Concurrent Rebalancing of AVL Trees: A Fine-Grained Approach *
(Extended Abstract)

Luc Bougé[1], Joaquim Gabarró[2], Xavier Messeguer[2], and Nicolas Schabanel[1]

[1] LIP, ENS Lyon, 46 alle d'Italie, F-69364 Lyon Cedex 07, France.
[2] LSI, Universitat Politcnica de Catalunya, Mòdul C5-C6, C/ Jordi Girona, 1-3, E-Barcelona 08034, Spain.

Abstract. We address the concurrent rebalancing of almost balanced binary search trees (AVL trees). Such a rebalancing may for instance be necessary after successive insertions and deletions of keys. We show that this problem can be studied through the self-reorganization of distributed systems of nodes controlled by local evolution rules in the line of the approach of Dijkstra and Scholten. This yields a much simpler algorithm that the ones previously known. As a by-product, this solves in a very general setting an old question raised by H.T. Kung and P.L. Lehman: where should rotations take place to rebalance arbitrary search trees?

Keywords: Concurrent algorithms, Search trees, AVL trees, Concurrent insertions and deletions, Concurrent generalized rotations, Safety and liveness proofs.

Note: The full version of this paper can be found in [BGMS97].

1 Introduction

Search trees are the key in implementing large data structures where keys are searched, inserted and deleted. The scheme introduced by Adel'son-Velskiĭ and Landis [AL62,Knu73], nowadays known as the *AVL scheme*, consists in keeping all internal nodes balanced, that is, the height of their subtrees differing at most by one. Sequential algorithms to insert one key at a time are well-known: once the key is inserted, the nodes along the access path are recursively updated by rotating their subtrees. But inserting and/or deleting many keys concurrently is much more difficult: the transient shapes of the tree may become *very unbalanced* in general, and no instantaneous update of the local registers maintained at each node can be assumed.

Many solutions have been proposed to this problem. Earlier *coarse-grain* solutions, e.g., by Ellis [Ell80], lock the access path to the inserted key to guarantee that two concurrent upwards update waves do not interfere. The degree of concurrency is obviously quite low. Later *medium-grain* solutions, e.g., by

* This work has been partly supported by the French CNRS Coordinated Research Program on Parallelism, Networks and Systems PRS, the EU HCM Program under Contract ERBCHGECT 920009 and the EU Esprit BRA Program ALCOM II 7141 and DGICYT under grant PB95-0787 (project KOALA).

Kessels [Kes83], split the upwards rebalancing wave into local atomic steps. Each insertion launches a new wave, so that several concurrent waves can be interleaved. The local rebalancing steps are described through a set of guarded rules. This idea has been later reworked and improved by by Nurmi, Soisalon-Soininen and Wood [NSSW87,NSSW92,NSS96]. Larsen [Lar94] shows that the reorganization process converges in $O(k.\log(n+2.k))$ steps in a tree with n nodes updated with k insertions.

The contribution of this paper is to go one step further in this direction by *completely uncoupling* the insertions (and/or deletions) and the rebalancing waves. The key idea for this *fine-grained* approach is taken from the work of Dijkstra, Lamport et al. [DLM+78] on concurrent "on the fly" garbage-collection. We see insertions/deletions as unpredictable *perturbations* on the tree data-structure, whilst rebalancing is independently performed by a number of *mutator* daemons based on local shape information only. The daemons flow information upwards through the tree (Propagation) or rotate the subtrees of a (apparently) unbalanced node (Rotation).

Our approach leads to fewer and simpler rules than previous ones, and it clarifies the essential nature of AVL rebalancing. As the shape of the tree may now be arbitrary, this amounts to solve an old question raised by H.T. Kung and P.L. Lehman [KL80]: where should rotations take place for to rebalance arbitrary trees? The answer is: anywhere.

