

# Efficient Reorganization of Binary Search Trees\*

Micha Hofri<sup>†</sup>  
Dept. of Computer Science  
Worcester Polytechnic Institute  
Worcester MA 01609  
e-mail: hofri@cs.wpi.edu

Hadas Shachnai<sup>‡</sup>  
Dept. of Computer Science  
The Technion – Israel Institute of Technology  
Haifa, 32000 ISRAEL  
e-mail: hadas@cs.technion.ac.il

## Abstract

We consider the problem of maintaining a binary search tree (BST) that minimizes the average access cost needed to satisfy randomly generated requests. We analyze scenarios in which the accesses are generated according to a vector of fixed probabilities which is *unknown*. Our approach is statistical.

We devise policies for modifying the tree structure dynamically, using rotations of accessed records. The aim is to produce good approximations of the optimal structure of the tree, while keeping the number of rotations as small as possible. The heuristics that we propose achieve a close approximation to the optimal BST, with lower organization costs than any previously studied.

We introduce the MOVE\_ONCE rule. The average access cost to the tree under this rule is shown to equal the value achieved by the common rule *Move to the Root* (MTR). The advantage of MOVE\_ONCE over MTR and similar rules is that it relocates each of the items in the tree at most once. We show that the total expected cost of modifying the tree by the MOVE\_ONCE rule is bounded from above by  $2(n+1)H_n - 4n$  rotations (in a tree with  $n$  records), where  $H_n$  is the  $n$ th harmonic number. Extensive experiments show that this value is an over-estimate, and in fact the number of rotations is linear for all the access probability vectors we tested. An approximate analysis is shown to match the experimental results, producing the expected number  $n((\pi^2/3) - 2) - 2\ln n + 0.1354$ .

Next we combine the MOVE\_ONCE rule with reference counters, one per record, that provide estimates of the reference probabilities. We call the resulting reorganization rule MOUCS. We show that, for any  $\delta$ ,  $\alpha > 0$  and sufficiently large  $n$ , it achieves a cost that approaches the optimum up to an absolute difference of  $\delta$  with probability higher than  $1 - \alpha$ , within a number of accesses that is proportional to  $n(\lg n)^2 / (\alpha\delta^2)$ .

**Key words:** Binary search tree, dynamic reorganization, move to the root, counter scheme statistics, access probabilities, stopping point.

---

\*The extended abstract of this paper appeared in *Proceedings of the 2nd Italian Conference on Algorithms and Complexity*, 1994.

<sup>†</sup>Part of the work was performed in the Dept. of Computer Science, The University of Houston, Houston TX 77204-3475, USA.

<sup>‡</sup>Author supported in part by the Technion V.P.R. Fund – E. and J. Bishop Research Fund, and by the Fund for the Promotion of Research at the Technion.

# 1 Introduction

## 1.1 Problem Statement

The *Binary Search Tree* (BST) is commonly used for storing files with elements that satisfy a total order. The advantage of a tree is that it allows an efficient search in the file. Typically, the search is most efficient when the tree is kept as balanced as possible, and when popular elements are close to the root. We study methods that maintain a BST in a near-optimal form.

Consider a fixed set of  $n$  records,  $F = \{R_1, \dots, R_n\}$ . Let  $[n]$  denote the set of the first  $n$  natural numbers, i.e.,  $\{1, \dots, n\}$ . The record  $R_i$  is uniquely identified by the key  $K_i$ , for  $i \in [n]$ . The keys satisfy a total order, and the set is maintained as a BST. The records are accessed according to a multinomial distribution driven by a *reference probability vector* (RPV),  $\mathbf{p} = (p_1, \dots, p_n)$ . We assume that in interesting cases  $p_i > 0$ , for all  $i \in [n]$ . Let us use the symbol  $\mathbf{e}$  to denote the uniform probability vector:  $\mathbf{e} = (\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n})$ .

At any stage, the record  $R_i$  may be requested with the fixed probability  $p_i$ , independently of previous requests and the state of the tree – and in particular, of the location of  $R_i$  in the tree. This defines the *independent reference model* (IRM).

In many scenarios it is natural to assume dependence between successive accesses, e.g., due to locality of reference [6, 8]. The IRM is unsuitable for such scenarios; however, the IRM captures well the nature of reference sequences generated from multiple sources: for such sequences, even when each source demonstrates locality, the correlation between successive references becomes negligible.

In our reference model the RPV and  $F$  do not change, hence the passage of time is realized by the sequence of references. There is no other notion of time in the model. Each reference requires a search for a record in the tree. The cost of a single access is defined as the number of key-comparisons needed to locate the specified record, which equals its *level* in the tree (where the level of the root is 1).

The order by which the records are initially inserted into the tree is assumed to be random (with equal probability over all possible permutations. The implications of this assumptions are discussed further in the note to Theorem 2 below). Different initial-insertion sequences usually result in different trees, with very large range of expected access costs; for a uniform RPV it varies from nearly  $\lg n^1$  to  $n/2$ .

