Chromatic Search Trees Revisited

Institut für Informatik — Report 91

Sabine Hanke
Institut für Informatik, Universität Freiburg
Am Flughafen 17, 79110 Freiburg, Germany
Email: hanke@informatik.uni-freiburg.de.

June 12, 1997

Abstract

Relaxed balancing of search trees was introduced with the aim of speeding up the updates and allowing a high degree of concurrency. In a relaxed version of a search tree the rebalancing operations are uncoupled from the updates and may be delayed. Using local transformations the rebalancing can be performed concurrently with updates and search operations.

In this paper we revisit the rebalancing of chromatic trees, a relaxed version of red-black trees. During the rebalancing of a chromatic tree it can occur that the rebalancing operations change the search structure of the tree in order to settle rebalancing requests from the history although the tree is already a balanced red-black tree. We propose how to perform the rebalancing task in such a way that each performed structural change indeed correspond to a real imbalance situation in the tree. The number of rebalancing operations needed to restore the red-black tree balance condition is still $O(i+d)$ pointer changes and $O((i+d) \cdot \log(n+i))$ colour changes, if i insertions and d deletions are applied to a chromatic tree of size n that initially fulfills the balance condition of red-black trees.

1 Introduction

A *dictionary* is a data structure for storing a set of data that supports the operations search, insert, and delete. Insert and delete operations are called the *update operations*. Standard implementations of dictionaries are balanced binary search trees like red-black trees [4, 15] or AVL trees [1], which allow to carry out search and update operations in $O(\log n)$ time, if n is the number of keys stored in the tree. In order to assure the logarithmic bound, each update operation is followed immediately by a sequence of rebalancing steps, which restore the respective balance condition.

If a dictionary is implemented in a concurrent enviroment, there must be a way to prevent simultaneous reading and writing of the same parts of the data structure. A common strategy for concurrency control is to *lock* the critical parts, so that other processes have no or only restricted access. For efficiency, only a small part of the tree should be locked at a time.

In a concurrent implementation of the conventional bottom-up rebalancing methods for search trees the whole search path from the root to the leaf beeing updated must be locked. This leads to the idea of uncoupling the updating from the rebalancing. Instead of requiring that the balance condition is restored immediately after each update operation the actual rebalancing transformations can be arbitrarily delayed and interleaved. This kind of rebalancing is called *relaxed balancing*.

The idea of uncoupling the update operations from the rebalancing operations was first suggested in [4] for red-black trees. The first actual solution for relaxed balancing is presented by Kessels [6]. The trees in Kessel's solution are AVL-trees and the only allowed updates are insertions. An extention to the general case where deletions are also allowed is presented by Nurmi et al. [13].

This solution is analyzed in [7], and it is shown that for each update operation in a tree with maximum size n, $O(\log n)$ new rebalancing operations are needed. Relaxed balancing for B-trees is introduced in [13] and further analyzed in [8]. Nurmi and Soisalon-Soininen [11, 12] propose a relaxed version of red-black trees which they call a *chromatic tree*. The proposal of [12] is analyzed in [3] by Boyar and Larsen, and a more efficient set of rebalancing transformations is defined. It is shown that the number of rebalancing operations per update is $O(\log(n+i))$, if i insertions are performed on a tree which initially contains n leaves. Furthermore, in [2] Boyar et al. prove for a slightly modified set of rebalancing operations that only an amortized constant amount of rebalancing is necessary after an update in a chromatic tree. A general method of how to make known rebalancing algorithms relaxed in an efficient way is proposed in [5]. With an example of red-black trees it is shown that essentially the same set of rebalancing transformations as used in the strict case can also be used for the relaxed case, and that the number of needed rebalancing operations known from the strict balancing scheme carry over to relaxed balancing. In [14] this method is applied to stratified trees. Furthermore, in [9] the technique of [5] is described in an abstract way for the class of all search trees with local bottom-up rebalancing. In [16] Soisalon-Soininen and Widmayer propose a relaxed version of AVL-trees, which is based on the standard balancing transformations for AVL trees. This solution is analyzed in [10]. It is shown that the number of rebalancing operations needed to restore the AVL balance condition is bounded by $O(M\log(M+N))$, where M is the number of updates to an AVL tree of initial size N.

