Exponentially Decreasing Number of Operations in Balanced Trees *

Lars Jacobsen
Systematic Software Engineering A/S, Aarhus, Denmark
Email: lja@systematic.dk.

Kim S. Larsen
Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark.
Email: kslarsen@imada.sdu.dk.

Abstract

While many tree-like structures have been proven to support amortized constant number of operations after updates, considerably fewer structures have been proven to support the more general exponentially decreasing number of operations with respect to distance from the update. In addition, all existing proofs of exponentially decreasing operations are tailor-made for specific structures. We provide the first formalization of conditions under which amortized constant number of operations imply exponentially decreasing number of operations. Since our proof is constructive, we obtain the constants involved immediately. Moreover, we develop a number of techniques to improve these constants.

1 Introduction

When asynchronous processes work on a shared tree-like structure, operations which are carried out by one process near the root are likely to interfere with and slow down other processes. In contrast, if the tree structure is large, then operations near the leaves will most likely not disturb others (this of course is application-dependent). This scenario is one motivation for considering analyses of where operations are carried out.

*Supported in part by the Danish Natural Science Research Council (SNF) and in part by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT). A preliminary version of this paper appeared in the *Seventh Italian Conference on Theoretical Computer Science*, Lecture Notes in Computer Science, vol. 2202, pages 293–311, Springer-Verlag, 2001.

Many tree-like structures have been proven to have amortized constant time operations. Amortized complexity analysis [14] aims at proving worst-case bounds for a sequence of operations which are better than the length of the sequence times the worst-case bound of each operation. A popular method for carrying out such analyses is the potential function technique. Using this technique, for purpose of the analysis, one pretends that operations which only take a short time pay some "time" in to the potential function which can then be used to reduce the time of operations which take a long time to complete. In this way, time is averaged out and one can sometimes obtain an *amortized* time for the operations which is better than their worst-case times. If Φ is a potential function assigning an integer value to a state T, then the amortized time of carrying out some operation which takes time t and changes the state from T to T' is defined to be $t + \Phi(T') - \Phi(T)$. It is easy to prove that the sum of the amortized times of carrying out all operations in a sequence is an upper bound on the sum of all the actual times of carrying out all these operations, provided that the final potential is at least as large as the initial potential. For examples of amortized constant time operations, which are relevant for this paper, see for instance [4, 5, 9, 10, 14].

If operations are initiated at the leaves of trees and move towards the root by local operations, this gives some hope that operations will not often be carried out close to the root. In particular, this hope is justified if some balance constraints on the trees guarantee that all leaves have some minimum non-constant distance to the root. A typical example of such a scenario is bottom-up rebalancing in balanced search trees. Proofs of exponentially decreasing operations have been published for (a, b)-trees [5] and insertions into AVL-trees [10].

The proofs in the literature that tree-like structures have amortized constant operations are often based on the potential function technique, or can easily be rephrased into that paradigm such that all the operations, excluding the updates, decrease the potential function. In other words, these operations are *amortized free*, meaning that the decrease in potential pays for the cost of the operation.

The main contribution of this paper is a formalization of these concepts of balance and locality, and a proof that under these conditions, amortized free operations imply that the number of rebalancing operations carried out at a certain level in the tree decreases exponentially in the distance from the leaves.

We have focused on formulating sufficient conditions that are as weak as possible such that our theorem is as broadly applicable as possible. This means that the many structures on which operations have been shown to be amortized constant can now, with very few extra arguments, claim to provide the stronger and more directly useful exponentially decreasing operations.

However, our proof is constructive, meaning that once exponentially decreasing operations have been established, constants can also be derived. More precisely, we obtain constants c_1 and c_2 such that the theorem guarantees that at most $c_1 \frac{u}{{c_2}^i}$ operations are carried out at a distance i from the leaves in response to u

initiations of amortized constant time operations from the leaves. The constant c_2 is of course particularly interesting; the larger it is, the better.

For theoretical results, it is interesting to be able to claim that some structure has exponentially decreasing operations, whereas the constants may not be all that interesting, but as described above, some constants automatically come out of applying the main theorem. In practical applications, it might be interesting to know the best possible constants, and the rest of the paper, after the main theorem, is devoted to this. A number of techniques are introduced which can often improve the constants, if the best possible constants are not automatically provided by a direct application of the main theorem.

Example applications are given in different sections with the purpose of illustrating these techniques. Some familiarity with the application areas partial persistency and AVL-trees would be an advantage. Though our treatment of these application areas is brief, our presentations should be complete and we give the relevant references.

Through the applications, we demonstrate that we can obtain new results, but also that we can derive results matching the published results for existing structures. This shows that if the constants are of interest, then in many cases the best possible constants can be found using this collection of techniques. In fact, the new concrete results in this paper are tight (a pebble game example related to partial persistency and deletions in AVL-trees).

In summary, we believe that a deeper understanding of complexity issues in trees is obtained through our results. On the practical side, when other researchers are interested in establishing new results of exponentially decreasing operations, our results may be useful. The conditions for when our result can be applied are fairly straight-forward and easy to remember having seen them once, and this should make it easier for researchers to recognize these conditions in their work with tree-like structures and quickly realize whether or not a given concrete structure has this property. If a researcher is interested in finding the best possible constants, it may be as easy to create a proof from scratch than to first obtain a sufficient understanding of our techniques. However, for difficult cases, our techniques for improving these constants can be an inspiration.

The rest of this paper is organized as follows: In Section 2, we state and prove the main theorem. In Section 3, we show how to improve the constants obtained from the theorem for certain structures. In Section 4, we give a first simple example of how to use the main theorem. In Section 5, we give two additional examples, and demonstrate a technique for making non-local structures, which are not directly covered by the theorem, satisfy the theorem as well. In Section 6, we conclude.

2 The Main Theorem

We now address the statement and proof of the main theorem. In between definitions, we try to provide the necessary intuition. However, as a supplement to this intuition, it might be beneficial to read the first example (Section 4) in conjunction with the formal part. In Table 1, we summarize the notation which is introduced gradually in the set-up discussed in this section and the next.

$\mathcal{U}$	the set of updates
$\mathcal{R}$	the set of (post-processing) operations
$\mathcal{L}$	layer function assigning a non-negative integer to each node
$\mathcal{C}$	a configuration
k_{size}	maximal number of nodes in a configuration
k_{height}	maximal layer difference in a configuration
$\ell_i(T)$	the set of nodes in layer i
Φ	a local potential function
T_i	tree at time i
T'	T after the application of a transformation
$\Delta\Phi_j$	potential change on layer j, $\Phi(\ell_j(T')) - \Phi(\ell_j(T))$
j_0	first layer with potential change
$k+1$	span of a transformation
$\mathcal{N}_i$	potential change on layers j_0 through i
$\mathcal{P}_i$	potential change on layers $i+1$ through j_0+k
$\mathcal{I}$	layers where $\mathcal{N}_i$ is negative and $\mathcal{P}_i$ non-negative
c_1, c_2	constants of main theorem in $c_1 \#\text{upd}/c_2^i$
$\mathcal{N}'$	negated sum of all negative potential changes
$\mathcal{P}'$	sum of all positive potential changes
f	ratio $\frac{\mathcal{P}'}{\mathcal{N}'} < 1$
c_t	maximum f for all applications of transformation t
k_t+1	maximal span of any application of transformation t
k_r	maximal k_t over all post-processing operations
k_u	maximal k_t over all updates
c	maximal c_t over all post-processing operations
c_u	maximal potential increase by any update
c_r	minimal potential decrease on layers where potential is decreased

Table 1: Summary of notation.

First, we try to give an intuitive explanation of what we are trying to obtain. We have observed the following:

- When establishing proofs of rebalancing in trees being amortized constant, the proofs are often structured via a potential function which is incremented by updates and decremented by rebalancing operations. Thus,

the potential pays for the rebalancing.