The access probabilities listed in the RPV  $\mathbf{p}$  are assumed unknown. *Were* they known, we could restructure the tree, using a known dynamic-programming approach, to provide the smallest possible expected cost. Since the RPV is constant, so would be the optimal structure. With  $\mathbf{p}$  unknown, we look at policies that use the accumulating reference history to adapt the tree structure. Our goal is to rearrange the records so that the expected access cost is minimized.

The reorganization process incurs a cost as well: the manipulations performed on the tree when its structure is modified. The only operations used for this are *rotations*, operations that exchange the ‘rotated’ node with its parent, while maintaining the key-order in the tree. Figure 1 shows the tree modifications that result. Note that the inverse of the rotation operation is a rotation as well. The cost of the reorganization is defined as the number of rotations, since each rotation requires essentially the same computing time.

The only cost components we use are key comparisons and rotations. A few performance measures which are of interest in this context are:

- The access cost following the  $m$ th reference (and possible reorganization), for  $m \geq 0$ .

---

<sup>1</sup> $\lg n$  is the logarithm of  $n$  to base 2.

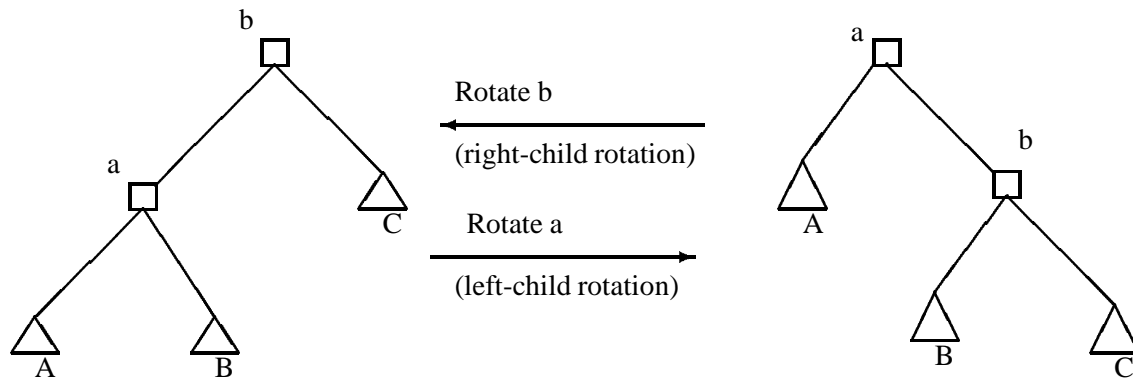


Figure 1: Single left-child and right-child rotations that reflect  $(A) < K_a < (B) < K_b < (C)$

- The asymptotic access cost, especially its expected value.
- The overall number of rotations done during a sequence of references to the tree.

## 1.2 Related Work and Our Results

The problem of reorganizing a BST to minimize the average access cost has been widely studied. Most of the works we have seen either focus on the asymptotic average access cost or present an *amortized* analysis (typically combining the cost of accessing an element with the required reorganization steps). Amortized analysis considers worst-case scenarios, and therefore does not apply to the present model [13, 29, 23].

Some of the well-known rules for reorganization of BSTs with the IRM assumption, and where the access probabilities are *unknown*, are studied in [1, 4, 6, 10, 9]. Allan and Munro introduced in [1] the *Move To the Root* (MTR) rule; this rule is the counterpart of the *Move To the Front* (MTF) rule for linear lists: a referenced record is rotated to the root of the tree (unless it is there already). They showed an upper bound on the ratio  $C(n, \text{MTR}|\mathbf{p})/C(n, \text{OPT}|\mathbf{p})$  for any distribution, and also estimated its rate of convergence.

Comprehensive studies of the MTF and the MTR rules under a Markovian reference model appear in [6, 8]. The IRM is subsumed under this model, but its simplicity allows us more detailed calculations (especially in Section 4), just as Dobrow and Fill combined the IRM and MTR for deeper investigations, in [9, 10].

Some of the published research focused on situations where the RPV is *known*, with the goals of finding the optimal tree – or achieving a nearly optimal one, with a smaller computational effort. A thorough treatment is given in [22, §6.2.2]. Further work, especially on balanced BSTs, is surveyed in [27]. Parallel algorithms have also been considered for construction of optimal and nearly optimal BSTs (see, e.g., [2, 20]). The results in [4] refer to the case where the elements of the RPV are known only up to a permutation.

A general account of the properties of BSTs created under the same assumptions we use here is given by Mahmoud in his book [25].

In this paper we introduce and analyze policies that achieve a close approximation to the optimal BST, with lower organization costs than any of the previously studied heuristics. In Section 2 we define notation, and present the dynamic-programming algorithm that constructs the optimal tree for a known RPV. In Section 3 we discuss the MOVE\_ONCE rule, which achieves the same asymptotic average access cost as the *Move to the*

*Root* (MTR) rule ([1, 6, 10, 9]), but requires at most  $n - 1$  reorganization steps, for *any* reference sequence to the tree.

