Università di Pisa

Dipartimento di Informatica

Technical Report: TR-05-03

k-Restricted Rotation with an Application to Search Tree Rebalancing

Alejandro Almeida Ruiz
Antonio Mesa Enriquez
Fabrizio Luccio
Linda Pagli

ADDRESS: via F. Buonarroti 2, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

k-Restricted Rotation with an Application to Search Tree Rebalancing

Alejandro Almeida Ruiz * Fabrizio Luccio †
Antonio Mesa Enriquez ‡ Linda Pagli§

Abstract

The *restricted rotation distance* $d_R(S,T)$ between two binary trees S, T of n vertices is the minimum number of rotations by which S can be transformed into T, where rotations can only take place at the root of the tree, or at the right child of the root. A sharp upper bound $d_R(S,T) \leq 4n-8$ is known, based on the word metric of Thompson's group. We refine this bound to a sharp $d_R(S,T) \leq 4n-8-\rho_S-\rho_T$, where ρ_S and ρ_T are the numbers of vertices in the rightmost vertex chains of the two trees, by means of a very simple transformation algorithm based on elementary properties of trees. We then generalize the concept of restricted rotation to *k-restricted* rotation, by allowing rotations to take place at all the vertices of the highest k levels of the tree. For $k=2$ we show that not much is gained in the worst case, although the classical problem of rebalancing an AVL tree can be solved efficiently, in particular rebalancing after vertex deletion requires $O(\log n)$ rotations as in the standard algorithm. Finding significant bounds and applications for $k \geq 3$ is open.

Keywords: Rotation, Rotation distance, Binary tree, Search tree, AVL tree, Rebalancing, Data structures, Design of algorithms.

1 Tree transformation via rotations.

We consider rooted binary trees of n vertices, simply called *trees* in the following. These trees are relevant in different fields of computing, in particular they are used as *search trees* storing n keys at the vertices in *infix order*, i.e. the keys in the left (respectively, right) subtree of each vertex v precede (respectively, follow) the key

*Departamento de Informatica, Universidad de Matanzas, alejandro.almeida@umcc.cu
†Dipartimento di Informatica, Università di Pisa, luccio@di.unipi.it
‡Facultad de Matemàtica y Computación, Universidad de la Habana, tonymesa@matcom.uh.cu
§Dipartimento di Informatica, Università di Pisa, pagli@di.unipi.it

in v according to a given ordering [3]. Although completely general the following considerations are carried out having search trees in mind.

Rotations are well known local changes in the tree structure, preserving key ordering in search trees. A *right rotation* at a vertex y raises its left child x to the place of y while y becomes the right child of x, and the original right child of x becomes the left child of y (the rest of the tree remains unchanged). A *left rotation* is symmetrical. If applied to vertex x, after the right rotation, it reconstructs the original tree. Rotations maintain the infix ordering of the keys, and are used to keep search trees *balanced*. That is, the height of a tree, hence the search time for a key, remains of $O(\log n)$ after insertions or deletions of vertices [3].

The *rotation distance* $d(S,T)$ between two trees S, T of n vertices is the minimum number of rotations by which S can be transformed into T, where the rotations can take place at any vertex. As right and left rotation are one the inverse of the other, we have $d(S,T) = d(T,S)$. In a scholarly paper [9], Sleator, Tarjan, and Thurston proved that $d(S,T) \leq 2n-6$ for any pair S, T using hyperbolic geometry, and that this bound is sharp for large values of n. Mäkinen [6] showed that slightly weaker results can be obtained by simple arguments, and Luccio and Pagli [4] then gave an elementary constructive proof for the upper bound of [9]. No polynomial-time algorithm is known to compute $d(S,T)$, but estimates of this value were given by Pallo [7] and Rogers [8].

The new *restricted rotation distance* $d_R(S,T)$ was introduced by Cleary [1], where rotations can take place only at the root of the tree, or at the right child of the root. Cleary and Taback [2] then strongly improved the results of [1], proving a sharp upper bound $d_R(S,T) \leq 4n-8$ for any pair S, T, and a stricter lower bound if S and T satisfy a special condition. Both these works are based on the word metric of Thompson's group.

The works of Cleary and Taback are remarkable for several reasons. First, the upper bound for $d_R(S,T)$ is only about twice the one for $d(S,T)$ "despite of the fact that it may take many rotations to move a vertex to one of the places where rotations are allowed" [1]. Second, the properties of an abstract algebric structure are transferred into the theory of data structures with brillant results. Finally, and important from an information processing point of view, allowing rotations only at selected vertices is relevant for handling very large files where only a fraction of the vertices are accessible in main memory where the rotations are performed.