- When layers can be defined in trees in a natural manner, the dominating rebalancing operations can only move a fraction of potential from one layer to another closer to the root. By dominating, we mean the operations which do not have an immediate constant bound on how many times they can be applied after an update. Thus, the amount of potential available at a layer decreases exponentially towards the root, and then the number of rebalancing operations must too. Realizing this by using that only a fraction of the potential on one layer can be moved to the next is also at the core of the proofs from [5] and [10].

We explore the generality of this intuition by formalizing all the involved concepts.

We aim at stating and proving a result which can be applied to a variety of tree schemes. By a tree scheme, we think of something such as AVL-trees or B-trees, i.e., at least some operations for modifying the trees, but also possibly definitions of nodes in the tree and allowed tree shapes, though these definitions could be implicit. For instance, one could define the allowed tree shapes as all the tree shapes which can be created by a finite number of modifications to an empty tree.

We are particularly concerned with establishing results regarding the properties of the operations which modify the trees. Such operations can be specified in many different ways. Sometimes it is done using pseudo-code, sometimes using a concrete programming language, and sometimes by a collection of local transformations which can be applied to a tree to modify it. To obtain a unified and manageable framework, we require that the specifications of the modifying operations are in the form of local transformations, and we formalize this notion.

Let k_{size} and k_{height} be universal non-negative integer constants. If T is a tree, then a *configuration* is a constant number of at most k_{size} connected nodes. A function $\mathcal{L} : T \to \mathbb{N}_0$ is a *layer function* on T if the following holds:

- for all nodes $u, v \in T$, where v is a proper ancestor of u, $\mathcal{L}(v) \geq \mathcal{L}(u)$
- for all nodes $u, v \in \mathcal{C}$ for any configuration $\mathcal{C}$, $|\mathcal{L}(u) - \mathcal{L}(v)| \leq k_{\text{height}}$
- for all leaves u, $\mathcal{L}(u) = 0$

This naturally defines a *layer* $\ell_i(T) = \{v \in T \mid \mathcal{L}(v) = i\}$ as a subset of T.

We can prove Theorem 1 at the end of this section based on these requirements to the layer function. However, the result which comes out of applying Theorem 1 to a structure will only be interesting if there are a non-constant number of layers; typically, it will be $\Theta(\log n)$.

A *local rule* is a transformation on T describing *before* and *after* configurations. If a configuration in the tree matches the before-configuration $\mathcal{C}$, it may be transformed to the after-configuration $\mathcal{C}'$.

We distinguish between transformations controlled by external events (from here on referred to as *updates*) and transformations where the before configuration matches a configuration in the tree created by another transformation (from here on referred to as *post-processing operations* or just *operations*). We let $\mathcal{U}$ denote the set of updates and let $\mathcal{R}$ denote the set of operations.

Consider an update, which by definition is a local rule. Thus, it replaces some configuration $\mathcal{C}$ by $\mathcal{C}'$. We require that for all nodes $u \in \mathcal{C} \cup \mathcal{C}'$, $\mathcal{L}(u) \leq k_{\text{height}}$. This captures that updates must take place *at the leaves*, and we use that as our definition.

In the theorem and proof, potential functions are used, and also for these do we need a notion of locality. The properties we need are that the global potential is defined as the sum of local contributions and that a modification in one part of the tree can only cause changes in local contributions in the vicinity of the modification in question. Since configurations have already been defined, we can reuse this concept to express some of these constraints.

A *local potential function* Φ on T is a potential function fulfilling the following requirement. It assigns non-negative bounded potential to each node of the tree, i.e., for all nodes u, $0 \leq \Phi(u) \leq K$, where K is a universal non-negative constant. We define the potential of a set of nodes as the sum of the potentials of the nodes in the set.

Our result discusses what happens to a tree over time. So, we are studying a sequence of trees $T_0, T_1, T_2, \ldots$, where T_0 is the initial tree, and each T_i, $i \geq 1$ is obtained from T_{i-1} by the application of one local rule. Most of the time, we will be concerned with one transformation, i.e., the step from one tree T_i to the next T_{i+1}. To avoid too many indices, we use T to describe the situation before the transformation in question and T' to describe the situation after the transformation.

Now, when we say "let T be a tree with layer function $\mathcal{L}$ and local potential function Φ", T refers to a tree scheme, and we assume that for this tree scheme we have a definition of layers and potential such that for any sequence of transformations applied to an initial tree, if a transformation changes the tree from one state T to another T', then all the requirements to layer functions and local potential functions hold.

We say that post-processing is *amortized free* if every post-processing operation decreases the potential. Note that since an update is an application of a local rule, which by definition only involves a constant number of nodes, and since each node can be assigned at most a constant potential by a local potential function, the increase in potential by an update is at most a constant. Likewise, a post-processing operation also only involves a constant number of nodes with constant potential associated with each node, so negative potential changes are also constant.

To formally express a statement saying that the number of operations decrease exponentially as we move up in the tree, we must define formally where an

operation takes place.

Updates as well as operations may involve several layers. We now choose one of these layers and define it to be the layer where the transformation is carried out. We could choose any of the constant number of layers involved, but technically it is most convenient to choose a layer where the potential decreases when the transformation is carried out. Assume that $t \in \mathcal{U} \cup \mathcal{R}$ is a transformation, changing the tree T to T'. Since updates and operations are defined in terms of local rules, they can only affect the potential on a constant number of consecutive layers. We let $\Delta\Phi_j$ denote the potential change on layer j given by $\Phi(\ell_j(T')) - \Phi(\ell_j(T))$. Now the entire change in potential caused by t can be expressed as:

$$\sum_{j=j_0}^{j_0+k} \Delta\Phi_j$$

for some uniquely defined constants j_0 and k such that $\Delta\Phi_{j_0}$ and $\Delta\Phi_{j_0+k}$ are both non-zero. We say that the transformation *spans* $k+1$ layers. We emphasize that this means that the update or operation affects the potential on up to $k+1$ layers. The number of layers where structural modifications are made could easily be smaller.

The layer of the transformation t, changing T to T', is now defined as follows. For $j_0 \leq i \leq j_0 + k$, let

$$\mathcal{N}_i = \sum_{j=j_0}^{i} \Delta\Phi_j, \quad \text{and} \quad \mathcal{P}_i = \sum_{j=i+1}^{j_0+k} \Delta\Phi_j.$$

We define the set $\mathcal{I} = \{ j_0 \leq i \leq j_0 + k \mid \mathcal{N}_i < 0 \text{ and } \mathcal{P}_i \geq 0 \}$. In the theorem, we require that post-processing is amortized free. This means that the set $\mathcal{I}$ will be non-empty since every operation decreases the potential of the tree, and thus $j_0 + k \in \mathcal{I}$. We say that the operation t is performed on layer $\min \{ i \in \mathcal{I} \mid i \text{ minimizes } \mathcal{N}_i \}$.

Let #upd denote the number of updates made to the tree.

Theorem 1 Let T be a tree with layer function $\mathcal{L}$ and local potential function Φ. Assume that updates are made at the leaves and that post-processing is amortized free. Then there exist constants c_1, c_2, where $c_2 > 1$, such that the number of operations performed on layer i is bounded by $c_1 \dfrac{\#\text{upd}}{c_2^i}$, when starting with a tree with potential zero. ◇

Proof To show the exponentially decreasing post-processing, we want to bound the amount of potential that is moved upwards in the tree, and we want to bound the number of layers this potential is moved.