We then propose in Section 4 a method for approximating the optimal search tree, which improves the asymptotic average cost obtained by either the MOVE\_ONCE or MTR. Our method, that we call *Move Once and Use Counter Scheme* (MOUCS) guarantees that for any  $\delta$  and  $\alpha > 0$ , the tree is dynamically reorganized until the average access cost is within  $\delta$  of the optimal cost, with probability of at least  $1 - \alpha$ . We obtain a distribution-free bound on the running time of the algorithm, which is proportional to  $n(\lg n)^2/(\alpha\delta^2)$ .

Some of the proofs of our results in Section 3.1 use ideas similar to those used in [9].

## 2 Preliminaries

Let  $T_n$  be a BST of  $n$  records,  $R_1, \dots, R_n$ , with access probabilities  $p_1, \dots, p_n$ , respectively. We denote by  $l_i$  the level of  $R_i$ ; then, the expected access cost to  $T_n$  is given by

$$C(T_n) = \sum_{i=1}^n p_i l_i. \quad (1)$$

To argue about *subtrees*, where the sum of probabilities is smaller than 1, we use also the notion of *weighted average access cost* of a BST, defined as follows. In a BST  $T_n$ , for some  $m \leq n$ , let the records  $F' = \{R_{i_1}, \dots, R_{i_m}\} \subseteq F$  be a subset of  $m$  records in  $F$  which form a subtree  $T'_m$  in  $T_n$ . We make the natural association of the weight  $w_{i_j} = p_{i_j}$  with the record  $R_{i_j}$ ,  $1 \leq j \leq m$ .

Let  $l'_{i_j}$  be the level of  $R_{i_j}$  in the subtree  $T'_m$ . Then, the weighted average access cost of  $T'_m$  is given by  $C(T'_m) = \sum_{j=1}^m w_{i_j} \cdot l'_{i_j}$ .

### 2.1 Finding the Optimal BST

Under the IRM, for any set of keys with a given RPV, there exists an optimal *static* BST. We denote by  $C(n, \text{OPT}|\mathbf{p})$  the average access cost in such an optimal tree.

The optimal tree structure and its expected cost are straightforward to compute using the following dynamic programming equations, which need to be satisfied at every internal node (adapted from [22]).

Let  $C(i, j)$  be the weighted average access cost of the optimal subtree built from the records  $R_{i+1}, R_{i+2}, \dots, R_j$ . Then  $C(0, n)$  is  $C(n, \text{OPT}|\mathbf{p})$ , as defined above. We also define  $\pi_j := \sum_{k=i}^j p_k$ . These costs satisfy the Bellman equations

$$\begin{aligned} C(i, i) &= 0, \\ C(i, j) &= \pi_{i+1, j} + \min_{i < k \leq j} (C(i, k-1) + C(k, j)), \quad 0 \leq i < j \leq n. \end{aligned} \quad (2)$$

### 2.2 Performance Measures for Reorganization Rules

When the access probabilities are *unknown*, continual reorganization of the tree may be used to achieve an approximation of the optimal order. Various performance measures were considered for this scenario. Denote by  $L_i^{(m)}$  the level of  $R_i$  in  $T_n$  after  $m$  references;  $L_i^{(0)} = L_i$  is the level of  $R_i$  in the initial tree. With a given reorganization policy  $B$  and an unknown RPV  $\mathbf{p}$ , the following costs are used below:

1. The average access cost after the  $m$ th reference,  $m \geq 0$ , is given by

$$C_m(n, B|\mathbf{p}) = \sum_{i=1}^n p_i \cdot E[L_i^{(m)} | B, \mathbf{p}]. \quad (3)$$

*Note:* The expected level at which an item may be found, under a policy  $B$ , is determined not only by the sequence of accesses: it also depends on the initial state of the tree, possibly as a result of the order the records were inserted into the tree. As a rule we average over all possible insertion sequences, considering them equiprobable<sup>2</sup>.

Under certain policies, the initial state becomes irrelevant following a large number of references and the changes in the tree they trigger. In particular, this holds for any reorganization policy which achieves the optimal tree state (or even the optimal expected cost only) after a sufficiently long sequence of searches.

2. The expected access cost in the limiting state<sup>3</sup>:

$$C(n, B|\mathbf{p}) = \lim_{m \rightarrow \infty} C_m(B|\mathbf{p}). \quad (4)$$

In addition to the limiting expectation of the random variable  $\mathbf{C}_m$ , it is interesting to consider the rate at which  $\mathbf{C}_m$  approaches this limit: since data processing systems exist for a finite time only, a policy which reduces the access cost promptly may be preferable to one that does it more slowly, even if the limiting cost of the latter is somewhat lower.