Given an arbitrary tree T, let the *left arm* (respectively, *right arm*) of a vertex v be the longest sequence of vertices reached from v when descending T via left (respectively, right) edges only. The arms of the root will be simply called the left and right arm of T, and the number of vertices in such arms will be denoted by λ_T and ρ_T, respectively (see figure 1). The union of the left arm, the right arm, and the root, is called the *border* of T. In this work we do the following:

1. We refine the upper bound $d_R(S,T) \leq 4n-8$ of [2] to $d_R(S,T) \leq 4n-8-$

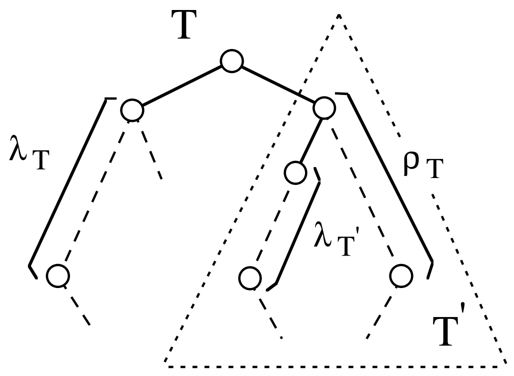

Figure 1: The left and right arms of a tree T, of lengths λ_T and ρ_T. T' is a subtree of T, with left arm of length $\lambda_{T'}$.

$\rho_S - \rho_T$, and prove that the new bound is sharp.[1] The bound of [2] can then be seen as a special case of the new one. The two bounds coincide for $\rho_S = \rho_T = 0$, but we shall see in the next section that the probability of this case to occur goes to zero for increasing n, and is strictly equal to zero for balanced search trees. We underline that the new upper bound is the result of a very simple algorithm to transform S into T, based on elementary properties of trees.

2. The concept of restricted rotation is generalized by performing rotations at all the vertices in the highest k levels of the tree, as motivated by searching into large files. We study the new distance for $k = 2$, that is the rotations are allowed only at the root and at both its children, showing that not much is necessarily gained over the previous case. We then discuss how the case $k \geq 3$ can be approached, although the problem will be left substantially open.

3. We finally show how restricted rotations are sufficient for rebalancing AVL trees efficiently, giving an insertion and a deletion algorithms that use rotations at the highest $k = 2$ levels only.

2 The basic algorithm.

To present our rotation algorithm we pose:

Definition 1 *A tree T is an* (i,j)-chain*, with $i, j \geq 0$ and $i + j + 1 = n$, if all the vertices of T lie on the border, with $\lambda_T = i$ and $\rho_T = j$.*

First we give a basic algorithm to transform any given tree into a fixed reference tree R consisting of an (n-3,2)-chain. As the transformation process can be reverted by exchanging right and left rotations, transforming S into T will amount to

[1]This bound was anticipated in [5] without a matching lower bound.

transforming both S and T into R, then reverting the second transformation to get $S \rightarrow R \rightarrow T$.

Let *rot1-right*, *rot1-left* respectively denote the right and left rotation at the root, and let *rot2-right*, *rot2-left* denote the same rotations at the right child of the root. We have:

Algorithm 1 *Transforming of a tree T with $n \geq 3$ into an (n-3,2)-chain. T' is the subtree rooted at the right child of the root of T (see figure 1). $\overline{T}$ and $\overline{T}'$ denote the trees generated at each step of the transformation, starting with $\overline{T} = T$ and $\overline{T}' = T'$.*

while $\lambda_{\overline{T}} > 0$ **do** rot1-right;

while $\overline{T}' \neq$ (0,1)-chain **do**

  if $\lambda_{\overline{T}'} > 0$ **then** rot2-right **else** rot1-left.

Theorem 1 *Algorithm 1 transforms any tree T with $n \geq 3$ into an (n-3,2)-chain performing $2n - 4 - \rho_T$ rotations.*

Proof The first **while** cycle transforms T into a new tree $\overline{T}$ with an empty left arm, and a right arm of length $\lambda_T + \rho_T$ to whose left are appended all the subtrees of the two arms of T. In the following **while** cycle either the root of one subtree is brought into the right arm of $\overline{T}$ and its left and right subtrees are directly appended to that arm (**then** branch), or the top vertex of the right arm is lifted to the root of $\overline{T}$ (**else** branch). Once a vertex is in the border of T it never leaves from it, therefore the algorithm eventually transforms T into a (n-3,2)-chain. Let now count the total number of rotations.

Let n_ℓ, n_r be the total number of vertices contained in the subtrees appended to the right of the left arm of T, and to the left of the right arm of T, respectively. We have: $n = 1 + \lambda_T + n_\ell + \rho_T + n_r$. For a rotation at a vertex y let us focus on its child x which is lifted to the place of y. x will be called the *actor* of the rotation. We distinguish among four types of vertices. (i) Each vertex of a subtree originally appended to the left or to the right arm of T is the actor of two rotations during the second **while** cycle, one for being inserted into the right arm (**then** branch), and one to be lifted to the root (**else** branch). (ii) The vertices originally contained in the left arm of T are firstly moved to the root as actors in one rotation during the first **while** cycle, thus being transferred into the right arm. Then each one of them returns to the root as actor of another rotation during the second **while** cycle, with the exception of the leftmost one (i.e., the smallest in the ordering) that resides in the root at the end of the first **while** cycle. (iii) The original root of T is actor of only one rotation when is lifted to the root of $\overline{T}$ during the second **while** cycle. (iv) The vertices originally contained in the right arm of T are actors in only one rotation during the second **while** cycle. The only exception to the above is with the final two vertices in $\overline{T}' =$ (0,1)-chain that undertake one rotation less. The total number of rotations is then: $2(n_\ell + n_r) + (2\lambda_T - 1) + 1 + \rho_T - 2 = 4n - 4 - \rho_T$. *Q.E.D.*

A tree S can be transformed into another tree T by applying algorithm 1 to transform S into an (n-3,2)-chain R, then transforming R into T by reverting the rotations of algorithm 1 that would transform T into R. We have:

Corollary 1 *For two arbitrary trees S, T with $n \geq 3$ we have: $d_R(S,T) \leq 4n - 8 - \rho_S - \rho_T$.*

The upper bound of corollary 1 coincides with the one of [2] for $\rho_S = \rho_T = 0$, that is the roots of the two trees have no right descendants. In fact, this is the only condition under which the bound of [2] is sharp, and is very unlikely or even impossible to occur. For trees built at random, the probability P_0 of having $\rho_S = \rho_T = 0$ decreases exponentially with n. For search trees the major distinction is between trees built by random key insertions from the root, and balanced trees. In the first case we have $P_0 = 1/n^2$, as this corresponds to having the maximum key being inserted as the first item in both trees. For *AVL* balanced trees [3] it can be easily shown that both ρ_S and ρ_T must fall into the positive interval $[h/2, h-1]$, where h is the index of the h-th Fibonacci number F_h such that $F_h \leq n+1 < F_{h+2}$. Then we have the stricter upper bound: $d_R(S,T) \leq 4n - 8 - \Omega(\lg n)$.

Due to the result of [2], the bound of Corollary 1 is sharp because, for any n, there are trees with $\rho_S = \rho_T = 0$. However we shall proof that the bound is also sharp for arbitrary values of ρ_S and ρ_T. To this end we start with a basic result. Identify the vertices of a tree S with the integers 1 to n in infix order, and let $\Pi^S(v)$ be the sequence of vertices in the path from the root to a vertex v. We have:

Lemma 1 *Let S, T be two trees to be transformed into one another. And let v, $w = v + 1$ be two vertices such that $v \in \Pi^S(w)$ and $w \in \Pi^T(v)$. The transformation process must pass through an intermediate tree whose border contains v and w.*

Proof Consider transforming S into T, as the opposite case is symmetrical. If v and w are in the border of S we are done. Otherwise, let Q be any arbitrary intermediate tree in the transformation process, with v and/or w not in the border of Q, and $v \in \Pi^Q(w)$, as in S. We prove that no rotation can transform Q into a new tree Q' with $w \in \Pi^{Q'}(v)$ as required in T. Since v and w are consecutive integers and $v \in \Pi^Q(w)$, w is the smallest vertex in the right subtree of v in Q, and w cannot be in the border of Q if v is not in that border. Then we have two cases: 1) v and w are not in the border of Q; and 2) v, and not w, is in the border of Q. Case 1 is easy to examine, as any rotation can invert the relative ordering of two vertices in a path Π only if at least one of them is in the border. Case 2 has the following subcases, one of which will be also illustrated in figure 2 (the others are similar or easier).

2.1) v is the root of Q, hence w is in the left subtree α of the right child x of v. rot1-right lowers v one level to the right together with its right subtree containing

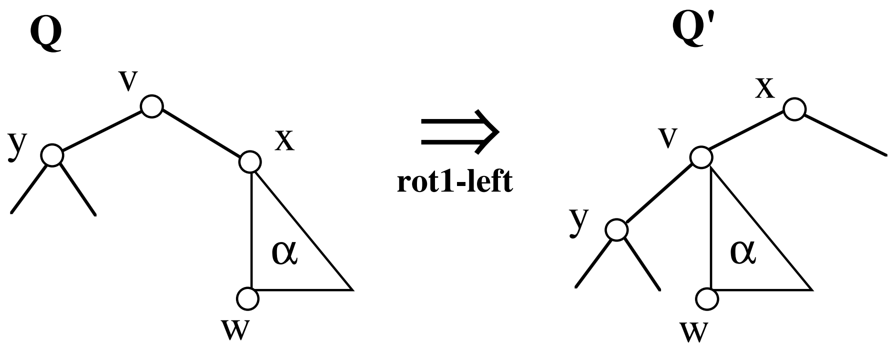

Figure 2: Illustation of an instance of case 2.1 of Lemma 1.

w. rot1-left lowers v one level to the left, acquiring α, that contains w, as its right subtree (see figure 2). rot2-right and rot2-left leave v at the root, hence we still have $v \in \Pi^{Q'}(w)$.

2.2) v is the left child of the root x, and w is in the right subtree β of v. rot1-right lifts v to the root, hence $v \in \Pi^{Q'}(w)$. rot1-left lowers v one level to the left together with its right subtree containing w. rot2-right and rot2-left do not affect the relative ordering of v, w in Π.

2.3) v is the right child of the root x, and w is in the left subtree γ of the right child z of v. rot1-right and rot2-right lower v one level to the right together with its right subtree containing w. rot1-left lifts v to the root, hence $v \in \Pi^{Q'}(w)$. rot2-left lowers v one level to the left, acquiring γ, that contains w, as its right subtree.

2.4) v is the right child of the right child x of the root, and w is in the left subtree δ of the right child z of v. rot1-right and rot1-left do not affect the relative ordering of v, w in Π. rot2-right lowers v one level to the right together with its right subtree containing w. rot2-left lowers v one level to the left, acquiring δ, that contains w, as its right subtree. *Q.E.D.*

It is interesting to note that the only way of inverting the relative ordering of v and w in a path Π, as required in the transformation of S into T, is passing through an intermediate tree Q with v and w on the border, where either v is the root, w is the right child of v, and rot1-left is applied to Q; or v is the right child of the root, w is the right child of v, and rot2-left is applied to Q.