This recent balancing scheme for AVL trees [16, 10] differs from all previous solutions for relaxed balancing in the important aspect that never a rotation is performed if the underlying search tree happens to fulfill the balance condition before any rebalancing has been done. All previous solutions work in such a way that balance conflicts from the history are remembered and gradually resolved, where only local parts of the tree have influence on the rebalancing transformation to be carried out. Therefore, during the rebalancing of a chromatic tree [2, 12] for example it may happen that rotations are performed in order to resolve balance conflicts from the history although the tree is already perfectly balanced.

In this paper we propose how to rebalance a chromatic tree so that each performed structural change indeed correspond to a real imbalance situation in the tree. For this purpose, in Section 3 we redefine the context of a balance conflict in a chromatic tree in a global way. Since the rebalancing task is to resolve balance conflicts from the tree, our aim is that a chromatic tree contains balance conflicts if and only if it does not fulfill the balance conditions of red-black trees. In Section 4 we show that those balance conflicts can be resolved by using the rebalancing transformations of Boyar and Larsen [2], if the order is restricted in which the transformations are carried out. In this way we get a relaxed balancing scheme for red-black trees, which needs only a constant number of rotations per update and which guarantees that each performed structural change indeed correspond to a real imbalance situation in the tree.

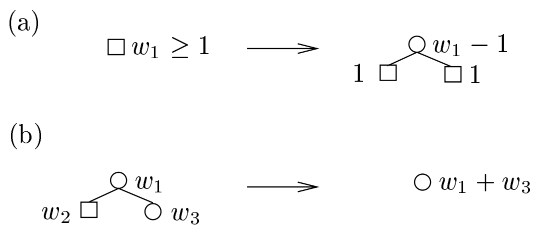

Figure 1: (a) Insertion and (b) Deletion of an item in a chromatic tree (Squares denote leaves, circles denote general nodes, i.e. either internal nodes or leaves, and the labels denote weights)

2 Chromatic Trees

The trees in this paper are leaf-oriented binary search trees, which are full binary trees (each node has either two or no children). The keys (chosen from a totally ordered universe) are stored in the leaves of the binary tree. The internal nodes contain only routers, which guide the search through the structure. A binary search tree is *balanced* if its height is bounded by $O(\log n)$, where n is the number of keys stored in the tree.

A *red-black tree* is a balanced binary search tree that is defined as follows. The nodes of a red-black tree are coloured red or black. As balance condition it is required that

(a) all leaves are black,

(b) each search path from the root to a leaf consists of the same number of black nodes, and

(c) each red node (except for the root) has a black parent.

In the following we consider chromatic trees, a relaxed version of red-black trees. The nodes of a *chromatic tree* are coloured red, black, or overweighted. The colour of each node p is represented by an integer value $w(p)$, the *weight* of p. The weight of a red node is zero, the weight of a black node is one, and the weight of an overweighted node is greater than one. As relaxed balance condition it is required that

(i) all leaves are not red and

(ii) the sum of weights on each search path from the root to a leaf is the same.

Insertion and deletion in a chromatic tree proceed in the following way.

In order to insert a key x first we search for x in the chromatic tree. If the search ends up unsuccessful, then we replace the leaf with weight w by an internal node with weight $w-1$ and two black leaves as its children, cf. Figure

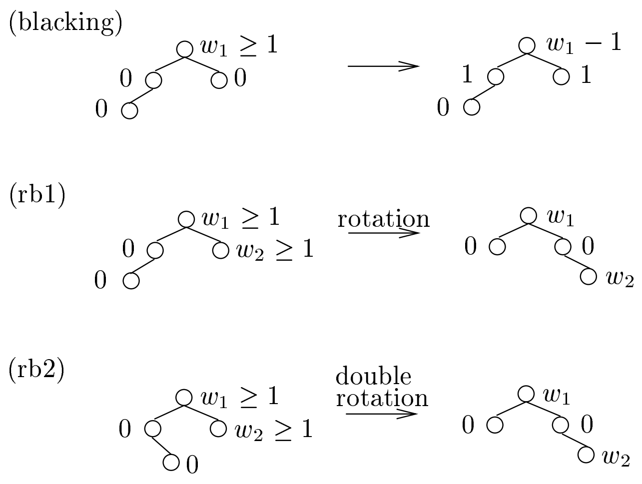

Figure 2: Handling a red-red conflict. (Symmetric cases are ommitted.)

