Relaxed Balanced Red-Black Trees

S. Hanke¹, Th. Ottmann², and E. Soisalon-Soininen³

¹ Institut für Informatik, Universität Freiburg, Am Flughafen 17, D-79110 Freiburg
e-mail: hanke@informatik.uni-freiburg.de

² Institut für Informatik, Universität Freiburg, Am Flughafen 17, D-79110 Freiburg
e-mail: ottmann@informatik.uni-freiburg.de

³ Laboratory of Information Processing Science, Helsinki University of Technology,
Otakaari 1 A, FIN-02150 Espoo, Finland. e-mail: ess@cs.hut.fi

Abstract. Relaxed balancing means that, in a dictionary stored as a balanced tree, the necessary rebalancing after updates may be delayed. This is in contrast to strict balancing meaning that rebalancing is performed immediately after the update. Relaxed balancing is important for efficiency in highly dynamic applications where updates can occur in bursts. The rebalancing tasks can be performed gradually after all urgent updates, allowing the concurrent use of the dictionary even though the underlying tree structure is not completely in balance. In this paper we propose a new scheme of how to make known rebalancing techniques relaxed in an efficient way. The idea is applied to the red-black trees, but can be applied to any class of balanced trees. The key idea is to accumulate insertions and deletions such that they can be settled in arbitrary order using the same rebalancing operations as for standard balanced search trees. As a result it can be shown that the number of needed rebalancing operations known from the strict balancing scheme carry over to relaxed balancing.

1 Introduction

A dictionary is a scheme for storing a set of data such that the operations search, insert, and delete can be carried out efficiently. Standard implementations of dictionaries using balanced search trees like red-black trees, AVL-trees, half-balanced trees, and others presuppose that each update operation is followed by a sequence of rebalancing steps which restore the respective balance condition. Maintaining the balance conditions assures that the trees cannot degenerate into linear lists and search and update operations can be performed in a number of steps which is always logarithmic in the number of keys stored in a tree. In a concurrent environment, however, uncoupling the updating (insertion and deletion) from the rebalancing transformations may increase the possible amount of concurrency and speed up updates considerably. This leads to the notion of relaxed balance. Instead of requiring that the balance condition is restored immediately after each update operation the actual rebalancing transformations can be delayed arbitrarily and interleaved freely with search and update operations.

Relaxed balancing was first suggested in [8] for red-black trees. The first actual solution, presented by Kessels [10], is for relaxed balancing in AVL-trees [1] where the allowed updates are only insertions. Nurmi et al. [14] extend the
work of Kessels to the general case in which deletions are also allowed. In [11] the solution of [14] is analyzed, 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 balanced B-trees are introduced in [14] and further analyzed in [12]. In [13] Nurmi and Soisalon-Soininen propose a relaxed version of red-black trees which they call a chromatic tree. Boyar and Larsen [5] analyze the proposal of [13] and show that after a minor modification 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. Boyar et al. [3] 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. In [21] Soisalon-Soininen and Widmayer propose a relaxed version of AVL-trees which fulfills, despite the local nature of its operations, some global properties. For example, they show that in their solution no rotation will be performed if the underlying search tree happens to fulfill the AVL tree balance condition before any rebalancing has been done.

Except for the recent paper [21], all previous solutions are not wholly based on the standard balancing transformations but require a large number of different new transformations.

In this paper we propose a new technique of how to make known rebalancing algorithms relaxed in an efficient way. We show 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 order to illustrate the key ideas and to clarify the ideas underlying our solution as much as possible, we restrict ourselves to the case of red-black trees. But we emphasize that the ideas of marking items for deletions, allowing trees to grow randomly below the balanced part, and to settle accumulated update and rebalancing requests in top-down manner using the standard rebalancing operations carry over to many other classes of search trees as well. The aim of our proposal is to extend the constant-linkage-cost update algorithm for red-black trees [19] in such a way that updates and local structural changes are uncoupled and may be arbitrarily delayed and interleaved. A key idea in our solution is the assumption that the deletion of a key in a tree leads to a removal request only; the actual removal of a leaf is considered to be a part of the structural change to restore the balance condition. In this way we put completely aside the problem of deletion which has complicated the previous solutions of relaxed balancing.

## 2 Red-black 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 nullary nodes of a tree are called the external nodes or leaves while the other nodes are said to be internal nodes. We assume that the keys (chosen from a totally ordered universe) are stored in the leaves of the binary tree. The internal nodes contain routers, which guide the search from the root to a leaf.