We now prove that the bound of Corollary 1 is sharp for arbitrary values of ρ_S and ρ_T. We have:

Theorem 2 *For any $n \geq 3$ there are trees S, T with arbitrary values of ρ_S, ρ_T such that $d_R(S,T) = 4n - 8 - \rho_S - \rho_T$.*

Proof For $n = 3$ the proof trivially goes by exhaustion. For $n \geq 4$, consider the two trees S, T of figure 3, with arbitrary values $\rho_S = s$ and $\rho_T = t$. The two vertices $n-2$, $n-1$ are in the same condition as v, w of Lemma 1 because $(n-2) \in \Pi^S(n-1)$ and $(n-1) \in \Pi^T(n-2)$. Therefore S must be transformed into an intermediate tree Q with both $n-2$ and $n-1$ on the border. Since the subsequent rotations for transforming Q into T can be inverted, $d_R(S,T)$ can be computed as the minimum number of rotations for transforming both S and T into the same Q.

For bringing vertex $n-1$ to the border of S, this vertex must be lifted at least up to level 3. This requires an initial sequence of at least $\rho_S - 1$ rot1-left rotations to lift the chain $s+1, s+2, \ldots, n-1$, bringing $s+1$ at level 3. The vertices of the chain can be moved from it, one after the other, only if the chain itself resides in the right subtree of the root, and a rot2-right rotation is applied to each of its vertices. For each vertex x in $s+1, s+2, \ldots, n-2$ such a rotation must be followed by at least one rot1-left, because rot2-right does not lift the vertices following x in the chain which are attached to the new right child of x. As there are $n - \rho_S - 2$ vertices in $s+1, s+2, \ldots, n-2$, at least $2(n - \rho_S - 2)$ rotations are needed to bring vertex $n-1$ at level 3, and a final rot2-right is needed to bring $n-1$ into the right arm, thus obtaining a tree with $n-2$ and $n-1$ on the border. The total number of rotations mentioned above is $2n - \rho_S - 4$. If only these rotations are performed, an (n-3,2)-chain Q_1 is obtained from S, that is the tree at minimum distance from S with the vertices $n-2$ and $n-1$ on the border.

A similar reasoning holds for bringing vertex $n-2$ to the border of T. Only one rot2-right is now required for vertex $n-1$ as this lifts the chain $t+1, t+2, \ldots, n-2$ one level up. After $2n - \rho_T - 3$ rotations T is then transformed into an (n-4,3)-chain Q_2, that is the tree at minimum distance from T with vertices $n-2$ and $n-1$ on the border. Another rotation is now required to transform Q_1 into Q_2, for a total number of $4n - 8 - \rho_S - \rho_T$ rotations. *Q.E.D.*

3 Rotating at different tree levels.

Allowing rotations only at selected vertices may be relevant for handling very large files that mainly reside in secondary storage, while the root, and the vertices in its neighborhood, are brought into main memory where the rotations are performed. We then introduce the *k-restricted rotation distance* $d_{k-R}(S,T)$ with $k \geq 2$, where rotations (now called "k-restricted") can take place at the highest k levels of the tree.

For $k = 2$, that is the rotations are allowed only at the root and at both its children. In addition to the notation *rot1-right*, *rot1-left*, *rot2-right*, *rot2-left* introduced before, we shall denote by *rot3-right*, *rot3-left* the right and left rotation at the left child of the root. We pose:

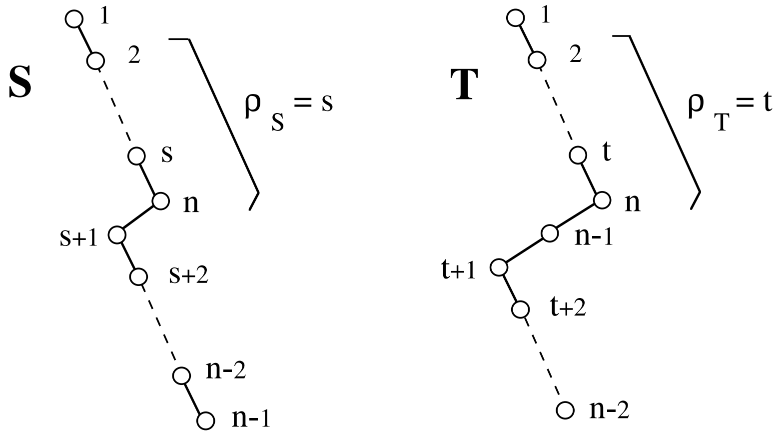

Figure 3: Two trees for which the bound of Theorem 2 is met.