1a. The two leaves now store the old key (where the search ended) and the new key x.

In order to delete a key x from a chromatic tree first we locate the leaf where the key is stored. If the search ends up successful, then we remove the leaf together with its parent and increase the weight of the remaining sibling by the weight of the parent node, cf. Figure 1b.

Inserting a key can introduce two adjacent red nodes on the search path from the root to the inserted leaf. This is called a *red-red conflict*. We say there is a *red-red conflict* at a node p, if p is red and has a red child node. Deleting a key can introduce an overweighted node. This is called an *overweight conflict*.

Since a chromatic tree that contains no red-red and no overweight conflicts clearly fulfills the balance condition of red-black trees, the task of rebalancing a chromatic tree in the previous solutions is to remove all red-red and overweight conflicts from the tree. For this purpose Nurmi and Soisalon-Soininen define a set of rebalancing operations for chromatic trees in [11, 12]. A more efficient transformation set is defined by Boyar and Larsen [2, 3].

The set of rebalancing operations by Boyar and Larsen [2, 3] includes four different types of transformations.

1. The *blacking* transformation handles a red-red conflict by using only colour changes. Thereby either the red-red conflict is resolved or it is shifted to the grandparent node, cf. Figure 2.

2. The *red-balancing* transformations remove a red-red conflict by performing a rotation or a double rotation, cf. Figure 2.

