PREPRINT. In Proc. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 382, Springer Verlag, pages 393-402, 1989.

# Improving Partial Rebuilding by Using Simple Balance Criteria

Arne Andersson
Department of Computer Science
Lund University
Lund, Sweden

**Abstract**

Some new classes of balanced trees, defined by very simple balance criteria, are introduced. Those trees can be maintained by partial rebuilding at lower update cost than previously used weight-balanced trees. The used balance criteria also allow us to maintain a balanced tree without any balance information stored in the nodes.

## 1 Introduction

Partial rebuilding is a general method to maintain balanced tree structures introduced by Overmars and van Leeuwen [9, 10]. The idea is brutal but powerful; each time a given balance criterion is violated at a node $v$ we rebuild the subtree rooted at $v$ to perfect balance. The simplicity of this method makes it useful in applications where other balancing methods do not work. The worst case cost for updates is high since rebuilding of a large subtree is expensive. However, Overmars and van Leeuwen showed that weight-balanced trees can be maintained at low amortized cost, $O(\log n)$ per update. This is due to the fact that in a weight-balanced tree a large number of updates is required to make an initially well-balanced subtree become unbalanced.

In this paper we improve the method of partial rebuilding by presenting some new classes of balanced trees which can be maintained at lower cost than weight-balanced trees. In Section 2 we show how to replace the weight-balancing criterion by height-balancing. The new balance criterion gives a lower average cost for updates and also allow us to give a a better upper bound on the amortized cost than for weight-balancing. In section 3 we present *general balanced trees*. For this class of trees the only requirement is that the height is logarithmic. Those trees can be represented without any balance information stored in the nodes.

In addition to the improvements obtained in time and space requirement the presented trees are interesting due to their simple and natural definitions.

We use $T$ to denote an extended binary tree and $v$ to denote a node in that tree. The number of leaves in the tree $T$, also called the size of $T$, is denoted $|T|$ or $n$. The size of the subtree rooted at $v$, or the size of $v$, is denoted $|v|$. The node difference of $v$, $\mathrm{diff}(v)$, is the difference in size between $v$'s two children, and $\mathrm{height}(v)$ denotes the number of edges on the longest path from $v$ down to a leaf. The number of updates made below $v$ since the last time $v$ was involved in a rebuilding are denoted $\mathrm{upd}(v)$. In the analysis below we assume that a rebuilding of a subtree takes linear time, $R|v|$ where $v$ is the root of the subtree. Partial rebuilding is also applyable in cases when a rebuilding at $v$ takes longer time, for example when maintaining quad trees [4]. The improvements we make are also valid in those cases.

## 1.1 Weight-Balancing

The idea of weight-balancing is to control the height of a tree by limiting the quotient between the sizes of the two subtrees of each node. Trees maintained in this way is called trees of bounded balance or BB($\alpha$)-trees and was first presented by Nievergelt and Reingold [7]. The maintenance of BB($\alpha$)-trees by partial rebuilding is briefly analyzed in [9] We give a short analysis here. An alternative definition is also given (equation (3) below).

Associated with the tree is a constant $\alpha$, $0 < \alpha < 1/2$. Each node $v$ has to fulfill

$$\frac{|v\text{'s smallest subtree}|}{|v|} \geq \alpha \tag{1}$$

From the balance criterion we can compute the maximum height of a BB($\alpha$)-tree.

$$\mathrm{height}(T) \leq \frac{\log n}{\log \frac{1}{1-\alpha}} \tag{2}$$

As shown in [7] the balance criterion (1) is meaningless for $1/3 < \alpha < 1/2$. This problem can be avoided by using the following slightly different balance criterion:

$$\frac{|v\text{'s smallest subtree}| + 1}{|v| + 1} \geq \alpha \tag{3}$$

The criterion (3) allows us to let $\alpha$ take any value between 1/2 and 1. The differences in the two balance criteria are of low significance and will result in a slightly higher tree. It is not hard to show that the criterion (3) gives

$$\text{height}(T) \leq \frac{\log n}{\log \frac{1}{1-\alpha}} + 1 \tag{4}$$

Given a constant $u$, $u > 1$ a maximum height of $u \log |v| + 1$ for each node $v$ is achieved by setting