Corollary 2 *For two arbitrary trees S, T with $n \geq 3$, we have: $d_{2-R}(S,T) \leq 4n - 8 - \mu$, where $\mu = max(\lambda_S + \lambda_T, \rho_S + \rho_T)$.*

Corollary 2 is an immediate consequence of Corollary 1, using Algorithm 1 or a symmetrical version of it, whichever requires less rotations. It also gives an upper bound of $d_{2-R}(S,T) \leq 4n - 9$ as $\mu \geq 1$. With this strategy not much is gained with the introduction of rot3-right, rot3-left, however, we now prove a lower bound for 2-restricted rotation, showing that only marginal room for improvement remains over the result of Corollary 2. First note that Lemma 1 holds unchanged since its proof can be immediately extended to the new allowed rotations. To apply the lemma to whole portions of the two trees, we pose:

Definition 2 *Two* opposite trees *A, B are built as search trees on a set of consecutive integers $\{h, h+1, ..., h+2^k-1\}$, with $k \geq 1$, such that A has vertex $h+2^k-1$ at the root, and the other integers of the set follow, arranged in a complete binary tree; and B has vertex $h+2^{k-1}$ at the root of a complete binary tree containing all the integers of the set, except for $h+1$ that is displaced in the ($k+1$)-th level.*

Figure 4 shows two opposite trees for the set $\{1, 2, ..., 8\}$, that is for $h = 1$ and $k = 3$. Due to Lemma 1, if two opposite trees A, B are respectively part of two trees to be transformed into one another, all the vertices of A and B must be brought into the border of the tree under transformation. This consideration allows to define two trees S, T for which a significant, although not yet sharp, lower bound on $d_{2-R}(S,T)$ can be found. First we prove:

Lemma 2 *Let a tree T contain a complete binary subtree C of $m = 2^k - 1$ vertices, with the root of of C not in the border of T. Then at least $2m - 1$ 2-restricted rotations are needed to move all the vertices of C into the border of T.*

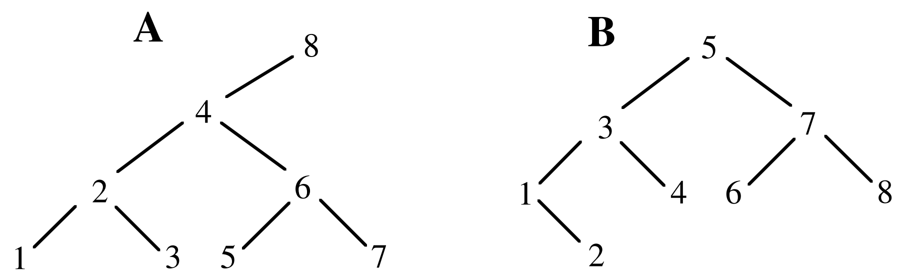

Figure 4: Two opposite trees A, B for the set $\{1, 2, ..., 8\}$.

Proof Each internal vertex v of C must enter the border of T before its two children $u < v$ and $w > v$. If v enters the left arm via a rot3-left, moving from level 3 to level 2, then v remains at level 4, and another rotation is needed to lift u one level up for successive insertion into the border. If v enters the right arm, a symmetrical situation holds for w. Therefore each of the $(m-1)/2$ internal vertices of C accounts for at least two rotations, for a total of $m-1$ rotations.

Consider now a leaf v of C that enters the border before two other leaves $u < v$ and $w > v$. If v enters the left arm, then u is pushed one level down, and two other rotations are needed to restore the level of u, and then to lift u one level up for successive insertion into the border. If v enters the right arm, a symmetrical situation hold for w. Therefore each leaf in the condition above accounts for at least three rotations.

Consider now a leaf v that enters the border when all the leaves $< v$ are already in the border. (The situation with all the leaves $> v$ already in the border is symmetrical). Since v enters the border at level 2, all the leaves $> v$ not already in the border must be descendants of vertices of the right arm. If v enters the right arm, all the leaves above are pushed one level downd and one rotation is needed to restore their level, plus another rotation to allow their insertion. If v enters the left arm we must consider two cases, namely: (i) for each leaf $w > v$ that will later enter the border in the left arm, another rotation is needed to bring w to the left side of the tree before its insertion takes place; (ii) for each leaf $w > v$ that will later enter the border in the right arm, at least two other rotations are needed to restore the level of the remaining leaves not already in the border, except for the last leaf that may require only one rotation. Collectively the $(m+1)/2$ leaves require at least $2(m+1)/2 - 1 = m$ rotations to enter the border, and the result follows. *Q.E.D.*

We now prove our lower bound, namely:

Theorem 3 *For any $n \geq 7$ there are trees S, T with non-empty arms such that $d_{2-R}(S,T) \geq 4n - 16 - \lambda_S - \rho_S - \lambda_T - \rho_T$.*

Proof Let A_1, B_1 and A_2, B_2 be two pairs of opposite trees containing m_1 and m_2 vertices, respectively. Let S, T be two trees of $n \geq m_1 + m_2 + 3$ vertices (i.e.,

$n \geq 7$), composed of a border with non-empty arms, with A_1, B_1 attached to the right of the extremes of the left arms of S and T, and A_2, B_2 attached to the left of the extremes of the right arms of S and T. Note that vertices 1 and n are respectively the extremes of the left and the right arms, and all the vertices of A_1, B_1, A_2, B_2 are not in the borders of S, T and must be brought into such borders by Lemma 1. This requires a total of at least $4m_1 + 4m_2 - 8$ 2-restricted rotations by Lemma 2, observing that opposite trees contain one vertex more than complete binary trees. To start such rotations, however, the roots of A_1, B_1, A_2, B_2 must be brought to level 3, requiring at least $\lambda_S - 1 + \lambda_T - 1 + \rho_S - 1 + \rho_T - 1$ rotations. Since $n = m_1 + m_2 + \lambda_S + \rho_S + 1 = m_1 + m_2 + \lambda_T + \rho_T + 1$, the result immediately follows. *Q.E.D.*

The lower bound of Theorem 3 is not sharp, however, the difference with the upper bound of Corollary 2 depends only on the values of the arm lengths, besides a constant, while the term $4n$ is unchanged. In the limit case $\lambda_S = \rho_S = \lambda_T = \rho_T = 1$ we have: $4n - 20 \leq d_{2-R}(S, T) \leq 4n - 10$.