For a given application of an operation $t \in \mathcal{R}$, changing T to T', we define the following:

$$\mathcal{N}' = -\sum_{\Delta\Phi_i<0} \Delta\Phi_i, \quad \mathcal{P}' = \sum_{\Delta\Phi_i\geq 0} \Delta\Phi_i, \quad \text{and} \quad f = \frac{\mathcal{P}'}{\mathcal{N}'} < 1.$$

Thus, f is an upper bound on the fraction of potential which is moved upwards by this application of t.

Now let c_t denote the maximum f for any application of t to any tree. Thus, c_t is the largest possible fraction of potential that is moved upwards by t. Similarly, let k_t be the maximal k for any application of t to any tree such that t spans $k+1$ layers, and let $k_r = \max_{t\in\mathcal{R}}\{k_t\}$, $k_u = \max_{t\in\mathcal{U}}\{k_t\}$, and $c = \max_{t\in\mathcal{R}}\{c_t\}$.

Finally, let c_u be the maximal increase in potential which can be caused by any update to any tree. Since configurations are of constant size and we have a local potential function, this is a constant.

An update to the tree may increase the potential. For the purpose of this proof, we now discuss how to mark and trace this potential. If the potential has increased by an amount δ, this means that when the before-configuration $\mathcal{C}$ was replaced by the after-configuration $\mathcal{C}'$, the two of which form the local rule constituting the update, the sum of potential of nodes in $\mathcal{C}'$ is δ larger than the sum of potential of nodes in $\mathcal{C}$. For the sake of this proof, we mark this amount δ as coming from the current update. Note that δ may be spread out over several nodes. All of these nodes would then have (parts of) their potential marked as stemming from the current update. We do this for every update such that all potential in the tree is marked at all times. Thus, the potential of any node in the tree can be viewed as a sum of pieces of potential stemming from different updates.

When a post-processing operation is carried out, it must, by assumption, decrease the total potential. Thus, some of the marked potential disappears, and the remaining potential may be moved, meaning that the potential of one node increases while it decreases at another. We make no assumptions as to how this is done. For instance, for the potential on a given node stemming from a given update, some of it may disappear, some may remain at the same node, and the rest may be spread to a number of other nodes. It is now meaningful to talk about potential from a given update being created, being moved around in the tree, and eventually disappear.

Since post-processing operations decrease the potential, bounding the movement of potential up to a given level in the tree can be used to bound the number of operations carried out at that level.

We now bound the total amount of potential which can be created on layer i due to any one update $t \in \mathcal{U}$ by $c_u \cdot c^{\lceil \frac{i-k_u}{k_r} \rceil}$.

This is clearly true for $i \in [0, \ldots, k_u]$ since the expression becomes c_u, which by definition is a bound. What remains is to show that for all $j \geq 1$ and

$i \in [k_u + (j-1) \cdot k_r + 1, \ldots, k_u + j \cdot k_r]$, $c_u \cdot c^j$ is an upper bound.

Neither the layer function nor the local potential function are allowed to change outside the before- and after-configurations, and within the configurations, k_r is an upper bound on how far potential can be moved. Thus, no potential can reach any layer within this range unless it has been moved at least j times by an operation. Since any operation moves at most the fraction $c < 1$ of the potential, $c_u \cdot c^j$ is an upper bound on the potential that might reach layer i. Note that if an operation spans $k+1$ layers, then potential can be moved at most k layers, namely from the first to the last of the $k+1$ layers.

By straightforward calculations, we obtain:

$$c_u \cdot c^{\lceil \frac{i-k_u}{k_r} \rceil} \leq c_u \cdot \sqrt[k_r]{c}^{(i-k_u)} = \frac{c_u}{\sqrt[k_r]{c}^{k_u}} \sqrt[k_r]{c}^{i}.$$

Thus, after #upd operations, $\frac{c_u}{\sqrt[k_r]{c}^{k_u}} \sqrt[k_r]{c}^{i} \cdot \#\text{upd}$ is an upper bound on the total potential created on layer i.

Now by definition of the layer of an operation, if an operation is carried out on layer i, the potential decreases on layer i. Let $\overline{c_r}$ denote the maximum of all these negative potential changes (thus, we choose the smallest absolute value) taken over all applications of any operation to any tree. Let $c_r = -\overline{c_r}$. Thus, c_r is positive. In total, the potential on layer i is decreased by at least $c_r \cdot \#\text{ops}_i$, where $\#\text{ops}_i$ denotes the total number of operations performed on layer i. Combining these inequalities, we get

$$\begin{aligned} & 0 \leq \Phi(\ell_i(T)) \leq \tfrac{c_u}{\sqrt[k_r]{c}^{k_u}} \sqrt[k_r]{c}^{i} \cdot \#\text{upd} - c_r \cdot \#\text{ops}_i \\ \Downarrow & \\ & \#\text{ops}_i \leq \tfrac{c_u}{c_r \cdot \sqrt[k_r]{c}^{k_u}} \sqrt[k_r]{c}^{i} \cdot \#\text{upd}. \end{aligned}$$

If we let $c_1 = \frac{c_u}{c_r \cdot \sqrt[k_r]{c}^{k_u}}$ and $c_2 = \left(\sqrt[k_r]{c}\right)^{-1}$, the theorem follows. □

Observe that the layer function should define a non-constant number of layers in order for the theorem to provide new information. Moreover, if h is the length of the shortest path of T, then the layer of the top-most node in the tree must be in $O(h)$, since otherwise we would have that some configuration spans a non-constant number of layers, or updates are made at a non-constant distance from layer 0.

If the initial tree has a non-zero potential of Φ^{init}, then this potential can create at most a linear amount of extra work on each layer. Thus, as a crude upper bound, $\#\text{ops}_i \in O\left(\Phi^{init} + \frac{\#\text{upd}}{c_2^i}\right)$. This is particularly interesting if arbitrarily large trees with constant potential can be build, and according to [7], this is indeed possible for (a,b)-trees, for example. In such situations, the results hold immediately also starting with a nonempty tree.

3 Splitting Large Transformations

In this section, we present a construction to reduce the span of a transformation by one, by replacing the transformation by two new transformations with reduced span. This will reduce k_r, in particular, and therefore, potentially, provide better constants, as it can be seen from the formulas last in the proof of Theorem 1. The two new transformations should of course together preserve the semantics of the transformations as a whole, i.e., the result of applying the two operations one after another should be the same as applying the original transformation. By introducing nodes on every layer using the first transformation, we are capable of redefining the potential changes on every layer and we can obtain the result that for the two new transformations, the span of layers where potential is changed is strictly smaller than for the old transformation.

We emphasize that the intention is not to change the data structure which is implemented, but to argue that the data structure has desirable properties. Thus, the lemma below is an analytical tool. If we can prove good constants after the application of the lemma below, these constants will apply to the original structure as well, since fewer transformations are applied in the original structure.

Let t be a transformation on T. We use the notation $B \stackrel{t}{\longmapsto} A$ to denote that applying t to the before configuration B yields the after configuration A.

Lemma 1 Let T be a tree satisfying the conditions in Theorem 1 with layer function $\mathcal{L}$ and local potential function Φ. Let $t \in \mathcal{R}$ be a post-processing operation on T with $k_t \geq 1$. Then there exists a sequence of configurations $C_2, C_3, \ldots, C_{k_t}$, a sequence of operations $t_1, t_2, \ldots, t_{k_t}$, satisfying for all i, $1 \leq i \leq k_t : k_{t_i} = 1$, a layer function $\mathcal{L}^{new}$, and a potential function Φ^{new} on T, such that $B \stackrel{t_1}{\longmapsto} C_2 \stackrel{t_2}{\longmapsto} \ldots \stackrel{t_{k_t}}{\longmapsto} A$ and T satisfies Theorem 1 using $\mathcal{L}^{new}$ and Φ^{new}.