The price to pay for this "fine-grained approach" is that $\Theta(n^2)$ steps are needed to rebalance an arbitrary binary tree in the worst case, instead of Larsen's $O(n.\log(n))$ for an empty tree filled by n successive insertions. Note however that a single atomic step of Larsen corresponds to several steps here which makes the comparison slightly more balanced. Also, we provide the user with a better degree of concurrency. Finally, there is good experimental evidence that the convergence is obtained in $O(n)$ steps in the average.

2 A Concurrent AVL Rebalancing Scheme

The challenge is to design a set of local guarded rules such that, if no external perturbation occurs, than any sequence of *local* rule applications eventually leads to a *globally* balanced tree.

2.1 General Description

Let u be a node of the search tree. We denote respectively by $u{\to}p$, $u{\to}ls$, $u{\to}rs$ the parent, the left son and the right son of u in the tree. The empty tree is denoted 'nil' and the root of the tree 'root'. The *real height* $\mathsf{realh}(u)$ is defined as usual:

$$\begin{cases} \mathsf{realh}(\mathsf{nil}) = 0 \\ \mathsf{realh}(u \neq \mathsf{nil}) = 1 + \max(\mathsf{realh}(u{\to}ls), \mathsf{realh}(u{\to}rs)) \end{cases}$$

As concurrent modifications in the tree prevent from maintaining realh on each node, each node $u \neq$ nil encodes its *local* knowledge of the state of the structure in two *private* registers in addition to the key register:

- $\mathsf{lefth}(u)$ and $\mathsf{righth}(u)$ are respectively the *apparent heights* of the left and right sons of u, at the best of the knowledge of u.

Definition 1 We call *height-relaxed search tree (HRS-tree)* a search tree whose nodes are equipped with the two private registers lefth and righth satisfying the following consistency condition: $\mathsf{lefth}(u) = 0$ (resp. $\mathsf{righth}(u) = 0$) for any node u with an empty left (resp. right) son.

The following auxiliary functions on the nodes of HRS-trees will be useful.

- $\mathsf{localh}(u)$ is the *apparent local height* of u, as computed from the two previous registers: $\mathsf{localh}(u) = 1 + \max(\mathsf{lefth}(u), \mathsf{righth}(u))$
- $\mathsf{car}(u)$, the *carry* of u, is the gap of knowledge between u and its parent:
  $$\mathsf{car}(u) = \begin{cases} \mathsf{lefth}(u{\to}p) - \mathsf{localh}(u) & \text{if } u \text{ is the left son of its parent} \\ \mathsf{righth}(u{\to}p) - \mathsf{localh}(u) & \text{otherwise} \end{cases}$$
  The car function measures the inconsistency of local information on the structure of the tree. A node u is said *reliable* if $\mathsf{car}(u) = 0$.
- $\mathsf{bal}(u)$ of u is the *apparent balance* of u, defined as follow:
  $$\mathsf{bal}(u) = \mathsf{lefth}(u) - \mathsf{righth}(u)$$
  A node u is said *apparently balanced* if $|\mathsf{bal}(u)| \leqslant 1$.

The following fact holds: *If each node of an HRS-tree T is reliable and apparently balanced, then T is an AVL.*

2.2 Description of the Daemons

Propagation Rule. It propagates information upwards from a son to its parent. As a convention the final state of a node u after application of a rule is denoted u'.

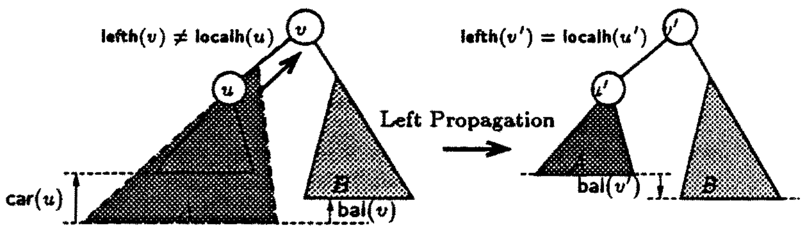


Fig. 1. Propagation rules: Rule (LP) left propagation if $\mathsf{car}(u) \neq 0$

Rule (LP) – Left Propagation (Figure 1)
Guard: Node u is the left son of node v and u is not reliable: $\mathsf{car}(u) \neq 0$
Action: the apparent left height of v is updated: $\mathsf{lefth}(v') = \mathsf{localh}(u)$
Spatial scope: u and its parent $v = u{\to}p$.