3. The average number of rotations induced by an input string of size  $m$ , under a reorganization rule  $B$  is given by

$$\Gamma_m(n, B|\mathbf{p}) = \sum_{r=1}^m \sum_{i=1}^n p_i E[\Gamma_i^{(r)} | B, \mathbf{p}], \quad (5)$$

where  $\Gamma_i^{(r)}$  is the number of rotations done when the  $r$ th reference is to  $R_i$ . A most interesting case in our study of the MOVE\_ONCE rule (in Section 3.2) is the limit  $m \rightarrow \infty$ ; with probability one, every record is referenced in such a string.

### 3 The Move Once (MO) Rule

#### 3.1 The Average Cost of a Single Access

The MTR rule, introduced in [1], is the analog for trees of the MTF rule for linearly ordered lists of records, but they differ in two significant ways. One is that MTR brings the access cost much closer to the optimum, in the limit, than MTF does. On the other hand, the reorganization cost of a tree is much higher than that of

---

<sup>2</sup>This does *not* translate to a uniform distribution over initial tree states, since the number of sequences that result in a given tree state is not the same for all tree states. Happily, more-balanced trees occur more frequently than badly skewed ones, which have normally (much) higher access costs (see [11, 25]).

<sup>3</sup>In general, such a state may not exist; however, in this paper we discuss two policies for which the random variable  $C$  exists (see Sections 3.1 and 4.1).

a list: there moving any record to the front of the list uses the same time; but moving a record to the root of a tree uses a number of rotations, equal to the number of steps required to reach the record in the first place. Hence it is of interest to look for rules that use less expensive rearrangements.

In [17] we showed that for a linear list, moving a record at most once (when it is first referenced), to the tail of the sublist of records that were moved before, achieves the same expected cost as the MTF, at any time, not only in the limit. We propose to use the same principle for reorganizing BSTs: a record is only moved the first time it is referenced. It is then rotated towards the root, until its parent is a record that has already been referenced. (The first referenced record goes of course all the way to the root.) Hence the name MOVE\_ONCE.

Allan and Munro [1] considered a similar rule, calling it the FIRST REQUEST rule, and showed it has the same *asymptotic* cost as the MTR. However, for BSTs, just as for linear lists, more can be claimed:

**Theorem 1:** *Let a binary search tree be referenced according to the independent reference model with the RPV  $\mathbf{p}$ . The rules MOVE TO THE ROOT and MOVE\_ONCE have the same expected access costs in the  $m$ th request, for any  $m \geq 1$ .*

We use in the proof the following result:

**Lemma 1:** For a given initial BST, let  $T_B(I)$  be the BST resulting from processing the reference string  $I$  with the reorganization rule  $B$ . Then

$$T_{MTR}(I) = T_{MO}(I^R), \quad (6)$$

where the string  $I^R$  is the reverse of  $I$ .

We give a detailed proof<sup>4</sup> of the lemma in the Appendix.

**Proof of Theorem 1:** We naturally assume that both rules start with the same tree (or with trees selected at random using the same initial distribution). Under the IRM, any reference string  $I$  and its reverse  $I^R$  have precisely the same probability, hence the access costs of the two rules are identically distributed; for our needs only the equality of the expectations matters. ■

The equality in Lemma 1 may seem surprising, since usually the rules construct for the *same* input string two entirely different trees. The important difference is that MOVE\_ONCE uses far fewer rotations than MTR, and moreover, the latter churns its tree indefinitely, whereas MOVE\_ONCE rests after a time which has a finite expectation. We can now use the results in [1] for the average cost under MTR to state the following theorem.

**Theorem 2:** [1] *The expected access cost to a BST reorganized with the MOVE\_ONCE rule, after  $m$  references, is given by*

$$C_m(n, \text{MOVE\_ONCE} | \mathbf{p}) = 1 + \sum_{1 \leq i < j \leq n} \left[ \frac{2p_i p_j}{\pi_{i,j}} + (1 - \pi_{i,j})^m \left( \frac{p_i + p_j}{j - i + 1} - \frac{2p_i p_j}{\pi_{i,j}} \right) \right] \quad (7)$$

where  $\pi_{i,j} = \sum_{k=i}^j p_k$ .

**Note:** This result applies when the tree is considered to have been initiated by a *uniformly* distributed insertion sequence. When the initial tree is created by the same RPV that governs the accesses to the records, we get

<sup>4</sup>A similar line of reasoning appears in [9, Lemma 3.1], referring, naturally, to the MTR only.

that the initial form of the tree has the asymptotic distribution induced by the MOVE\_ONCE policy. Thus, the expected access to the initial tree gets the limiting expected value

$$C(n, \text{MOVE\_ONCE} | \mathbf{p}) = 1 + \sum_{1 \leq i < j \leq n} \frac{2p_i p_j}{\pi_{i,j}}, \quad (8)$$

and the MOVE\_ONCE policy calls for no more modifications.

### 3.2 The Number of Rotations

Let  $\Gamma_i$  denote the number of rotations required for moving  $R_i, i \in [n]$  up in the tree, when it is first accessed. The first record to be referenced is rotated all the way to the root, and for this record the  $\Gamma_i$  equals  $D_i$ , the depth of that record in the tree<sup>5</sup>. Subsequently referenced records are rotated to the root of the subtree of the closest ancestor that has been moved before.

