not allow us to maintain an α-balanced tree efficiently by partial rebuilding. Fortunately, a better complexity may be obtained by a modification of the maintenance algorithms. We can show that when only insertions are made the tree can be maintained by partial rebuilding at a logarithmic amortized cost. Together with a different strategy for deletion we can exploit this fact to obtain a logarithmic amortized bound for updates.

Lemma 6 *Let v be a node in an α-balanced tree. Then*

$$|v| \geq \frac{1}{2} \cdot \frac{2^{(\mathrm{h}(v)+1)\alpha} - 1}{2^{\alpha} - 1} \tag{35}$$

Proof: The proof is similar to the proof of Lemma 1. The smallest possible weight of v is given by

$$\begin{aligned} |v| &= 1 + \sum_{i=1}^{\mathrm{h}(v)} 2^{\lceil \alpha i - 1 \rceil} \\ &\geq \sum_{i=0}^{\mathrm{h}(v)} 2^{\alpha i - 1} \\ &= \frac{1}{2} \cdot \frac{2^{\alpha(\mathrm{h}(v)+1)} - 1}{2^{\alpha} - 1} \end{aligned} \tag{36}$$

which completes the proof. $\square$

Lemma 7 *Let v be a node in an α-balanced tree where no deletions are made. When v has become out of balance, the following is true:*

$$\mathrm{ins}(v) > (2^{1-\alpha} - 1)|v| + O(1) \tag{37}$$

Proof: When v is out of balance

$$\frac{\mathrm{s}(v)}{\mathrm{h}(v)} < \alpha \tag{38}$$

Without loss of generality we may assume that v_o was a complete binary tree. This implies that

$$\mathrm{h}(v_o) = \mathrm{s}(v_o) < \mathrm{s}(v) < \alpha \mathrm{h}(v) \tag{39}$$

which implies that

$$|v_o| < 2^{\alpha \mathrm{h}(v)} \tag{40}$$

This together with Lemma 6 gives that

$$\begin{aligned} \frac{|v_o|}{|v + \frac{1}{2^\alpha - 1}|} &< \frac{2^{\alpha \mathrm{h}(v)}}{\frac{1}{2} \cdot \frac{2^{(\mathrm{h}(v)+1)\alpha}}{2^\alpha - 1}} \\ &= 2 - 2^{1-\alpha} \end{aligned} \tag{41}$$

This gives that

$$\begin{aligned} \mathrm{ins}(v) &= |v| - |v_o| \\ &= (1 - \frac{|v_o|}{|v|})|v| \\ &= (1 - \frac{|v_o|}{|v| + \frac{1}{2^\alpha - 1}})|v| + O(1) \\ &> (1 - (2 - 2^{1-\alpha}))\left(|v| + \frac{1}{2^\alpha - 1}\right) + O(1) \\ &= (2^{1-\alpha} - 1)|v| + O(1) \end{aligned} \tag{42}$$

which completes the proof. □

Lemma 8 *An α-balanced tree in which no deletions are made can be maintained by partial rebuilding at an amortized cost of $O(\log n)$ per insertion.*

Proof: From Lemma 7 we know that $\Omega(|v|)$ insertions are required to make a node v become unbalanced. Thus, the amortized cost per insertion below v is $O(1)$. Since each insertion is made below $O(\log n)$ nodes, the total amortized cost per update is $O(\log n)$. □

Given the result of Lemma 8 we can use the method of global rebuilding and weak deletions [12] to maintain a set in an αBB-tree during both insertion and deletion, for any value of α, $0 < \alpha < 1$.

The idea is simple: Making deletions by marking the element as deleted instead of removing it, we can perform $\Theta(n)$ deletions in the tree while the height of the tree is still logarithmic. After $\Theta(n)$ deletions the entire tree is rebuilt and deleted elements are removed. The amortized cost of this global rebuilding is $O(1)$ per update.

Theorem 2 *For $0 < \alpha < 1$ an ordered set may be maintained in an α-balanced tree at an amortized cost of $O(\log n)$ per update.*

Proof: For each node in the tree we add a boolean variable, telling whether the stored element is present or deleted. We maintain the set by the following algorithm:

Insertion: If the element is found in the tree we mark it as present. Otherwise we insert a new node. Whenever the balance criterion becomes violated at a node, we make a partial rebuilding there.

Deletion: We mark the element as deleted. When $\frac{n}{2}$ deletions have been made, we remove all elements marked as deleted and rebuild the entire tree to perfect balance.

