Università di Pisa
Dipartimento di Informatica

Technical Report: TR-04-22

General Restricted Rotation Distance

Fabrizio Luccio Linda Pagli

December 22, 2004

Address: via F. Buonarroti 2, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

General Restricted Rotation Distance

Fabrizio Luccio　　Linda Pagli

December 22, 2004

Abstract

The *restricted rotation distance* $d_R(S,T)$ between two trees S, T of n vertices is the minimum number of rotations by which S can be transformed into T, where 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 $d_R(S,T) \leq 4n-8-\rho_S-\rho_T$, where ρ_S and ρ_T are the numbers of vertices in the rightmost vertex chains of the two trees, providing a very simple transformation algorithm based on elementary properties of trees. We also generalize the concept of restricted rotation by allowing rotations at the highest k levels of the tree, and study the new distance for $k=2,3$.

Keywords: Rotation distance, Binary trees, Search trees, 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, such that a preorder traversal of a tree meets the keys in ascending order [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, fully defined in figure 1. In a search tree T rotations maintain the ordering of the keys, and are used to keep the tree *balanced*, that is the height of T, 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 $d(S,T) = d(T,S)$. [1] In a scholarly paper [8], 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 [5] 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 [8]. No polynomial-time algorithm is known to compute $d(S,T)$, but estimates of this value were given by Pallo [6] and Rogers [7].

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 remerkable for three 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

[1] If S and T are search trees, assume w.l.o.g that the keys are the integers from 1 to n with a natural ordering. The rotations have the only role of changing the shape of S to the one of T.

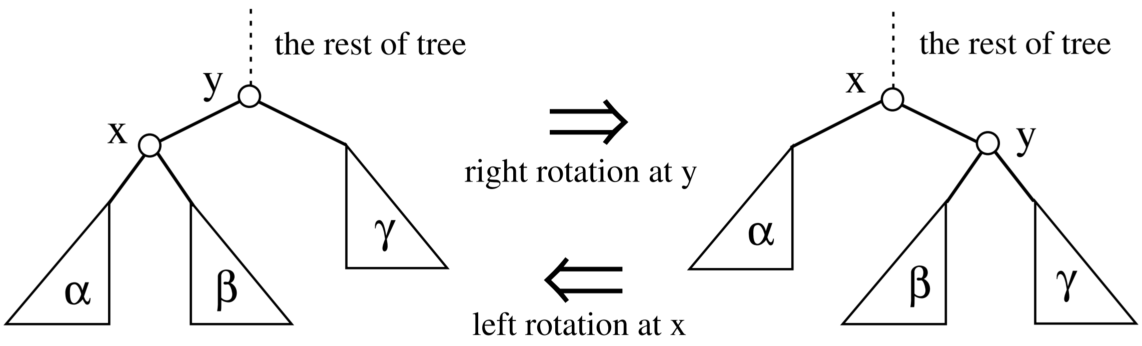

Figure 1: A *right rotation* at vertex y brings the left child x of y one level up, and y one level down to the right. α, β, and γ are (possibly empty) subtrees that remain unchanged. The rest of the tree is also unchanged, and the search ordering is preserved. A *left rotation* is the inverse operation.

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. Third, 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 2). In this note we refine the upper bound $d_R(S,T) \leq 4n - 8$ of [2] to $d_R(S,T) \leq 4n - 8 - \rho_S - \rho_T$. Clearly 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.

We then generalize the concept of restricted rotation by performing rotations at 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; and for $k = 3$, where the rotations can also occur at a further level down.

2 The basic algorithm.

To present our rotation algorithm we pose:

Definition 1 *A tree T is an* (i,j)-chain *if all the vertices of T besides the root reside in the left and/or right arm of T, with $\lambda_T = i$ and $\rho_T = j$.*

See figure 3 for examples. 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 transforming both S and T into R, then reverting the second transformation to get $S \rightarrow R \rightarrow T$.