Denote by  $\Gamma(n) \equiv \Gamma(n, \text{MOVE\_ONCE} | \mathbf{p})$  the random variable which equals the total number of rotations under the MOVE\_ONCE rule in a tree of size  $n$ , assuming the uniform insertion model mentioned above. Let  $\eta(n) \equiv E[\Gamma(n)]$  and  $v(n) \equiv V[\Gamma(n)]$  denote the mean and variance of  $\Gamma(n)$ .

We first give a bound on  $\eta(n)$ . The bound is not the best possible, since it is super-linear in  $n$ , while experiments that we report below suggest that as  $n$  increases the expected number of *rotations per node* approaches a limit. The bound is provided since the analysis we can produce is somewhat limited: we can only do an approximate analysis (albeit exact for the first moment when  $\mathbf{p} = \mathbf{e}$ ). The analysis produces such an essentially linear function for  $\eta(n)$ .

To derive an upper bound on  $\eta(n)$ , we use a famous result of Hibbard ([15], cf. [24]). Let  $T_n$  be a BST constructed from a random permutation of size  $n$ . The expected *internal path length* of  $T_n$ , denoted by  $D(T_n)$ , is the sum of the expected path lengths from the root to each of the keys, i.e.,  $D(T_n) = \sum_{i=1}^n E[D_i]$ .

**Theorem 3:** [15] *The average internal path length of  $T_n$  is given by*

$$D(T_n) = 2(n+1)H_n - 4n, \quad (9)$$

where  $H_n$  is the  $n$ th harmonic number.

Theorem 4 is our initial bound.

**Theorem 4:** *For any tree  $T_n$  created from a random permutation of  $[n]$ ,  $\eta(n)$ , the expected number of rotations under the MOVE\_ONCE rule is at most  $D(T_n)$ .*

Denote by  $\gamma_i^{(m)} = \Gamma_i^{(m)}(r_1, \dots, r_m, T_n)$  the number of rotations incurred in the  $(m+1)$ st access, if the requested item is  $R_i$ , the initial tree is  $T_n$ , and the first  $m$  references are  $r_1, \dots, r_m$ . Similarly, let  $\delta_i^{(m)} = D_i^{(m)}(r_1, \dots, r_m, T_n)$  be the depth of  $R_i$  after the  $m$ th reference, if the initial tree is  $T_n$  and the reference string is  $r_1, \dots, r_m$ . We use in the proof the following result.

**Lemma 2:** *For any element  $R_i, i \in [n]$ , initial tree  $T_n$ , and reference sequence  $r_1, \dots, r_m$  with  $m \geq 1$ , we have  $\gamma_i^{(m)} \leq \gamma_i^{(m-1)}$ .*

---

<sup>5</sup>The depth of the root is zero; hence depth and  $L_i$ , the level of the node, are related by  $L_i = 1 + D_i$ .

A proof of the Lemma is given in the Appendix.

**Proof of Theorem 4:** From Lemma 2 we find that for any initial tree  $T_n$  and reference sequence  $r_1, \dots, r_m$  with  $m \geq 1$ , it holds that  $\gamma_i^{(m)} \leq \gamma_i^{(0)} = \delta_i^{(0)} = D_i$ . Hence we can bound the number of rotations of  $R_i$  by  $D_i$ , and  $\eta(n) \leq D(T_n)$ . ■

### Direct Analysis

We proceed now from bounding  $\Gamma(n)$  to computing it. This random variable derives its distribution from the initial sequence used to construct the tree (assumed uniform over all  $n!$  permutations) and from the reference sequence (in which the RPV  $\mathbf{p}$  plays a role).

Unlike the initial sequence which is of fixed size  $n$ , the reference sequence of the reorganization phase is of unbounded length. It continues till all records have been accessed at least once<sup>6</sup>. The length of such a sequence is known as the duration of the coupon collector search. This process is discussed in detail in the survey [5], where results and references for many aspects of this duration are discussed. The interest in it *here* derives from the fact that while (as we show below)  $\eta(n)$  is not very sensitive to the RPV  $\mathbf{p}$ , the *length* of the sequence, which is a bound of the time until the tree quiesces, depends critically on  $\mathbf{p}$ . The expected value of this length is smallest when  $\mathbf{p}$  is uniform, and equals then  $nH_n$ ; however, it can be much larger: for the Linear RPV, with  $p_i \propto i$  (i.e.,  $p_i$  is proportional to  $i$ ), its expected length is greater than  $n(n+1)/2$ .

In this subsection we evaluate  $\Gamma(n)$  for the uniform RPV:  $p_i = 1/n$ . Experimental results below show the variations about these values arising in scenarios with other reference probabilities.