3. The *push* transformation handles an overweight conflict by using only

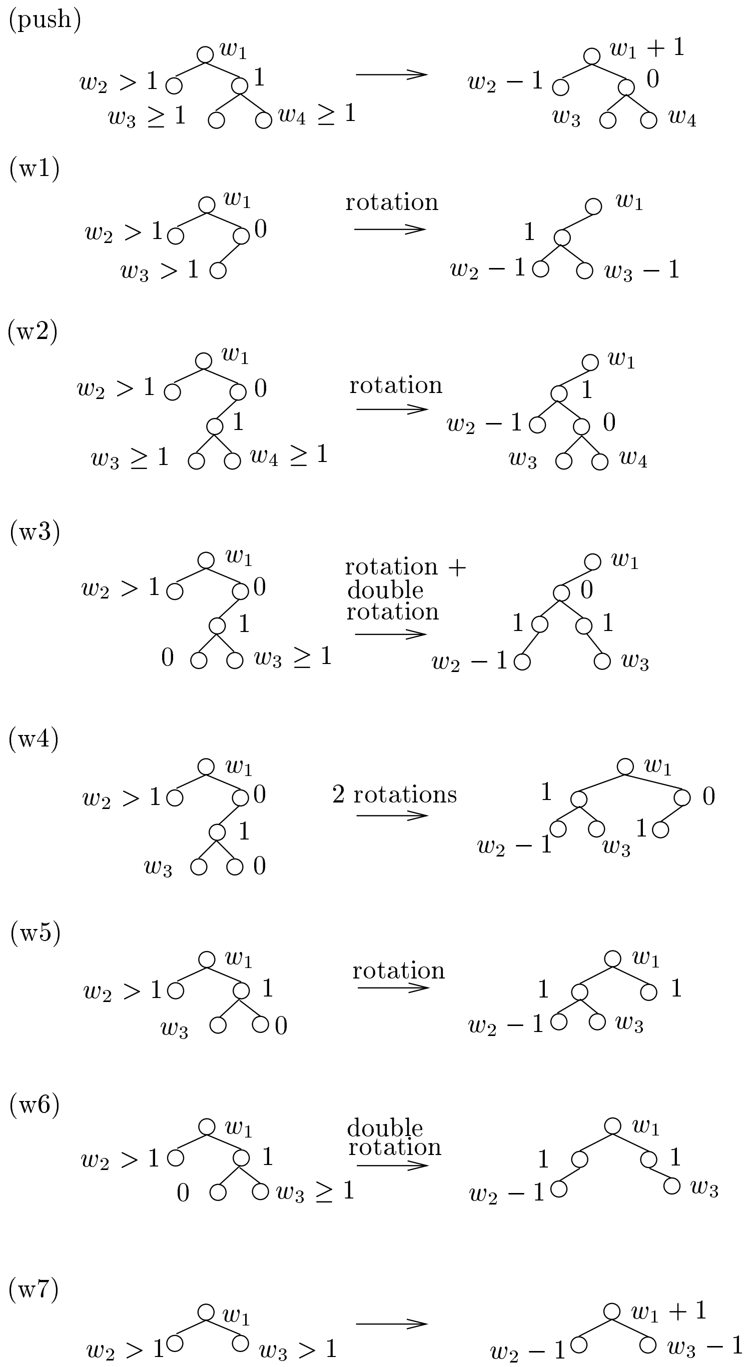

Figure 3: Handling an overweight conflict. (Symmetric cases are ommitted.)

colour changes. Thereby either the overweight conflict is resolved or it is shifted to the parent node, cf. Figure 3.

4. The *weight-decreasing* transformations remove an overweight conflict by performing a structural change (at most two rotations or a rotation plus a double rotation), cf. Figure 3.

It is required that several red-red conflicts at consecutive nodes are handled in top-down manner. Despite of this exception the rebalancing transformations can be applied in any arbitrary order.

Assume that a chromatic tree T of size n is given that is a red-black tree and i insertions and d deletions are applied to T. Boyar and Larsen [3] show that $O(i \cdot \log(n+i))$ blacking operations, $O(i)$ red-balancing transformations, $O(d \cdot \log(n+i))$ push tranformation, and $O(d)$ weight-decreasing transformations are needed in order to restore the balance condition of red-black trees.

3 Balance Conflicts

In all previous solutions [2, 3, 5, 11, 12, 13] the rebalancing of a relaxed balanced red-black tree (chromatic tree) is done in such a way that only the local position where conflicts occur are remembered. So, only in a very small vicinity several updates may have influence on a rebalancing transformation to be carried out. That is the reason why in this solutions the tree may be full of rebalancing requests from the history even though it is perfectly balanced. For example, assume that we have given a chromatic tree that is a red-black tree. We apply several deletions interleaved with rebalancing transformations on the tree. Finally, we obtain a tree that contains overweighted nodes but fulfills the balance condition of red-black trees if we regard non-red nodes as coloured black. In this situation the previous solutions [2, 3, 5, 11, 12, 13] associate each overweight conflict with a balance conflict and may perform structural changes in order to resolve this conflicts although the tree is full in balance. This example shows that it is necessary to redefine the concept of a balance conflict if we want to avoid such a problem. During the rebalancing it is important to know whether both subtrees of a node have lost black nodes since by suitable deletions in both subtrees the red-black tree balance condition might be restored by itself.

In order to allow an easier way of speaking in the following we assume that the overweight information is seperated from the colour information. That means, a node is always either red or black and each node has an extra value to store the units of overweight. In Figure 1–3 label $w \geq 1$ always denote that the corresponding node is black with $w-1$ units of overweight. Now the relaxed balance condition of chromatic trees can be formulated like follows.

(i) All leaves are black and

(ii) the number of black nodes plus the sum of overweights on each search path from the root to a leaf is the same.

In contrast to the previous solutions [2, 3, 5, 11, 12, 13] in the following we differ between red overweighted nodes and black overweighted nodes.

Let the *power* of a node p be the minimum number of black nodes on a search path from p to a leaf of the subtree rooted at p. We say there is a *power conflict* of unit $k > 0$ at a node p if the difference of the powers of p's children is k. A power conflict at a node p indicates that a subtree of p contains at least one search path that has lost more black nodes than all search paths in p's other subtree.

Lemma 1 *Assume that we have given a chromatic tree T that contains no red-red conflicts. T is a red-black tree if and only if it contains no power conflicts.*

Proof: Let T be a red-black tree. Since each search path from the root to a leaf has the same number of black nodes, it is clear that T cannot contain any power conflicts.