For $k \geq 3$ the rotations can also occur at further levels down, and the situation is much more complex. For example, for $k = 3$ the left and the right subtrees of the root can be transfored independently, making use of the 2-restricted rotations in their upper two levels where the case $k = 2$ applies (note, however, that S and T must have the same root). Such a divide and conquer approach can then be extended to greater values of k. The study of ($k \geq 2$)-restricted rotation is fully open, some ideas can be found in [5].

4 Rebalancing AVL trees.

A natural question arises under the restricted rotations model, namely, can a balanced search tree, unbalanced after insertion or deletion of a vertex, be rebalanced efficiently? The answer is postive at least for AVL trees, in fact we will show how rebalancing one such a tree after vertex insertion or deletion with $O(\log n)$ 2-restricted rotations. [2]

Consider rebalancing after insertion. While the standard algorithm performs a unique (simple or double) rotation around the critical vertex, the key idea in the restricted rotation model is lifting the critical vertex up to the root with a sequence of rotations in the two highest levels of the tree, then performing the required AVL rotation at the root to rebalance the subtree locally unbalanced, and then moving this subtree down to its original position. In fact, when the critical vertex is at the root, simple and double AVL rotations can be expressed in terms of restricted rotations. In particular a simple right rotation and a simple left rotation correspond to rot1-right and rot1-left, respectively; a double right rotation corresponds to rot3-left

[2]The same result can be obtained with rotations at the root and at one of its children only, using a slightly more complicated procedure.

followed by rot1-right; and a double left rotation corresponds to rot2-right followed by rot1-left.

In an AVL tree T, let $\Pi = a_0, a_1, ..., a_{k+1}$ be a subsequence of vertices encountered in the search path for insertion of a vertex. Π starts with the root a_0 of T, reaches the critical vertex a_{k-1} and its child a_k, and ends with the child a_{k+1} of a_k. Note that a_{k-1} and a_k enter in a simple AVL rotation, while a_{k-1}, a_k, and a_{k+1} enter in a double AVL rotation. For performing at the root the required AVL rotation, all the vertices in Π, starting from a_1 and up to a_k (for simple rotation) or to a_{k+1} (for double rotation) are lifted to the root one after the other, then the AVL rotation is done, and then the vertices are moved down to their original postitions. These up and down movements are better expressed with an *ad hoc* notation.

For a vertex x residing in level 2 or 3 of the tree we define $UP(x)$ as the operation of lifting x up to the root, and $UP^{-1}(x)$ as the one of moving x from the root down to its original position. $UP(x)$ must be performed before $UP^{-1}(x)$, and both operations depend on the position of x in the tree when $UP(x)$ is performed. However, since the two operations may be separated by several time steps during which the tree may change, the actions taken by $UP(x)$ must be recorded for later execution of $UP^{-1}(x)$. The two operations are defined as follows:

if x is the left child of the root **then** $UP(x)$=rot1-right, $UP^{-1}(x)$ = rot1-lef;
if x is the right child of the root **then** $UP(x)$=rot1-left, $UP^{-1}(x)$=rot1-right;
if x is the left child of the right child of the root **then**
$UP(x)$= rot2-right, rot1-left, $UP^{-1}(x)$=rot1-right, rot2-left;
if x is the right child of the left child of the root **then**
$UP(x)$= rot3-left, rot1-right, $UP^{-1}(x)$=rot1-left, rot3-right.

Note that UPs are equivalent to the AVL rotations executed in the upper two levels of the tree, and $UP^{-1}s$ are their inverses.

We now define the rebalancing algorithm, whose functioning can be followed on the example of figure 5 (integer keys not in the range $1 : n$). Recall that $\Pi = a_0, a_1, ..., a_{k+1}$ is the search sequence. Note that the vertices in Π get 'marked'.

Algorithm 2 *Rebalancing an AVL tree after insertion, using 2-restricted rotations. In the search sequence, $A = a_{k-1}$ is the critical vertex, $B = a_k$ is the child of A, and $C = a_{k+1}$ is the child of B.*

if an AVL simple rotation on A is required **then**

1. mark a_0;
   for $i = 1$ **to** k **do** mark a_i, $UP(a_i)$;

2. **if** A is the left-child of B **and** the left-child of A is marked **then** rot3-right;
   if A is the right-child of B **and** the right-child of A is marked **then** rot2-left;
   $a_{k-1} \leftarrow B$;

3. **for** $i = k-1$ **downto** 1 **do** $UP^{-1}(a_i)$;

if an AVL double rotation on A is required **then**

1. mark a_0;
   for $i = 1$ **to** $k+1$ **do** mark a_i, $UP(a_i)$;

2. **if** the left-child of the left-child of C is marked **then** rot3-right;
   if the right-child of the right-child of C is marked **then** rot2-left;
   $a_{k-1} \leftarrow C$;

3. **for** $i = k-1$ **downto** 1 **do** $UP^{-1}(a_i)$.