Let the record  $R_I$ , where  $I$  is the position of that record in the total order of the keys, be the first record to be accessed; then  $D_I$  is its depth. Hence the number of rotations that bring  $R_I$  to the root is also distributed as  $D_I$ . We plan to use in some of the calculations below the approximation  $D_I \sim D(n)$ , where  $D(n)$  is the depth of a randomly selected node in a randomly constructed BST. This replacement is not exact even when the RPV is  $\mathbf{e}$ , but then it is at least (trivially) correct on the average:

$$D(n) \sim \frac{1}{n} \sum_I D_I, \quad (\mathbf{p} = \mathbf{e}). \quad (10)$$

Another distributional property of the BSTs holds precisely, also for an arbitrary RPV  $\mathbf{p}$ .

**Lemma 3:** *After rotating  $R_I$  to the root, the root has two subtrees of sizes  $I-1$  and  $n-I$ . These subtrees have exactly the same shape (and structure) distribution of randomly created BSTs, with the indicated number of nodes.*

The notion of “distribution of tree shape” is explained, and the Lemma is proved, in the Appendix.

Hence we can write a recursion for  $\Gamma(n)$ , the number of rotations incurred by the MOVE\_ONCE rule for a uniform  $\mathbf{p}$ . We randomize on the first record referenced,  $R_I$ , and sum the number of rotations incurred by it with the number of rotations then generated in the subtrees of  $R_I$ , once it is moved to the root.

$$\Gamma(n|I) \sim D_I + \Gamma(I-1) + \Gamma^*(n-I), \quad n \geq 1, \quad \Gamma(0) \equiv 0. \quad (11)$$

---

<sup>6</sup>Strictly speaking, until all the nodes in the tree have parents that were referenced, which means that under MOVE\_ONCE, they will not be moved any more. The number of moved records can be at most  $n-1$ , but since the average number of leaves in  $T_n$  is  $(n+1)/3$ , it is not likely to exceed  $2n/3$ .

All three components of the right-hand side are independent, due to Lemma 3; the purpose of the asterisk in  $\Gamma^*(n-I)$  is to mark it as being sampled independently of the component  $\Gamma(I-1)$ . By derandomizing on the first-referenced record we can evaluate all moments of  $\Gamma(n)$ :

$$E[\Gamma^k(n)] = \sum_{i=1}^n p_i E[D_i + \Gamma(i-1) + \Gamma^*(n-i)]^k, \quad k \geq 1. \quad (12)$$

In particular, for  $k=1$  and a uniform reference probability vector we can use the relation (10) and write

$$E[\Gamma(n)] = \eta(n) = \frac{1}{n} \sum_{i=1}^n (D(n) + \eta(i-1) + \eta(n-i)), \quad \eta(1) = \eta(0) = 0. \quad (13)$$

The properties of  $D(n)$ , the depth of a random node in a random BST  $T_n$  are well known, having been presented in [7, §3] and in [25, §2.5]. It satisfies a recursion similar to Eq.(11) but simpler. We randomize on the size of the key at the root,  $I$ , and then the conditional distribution of the depth is distributed as the mixture

$$D(n|I) \sim \left( \frac{1}{n} 0 + \frac{n-1}{n} \delta(I) \right) + \frac{I-1}{n} D(I-1) + \frac{n-I}{n} D^*(n-I), \quad n \geq 1,$$

where the first term—zero in probability  $1/n$  and one otherwise—refers to the contribution of the root. From this it is easy to obtain its *probability generating function* (PGF), the expected value and its variance, respectively (all for  $n \geq 1$ ):

$$\begin{aligned} g_n(z) &= [n(1-2z)]^{-1} \left( 1 - (-1)^n \binom{-2z}{n} \right), \\ d_n &\equiv E[D(n)] = 2 \frac{n+1}{n} H_n - 4, \\ u_n &\equiv V[D(n)] = 2 \frac{n+5}{n} H_n + 4 \left( 1 - (n+1) \left[ \frac{H_n^{(2)}}{n} + \left( \frac{H_n}{n} \right)^2 \right] \right). \end{aligned}$$

$H_n^{(2)}$  denotes the  $n$ th second-order harmonic number, that has the asymptotics  $\pi^2/6 - 1/n + O(n^{-2})$ .

Now, Eq.(13) can be used to derive the first-order difference equation

$$\eta(n+1) \equiv E[\Gamma(n+1)] = \frac{n+2}{n+1} \eta(n) + \left[ d_{n+1} - \frac{n}{n+1} d_n \right], \quad n \geq 1 \quad (14)$$

which has the immediate solution

$$\eta(n) = 2[(n+1)H_n^{(2)} - n - H_n], \quad n \geq 1. \quad (15)$$

For not-too-small  $n$ , a good approximation of  $\eta(n)$  is given by  $\eta(n) \approx n(\pi^2/3 - 2) - 2 \ln n + 0.135437$ .

The total expected number of rotations *per record* in the tree is then less than 1.3.