Let T be a chromatic tree without red-red conflicts that does not fulfill the balance condition of red-black trees. We have to show that T contains power conflicts. Since T is no red-black tree, at least one search path s_1 exists with less black nodes than another search path s_2. Let s_1 have a minimum number and s_2 have a maximum number of black nodes. Consider the node p where this two paths seperate. Let p_1 denote p's child that belongs to path s_1 and p_2 denote p's child that belongs to path s_2. If there is no power conflict at p, then in the subtree rooted at p_2 there is a search path that contains the same number of black nodes than the part of s_1 that begins at p_1 and ends at the leaf. Therefore, the subtree rooted at p_2 does not fulfill the balance condition of red-black trees. We continue recursively with this subtree, etc. Finally, if the only black node on the path s_1 is a leaf, then there must be a power conflict at its parent or its sibling. $\square$

Lemma 2 *Assume that we have given a chromatic tree T that contains no power conflicts. T is a red-black tree if and only if it contains no red-red conflicts.*

Proof: Since T contains no power conflicts, it fulfills the balance condition (a) and (b) of red-black trees. Balance condition (c) holds if and only if T contains no red-red conflicts. $\square$

We say there is a *balance conflict* at a node p, if there is either a red-red conflict or a power conflict at p. From Lemma 1 and Lemma 2 follows

Theorem 1 *A chromatic tree contains balance conflicts if and only if it does not fulfill the balance condition of red-black trees.*

4 Rebalancing a Chromatic Tree

The task of rebalancing a chromatic tree is to remove all balance conflicts from the tree. This should be done by performing local transformations so that only a constant number of nodes has to be locked at a time.

A single deletion may introduce power conflicts along the hole search path. Analogously, a single weight-decreasing transformation may resolve several power conflicts along the search path at the same time. Therefore, the difficulty in dealing with power conflicts is to determine power conflicts by considering only a constant number of nodes at a time.

Lemma 3 *Assume that we have given a node p both subtrees of which contain no overweighted nodes except for p's children. Then there is a power conflict at p if and only if p's children have different overweights. Furthermore, the subtrees of p contain no power conflicts.*

Proof: Let p_1 and p_2 be p's children, u_1 and u_2 the units of overweight of p_1 and p_2, and T_1 and T_2 the subtrees rooted at p_1 and p_2 respectively.

Since both subtrees of p_1 contain no overweighted nodes, from the relaxed balance condition (ii) follows that all search paths in T_1 have the same number of black nodes, denoted by b_1. So, T_1 contains no power conflicts. Analogously, all search paths in T_2 have the same number of black nodes, denoted by b_2, and T_2 contains no power conflicts. Therefore, p_1 has power b_1 and p_2 has power b_2. Because u_1 and u_2 are the only overweights in T_1 and T_2, from the relaxed balance condition (ii) follows that $u_1 + b_1 = u_2 + b_2$. So, $b_1 \neq b_2$ if and only if $u_1 \neq u_2$. □

Lemma 4 *Assume that we have given a node p both subtrees of which contain no overweighted nodes except for p's children. If there is a power conflict at p than the power conflict at p can be resolved by handling a corresponding overweight conflict at p's child node.*

Proof: Since there is a power conflict at p, from Lemma 3 follows that p's children have different overweights. Shifting units of overweight from the children to the parent p without performing colour changes has no influence on any power conflicts. Therefore, without restriction in the following we assume that only one of p's children is overweighted of unit $k > 0$. This child is denoted by q. Since both subtrees of p contain no power conflicts, the power conflict at p is of unit k.