The right propagation rule (RP) where u is the right son of v, can be deduced symmetrically from (LP). It is easy to see that applying these rules repeatedly will eventually set the apparent local height of each node to its real height.

Rotation Rules. These rules are inspired from the original AVL rules [AL62] but extended to the case where the balances of the nodes may exceed 2. These relaxed preconditions allow to rebalance any tree with any initial local knowledge. The rotation rules tend to reduce the apparent balance, but of course, can worsen not only the consistency of the local heights but also the real balance if the apparent balance was wrong.

Rule (RR*) – Right Rotation, Unbalanced case (Figure 2.2)
Guard: Node u is the left son of node v, u is reliable, $\mathsf{bal}(u) > 0$ and $\mathsf{bal}(v) \geqslant 2$
Action: u and v execute a right rotation [FIG. 2.2] with the obvious updating:

$$\begin{array}{ll} \mathsf{lefth}(u') = \mathsf{lefth}(u) & \mathsf{righth}(u') = \mathsf{localh}(v') \\ \mathsf{lefth}(v') = \mathsf{righth}(u) & \mathsf{righth}(v') = \mathsf{righth}(v) \end{array}$$

Spatial scope: u and its parent $v = u \rightarrow p$.

The rule (LR^*), where u is the left son of v, and u and v execute a left rotation when $\mathsf{bal}(u) < 0$ and $\mathsf{bal}(v) \leqslant -2$, is obtained symmetrically from (RR^*).

Rule (RR$_=$) – Right Rotation, Balanced case (Figure 2.2)
Guard: Node u is the left son of node v, u is reliable, $\mathsf{bal}(u) = 0$ et $\mathsf{bal}(v) \geqslant 2$
Action: u and v execute a right rotation [FIG. 2.2] with the obvious updating:

$$\begin{array}{ll} \mathsf{lefth}(u') = \mathsf{lefth}(u) & \mathsf{righth}(u') = \mathsf{localh}(v') \\ \mathsf{lefth}(v') = \mathsf{righth}(u) & \mathsf{righth}(v') = \mathsf{righth}(v) \end{array}$$

Spatial scope: u and its parent $v = u \rightarrow p$.

The rule $(LR_=)$, where u is the left son of v and, u and v execute a left rotation when $\mathsf{bal}(u) = 0$ and $\mathsf{bal}(v) \leqslant -2$, is obtained as before symmetrically from $(RR_=)$.

Rule (LRR) – Left-Right double Rotation (Figure 2.2)
Guard: Node w is the right son of the left son u of node v, w and u are reliable, $\mathsf{bal}(u) < 0$ et $\mathsf{bal}(v) \geqslant 2$
Action: u, v and w execute a left-right double rotation [FIG. 2.2] with the obvious updating:

$$\begin{array}{ll} \mathsf{lefth}(u') = \mathsf{lefth}(u) & \mathsf{righth}(u') = \mathsf{lefth}(w) \\ \mathsf{lefth}(v') = \mathsf{righth}(w) & \mathsf{righth}(v') = \mathsf{righth}(v) \\ \mathsf{lefth}(w') = \mathsf{localh}(u') & \mathsf{righth}(w') = \mathsf{localh}(v') \end{array}$$