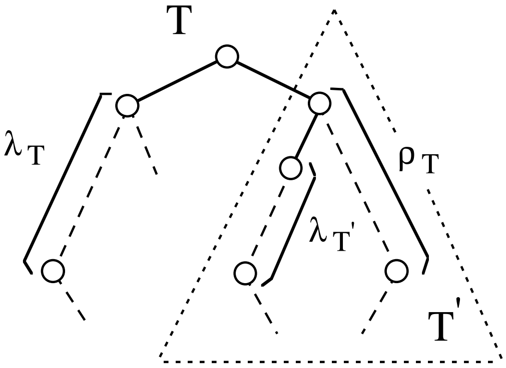

Figure 2: The left and right arms of a tree T, of lengths λ_T and ρ_T. T' is a subtree of T, with left arm of length $\lambda_{T'}$.

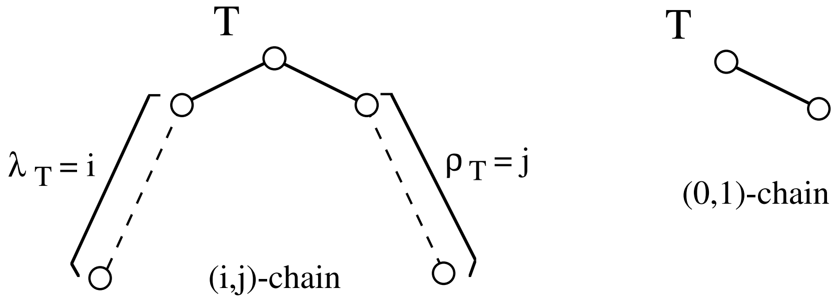

Figure 3: (i,j)-chains.

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 into an (n-3,2)-chain. T' is the subtree rooted at the right child of the root of T (see figure 2). $\overline{T}$ 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 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). Let the union of the root, the left, and the right arms be the *contour* of T. Once a vertex is in the contour 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 any rotation let us focus on the vertex which is directly lifted up one level (i.e., vertices x and y in the left and right side of figure 1, respectively), 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 coincide 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, if not impossible, to occur. For trees built at random, the probability P_0 of having $\rho_S = \rho_T = 0$ decreases exponentially with n. If we regard the trees as search structures, the major distintion 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: $d_R(S,T) \leq 4n - 8 - \Omega(\lg n)$.

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 can take place at the highest k levels of the tree, and study what can be done for $k = 2, 3$.

For $k = 2$, that is the rotations are allowed only at the root and at both its children, we have:

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 the result for restricted rotation, and also gives an upper bound of $d_{2,R}(S,T) \leq 4n - 9$ as $\mu \geq 1$. Not much has then been gained, in the worst case, by allowing a new rotation at the left child of the root. However we have been unable to establish a significant lower bound, so there may be room for improvement in the rotation algorithm.

For $k = 3$ the rotations can also occur at a further level down, and the situation is much more interesting. In fact the left and the right subtrees of the root can be transfored independently, making use of the rotations in their upper two levels where the case $k = 2$ applies. To take advantage of this possibility, however, the two trees S, T must have the "same" root, that is, the two roots must carry the same integer value in the key ordering from 1 to n. The scheme of a new rotation algorithm for $k = 3$ is given below, under the following:

Condition 1 *Given two arbitrary trees S, T, identify the vertices with the integer values of their keys in the range $[1, n]$. Let $x = \mathrm{root(T)} \leq \mathrm{root(S)}$, i.e. x is either at the root of S or in its left subteree. Let n_ℓ, n_r be the number of vertices contained in left and right subtrees of the root of T, respectively. Let S_ℓ, S_r and T_ℓ, T_r be the left and right subtrees of the roots of S and T, respectively. Denote by L and R the left and right arms of S, and denote by π the (possibly empty) path from* root(S) *to x, extremes excluded. π contains $f \geq 0$ vertices greater than x in L, $g \geq 0$ vertices greater than x not in L, and $h \geq 1$ vertices smaller than x, one of which is in L.*

We also extend the notation of the previou section, denoting by *rot1-right*, *rot1-left*, *rot2-right*, *rot2-left*, *rot3-right*, *rot3-left* the right and left rotation at the root (1), and at the right child (2) and left child (3) of the root, respectively.