In order to settle one unit of the power conflict at p either we have to increase the power of q by one or to perform a transformation such that q gets a sibling with one power less than q's old sibling, where it must be assured that all search paths ending at leaves that does not belong to the subtree rooted at q still have the same number of black nodes after the transformation as before. Observe, if q is black, then the push and weight-decreasing transformations are suitable transformations to achieve this. If q is red, then colouring q black and decreasing its overweight by one increases the power of q and settles one unit of the power conflict at p. Therefore, handling the overweight conflict at p's child q resolves the power conflict at p. □

Lemma 3 and Lemma 4 lead to the idea of handling power conflicts from bottom-up. That means, we allow to handle a power conflict at a node p (by

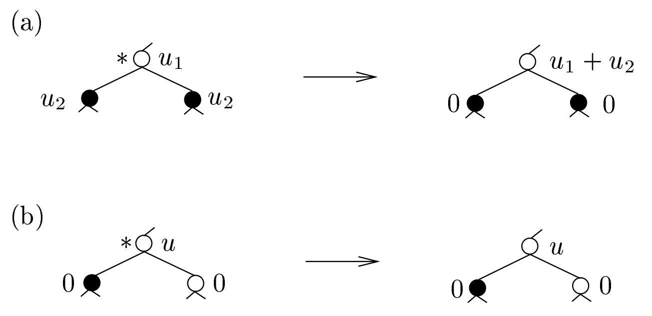

Figure 4: Examples of rebalancing after a deletion. The unsafe node, marked by *, has no power conflict.

handling the corresponding overweight conflict at one of p's children) only if both subtrees of p contain no other overweight conflicts.

For this we proceed similar as in [10]. To each node p we assign a bit $v(p)$. We say p is *safe* if $v(p) = 0$ and p is *unsafe* if $v(p) = 1$. Leaves are always safe. If p is safe, then p's children are both safe as well and the subtree rooted at p is a chromatic tree that contains no power conflicts. Furthermore, except for p this subtree contains no overweighted nodes.

The idea is to mark the search path (along which a power conflict may be bubbled up) during the update by setting all nodes to be unsafe. In this way at a node with a power conflict there is always information whether there may be other power conflicts in the subtree below.

A deletion proceed in the following way. First we locate the leaf where the key is stored. During the search all internal nodes on the search path become unsafe. If the search ends up successful, then we remove the leaf together with its parent. If the parent node was black either we colour the remaining sibling black (if it is red) or we increase its overweight by one unit. Furthermore, the overweight of the remaining sibling is increased by the overweight of the removed parent node.

In the following we describe how to rebalance after a deletion. Let p be an unsafe node the children p_1 and p_2 of which are both safe. (Note that such a node must exist since leaves are always safe.) Assume that the overweight of p_1 is greater or equal than the overweight of p_2.

If the overweight of p_2 is greater than zero, then we increase the overweight of p by the overweight of p_2. Furthermore, we decrease the overweight of p_1 by the overweight of p_2 and set the overweight of p_2 to zero. (This transformation corresponds to the weight-decreasing transformation (w7). Note, that in contrast to transformation (w7) we do not colour p black, if it is red, before considering p's parent and sibling.) After that we distinguish the following two cases.

Case 1: [p_1 has overweight zero.] In this case there is no power conflict at p.

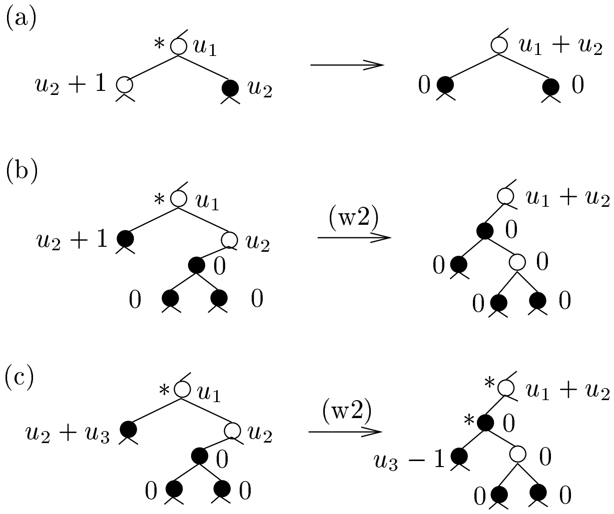

Figure 5: Examples of handling power conflicts. Unsafe nodes are marked by *.

No structural change or colour change is performed, and p becomes safe. Figure 4 shows an example for this event.