We have:

Theorem 4 *Algorithm 2 correctly rebalances an AVL tree, unbalanced after vertex insertion, with $O(\log n)$ 2-restricted rotations.*

Proof If a simple rotation is required, step 1 brings the critical vertex A to the root with a sequence of 2-restricted rotations (UP operations) on the vertices of Π, that are lifted one after the other. At this point the AVL rotation for exchanging A and B is performed at the root (k-th UP operation of step 1), so that the subtree rooted at B is now balanced. To rebuild the initial tree, with the former subtree rooted in A now rebalanced and rooted in B, all the vertices previously moved up must be moved down. At the beginning of step 2, B is at the root. A is the left (or the right) child of B, and is moved down only if its left (or right) child is marked, as this vertex belongs to Π and must reach a position where a restricted rotation can be performed. B will be eventually moved down from the root by an UP^{-1} operation, that justifies the statement $a_{k-1} \leftarrow B$. In fact, in step 3 B is first moved down, followed by all the other vertices in inverted order. Since the operations in this step, except for the ones on A and B, are the inverses of those in step 1, the resulting tree is equal to T except for the rebalanced subtree now rooted in B. A similar reasoning holds if an AVL double rotation is required, considering also the rotations for vertex C.

Finally, since the number of performed UPs and $UP^{-1}s$ is $O(\log n)$, and each one requires one or two 2-restricted rotations, the result on the total number of rotations follows. Q.E.D.

In AVL trees rebalancing after deletion is more complicated than for insertion, because the height of the subtree rooted at the critical vertex always decreases by one after rebalancing at that vertex, thus possibly unbalancing the parent of the critical vertex. The unbalancing can then propagate up to the root so that $O(\log n)$ rotations are required in the worst case. However we can detect in advance the vertices on the search path requiring a rotation, called *candidates*, as the ones whose balance factor is different from zero and is of opposite sign with respect to the search path [3].

The stategy for rebalancing a tree after deletion, using a restricted set of rotations, is similar to that of insertion. As before all the vertices of the critical path are lifted one by one to the root. In addition, when each such a vertex is lifted, it is also rotated if satisfies the candidate condition. A rebalancing after deletion algorithm can then be derived with minor modifications from Algorithm 2, still requiring $O(\log n)$ rotations. This result seems to be interesting because the same number of rotations, in order of magnitude, is required by the standard AVL deletion algorithm where rotations are performed at all levels.

As before $a_0, \ldots, a_{k-1}$ is the search subsequence from the root to the critical vertex. Unlike for insertion, a_k is now the child of a_{k-1} *not* in the search path, and a_{k+1} is the left child (respectively, right child) of a_k if a_k is the right child (respectively, left child) of a_{k-1}. The rebalancing algorithm is as follows. As usual we assume that the candidate vertices have already been detected during the search for the node to be deleted.

Algorithm 3 *Rebalancing an AVL tree after deletion, using 2-restricted rotations. In the search sequence, $A = a_{k-1}$ is the critical vertex, $B = a_k$ is the child of A, and $C = a_{k+1}$ is the child of B.*

if an AVL simple rotation on A is required **then**
    1. for $i = 0$ **to** k **do**
        if a_i is a candidate **then**
            if a_{i+1} is the left-child of a_i **then let** b_i = right-child of a_i;
            if a_{i+1} is the right-child of a_i **then let** b_i = left-child of a_i;
            mark b_i, $UP(b_i)$;
        mark a_i, **if** $i \neq 0$ **then** $UP(a_i)$;
    2. if A is the left-child of B **and** the left-child of A is marked **then** rot3-right;
        if A is the right-child of B **and** the right-child of A is marked **then** rot2-left;
        $a_{k-1} \leftarrow B$;
    3. for $i = k-1$ **downto** 0 **do**
        if $i \neq 0$ **then** $UP^{-1}(a_i)$;
        if a_i is a candidate **then** $UP^{-1}(b_i)$;

if an AVL double rotation on A is required **then**
    1. for $i = 0$ **to** $k+1$ **do**
        if a_i is a candidate **then**
            if a_{i+1} is the left-child of a_i **then let** b_i = right-child of a_i;
            if a_{i+1} is the right-child of a_i **then let** b_i = left-child of a_i;
            mark b_i, $UP(b_i)$;
        mark a_i, **if** $i \neq 0$ **then** $UP(a_i)$;
    2. if the left-child of the left-child of C is marked **then** rot3-right;
        if the right-child of the right-child of C is marked **then** rot2-left;
        $a_{k-1} \leftarrow C$;
    3. for $i = k-1$ **downto** 0 **do**
        if $i \neq 0$ **then** $UP^{-1}(a_i)$;
        if a_i is a candidate **then** $UP^{-1}(b_i)$.

We then have the following corollary as a simple extension of Theorem 4:

Corollary 3 *An AVL tree can be rebalanced after deletion with $O(\log n)$ 2-restricted rotations.*

5 Concluding remarks.

Much work has to be done to complete this work. First, a sharp lower bound and/or a better algorithm for 2-restricted rotation have yet to be found. Moreover the present study is directed to worst case performance, while an efficient transformation algorithm for the average case is still unknown. ($k \geq 3$)-restricted rotations open a promising new line of investigation. Perhaps more importantly, other applications of restricted rotations should be studied besides the AVL rebalancing problem.