$$\alpha = 1 - 2^{-1/u} \tag{5}$$

Using the fact that

$$|v| - \text{diff}(v) = 2|v\text{'s smallest subtree}| \tag{6}$$

the balance criterion (3) can be rewritten as

$$\text{diff}(v) + 1 \leq (1 - 2\alpha(|v| + 1) \tag{7}$$

Combining (5) and (7) gives that when a node $v$ becomes unbalanced the following is true

$$\begin{aligned} \text{diff}(v) &> (1 - 2(1 - 2^{-1/u}))(|v| + 1) - 1 \\ &= (2^{1-1/u} - 1)|v| + 2^{1-1/u} \end{aligned} \tag{8}$$

The subtree rooted at $v$ is perfectly balanced immediately after a rebuilding at $v$ and thus $\text{diff}(v)$ is 0 or 1. To simplify the analysis we assume that $\text{diff}(v) = 0$. From this follows that

$$\text{upd}(v) \geq \text{diff}(v) \tag{9}$$

Since a partial rebuilding at $v$ requires $R|v|$ time the amortized cost for rebuilding at $v$ is $\frac{R}{2^{1-1/u}-1}$ per update below $v$. This gives an amortized cost of $\frac{R}{2^{1-1/u}-1}(\text{height}(T) - 1)$ per update. Since $\text{height}(T) \leq u \log n + 1$ we get an amortized cost of

$$\frac{Ru \log n}{2^{1-1/u} - 1} \tag{10}$$

where $u$ is an arbitrary constant larger than 1.

## 2 Trees of Bounded Height

In the previous section we saw that a maximum height of $u \log |v| + 1$ for each node $v$ can be achieved for a BB($\alpha$)-tree by choosing $\alpha$ to be $2^{1-1/u} - 1$. As we will show in this section, we can remove the weight-balancing and use the more natural "heuristic" balance criterion that $\text{height}(v) \leq u \log |v| + 1$. The obtained class of balanced trees, called trees of bounded height, is defined below.

**Definition 1** *A binary tree is of bounded height if*

$$\text{height}(v) \leq u \log |v| + 1 \tag{11}$$

*for each node $v$ where $u$ is a constant, $u > 1$.*

The constant 1 in the definition is added to allow $u$ to take any value greater than one. The algorithms to maintain a BH($u$)-tree are the same as for BB($\alpha$)-trees; when the balance criterion is violated at a node we make a rebuilding. To decide when a node needs to be rebuilt we store two integers in each node telling its size and height.

In Lemma 1 below we show that when a node $v$ needs rebalancing diff($v$) will be at least the same as when a rebalancing is required in a BB($\alpha$)-tree with the same maximum height.

**Lemma 1** *Given a tree of bounded height where the node $v$ has become unbalanced. Then*

$$\text{diff}(v) > (2^{1-1/u} - 1)|v| \tag{12}$$

**Proof:** Since $v$ is unbalanced we have

$$\text{height}(v) > u \log |v| + 1 \tag{13}$$

Let $v_1$ be $v$'s highest child. We have

$$\text{height}(v) = \text{height}(v_1) + 1 \tag{14}$$

Combining equations (13) and (14) with the fact that $v_1$ satisfies the definition (11) gives

$$u \log |v_1| + 1 > u \log |v|$$

$$2^{1/u} |v_1| > |v|$$

$$|v_1| > 2^{-1/u}|v| \tag{15}$$

This implies that

$$\begin{aligned}
\text{diff}(v) &> |v_1| - (|v| - |v_1|) \\
&= 2|v_1| - |v| \\
&= (2^{1-1/u} - 1)|v|
\end{aligned} \tag{16}$$

which completes the proof. □

Comparing the result of Lemma 1 with equation (8) we see that when a node gets too high it will also be out of weight-balance. Thus both average and amortized cost will be at least as good as for trees of bounded balance. In fact, we can prove a better average behaviour. This is based on Lemma 2 below.