Proof The proof is by induction on k_t. The base case ($k_t = 1$) is trivial. Assume that the lemma is true for all $k_t < N$. Consider the case $k_t = N$. We show how to replace an operation $B \stackrel{t}{\longmapsto} A$ by two operations t_1 and t' such that $B \stackrel{t_1}{\longmapsto} C \stackrel{t'}{\longmapsto} A$ for some configuration C such that $k_{t_1} = 1$ and $k_{t'} = k_t - 1$. The lemma will follow by application of the induction hypothesis to t'.

Observe that by the definition of the local potential function, the potential on layer i cannot change, unless some node is present on this layer. Therefore, to control the potential on every layer from j_0 to $j_0 + k_t$, we must ensure that a node is present on each layer. We add $k_t + 1$ nodes as illustrated in Fig. 1. The middle configuration is configuration C referred to above. Each node is marked with the layer it belongs to. All other nodes belong to the layer they belonged to before the application of t_1.

Below, we define the potential function for the "intermediate state", where configuration B has been replaced by configuration C. Note that we have significant

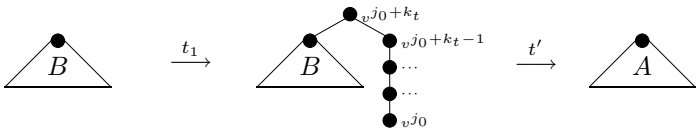

Figure 1: Transformations by first t_1 and then t'.

Operation	layer j_0	layer j_0+1	layer j_0+2	layer j_0+k_t
t_1	$\Delta\Phi_{j_0}$	$-x\cdot\Delta\Phi_{j_0}$	0	0
t'	0	$\Delta\Phi_{j_0+1}+x\cdot\Delta\Phi_{j_0}$	$\Delta\Phi_{j_0+2}$	$\Delta\Phi_{j_0+k_t}$
t	$\Delta\Phi_{j_0}$	$\Delta\Phi_{j_0+1}$	$\Delta\Phi_{j_0+2}$	$\Delta\Phi_{j_0+k_t}$

Table 2: Potential changes by t_1, t' and t.

freedom in choosing the potential function for this intermediate state. The only constraint is that all operations should decrease the potential. We decide on the form shown below, where we leave one constant, x, to be determined. See Table 2 for potential changes on the different layers.

$$\Phi^{new}(u)=\begin{cases}\Delta\Phi_{j_0} & \text{, if } u=v^{j_0}\\ -x\cdot\Delta\Phi_{j_0} & \text{, if } u=v^{j_0+1}\\ 0 & \text{, if } u\in\{v^{j_0+2},\ldots,v^{j_0+k}\}\\ \Phi(u) & \text{, otherwise}\end{cases}$$

Some notation will make it easier to express the constraints: for some operation p, let $\Delta\Phi^p$ denote the potential change incurred by applying p. We must show that we can choose x such that $\Delta\Phi^{t_1}<0$ and $\Delta\Phi^{t'}<0$.

We have that $\Delta\Phi^t=\sum_{i=j_0}^{j_0+k_t}\Delta\Phi_i$, $\Delta\Phi^{t_1}=\Delta\Phi_{j_0}-x\cdot\Delta\Phi_{j_0}$, and $\Delta\Phi^{t'}=\Delta\Phi^t-\Delta\Phi^{t_1}$.

We consider the cases $\Delta\Phi_{j_0}>0$ and $\Delta\Phi_{j_0}<0$ separately.

Assume that $\Delta\Phi_{j_0}<0$. Then $\Delta\Phi_{j_0}-x\cdot\Delta\Phi_{j_0}<0$ implies that $x<1$, while $\Delta\Phi^{t'}=\Delta\Phi^t-\Delta\Phi^{t_1}=\Delta\Phi^t-(\Delta\Phi_{j_0}-x\cdot\Delta\Phi_{j_0})<0$ implies that $x>1-\frac{\Delta\Phi^t}{\Delta\Phi_{j_0}}$. Since $\Delta\Phi^t<0$ and $\Delta\Phi_{j_0}<0$, we have that $1-\frac{\Delta\Phi^t}{\Delta\Phi_{j_0}}<1$, so we can choose an x such that $1-\frac{\Delta\Phi^t}{\Delta\Phi_{j_0}}<x<1$.

Analogously, assume that $\Delta\Phi_{j_0}>0$. Then $\Delta\Phi_{j_0}-x\cdot\Delta\Phi_{j_0}<0$ implies that $x>1$, while $\Delta\Phi^{t'}=\Delta\Phi^t-\Delta\Phi^{t_1}=\Delta\Phi^t-(\Delta\Phi_{j_0}-x\cdot\Delta\Phi_{j_0})<0$ implies that $x<1-\frac{\Delta\Phi^t}{\Delta\Phi_{j_0}}$. Since $\Delta\Phi^t<0$ and $\Delta\Phi_{j_0}>0$, we have that $1-\frac{\Delta\Phi^t}{\Delta\Phi_{j_0}}>1$, so we can choose an x such that $1<x<1-\frac{\Delta\Phi^t}{\Delta\Phi_{j_0}}$. □

We can reduce the span of updates in the same manner, but it is simpler because of fewer restrictions. If, in the above lemma, t was an update, which is replaced

by t_1 and t', then t_1 would be considered the update and t' a post-processing operation. Thus, t' must decrease the potential, but there are no constraints on t_1.

4 Partially Persistent Binary Trees

We consider partial persistence as in [4]. A study of partial persistence through 'pebble games' has been carried out in [3], and we use a similar presentation here.

A data structure is called *persistent* if it supports access to more than one version, and called *partially persistent* if only the newest of these versions can be modified. Persistence has been studied in connection with separate concrete data structures, but a general method has been devised in [4], which credits [11, 13] for initiating the study of general methods. In brief, a standard data structure is made persistent by extending the nodes with extra fields. In this context, only some of these fields, namely the so-called extra pointers, are of interest. The idea is that when a pointer is changed in the newest version, this can be done by adding this version-stamped information as an extra pointer. Searching routines will then check the version number and proceed in a direction depending on which version is currently being searched. When no more extra pointers are available in a node, all the newest information is copied into a new node which then has all its extra pointers available for future updates. However, now all the nodes pointing to the node which has just been copied must have their information updated, which will use extra pointers and may involve copying other nodes.

This method can be related to *pebble games*. Pebble games are played between two players and involve a number of piles. In a turn, the first player increases the number of pebbles on a pile, and the second decreases the number of pebbles, according to more detailed rules. The object of the game is the maximum number of pebbles in any pile. Player one must maximize this number and player two must minimize it. In [12], results are presented for many variations of such games and it is demonstrated how these games can be useful in the analysis of data structures, including analyses of persistence.

The more detailed rules can be given by arranging the piles in a tree structure and imposing various constraints on the moves. The connection to partial persistence using the node-copying method is made by interpreting a used extra pointer as a pebble.

We consider the following game played on a complete binary tree of height h. Each internal node has, besides pointers to its children, room for e pebbles, for some constant e. An update consists of updating a leaf, and putting a pebble on the parent (this is the equivalent of copying a leaf and using an extra-pointer at the parent in the node-copying model [4]). Post-processing consists of making sure no internal node is storing more than e pebbles. The possible operations are to remove all pebbles from a node and put one on the parent (this is the

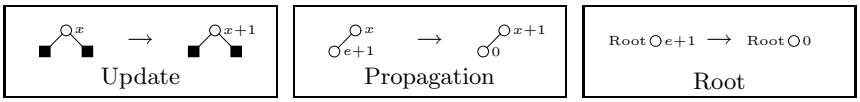

Figure 2: Transformations for pebble games.