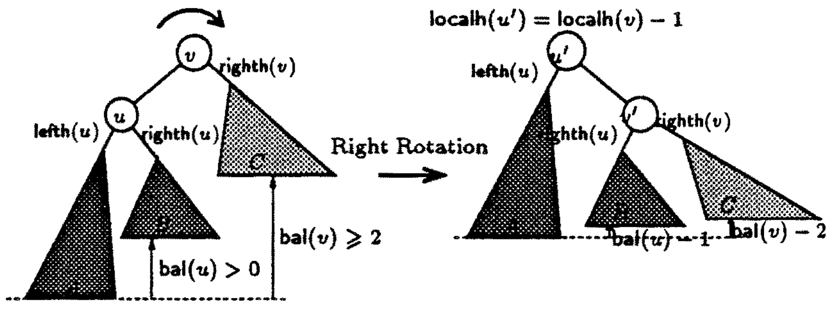
Spatial scope: u, its parent $v = u \rightarrow p$ and its right son $w = u \rightarrow rs$.

The symmetrical rule (RLR) where w is the left son of the right son u of v and u, v and w execute a right-left double rotation, applies when u and w are reliable, $\mathsf{bal}(u) > 0$ and $\mathsf{bal}(v) \leqslant -2$.

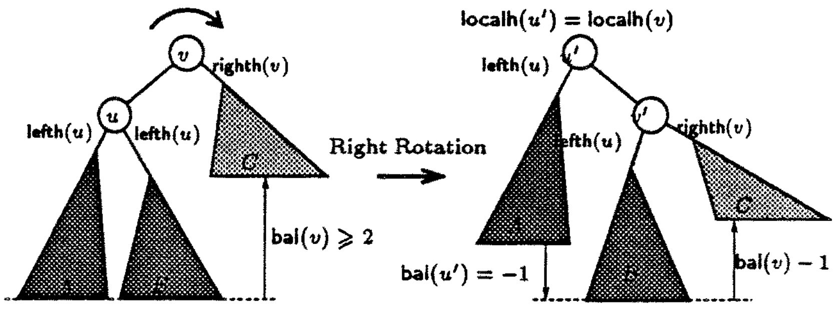
2.3 Invariant Properties

The following lemma ensures the *safety* of the algorithm: "nothing bad can happen: if the algorithm blocks, then we hold the right result".

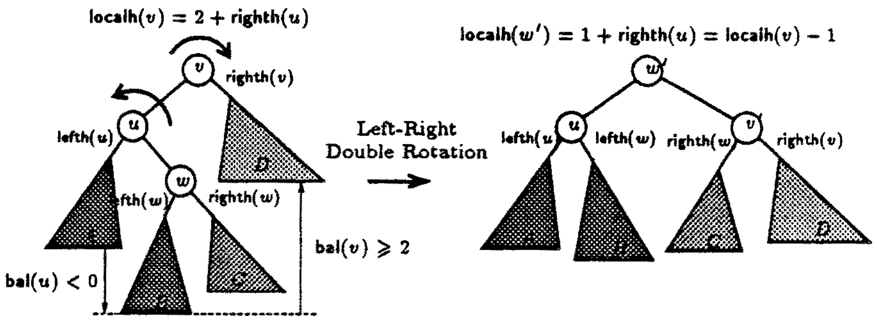
Lemma 1 (Safety property) *Let T be an HRS-tree. If T' is obtained by applying on T any one of the rules described above, then T' is an HRS-tree holding the same keys than T. Moreover if no rule applies on T, T is an AVL.*



(a) Rule (RR*) right rotation if car(u) = 0, bal(u) > 0 and bal(v) ⩾ 2.



(b) Rule (RR$_=$) right rotation if car(u) = 0, bal(u) = 0 and bal(v) ⩾ 2.



(c) Rule (LRR) left-right double-rotation if car(u) = car(w) = 0, bal(u) < 0 and bal(v) ⩾ 2.

Fig. 2. The rotation rules

A closer look at the rules reveals the following stable property which is actually the key to the proof of convergence below.