Case 2: [p_1 is still overweighted.] In this case there is a power conflict at p. If p_1 is red, then we colour it black and decrease its overweight by one, cf. Figure 5a for an example. Otherwise, we perform either the push transformation or one of the overweight-decreasing transformations (w2)–(w6), cf. Figure 5b for an example. (Note that the situation of transformation (w1) cannot occur since the subtree rooted at p_2 contains no overweights.) After the transformation p becomes safe. If there is still a power conflict (of one unit less) at the new parent node of p_1, then all ancestors of p_1 that were involved in the transformation become unsafe, cf. Figure 5c.

Red-red conflicts can be handled by applying one of the blacking or red-balancing transformations proposed by Boyar and Larsen [2]. Since we allow that red nodes may be overweighted as well, it must be assured that the units of overweight are handed on correctly to the children or grandchildren after a red-balancing transformation, cf. Figure 6. Furthermore, all ancestors of unsafe nodes that are involved in the transformation become unsafe.

During the rebalancing a structural change is performed only in order to resolve a balance conflict. So, from Theorem 1 follows

Theorem 2 *If red-red conflicts and overweight conflicts are settled as described above by using the rebalancing transformations proposed in [2], then each performed structural change indeed correspond to a real imbalance situation in the tree.*

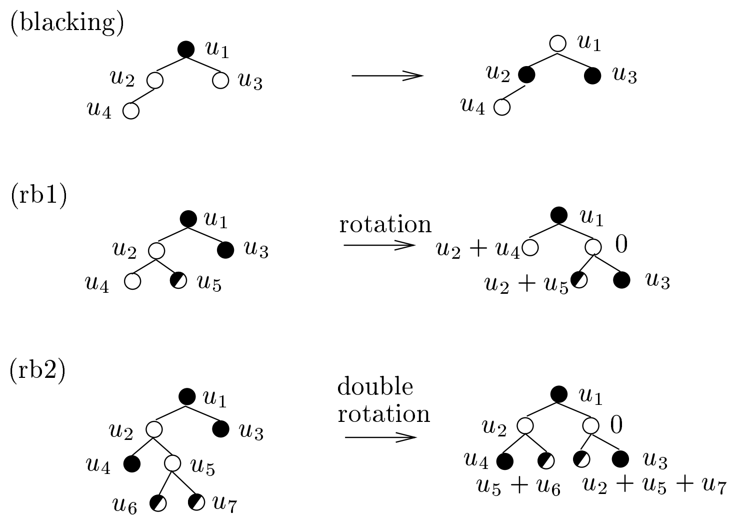

Figure 6: Handling a red-red conflict. (Filled nodes denote black nodes, nodes without fill denote red nodes, and half filled nodes denote nodes that are either red or black. The labes denote the units of overweight.)

We use the same set of rebalancing transformation as in [2]. The only requirement is that overweight conflicts are handled bottom-up. Furthermore, after a deletion we go back the hole search path in order to detect and resolve power conflicts (even if the deletion fails or does not introduce an overweight conflict). This needs a logarithmic number of overweight changes in addition, so that the bounds on the running time [3] still hold. Therefore we obtain

Lemma 5 *If i insertions and d deletions are applied to a chromatic tree of size n that fulfills the balance condition of red-black trees, then our balancing scheme needs $O(i+d)$ pointer changes and $O((i+d) \cdot \log(n+i))$ colour changes in order to restore the balance condition of red-black trees.*

5 Conclusions

Our aim was to do the rebalancing of a chromatic tree in such a way that whenever pointer changes are performed the chromatic tree really does not fulfill the balance condition of red-black trees. For this purpose we have redefined the context of a balance conflict caused by a deletion, such that in accordance with the new definition a chromatic tree contains balance conflicts if and only if it does not fulfill the balance conditions of red-black trees. We have shown that the rebalancing transformations proposed by Boyar and Larsen [3] can be used in order to resolve the balance conflicts from the tree if the handling of overweight conflicts is forced to happen bottom-up. In this way, it can be guaranteed that the rebalancing needs only a constant number of rotations per update, where each performed structural change indeed correspond to a real imbalance situation in the tree.