Operation t	$\Delta\Phi_i$	$\Delta\Phi_{i+1}$	c_t
Propagation	$-(e+1)$	1	$\frac{1}{e+1}$
Root	$-(e+1)$	0	0

Table 3: Potential changes by operations on layer i for the pebble game.

equivalent of copying an internal node and using an extra-pointer at the parent in the node-copying model), or removing all pebbles from the root.

Since all transformations are described by local rules, we assume that nodes are capable of storing $e+1$ pebbles to represent the intermediate state between two consecutive operations.

The transformations are depicted in Figure 2. The set of transformations showed in this example (and any of the following) is only complete up to symmetric variants of the transformations. The number to the right of a node u is the number of pebbles on this node, denoted $\pi(u)$. We then define a layer function $\mathcal{L}(u) = h(u)$, where $h(u)$ denotes the height of the node u (leaves at height 0) and a local potential function $\Phi(u) = \pi(u)$. Using these transformations and the functions $\mathcal{L}$ and Φ, we have proven the following corollary.

Corollary 1 The number of persistence operations in a balanced binary tree is exponentially decreasing with respect to the distance from the leaves. $\diamond$

We now use the constructive part of the proof of Theorem 1 to derive constants for the exponential expression. Since an update puts a pebble on an internal node on layer 1, we immediately have: $c_u = 1$ and $k_u = 1$.

To derive the remaining constants, we analyze each operation one by one. A table such as Table 3 can be constructed for the potential changes by each operation. From the table we derive that $k_r = 1$, $c_r = e+1$, and $c = \frac{1}{e+1}$. Using the formula mentioned last in the proof of Theorem 1, the number of persistence operations at height i, denoted $\#\text{ops}_i$, is bounded by:

$$\#\text{ops}_i \leq \frac{1}{(e+1)\cdot\frac{1}{e+1}} \cdot \frac{\#\text{upd}}{(e+1)^i} = \frac{\#\text{upd}}{(e+1)^i},$$

where #upd denotes the number of updates to the tree.

Observe that the constant e is completely independent of the size of the tree, so the exponential expression in the denominator is independent of the size

of the subtree. Moreover, this bound is tight, which can easily be verified by considering a sequence of updates to the same leaf.

5 Designing Local Rules for Non-Local Structures

In this section, we consider examples of AVL-tree schemes. In the following, #upd denotes the number of updates. The number of rebalancing operations on layer i, with respect to some layer function, is denoted $\#\text{ops}_i$. Moreover, the search trees considered in the following are leaf-oriented. This means that all keys stored in internal nodes are routers used only to guide the search, and all elements are stored in the leaves. This is to satisfy the requirement that updates are always made near layer 0.

An AVL-tree, which is binary, is balanced if for all nodes, the height difference between its two subtrees is at most one. When an update (insertion or deletion) is carried out, this may temporarily disturb the balance criteria. The purpose of the rebalancing operations is to fix the problem created by the update and thereby return the tree to a balanced state. This can be done by applying a set of local operations to nodes on the path from the leave where the update was made up to the root.

For an introduction to balanced trees with further references to the original publication as well as to textbooks, see [2].

5.1 Semi-Dynamic AVL-Trees: Insertions

The complexity of sequences of insertions into an AVL-tree [1] is treated in [10], where it is shown that the number of post-processing (rebalancing) operations is exponentially decreasing in the height. In this section, we prove matching bounds using our construction.

Again, as noted in the previous section, transformations on the tree must be described by local rules, i.e., no transformation (including updates) can make changes to more than a constant number of nodes. In particular, we cannot immediately update the balance of all nodes on the search path of an update.

To satisfy this requirement, we introduce a special node that indicates that the height of the tree below has been increased by one. Rebalancing then consists of propagating this information upwards, and it ceases when the *critical node* [10] or the root is reached. The critical node is the bottommost node on the search path with non-zero balance. The set of transformations in terms of local rules is depicted in Figure 3. The special node described above is the white node. We do not show symmetric operations. Thus, all the operations shown have symmetric versions where directions and signs are reversed, i.e., left and right are switched

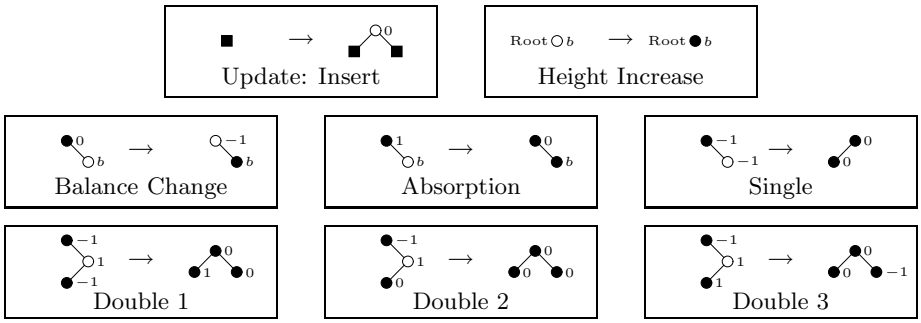

Figure 3: Insert and rebalancing operations on AVL-trees. The balance of a node is shown to the right of the node. For all operations: $b \in \{-1, 0, 1\}$. Symmetric transformations are omitted.

and all balance values are negated. Note that the potential function defined below only depends on the absolute value of the balance values.

We define the height of a node in a partially rebalanced AVL-tree as follows:

$$H(u) = \begin{cases} 0 & \text{, if } u \text{ is a leaf} \\ \max\{H(u.l), H(u.r)\} & \text{, if } u \text{ is white} \\ \max\{H(u.l), H(u.r)\} + 1 & \text{, otherwise} \end{cases}$$

where $u.l$ ($u.r$) denotes the left (right) child of the node u. The *balance* of a node, $b(u)$, is defined as $b(u) = H(u.l) - H(u.r)$. The number to the right of each node in Figure 3 is its balance. A balanced AVL-tree contains no white nodes and satisfies $b(u) \in \{-1, 0, 1\}$ for all nodes u.

The layer of a node is simply its height, $\mathcal{L}(u) = H(u)$. Observe that as a consequence of the definition of $\mathcal{L}(u)$, the layer of the topmost node of any operation (except *Height Increase*) is not changed by the operation.

Finally, we define a local potential function:

$$\Phi(u) = \mathbf{1}_{b(u)=0} + \phi^2 \cdot \mathbf{1}_{u \text{ is white}},$$

where u is an internal node, ϕ denotes $\frac{1+\sqrt{5}}{2}$, the golden ratio, and $\mathbf{1}_p$ is the indicator function, i.e., $\mathbf{1}_p = 1$ if p is true and $\mathbf{1}_p = 0$ otherwise. We define the potential of a leaf to be zero.

With the layer and potential functions we now have, by Theorem 1, that rebalancing is exponentially decreasing in AVL-trees with insertions. To determine the exact bound, we analyze each transformation on the tree. By analyzing each rebalancing operation one by one (see Figures 4, 5, 6, 7, 8, 9, and 10), Table 4 can be constructed.

We comment on the restriction $b \neq 0$ in Table 4 below. First, we note that we are aiming for the result that the number of rebalancing operations should

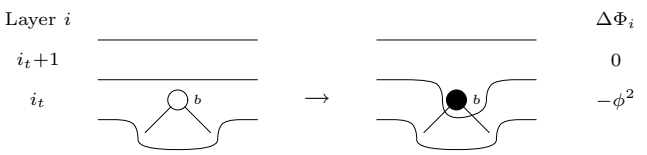

Figure 4: *Height Increase.*

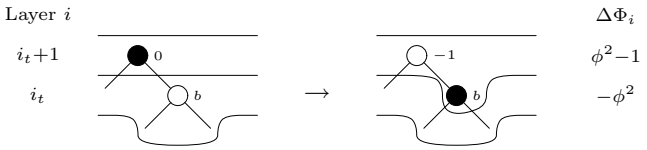

Figure 5: *Balance Change.*

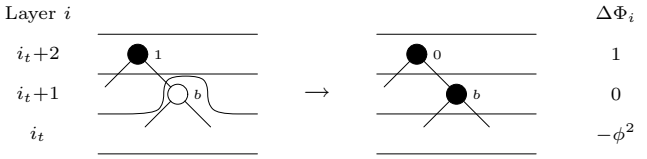

Figure 6: *Absorption* with $b \neq 0$.