**Lemma 2** *Given a weight-balanced tree with $\alpha = 1 - 2^{-1/u}$ where the node $v$ has become unbalanced Then it is possible that*

$$\text{height}(v) \leq u \log |v| + 1 \tag{17}$$

**Proof:** We give a counterexample. Clearly it is possible to construct a subtree $v$ where both subtrees are perfectly balanced but $v$ is out of weight-balance. The height of such a tree is $\leq u \log |v| + 2$ which is less than $u \log |v| + 1$, small trees excepted. □

**Theorem 1** *The average cost for rebuilding is less in a BH(u)-tree than in a BB($\alpha$)-tree with $\alpha = 2^{1-1/u} - 1$.*

**Proof:** From Lemma 1 we know that when a node in a BH($u$)-tree requires rebalanciong a rebalancing is required in the corresponding BB($\alpha$)-tree. From Lemma 2 we know that the opposite is not true. Thus rebuilding is required less often in BH($u$)-trees, which completes the proof. □

Trees of bounded height do not only offer a better average case behaviour than trees of bounded balance. It is also possible to give a better upper bound on the amortized cost per update than what is given for trees of bounded balance. This improvement is based on the fact that when a node becomes unbalanced there has to be a certain unbalance at the lower levels of the tree. This unbalance can be expressed as the sum of all node differences in the subtree $v$, denoted totaldiff($v$).

**Lemma 3** *Given a tree of bounded height where the node $v$ has become unbalanced. Then*

$$\text{totaldiff}(v) > \frac{2 - 2^{1/u}}{2^{1/u} - 1}(|v| - 1) \tag{18}$$

**Proof:** Let $v_d$ be a node on one of $v$'s longest paths such that

$$\text{height}(v) = \text{height}(v_d) + d \tag{19}$$

Combining equations (13) and (19) with the fact that $v_d$ satisfies the definition (11) gives

$$\begin{aligned} u \log |v_d| + d &> u \log |v| \\ 2^{d/u}|v_d| &> |v| \\ |v_d| &> 2^{-d/u}|v| \end{aligned} \tag{20}$$

Assume that each node $v_d$ have its smallest possible size. From (20) we get that

$$\begin{aligned} \text{diff}(v_d) &= |v_{d+1}| - (|v_d| - |v_{d+1}|) = 2|v_{d+1}| - |v_d| \\ &> (2 \cdot 2^{-(d+1)/u} - 2^{-d/u})|v| = (2^{1-1/u} - 1)2^{-d/u}|v| \end{aligned} \tag{21}$$

In this case, summing the node differences on $v's$ longest path we get a total sum of

$$\text{totaldiff}(v) >= (2^{1-1/u} - 1) \sum_{d=0}^{u \log |v|+1} 2^{-d/u}|v|$$

$$= (2^{1-1/u} - 1)\frac{1 - 2^{-(u \log |v|)/u}}{1 - 2^{-1/u}}|v| = (2^{1-1/u} - 1)\frac{1 - 1/|v|}{1 - 2^{-1/u}}|v|$$

$$= \frac{2 - 2^{1/u}}{2^{1/u} - 1}(|v| - 1) \tag{22}$$

Of course, the nodes $v_d$ do not necessary have minimal size, but there is no other configuration where totaldiff($v$) is lower than what is given in (22). To see this we show that no modification of the subtree rooted at $v$ results in a lower value of totaldiff($v$). Assume that we try to change the tree to decrease the potential at a node $v_d$. Since its largest subtree ($v_{d+1}$) has minimal size we can only decrease the potential at $v_d$ by adding nodes to its smallest subtree. Those nodes can not be taken from $v_d$'s largest subtree and thus we have to take nodes from the smaller subtree of one of $v_d$'s ancestors (but not above $v$ since $|v|$ then will change). This will result in an increase of that ancestor's node potential and the sum of the potential of the two nodes will be constant. Furthermore, at each node between $v_d$ and the affected ancestor the potential will increase. Thus the total sum of node potentials below $v$ will not decrease by any possible modification of the tree. From this follows that the sum of node differences is at least the one given in equation (22) which completes the proof. $\square$