The PGF of  $\Gamma(n)$  does not appear to be easy to obtain. Even the variance  $v(n) \equiv V[\Gamma(n)]$  is not easy to handle. To do it precisely we need the distribution of  $D_i$ . Since we have the first moment exactly, we are now satisfied with an approximate variance calculation, obtained by the direct replacement of  $D_i$  in Eq.(12)

by  $D(n)$ , and delay the exact calculation for a next stage. Proceeding to the variance calculation, we prefer to work with the recurrence for the somewhat simpler second moment  $g_2(n) \equiv E[\Gamma^2(n)]$ , and find that it satisfies a relation very similar to Eq.(14):

$$g_2(n+1) = \frac{n+2}{n+1}g_2(n) + U(n), \quad n \geq 1, \quad (16)$$

where

$$\begin{aligned} U(n) &= E[D^2(n+1)] - \frac{n}{n+1}E[D^2(n)] + 4\frac{d_{n+1}}{n+1}\eta(n) + \frac{4}{n+1}\sum_{k=1}^{n-1}\eta(k)\left(H_{n-k}^{(2)} - 1\right) \\ &+ \frac{8}{(n+1)^2}\left(\frac{n+2}{n+1} - \frac{H_n}{n}\right)\left(n(n+1)H_n^{(2)} - n(n-2) - (2n+1)H_n\right). \end{aligned}$$

We can then write

$$v(n) = g_2(n) - \eta^2(n) = (n+1)\sum_{j=1}^{n-1}\frac{U(j)}{j+2} - \eta^2(n). \quad (17)$$

There is no clear simplification for this expression, but for moderate values of  $n$ , as in the tables we show, numerical evaluation is straightforward.

The following tables show statistics of the number of rotations incurred by the MOVE\_ONCE rule for different underlying RPVs. The Uniform is  $\mathbf{p} = \mathbf{e}$ , and Zipf is the RPV with  $p_i = 1/(iH_n)$ ; the ‘‘Random’’ RPV was created by sampling from a uniform distribution  $n$  values and normalizing. The ‘‘Linear’’ RPV has  $p_i = i/\alpha$ , where the normalizing constant  $\alpha$  is  $n(n+1)/2$ . The simulation results were surprisingly insensitive to the variation between these four distributions. As seen in the second table, where we list values computed from Eqs.(15,17) for the *uniform approximation*, there is a good agreement with  $\eta(n)$ , but  $v(n)$  is significantly too high. In that table we have results for two very different types of RPV: The ‘‘simple exponential’’ has  $p_i \propto 2^{-i}$ ; the ‘‘double exponential’’ is an RPV where  $p_i \propto 2^{-|n/2-i|}$ . The simulation results for these distributions differ from the ones in Table 1 — but we are at a loss to explain the observed behavior, or even identify a mechanism behind the remarkable values shown there. One can make of course the following observation: no other RPV in our study contains probability values that differ by so many orders of magnitude.

*Simulation notes:* The simulation was performed by creating for each RPV 200 initial insertion sequences (*i.e.* 200 initial trees); for each such tree 500 reference strings were generated, according to the RPV. The strings were of length equal to the tree size, since keys were sampled ‘‘without replacement’’ (by removing each referenced key from the set of eligible keys). Finally the number of rotations was counted for each reference sequence. The pseudo-random number generator used was the standard function `random()` available with most UNIX systems. For control, several points were also redone with the generator suggested in [28].

## 4 Reference Counters and Approximately Optimal Trees

The essential difference between the use of reference counters for the reorganization of linear lists and of BSTs is that for the first storage mode, the *Counter Scheme* (CS)—the policy that keeps the records ordered by their counters—converges to the optimal order without any extra costs, while there appears to be no such simple rule for BSTs that results in an optimal structure<sup>7</sup>.

<sup>7</sup>Indeed, if tree reorganization is ‘‘free’’—*i.e.*, we do not pay for rotations, then we can keep counters for as much as needed, for obtaining close estimates for the  $p_i$ 's; then, we can rotate the records to locations in the *optimal* tree, using, *e.g.*, the algorithm

tree size	Reference probability vector distribution type							
	Uniform		Zipf		Random		Linear	
	E[ $\Gamma_n$ ]	V[ $\Gamma_n$ ]	E[ $\Gamma_n$ ]	V[ $\Gamma_n$ ]	E[ $\Gamma_n$ ]	V[ $\Gamma_n$ ]	E[ $\Gamma_n$ ]	V[ $\Gamma_n$ ]
3	1.2249	0.7288	1.2195	0.7164	1.2554	0.6846	1.2581	0.7239
7	4.9672	3.6504	5.1395	3.9546	4.9119	3.6205	5.0112	3.9624
16	15.1815	12.2940	15.7865	14.1060	14.9606	11.5106	15.2472	13.8854
50	56.7692	48.4028	58.5563	53.5591	56.6323	46.8707	56.7847	49.8825
100	119.9026	100.5981	122.3002	108.7423	120.1409	103.1231	120.1612	104.4347
200	247.5589	209.7895	250.9379	221.2509	248.0150	213.0836	247.6357	215.0428
400	504.1035	431.3339	508.6558	445.5725	504.1811	415.5026	504.1592	435.9159
800	1018.6715	857.7135	1024.8838	878.5015	1019.3165	880.2130	1019.5081	888.4060