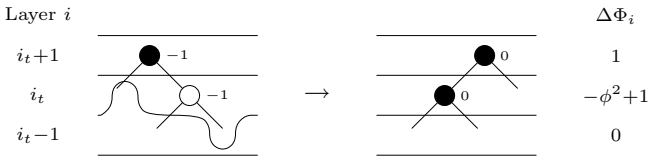

Figure 7: *Single.*

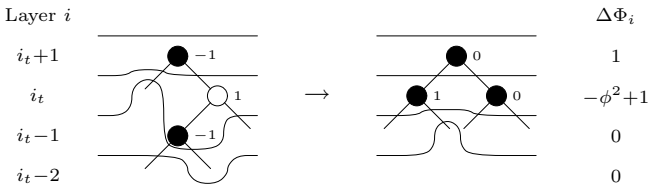

Figure 8: *Double 1.*

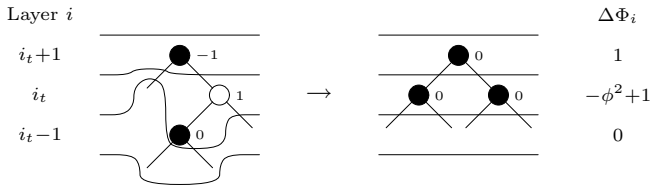

Figure 9: *Double 2.*

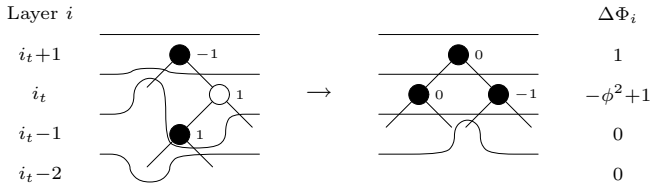

Figure 10: *Double 3.*

decrease with a factor of ϕ when moving from a layer to one closer to the root. The result seems to be within reach. The only problem (in Table 4) being *Absorption*, which spans three layers and therefore has $k_t > 1$. We apply Lemma 1 to split the operation in two. The result is shown schematically in Tables 5.

The update *Insert* increases the potential of just one layer as shown in Table 6.

We now comment on the assumption of $b \neq 0$ on the operations *Balance Change* and *Absorption*. The reason for analyzing the case $b = 0$ separately is that in that case, those two operations do not give as good results, i.e., the fraction c_t which is $\frac{1}{\phi}$ for the other cases and other operations are closer to one for the cases $b = 0$. However, considering the operations in Figure 3, one can observe that no rebalancing operation introduces a white node with balance zero. Such a node can only be introduced by an *Insert* and will then be removed by either *Balance Change* or *Absorption*.

To obtain the best possible result we consider *Insert* followed by either *Balance Change* or *Absorption* as one update operation. The combined effect of *Insert* followed by either *Balance Change* or *Absorption* is shown in Figures 11 and 12.

We get the best result if potential is only introduced on the lowest layer, so we split these two operations as shown in Tables 7 and 8. When splitting an update, only the first of the new transformations is considered an update. The others are rebalancing operations. The amount of potential introduced on layer zero by the new updates is chosen such that the rebalancing operations can decrease potential by a factor ϕ when moving it one layer up.

Though this whole process of first combining some operations and then splitting

Operation t	$\Delta\Phi_{i_t}$	$\Delta\Phi_{i_t+1}$	$\Delta\Phi_{i_t+2}$	c_t
Height Increase	$-\phi^2$	0	0	0
Balance Change ($b \neq 0$)	$-\phi^2$	$\phi^2 - 1$	0	$\frac{1}{\phi}$
Absorption ($b \neq 0$)	$-\phi^2$	0	1	$\frac{1}{\phi^2}$
Single, Double 1–3	$-\phi^2 + 1$	1	0	$\frac{1}{\phi}$

Table 4: Potential changes by rebalancing operations on AVL-trees carried out at layer i_t.

Operation t	$\Delta\Phi_{i_t}$	$\Delta\Phi_{i_t+1}$	$\Delta\Phi_{i_t+2}$	c_t
Absorption_1	$-\phi^2$	ϕ	0	$\frac{1}{\phi}$
Absorption_2	0	$-\phi$	1	$\frac{1}{\phi}$
Absorption ($b \neq 0$)	$-\phi^2$	0	1	$\frac{1}{\phi^2}$

Table 5: Potential changes by resulting transformations after splitting *Absorption*.

them in a new way can seem somewhat artificial, it is clear that it only affects the lowest layers. Nothing has changed further up in the tree, so the result will also hold for the original set of operations.

We can now apply the formula developed in Theorem 1. By examining the updates, *Insert*, Insert_1, and Insert_2, we find $c_u = \phi^3$ and $k_u = 0$. From Tables 4, 7, and 8, we have the values for all rebalancing operations, and find that $c = \frac{1}{\phi}$, $k_r = 2$, and $c_r = \phi$.

Finally, we find the number of rebalancing operations at height i by plotting into the formulas last in the proof of Theorem 1: $\#\text{ops}_i \leq \frac{\phi^3}{\phi \cdot \sqrt[1]{\frac{1}{\phi}^0}} \cdot \frac{\#\text{upd}}{\sqrt[1]{\phi^i}} = \phi^2 \cdot \frac{\#\text{upd}}{\phi^i}$, matching the result of [10].

We have proven the following theorem:

Theorem 2 For semi-dynamic AVL-trees where the only allowed update is insertion, rebalancing is exponentially decreasing and the number of rebalancing operations carried out at layer i is bounded by $\phi^2 \cdot \frac{\#\text{upd}}{\phi^i}$.

5.2 Semi-dynamic AVL-trees: Deletions

The amortized complexity of sequences of deletions with rebalancing in AVL-trees is treated in [14]. However, unlike for sequences of insertions, to our knowledge, rebalancing has not been shown to be exponentially decreasing. We show that in this section, and show that the constants obtained using Theorem 1 and the technique described in the previous section are tight.

Transformation t	$\Delta\Phi_0$
Insert	$1+\phi^2$

Table 6: *Insert*.

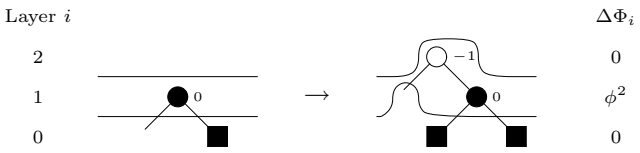

Figure 11: Combined effect of *Insert* and *Balance Change*.

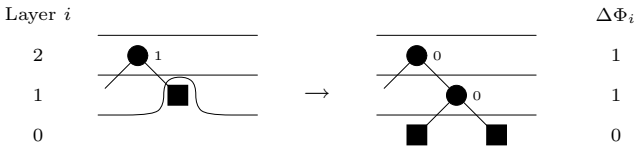

Figure 12: Combined effect of *Insert* and *Absorption*.

Transformation t	$\Delta\Phi_0$	$\Delta\Phi_1$	c_t
Insert_1	ϕ^3	0	
IBC_1	$-\phi^3$	ϕ^2	$\frac{1}{\phi}$
Insert; Balance Change	0	ϕ^2	

Table 7: Potential changes by resulting transformations after splitting the combined *Insert* and *Balance Change*.