Algorithm 2 *Transforming a tree S into a tree T under 3-restricted rotation. $\overline{S}$ is a tree generated at each step of the transformation, starting with $\overline{S} = S$, and the other overlined symbols have similar meaning. Condition 1 is assumed (if not, symmetric actions are taken througout the algorithm).*

Phase 1: lifting vertex x of S to the root of S.

1. **execute** a sequence of rot1-right and rot3-left in S **until** x is brought into $\overline{L}$;

   comment: each one of the f vertices $> x$ in L require one rot1-right, increasing $\rho_{\overline{S}}$ by one; each one of the g vertices $> x$ not in L require one rot3-left, followed by one rot1-right which increases $\rho_{\overline{S}}$ by one; each one of the h vertices $< x$ require one rot3-left, increasing $\lambda_{\overline{S}}$ by one;

2. **execute** a rot1-right to bring x to the root of S;

   comment: this rotation increases $\rho_{\overline{S}}$ by one;

Phase 2: Transforming $\overline{S}$ into T.

1. **execute** algorithm 1 extended to 2-restricted rotation on the left subtree $\overline{S_\ell}$ to transform this subtree into T_ℓ via a (2, n_ℓ-3)-chain;

2. **execute** algorithm 1 extended to 2-restricted rotation on the right subtree $\overline{S_r}$ to transform this subtree into T_r via an (n_r-3, 2)-chain.

Corollary 3 *For two arbitrary trees S, T with $n \geq 4$ we have: $d_{3,R}(S,T) \leq 4n - 14 - \lambda_S - \lambda_T - \rho_S - \rho_T + f + g$, where f and g are as in condition 1 or symmetrical.*

Proof Apply algorithm 2. Phase 1 requires $f + 2g + h + 1$ rotations. At the end of this phase we have:

$\lambda_{\overline{S_\ell}} = \lambda_{\overline{S}} - 1 = h + (\lambda_S - f - 1) - 1$,

where the term $\lambda_S - f - 1$ is the number of vertices smaller that x originally contained in λ_S,

$\lambda_{T_\ell} = \lambda_T - 1$,

$\rho_{\overline{S_r}} = \rho_{\overline{S}} - 1 = f + g + \rho_S - 1$,

$\rho_{T_r} = \rho_T - 1$.

The total number of rotations in phase 2 is given by:

$4n_\ell - 8 - \lambda_{\overline{S_\ell}} - \lambda_{T_\ell} + 4n_r - 8 - \rho_{\overline{S_r}} - \rho_{T_r}$.

Combining the above results for the two phases we get the stated bound. Q.E.D.

Again we do not have a significant lower bound for $d_{3,R}(S,T)$. Algorithm 2 can be modified in several ways, for example working on T instead of S in phase 1, or inverting the rotations in step 2 of phase 2, or even applying to S, T the algorithm for 2-restricted rotation without making use of the rotations at level 3, then comparing the results to get the best of them. A more interesting task would be searching for a common vertex x, not necessarily the root of S or T, to become the common root in the first phase.

4 Concluding remarks.

Much work has to be done to complete this work. First, the new upper bound on the restricted rotation distance has no matching lower bound as yet. This limitation is obviously encountered also for k-restricted rotation, where a strong improvemment on the proposed upper bounds for $k = 2, 3$ may be expected, while finding an efficiet algorithm for arbitrary k is still open.

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 proving the lower bounds still missing, 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] E. Mäkinen, On the rotation distance of binary trees, *Information Processing Letters* 26 (5) (1988) 271-272.

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

[7] 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.

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