Lemma 2 *Let T be an HRS-tree, so that $\forall u \in T\ \mathsf{car}(u) \geqslant 0$. If T' is obtained by applying on T any one of the rules described above, then $\forall u' \in T'\ \mathsf{car}(u') \geqslant 0$.*

3 Convergence

3.1 Proof of liveness

As long as perturbations occur in the tree (insertion or deletion of keys), the daemons just compete with the mutators. The resulting behavior essentially depends on their relative speeds. For the convergence analysis, we hence assume that no insertion or deletion occurs any longer and prove then that at most $\Theta(n^2)$, where n is the number of nodes of the tree, rules may be applied. By Lemma 1, the resulting tree is an AVL: the algorithm rebalances thus any arbitrary tree using at most $\Theta(n^2)$ rules.

The convergence proof is based on a number of global quantities expressing the progress towards a final state. The complete description can be found in [BGM95,BGMS97] and we only sketch here an intuitive description.

The first observation is that on each rule application, negative carries vanish or flow upwards the root. The tree progressively converges towards a state described by Lemma 2. Let us define $\mathsf{NEG} = \sum_{\mathsf{car}(u)<0} Out(u) \cdot |\mathsf{car}(u)|$, where $Out(u)$ denotes the *number of nodes* of the tree which are *not in the subtree* rooted in u, as introduced by Kessels [Kes83]. It can be shown that NEG cannot increase.

When it does not decrease, there is a subtle interaction between car and bal: rebalancing a node may increase its carry as its apparent local height may decrease; conversely, propagating a carry may increase the imbalance of its father. Let us define $\mathsf{POS} = \sum_{\mathsf{car}(u)>0} \mathsf{car}(u)$ and $\mathsf{BAL} = \sum_u |\mathsf{bal}(u)|$. It can be shown that the trade-off $(2\mathsf{POS} + \mathsf{BAL})$ cannot increase.

In the only case where NEG and $(2\mathsf{POS}+\mathsf{BAL})$ does not decrease (Rule LR=), the quantity $\mathsf{RBAL} = \sum_{|\mathsf{bal}(u)|\geqslant 2} |\mathsf{bal}(u)| - 1$ necessarily decreases.

Property 1 (Liveness property) $\langle \mathsf{NEG}, 2\mathsf{POS} + \mathsf{BAL}, \mathsf{RBAL} \rangle$ *is a valid variant: it strictly decreases for the lexicographic order on any rule application and it is greater than $\langle 0,0,0 \rangle$. Therefore, no infinite sequence of rule applications is possible.*

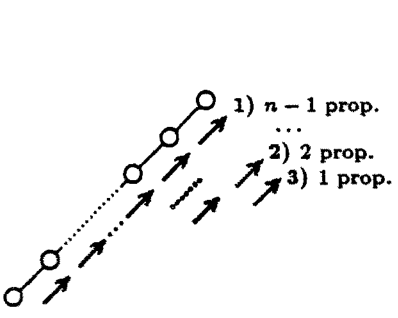
This proposition implies moreover that the algorithm converges on any tree after at most $3n^7$ rule applications.

A tedious exhaustive case analysis (summed up by a table in the complete paper [BGMS97]) reveals a more subtle interaction between those four quantities and leads to a simpler variant:

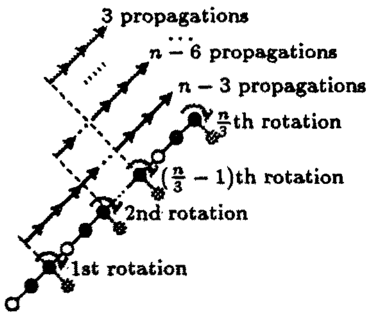
Theorem 1 $6(\mathsf{NEG} + \mathsf{POS}) + 2\mathsf{BAL} + \mathsf{RBAL}$ *is a valid variant for the algorithm.*

Therefore if c_{max} and b_{max} respectively denote the maximum absolute values of car and bal initially, our scheme applies at most $6c_{max}n(n+1)+3b_{max}n$ rules to rebalance *any* arbitrary HRS-tree with *any* initial state.