Transformation t	$\Delta\Phi_0$	$\Delta\Phi_1$	$\Delta\Phi_2$	c_t
Insert_2	$\phi(\phi+1)$	0	0	
IA_1	$-\phi(\phi+1)$	$\phi+1$	0	$\frac{1}{\phi}$
IA_2	0	$-\phi$	1	$\frac{1}{\phi}$
Insert; Absorption	0	1	1	

Table 8: Potential changes by resulting transformations after splitting the combined *Insert* and *Balance Change*.

Again, as in the previous section, we use a special node (this time indicating that the height of the subtree below has decreased by one) to capture the notion of a partially rebalanced tree. Again rebalancing is done by propagating this information upwards, and it ceases when the node is removed by some transformation. The set of transformations in terms of local rules is depicted in Figure 13. The special node described above is the white node. As in the previous section, we do not show symmetric operations. Thus, all the operations shown have symmetric versions where directions and signs are reversed, i.e., left and right are switched and all balance values are negated. As previously, the potential function only depends on the absolute value of the balance values.

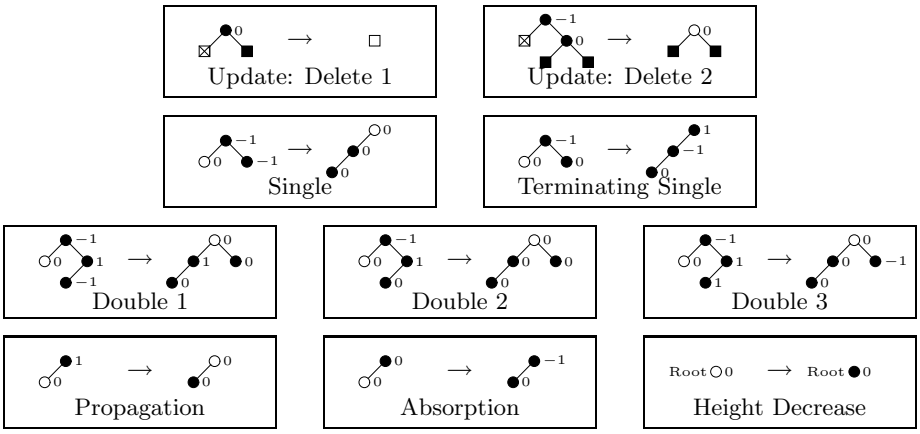

Figure 13: Operations on AVL-trees to handle sequences of deletions. Symmetric transformations are omitted.

We define the height $H(u)$ and the *adjusted height* $AH(u)$ of a node u as follows:

$$H(u) = \begin{cases} 0 & \text{, if } u \text{ is a leaf} \\ \max\{AH(u.l), AH(u.r)\} + 1 & \text{, otherwise} \end{cases}$$

$$AH(u) = H(u) + \mathbf{1}_{u \text{ is white}}$$

Here $u.l$ ($u.r$) denotes the left (right) child of the node u. The *balance* of a node $b(u)$ is defined as $b(u) = AH(u.l) - AH(u.r)$. The number to the right of each node in Figure 13 is its balance. Again, a balanced AVL-tree contains no white nodes and satisfies $b(u) \in \{-1, 0, 1\}$ for all nodes u.

The layer function $\mathcal{L}(u)$ is defined to be the adjusted height, i.e, $\mathcal{L}(u) = AH(u)$. Again, as a consequence of this definition, the layer of the top-most node is not changed by any operation, except for *Height Decrease*.

Finally, the local potential function is defined as follows:

$$\Phi(u) = x \cdot \mathbf{1}_{|b(u)|=1} + y \cdot \mathbf{1}_{u \text{ is white}},$$

where u is an internal node. We define the balance of a leaf to be zero, so the potential is 0 or y. In the formula, x and y are positive constants such that $y > x$. One can easily verify that this suffices to show amortized free rebalancing. Thus, Theorem 1 applies. To determine the constants, we analyze each transformation one by one, as in the previous section. This results in Figures 14, 15, 16, 17, 18, 19, 20, 21, 22, and 23 from which we extract the information in Table 9.

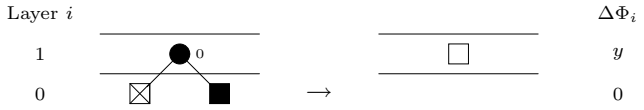

Figure 14: *Delete 1.*

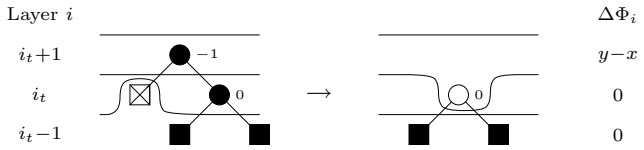

Figure 15: *Delete 2.*

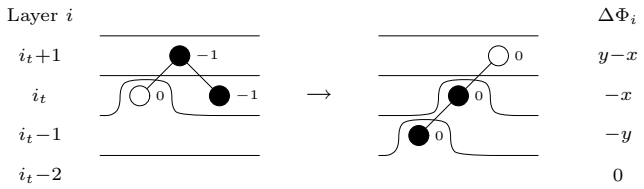

Figure 16: *Single.*

From the table we find that $k_r = k_u = 2$. However, this will again not yield the best c and therefore not the best c_2. For *Single*, *Double 1–3*, and *Delete 2* having $k_t > 1$, we apply Lemma 1 to reduce k_t. See Tables 10 and 11 for the new potential changes.

We now find from the tables that $k_r = 1$ and $c = \max\left\{\frac{x}{y}, \frac{y-x}{y}\right\} = \frac{1}{2}$ (by choosing $y = 2x$) and $c_r = y$. By examination of the *Delete* transformations,

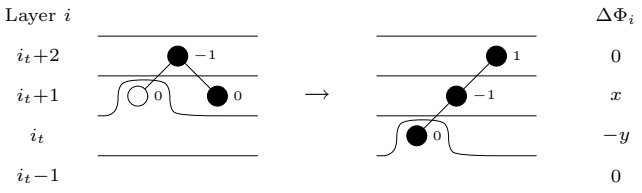

Figure 17: *Terminating Single.*

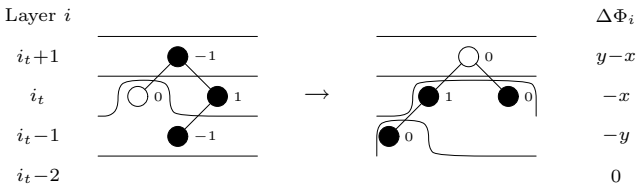

Figure 18: *Double 1.*

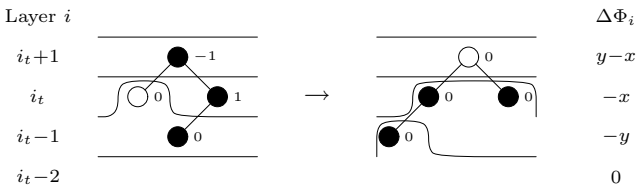

Figure 19: *Double 2.*

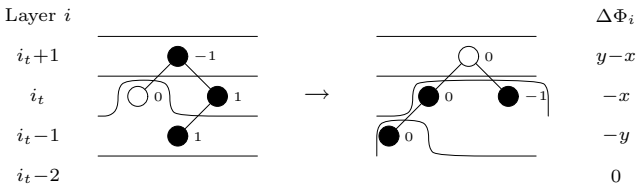

Figure 20: *Double 3.*

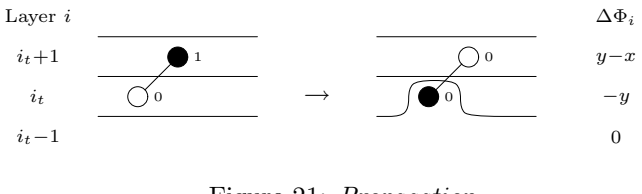

Figure 21: *Propagation.*

Figure 22: *Absorption.*

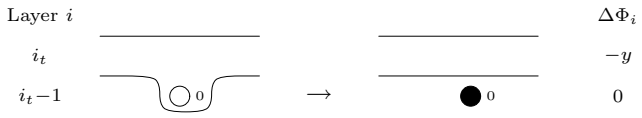

Figure 23: *Height Decrease.*

we find that $k_u = 1$ and $c_u = y$. Hence,

$$\#\text{ops}_i \leq \frac{y}{y \cdot \sqrt[1]{\frac{1}{2}}^1} \cdot \frac{\#\text{upd}}{\sqrt[1]{2}^i} = 2 \cdot \frac{\#\text{upd}}{2^i}.$$