The most interesting approach for further studies, however, could be investigating the relationship between group theory and our new results, along the line opened by Cleary and Taback. Not only this could lead to improve the bounds, but new significant interpretations of tree distance could possibly arise in group theory.

References

[1] S. Cleary, Restricted rotation distance between binary trees, *Information Processing Letters* 84 (2002) 333-338.

[2] S. Cleary and J. Taback, Bounding restricted rotation distance, *Information Processing Letters* 88 (5) (2003) 251-256.

[3] D.E. Knuth, *The Art of Computer Algorithms: Sorting and Searching*, Vol. 3, Addison-Wesley, Reading, MA, 1973.

[4] F. Luccio and L. Pagli, On the upper bound on the rotation distance of binary trees, *Information Processing Letters* 31 (2) (1989) 57-60.

[5] F. Luccio and L. Pagli, General restricted rotation distance, *Università di Pisa, Dipartimento di Informatica*, Tech. Rep. TR-04-22 (2004).

[6] E. Mäkinen, On the rotation distance of binary trees, *Information Processing Letters* 26 (5) (1988) 271-272.

[7] J. Pallo, An efficient upper bounfd of the rotation distance of binary trees, *Information Processing Letters* 73 (3-4) (2000) 87-92.

[8] R. Rogers, On finding shortest paths in the rotation graph of binary trees, in: *Proc. Southeastern Internat. Conf. on Combinatorics, Graph Theory, and Computing*, Vol. 137 (1999) 77-95.

[9] D.D. Sleator, R.E. Tarjan, W.R. Thurston, Rotation distance, triangulations, and hyperbolic geometry, *J. Amer. Math. Soc.* 1 (3) (1988) 647-681.

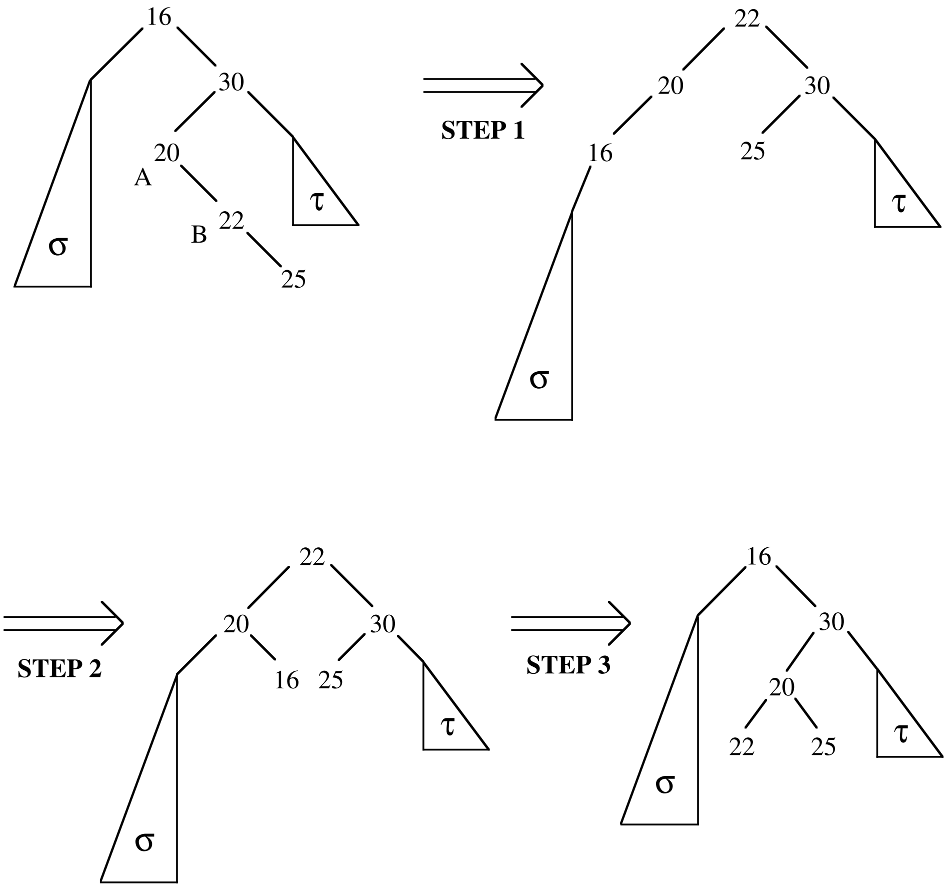

Figure 5: Rebalancing an AVL tree after the insertion of the unbalancing vertex 25 (simple rotation), using 2-restricted rotations,. We have: $\Pi = a_0, a_1, a_2, a_3 = 16, 30, 20, 22$ and $k = 3$, $a_{k-1} = A = 20$ (critical vertex), $B = 22$. Step 1 of Algorithm 2 makes: $UP(a_1 = 30) =$ rot1-left; $UP(a_2 = 20) =$ rot3-left, rot1-right; $UP(a_3 = 22) =$ rot2-right, rot1-left. Step 2 makes: rot3-right, $a_2 = 22$. Step 3 makes: $UP^{-1}(a_2 = 22) =$ rot1-left, rot3-right; $UP^{-1}(a_1 = 30) =$ rot1-right.