Two examples of worst cases applying $\Theta(n^2)$ rules are shown Figure 3. An



(a) Initially each node has its local height set to 1: $c_{max}=1$ and $b_{max}=0$



(b) Initially the local heights are the real heights: $c_{max}=0$ and $b_{max}=n-1$

Fig. 3. Two examples of $\Theta(n^2)$ rules executions highlighting the importance of the two terms $6c_{max}n(n+1)$ and $3b_{max}n$.

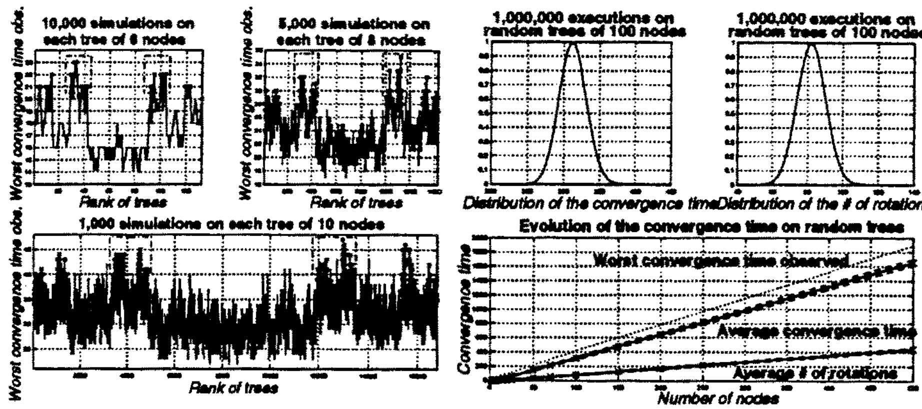
amazing fact is that we could not find any execution scheme involving more than $O(n)$ rotations. It is tempting to relate this to the two parts of the variant: $6c_{max}n(n+1) = O(n^2)$ may be related to the number of propagations, and $3b_{max}n = O(n)$ to the number of rotations. We therefore conjecture that *at most $O(n)$ rotations may be applied.* It is likely that such a bound would certainly *shed a new light on the intimate structure of AVL trees.*

3.2 Experimental studies

We have essentially proceeded to two kinds of experimental behavior studies: practical worst case and average convergence time studies.

Experimental Worst Convergence Time Analysis. First, we concentrate on small trees and record for each tree the worst convergence time measured on a large number of simulations. The results are displayed on [FIG. 3.2]. The diagrams are based on the following tree enumeration: we enumerate the binary trees of fixed size n simply by enumerating recursively all the possible right subtrees for all the possible left subtrees and we index each tree by its *rank in this enumeration.* The advantage of this method is that it respects the recursive structure of binary trees; in particular trees which have close indexes have close shapes.

It appears that these diagrams Figure 3.2 have fractal structures: this means that our rebalancing algorithm is somehow continuous with respect to the shape of the tree.



(a) Worst convergence times observed.The '*' point out the linear trees. The dashed rectangles highlight the worst convergence time localizations.

(b) Average convergence time observed. The dashed lines represent the dispersion intervals.

Fig. 4. Experimental Analysis

The worst worst cases are always obtained on linear trees and more precisely, on the second pair of the *regular zigzag tree*, i.e. the linear trees where each son of a right son (resp. left son) is a left son (resp. right son) (cf. [BGMS97]).

Experimental Average Convergence Time Analysis. A more precise analysis of the convergence time distribution confirms the above assumption. The result of the simulations is shown [FIG. 3.2].

The behavior of our algorithm appears to be very smooth: the convergence time seems to follow a "Gaussian-like" distribution as well as the number of rotation rule applications. The average convergence time appears to be $\alpha.n$ with $\alpha \cong 3.5$ with a standard deviation of $\beta\sqrt{n}$ with $\beta \cong 4.1$.