Moreover, this bound is tight. Consider a complete binary tree of height h. Thus, all internal nodes have balance 0 and the tree has zero potential. Delete every second element. The effect of this is illustrated by showing the lower left corner of the tree in Figure 24. Notice that the first deletion changes the

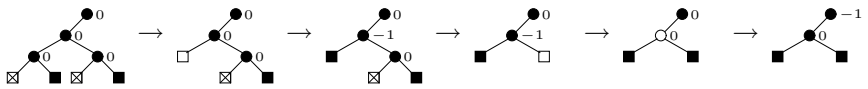

Figure 24: Sequence of operations on AVL-tree: Delete 1, Absorption, Delete 1, Propagation, Absorption.

balance of the grandparent of the deleted node to a -1 (via an Absorption). This means that the next deletion results in a Propagation. When this is followed by an Absorption, a balance of -1 is introduced at the level one higher, and the process repeats.

If we let n denote the number of leaves, then the $n/2$ deletions give rise to $n/4$ Absorption operations and $n/4$ Propagation operations at level 1; a total of $n/2$ operations. As the effects proceeds up the tree, the number of operations at a given level is halved compared to the previous. Thus, at level i, we carry out $\frac{n}{2^i} = 2\frac{n/2}{2^i} = 2\frac{\#\text{upd}}{2^i}$ operations.

Operation t	$\Delta\Phi$	$\Delta\Phi_{i_t-1}$	$\Delta\Phi_{i_t}$	$\Delta\Phi_{i_t+1}$	c_t
Term. Single	$x-y$	0	$-y$	x	$\frac{x}{y}$
Absorption	$x-y$	0	$-y$	x	$\frac{x}{y}$
Propagation	$-x$	0	$-y$	$y-x$	$\frac{y-x}{y}$
Height Decrease	$-y$	0	$-y$	0	0
Single, Double 1–3	$-2x$	$-y$	$-x$	$y-x$	$\frac{y-x}{y+x}$

Table 9: Potential changes by rebalancing operations on AVL-trees.

Operation t	$\Delta\Phi_{i_t-1}$	$\Delta\Phi_{i_t}$	$\Delta\Phi_{i_t+1}$	c_t
Single_1, Double 1_1–3_1	$-y$	$y-x$	0	$\frac{y-x}{y}$
Single_2, Double 1_2–3_2	0	$-y$	$y-x$	$\frac{y-x}{y}$
Single, Double 1–3	$-y$	$-x$	$y-x$	$\frac{y-x}{y+x}$

Table 10: Potential changes by resulting transformations after splitting *Single* and *Double 1–3*. The layer i_t refers to the layer of *Single* (and *Double 1–3*).

We have proven the following theorem:

Theorem 3 For semi-dynamic AVL-trees where the only allowed update is deletion, rebalancing is exponentially decreasing and the number of rebalancing operations carried out at layer i is bounded by $2 \cdot \frac{\#\text{upd}}{2^i}$. Furthermore, there exists sequences of operations where this bound is met.

6 Concluding Remarks

A reasonable question to consider is whether or not the theorem has found its right form. In particular, are all the requirements necessary?

Clearly, if operations are not local, then every operation could involve the root. Furthermore, requiring that there is a local potential function which assigns at most a constant amount of potential to each node gives a similar type of control over the progress of operations. Without this requirement, one can construct the scenario where every $(\log n)$'th operation progresses all the way to the root, while all other operations finish immediately at the leaves. Then, assuming that the height of the tree (as well as the number of layers) is $\Theta(\log n)$, the operation can clearly be made amortized free, while the number of operations cannot decrease exponentially, since the root is accessed too often.

At least this demonstrates a local form of completeness of our approach, in that no single requirement can be relaxed in isolation.

Finally, more results and examples of applications of our main theorem can be found in [6], where it is demonstrated how the techniques can be applied to

Operation t	$\Delta\Phi_1$	$\Delta\Phi_2$	c_t
Delete 2_1	y	0	
Delete 2_2	$-y$	$y-x$	$\frac{y-x}{y}$
Delete 2	0	$y-x$	

Table 11: Potential changes by resulting transformations after splitting *Delete 2*.

search trees with relaxed balance and to (a,b)-trees in particular, and results matching the ones developed specifically for (a,b)-trees [5, 9] are obtained. For references on (a,b)-trees with relaxed balance, see [2, 8].

Acknowledgment

The authors would like to thank the anonymous referees for many insightful comments and suggestions for improving papers.

References

[1] Georgiĭ M. Adel'son-Vel'skiĭ and Evgeniĭ M. Landis. An Algorithm for the Organisation of Information. *Doklady Akadamii Nauk SSSR*, 146:263–266, 1962. In Russian. English translation in *Soviet Math. Doklady*, 3:1259-1263, 1962.

[2] Arne Andersson, Rolf Fagerberg, and Kim S. Larsen. Balanced Binary Search Trees. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, Chapman & Hall/CRC Computer & Information Science Series, pages **10**–1–**10**–28. CRC Press, 2005.

[3] Paul F. Dietz and Rajeev Raman. Persistence, Amortization and Randomization. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 78–88, 1991.

[4] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making Data Structures Persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.

[5] Scott Huddleston and Kurt Mehlhorn. A New Data Structure for Representing Sorted Lists. *Acta Informatica*, 17:157–184, 1982.

[6] Lars Jacobsen. *Search Trees with Local Rules.* PhD thesis, Department of Mathematics and Computer Science, University of Southern Denmark, 2001.

[7] Lars Jacobsen, Kim S. Larsen, and Morten N. Nielsen. On the Existence and Construction of Non-Extreme (a, b)-Trees. *Information Processing Letters*, 84(2):69–73, 2002.

[8] Kim S. Larsen. Relaxed Multi-Way Trees with Group Updates. *Journal of Computer and System Sciences*, 66(4):657–670, 2003.

[9] Kurt Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms*. Springer-Verlag, 1984.

[10] Kurt Mehlhorn and Athanasios Tsakalidis. An Amortized Analysis of Insertions into AVL-Trees. *SIAM Journal on Computing*, 15(1):22–33, 1986.

[11] M. H. Overmars. Searching in the Past II: General Transforms. Technical Report RUU-CS-81-9, Department of Computer Science, University of Utrecht, The Netherlands, 1981.

[12] Rajeev Raman. *Eliminating Amortization: On Data Structures with Guaranteed Response Time*. PhD thesis, Department of Computer Science, University of Rochester, Rochester, New York, 1992.

[13] N. Sarnak. *Persistent Data Structures*. PhD thesis, Department of Computer Science, New York University, New York, 1986.

[14] Athanasios K. Tsakalidis. Rebalancing Operations for Deletions in AVL-Trees. *R.A.I.R.O. Informatique Théorique*, 19(4):323–329, 1985.