On the other hand, analogously to the previous solutions [2, 3, 5, 11, 12, 13] our scheme allows to increase the number of black nodes on each search path after multiple insertions only at the root. It does not notice if a sequence of insertions happens such that the balance condition of red-black trees can be restored very easily by colouring red nodes black in the bottom most part of the tree. This is caused by the kind of the blacking and red-balancing transformations, which assure a constant number of rotations per insertion.

If we do not mind a logarithmic number of rotations per insertion this feature can be achieved very easily. A red-red conflict can be transformed into an analogous problem as a balance conflict caused by a deletion. Colouring one of the involved nodes black increases the black height of the corresponding subtree by one and may introduce a power conflict at its parent node. This power conflict can be handled in the same way as caused by a deletion. The difference is that in order to resolve the conflict caused by a single insertion structural changes along the hole search path may be needed.

Whether a local balancing scheme can be found that needs only a constant number of rotations per insertion and that is able to notice if the red-black tree balance condition can be restored by colouring red nodes black in the bottom most part of the tree, is still an open question.

References

[1] G.M Adel'son-Vels'kii and E.M. Landis. An algorithm for the organisation of information. *Soviet Math. Dokl.*, 3:1259–1262, 1962.

[2] J. Boyar, R. Fagerberg, and K. Larsen. Amortization results for chromatic search trees, with an application to priority queues. In *4th International Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science, pages 270–281, 1995.

[3] J. Boyar and K. Larsen. Efficient rebalancing of chromatic search trees. *Journal of Computer and System Sciences*, 49:667–682, 1994.

[4] L.J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th IEEE Symposium on Foundations of Computer Science*, pages 8–21, 1978.

[5] S. Hanke, T. Ottmann, and E. Soisalon-Soininen. Relaxed balanced red-black trees. In *Proc. 3rd Italian Conference on Algorithms and Complexity, Lecture Notes in Computer Science*, volume 1203, pages 193–204, 1997.

[6] J.L.W. Kessels. On-the-fly optimization of data structures. *Communications of the ACM*, 26:895–901, 1983.

[7] K. Larsen. Avl trees with relaxed balance. In *Proc. 8th International Parallel Processing Symposium*, pages 888–893. IEEE Computer Society Press, 1994.

[8] K. Larsen and R. Fagerberg. B-trees with relaxed balance. In *Proc. 9th International Parallel Processing Symposium*, pages 196–202. IEEE Computer Society Press, 1995.

[9] K. Larsen, T. Ottmann, and E. Soisalon-Soininen. Relaxed balance for search trees with local rebalancing. Unpublished manuscript, 1997.

[10] K. Larsen, E. Soisalon-Soininen, and P. Widmayer. Relaxed balance through standard rotations. In *Workshop on Algorithms and Data Structures*, Halifax, Nova Scotia, Canada, August 1997. To appear.

[11] O. Nurmi and E. Soisalon-Soininen. Uncoupling updating and rebalancing in chromatic binary search trees. In *Proc. 10tm ACM Symposium on Priciples of Database Systems*, pages 192–198, 1991.

[12] O. Nurmi and E. Soisalon-Soininen. Chromatic binary search trees: A structure for concurrent rebalancing. *Acta Informatica 33*, pages 547–557, 1996.

[13] O. Nurmi, E. Soisalon-Soininen, and D. Wood. Concurrency control in database structures with relaxed balance. In *Proc. 6th ACM Symposium on Principles of Database Systems*, pages 170–176, 1987.

[14] T. Ottmann and E. Soisalon-Soininen. Relaxed balancing made simple. Technical Report 71, Institut für Informatik, Universität Freiburg, Germany, 1995. Also appeared as electronic version, anonymous FTP ftp.informatik.uni-freiburg.de, in /documents/reports/report71/, also http://hyperg.informatik.uni-freiburg.de/Report71.

[15] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29:669–679, 1986.

[16] E. Soisalon-Soininen and P. Widmayer. Relaxed balancing in search trees. In D.-Z.Du and K.-I. Ko, editors, *Advances in Algorithms, Languages, and Complexity: Essays in Honor of Ronald V. Book*. Kluwer Academic Publishers, Dordrecht, 1997. To appear.