Although the weight of the tree is larger than $n+1$, it still have a height of $O(\log n)$, since the number of deleted elements is at most $\frac{n}{2}$.

The logarithmic amortized cost for insertion follows from Lemma 8 and the logarithmic amortized cost for deletion follows from the fact that rebuilding is made only when $\Theta(n)$ deletions have been made. □

4.1 A Modified Shape

To keep deleted elements left in the tree may seem like an unnecessary waste of space. This problem may be circumvented by introducing yet another strategy for deletion. Instead of marking an element as deleted, we remove it.

In order to avoid rebuildings generated by deletions we just omit updating the balance information (i.e. the value of $s(v)$) during a deletion. The skewness caused by this method is taken care of by rebuilding the entire tree when $\Theta(n)$ deletions have been made. This tree is simpler and more natural to implement than the one described in Theorem 2 above. However, due to the fact that the balance information is not updated during deletions, the tree may contain nodes which violate the balance criterion. Therefore, the tree is not strictly α-balanced and we say that we have a modified version of α-balanced trees. In spite of those violations the height of the tree remains logarithmic, although it might be slightly larger than the height of the original αBB-tree.

Theorem 3 *A modified α-balanced tree can be maintained at an amortized cost of $O(\log n)$ per update without keeping deleted elements in the tree.*

Proof: We modify the tree in the following way:

- The stored value of $s(v)$ is only changed when the node v is involved in a rebuilding.
- The entire tree is rebuilt after every $\frac{n}{2}$ deletions.

The logarithmic height is guaranteed by the periodic rebuilding and the fact that a deletion can not increase the height of the tree.

From the fact that the stored values of $s(v)$ are only changed during a rebuilding follows that the number of insertions between two rebuildings of a subtree is the same as if no deletions were made. Thus, from Lemma 8 we know that the cost per insertion is $O(\log n)$.

The logarithmic amortized cost for deletion follows from the fact that rebuilding is made only when $\Theta(n)$ deletions have been made. $\square$

5 Conclusions

Contrary to the analysis of weight balanced trees [12, 13], the analysis of α balanced trees maintained by partial rebuilding is nontrivial. The fact that the amortized cost is $\Omega(\sqrt{n})$ is not intuitively clear.

By improving the cost to $O(\log n)$ we have presented simple and efficient maintenance algorithms that can be used for an arbitrary value of

α, $0 < \alpha < 1$. The fact that we use partial rebuilding implies that the α-balance criterion may be used to maintain tree structures where local restructuring does not work, such as k-d-trees.

References

[1] A. Andersson. Improving partial rebuilding by using simple balance criteria. In *Proc. Workshop on Algorithms and Data Structures, Springer Verlag*, pages 393–402, 1989.

[2] A. Andersson, Ch. Icking, R. Klein, and Th. Ottmann. Binary search trees of almost optimal height. *Acta Informatica*, 28:165–178, 1990.

[3] R. Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, 1972.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] H. Chang and S. S. Iynegar. Efficient algorithms to globally balance a binary search tree. *Communications of the ACM*, 27(7):695–702, 1984.

[6] A. C. Day. Balancing a binary tree. *Computer Journal*, 19(4):360–361, 1976.

[7] W. A. Martin and D. N. Ness. Optimizing binary trees grown with a sorting algorithm. *Communications of the ACM*, 15(2):88–93, 1972.

[8] J. Nievergelt and E. M. Reingold. Binary trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.

[9] H. J. Olivie. *A Study of Balanced Binary Trees and Balanced One-Two-Trees.* Ph.D. Thesis, Dept of Mathematics, University of Antwerp, 1980.

[10] H. J. Olivie. A new class of balanced search trees: Half-balanced binary search trees. *R. A. I. R. O. Informatique Theoretique*, 16:51–71, 1982.

[11] Th. Ottmann and D. Wood. Updating binary trees with constant linkage cost. In *Proc. Scandinavian Workshop on Algorithm Theory*, 1990.

[12] M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer Verlag, 1983. ISBN 3-540-12330-X.

[13] M. H. Overmars and J. van Leeuwen. Dynamic multi-dimensional data structures based on quad- and k-d trees. *Acta Informatica*, 17:267–285, 1982.

[14] Q. F. Stout and B. L. Warren. Tree rebalancing in optimal time and space. *Communications of the ACM*, 29(9):902–908, 1986.