**Theorem 2** *A BH(u)-tree can be maintained at an amortized cost of*

$$\frac{2^{1/u}-1}{2-2^{1/u}} Ru \log n + R \tag{23}$$

*per update.*

**Proof:** To prove the amortized cost for updates we define a potential function $\Phi(T)$ which depends on the shape of the tree $T$. The potential function is chosen in such a way that the decrease of potential caused by a rebuilding covers the cost for the rebuilding. The amortized cost for an update equals the increase in potential caused by the update. We choose the following potential function:

$$\Phi(T) = \frac{2^{1/u}-1}{2-2^{1/u}} R \cdot \mathrm{totaldiff}(T) + R(\mathrm{upd}(T) - \mathrm{reb}(T)) \tag{24}$$

where $\mathrm{reb}(T)$ denotes the total number of rebuildings made in $T$ since the last total rebuilding of $T$. Clearly, $\mathrm{upd}(T) \geq \mathrm{reb}(T)$, which implies that $\Phi(T)$ is always nonnegative. During an update the value of $\mathrm{totaldiff}(T)$ can be changed by at most the number of ancestors to the inserted/deleted node The number of ancestors is $\leq u \log n$ which gives the cost for an update

$$\Delta\Phi(T) \leq \frac{2^{1/u}-1}{2-2^{1/u}} Ru \log n + R \tag{25}$$

Combining (24) with the result of Lemma 3 gives that a rebuilding at the node $v$ results in a decrease of $T$'s potential such that

$$\Delta\Phi(T) \geq \frac{2^{1/u}-1}{2-2^{1/u}} R \cdot \mathrm{totaldiff}(v) + R$$

$$\geq \frac{2^{1/u}-1}{2-2^{1/u}} R \cdot \frac{2-2^{1/u}}{2^{1/u}-1} (|v|-1) + R = R|v| \tag{26}$$

The decrease in potential covers the cost for rebalancing and the proof is completed. □

If we compare the given costs for height-balancing with the cost for weight-balancing we get

$$\frac{\frac{2^{1/u}-1}{2-2^{1/u}}Ru\log n + R}{\frac{Ru\log n}{2^{1-1/u}-1}} \approx \frac{\frac{2^{1/u}-1}{2-2^{1/u}}}{\frac{1}{2^{1-1/u}-1}} = 1 - 2^{-1/u} < \frac{1}{2} \qquad (27)$$

Thus the upper bound on the rebuilding cost for partial rebuilding has been reduced by a factor of 2. Note that although we give a better upper bound we do not prove that the amortized cost actually is lower.

The simplicity of the balance criterion makes trees of bounded height a natural class which contains most other classes of balanced trees. This is the case for AVL-trees [1], BB($\alpha$)-trees [7], SBB-trees [3], and $\alpha$BB-trees [8]. For all those classes there exist a constant $u$ such that $\text{height}(v) \leq u \log |v| + 1$ for each node $v$ in the tree.

# 3 General Balanced Trees

Allthough the BH($u$)-trees are a general class of trees this class does not contain all balanced trees, for example not the $k$-neighbour trees [6]. The simpliest possible balance criterion we can have is to allow the tree to take any shape as long as its height is $O(\log n)$. Such a balance criterion results in a "superclass" containing all other classes of balanced trees. (Note that the splay tree [11] is not balanced in this sense, since its worst case height is $O(n)$.) In this section we show that such a superclass may be maintained by partial rebuilding. This class also has the advantage of requiring no balance information to be stored in the nodes. We call the structure a *general balanced tree* or GB($u$)-tree.

**Definition 2** *Given a constant $u$, $u > 1$, a tree $T$ is a GB($u$)-tree if*

$$\text{height}(T) \leq u \log |T| + 1 \qquad (28)$$

The maintenance algorithms for GB($u$)-trees differ from the algorithms for BB($\alpha$)-trees and BH($u$)-trees. Since there is no local balance criterion to be fulfilled in each node the balance criterion is checked only at the root. As shown by Baer and Schwab [2], if we rebalance the entire tree each time it becomes too high the amortized cost will be $O(n)$ per updating operation. To achieve a better result we make the following observation:

**Lemma 4** *Let $T$ be a binary tree in which there is a path longer than $u\log|T|+1$ and let $v$ be the lowest node on this path such that*

$$\text{height}(v) > u\log|v|+1 \tag{29}$$

*Then*

$$\text{totaldiff}(v) \geq \frac{2-2^{1/u}}{2^{1/u}-1}(|v|-1) \tag{30}$$

**Proof:** First, it is clear that on each long path there exist a node which satisfies (29), since at least the root satisfies this criterion. Since $v$ is the lowest node satisfying the criterion we know that each node $v_d$ on $v$'s longest path satisfies $\text{height}(v_d) \leq u\log|v_d|+1$. The rest of the proof is similar to the proof of Lemma 3. □

The result of Lemma 4 allows us to design efficient algorithms for the maintenance of general balanced trees. As soon as the tree becomes too high we make rebuildings at each path which is too long. On each path the rebuilding is made at the lowest node $v$ satisfying condition (29). In order to locate the paths and nodes where to make rebuilding we use the same data structure as for trees of bounded height, that is, in each node we store the height and size.

The amortized cost for general balanced tree is practically the same as for trees of bounded height. The only extra cost we have is the cost to locate the nodes where to make rebuilding. The location of a node requires logarithmic time. Since at least one update has been made below a node which is to be rebalanced the amortized cost per update for locating this node is $O(\log n)$. If we omit this extra cost, which anyway is small compared to the rebuilding cost, we can use the same analysis as for BH($u$)-trees. Therefore, we have

**Theorem 3** *The class of general balanced trees can be maintained with an amortized rebuilding cost of $\frac{2^{1/u}-1}{2-2^{1/u}}Ru\log n+R$ per update.*

**Proof:** The proof follows from the discussion above. □

## 3.1 Removal of the Balance Information

The balance information stored in the nodes of a general balanced tree is used to find nodes where rebuilding can be made at low amortized cost. However, when an insertion is made only one path — the path down to the inserted node — may become too long. Thus during insertion we do not need any stored information to find the path where rebuilding is required. The only thing we have to do is to find the lowest node $v$ on the path which satisfies condition (29). This location can also be made without using any stored information, which is shown below. The skewness caused by deletions can be dealt with by rebuilding the entire tree with periodic intervals. Altogether we obtain a balanced tree with no balance information stored in the nodes. The only balance information we need is two global integers containing the size of $T$ and the number of deletions made since the last rebuilding of the entire tree, here denoted $\mathrm{del}(T)$.

**Theorem 4** *There is a binary search tree which requires no stored balance information, except two global integers, with the following costs:*

*search* $O(\log n)$ *worst case*

*update* $O(\log n)$ *amortized*

**Proof:** We let the two global integers contain the values of $|T|$ and $\mathrm{del}(T)$. Given a constant $u > 1$ we use the following updating algorithms:

*Insertion:* If the depth of the inserted leaf exceeds $u \log |T| + 1$ we back up along the path making explicit examination of the subtrees of the visited nodes until a node $v$, $\mathrm{height}(v) > u \log |v| + 1$ is found. We make a partial rebuilding at that node.

*Deletion:* If $del(T) \geq |T|$ we make a rebuilding of the entire tree.

Let $|T|_{max}$ be the maximal value of $|T|$ since the latest rebuilding of $T$. Since we rebuild $T$ when $\mathrm{del}(T) \geq |T|$ we know that $|T|_{max} \leq 2|T| = 2n$. During insertion we make a rebuilding when a path longer than $u \log n + 1$ is found. This implies that the height of $T$ is never more than $u \log |T|_{max} + 1$. Thus

$$\mathrm{height}(T) \leq u \log |T|_{max} + 1 \leq u \log(2n) + 1 = u \log n + u + 1 \qquad (31)$$

which gives the logarithmic search time.

Rebuildings are made on two occasions; when $\mathrm{del}(T) = |T|$ and when the tree gets too high during insertion. The amortized cost for the first type of rebuilding is constant for a deletion. In the second case the rebuilding algorithm contains two steps: location of the node $v$ and a rebuilding at $v$. To find the node $v$ by explicit examination of its subtrees starting at a leaf takes $\Theta(|v|)$ time. Thus the time for locating $v$ can be included in the restructuring cost without affecting the asymthotic bound. The rest of the analysis is the same as for trees of bounded height, which gives a logarithmic amortized cost per update. This completes the proof. $\square$

The trees in the proof of Theorem 4 are not identical to GB-trees since their maximum height is $u \log n + u + 1$ instead of $u \log n + 1$. This difference, which is of low practical significanse, is due to the difference in the deletion algorithm. To differ between these two variants of general balanced trees we call the class without balance information $\mathrm{GB}_0(u)$-trees.