Unfortunately we do not have any theoretical estimation concerning the convergence time distribution.

4 Conclusion

This paper presents a fine-grained, distributed approach to the problem of managing concurrent request in AVL search trees. Our contribution is to show that completely uncoupling the insertion/deletion of keys from the rebalancing process yields fewer, simpler and clearer local rules. In fact, our scheme allows to rebalance *any* tree with n nodes in $O(n^2)$ local steps with a very high degree of parallelism: each steps only locks at most 3 nodes. However, extensive simulation results indicate that quadratic behaviors are extremely unlikely.

In fact, this fine-grained scheme yields a useful basis to design more complex algorithms by restricting the scheduling of the rules to "efficient" ones. It turns out that many existing algorithms previously proposed in the literature can be seen as such specializations (up to a suitable renaming of the registers). The extended version of this paper shows that this is the case for the algorithms of Ellis [Ell80], Kessels [Kes83], Nurmi and al. [NSSW87,NSSW92], and Larsen [Lar94]. The initial sequential AVL algorithm even appears as a limit case. As our scheme has been proved correct (safety and liveness), any non-deadlocking specialization of it yields a correct algorithm, too.

References

[AL62] G. M. Adel'son-Vel'skiĭ and E. M. Landis. An algorithm for the organization of the information. *Soviet Mathematics Doklady*, (3):1259–1263, 1962.

[BGM95] L. Boug, J. Gabarró, and X. Messeguer. Concurrent AVL revisited : self-balancing distributed search. Research Report RR95-45, LIP, ENS Lyon, France, 1995. Available at URL http://www.ens-lyon.fr/LIP.

[BGMS97] L. Boug, J. Gabarró, X. Messeguer, and N. Schabanel. Concurrent rebalancing of AVL trees: A fine-grained approach. Research Report RR97-13, LIP, ENS Lyon, France, 1997. Available at URL http://www.ens-lyon.fr/LIP.

[BS77] R. Bayer and M. Schkolnick. Concurrency of operations on B-trees. *Acta Informatica*, 9(1):1–21, 1977.

[DLM+78] E. W. Dijkstra, L. Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens. On the fly garbage collection : an exercice in cooperation. *Comm. ACM*, 21(11):966–975, 1978.

[Ell80] C. S. Ellis. Concurrent search and insertion in AVL trees. *IEEE Trans. Comp.*, C-29(9):811–817, 1980.

[Fra80] N. Francez. Distributed termination. *ACM Trans. Progr. Languages and Systems*, 2:42–55, 1980.

[Kes83] J. L. S. Kessels. On the fly optimisation of data structures. *Comm. ACM*, 26(11):895–901, 1983.

[KL80] H. T. Kung and P. L. Lehman. Concurrent manipulation of binary serch trees. *ACM Trans. Database Systems*, 5(3):354–382, 1980.

[Knu73] D. E. Knuth. *The art of computer programming, Fundamental algorithms*. Addison-Wesley, 1973.

[Lar94] K. S. Larsen. AVL trees with relaxed balance. In *Proc. Int. Parallel Processing Symp.*, pages 888–893. IEEE Comp. soc., 1994.

[NSS96] O. Nurmi and E. Soisalon-Soininen. Chromatic binary serach trees. *Acta informatica*, 33(6):547–557, 1996.

[NSSW87] O. Nurmi, E. Soisalon-Soininen, and D. Wood. Concurrency control in database structures with relaxed balance. *ACM Symp. Princ. Distributed Systems*, pages 170–176, 1987.

[NSSW92] O. Nurmi, E. Soisalon-Soininen, and D. Wood. Concurrent balancing and updationg of AVL trees. Technical Report 1992 ITKO-B76, Helsinki Univ. of Techn., Dept. Comp. Sciences, 1992.