**Example:** Figure 1 shows a $\mathrm{GB}_0(1.2)$-tree where the node $p$ has just been inserted. The path to the inserted node is too long since $\mathrm{height}(T) = 6 > 1.2 \log 15 + 1$. We have to make a partial rebuilding at one of the nodes on that path. By making explicit examination of the subtrees bottom-up we find that the node $v$ satisfies condition (29) since $\mathrm{height}(v) = 5 > 1.2 \log 10 + 1$. A partial rebuilding is made at $v$ and the insertion is completed. $\square$

## 4 Conclusions

We have presented some new classes of balanced trees which may be efficiently maintained by partial rebuilding. The balance criteria used are evident and the trees can be maintained at a lower cost than weight-balanced trees. In this way we have improved the time performance of partial rebuilding. In our analysis we assumed that a rebuilding of a subtree can be made in linear time but our improvements are valid also in other cases.

The general balanced trees are particularly interesting since they contain all other classes of balanced trees and can be maintained with no balance information stored in the nodes. They also use a natural and attractive maintenance strategy; we do not make any rebalancing until it is really needed. In the literature some attempts have been made to have a global

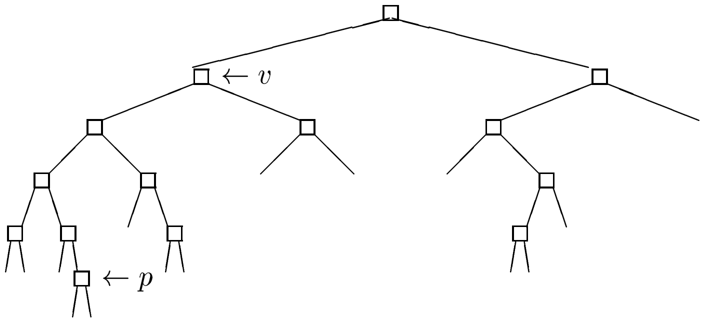


Figure 1: *A $GB_0$(1.2)-tree which requires rebuilding. Leaves are marked as empty edges.*

balance criterion and make restructurings only when this criterion is violated [2]. The amortized cost for those methods are $O(n)$ per operation. Here we have shown that by choosing carefully where to make rebuilding we can maintain a tree with only a global balance criterion at low amortized cost.

The pslay tree, presented by Sleator and Tarjan [11], does not require any balance information stored in the nodes. However, this tree is not balanced in the sense that the height is guaranteed to be $O(\log n)$. The logarithmic cost for searching in a splay tree is amortized while we here obtain logarithmic worst case bounds. As shown by Brown [5] the explicit balance information may in some classes of balanced trees be eliminated by coding the information by the location of empty pointers. However, in this case we still store the information although it is stored implicitly.

Besides being generally applyable on various data structures the method of partial rebuilding has the advantage of being simple to implement. This together with the fact that the height of the tree can be kept arbitrary close to optimal makes the method useful also for the dictionary problem, especially when comparisons are expensive.

## References


[1] G. M. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. *Dokladi Akademia Nauk SSSR*, 146(2):1259–1262, 1962.

[2] J-L. Baer and B. Schwab. A comparison of tree-balancing algorithms. *Communications of the ACM*, 20(5):322–330, 1977.

[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] M. R. Brown. Addendum to "A storage scheme for height-balanced trees". *Information Processing Letters*, 8(3):154–156, 1979.

[6] H. A. Maurer, Th. Ottmann, and H. W. Six. Implementing dictionaries using binary trees of very small height. *Information Processing Letters*, 5(1):11–14, 1976.

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

[8] 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.

[9] 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.

[10] 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.

[11] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. In *Proc. ACM Symp. Theory of Computing*, 1983.