Table 1: Statistics of the total number of rotations done under the MOVE\_ONCE policy

tree size	Reference probability vector distribution type				Uniform approximation	
	Simple exponential		Double exponential		E[ $\Gamma_n$ ]	V[ $\Gamma_n$ ]
	E[ $\Gamma_n$ ]	V[ $\Gamma_n$ ]	E[ $\Gamma_n$ ]	V[ $\Gamma_n$ ]		
3	1.2534	0.7651	1.1973	0.7375	1.2222	0.76543
7	5.3234	4.6919	4.7662	3.8986	5.0030	4.24904
16	17.9996	26.2657	15.7973	18.7015	15.1063	14.91014
50	82.7853	297.4406	71.0903	142.8698	56.7651	61.06720
100	148.5056	833.1301	170.7486	567.8173	119.8920	131.67843
200	185.0276	2016.7063	292.4833	1791.9197	247.5024	274.62992
400	217.5541	2782.6140	377.9811	3492.0348	504.0948	562.13599
800	250.1933	3828.8917	449.0512	5977.6996	1018.6582	1138.6651

Table 2: Continued – Statistics of the total number of rotations done under the MOVE\_ONCE policy

#### 4.1 The Counter Scheme and Dynamic Programming

In [18] we showed that the CS is optimal for *linear lists*, not only in the limit, but also for every *finite* reference string. It is tempting to search for such a rule for BSTs. Consider the *monotonic* BST, which like the CS keeps records with higher counters closer to the root.

Let  $C_i^{(m)}$  be the number of times  $R_i$  is addressed during the first  $m$  references. A recursive definition of the monotonic BST is as follows: its root is the record  $R_j$  determined via  $C_j^{(m)} = \max_{1 \leq i \leq n} C_i^{(m)}$ . All the records with keys smaller than  $K_j$  are in the left subtree. All the records with keys larger than  $K_j$  are in the right subtree. Both subtrees are structured recursively in the same way. An operational definition of this reorganization rule states that whenever  $R_i$  is referenced,  $C_i^{(m)}$  is incremented by one, and  $R_i$  is rotated up in the tree, as long as the counter of its parent in the tree is smaller than  $C_i^{(m)}$ . The monotonic BST will usually fail to result in the optimal structure, for the same reason that using a *known* RPV to structure a BST monotonically

---

described in Section 2.

fails to reproduce the optimal tree (which is computed by the dynamic programming algorithm of Section 2).

It is also known that the cost of the monotonic tree can be unboundedly higher than that of the optimal one (specifically – their ratio can be as high as roughly  $n/\log n$ , see [26]).

But not all is lost. We can combine the MOVE\_ONCE and the CS with the dynamic programming algorithm as follows. The counters provide estimates for the RPV; the dynamic programming algorithm can then be used to construct a tree that is optimal for the estimates (usually, this tree would be suboptimal). The computation time is  $\Theta(n^2)$ : this is non-trivial for a large tree. Hence we would like to do it once, and do it right. This requires

1. that the estimates should be good enough for the deviation from optimality to be tolerable;
2. an efficient management of the counters, that will minimize their space overhead. As we show below, the total number of references needed is typically a moderate multiple of  $n$ . Hence, unless for some reason very small counting registers must be used, we need not worry about their potential overflow.

These points—especially the second one—suggest that the procedure needs a stopping criterion, a way to determine when the estimates are good enough to stop counting. The criterion must connect the total number of references (possibly with some information about the estimated RPV) and the nearness of the suboptimal cost value to the optimal one. We suggest the following compound policy MOUCS (for Move Once and Use Counter Scheme):

**Reorganization Rule MOUCS.** This rule has two phases:

Phase A: Use the rule MOVE\_ONCE and also compile reference counters,  $C_i^{(m)}$  for  $R_i$ , during a total of  $m = m_0$  references. A suitable value for  $m_0$  is shown below.

Phase B: Compute the estimators<sup>8</sup> of  $p_i$ , denoted by  $\hat{p}_i$ , and equal to the normalized value of  $(1/n^5) + (C_i^{(m)}/m)$ :

$$\hat{p}_i = \left( \frac{C_i^{(m)}}{m} + \frac{1}{n^5} \right) \frac{n^4}{n^4 + 1} = \frac{C_i^{(m)}}{m} \frac{n^4}{n^4 + 1} + \frac{1}{n(n^4 + 1)}. \quad (18)$$

Compute the “ostensibly optimal” BST using these weights as input for the dynamic programming algorithm, restructure the tree accordingly and stop reorganizing.

For the access cost during Phase A we have an explicit, if cumbersome, result in equation (7). Allen and Munro [1] use equation (7) to show that the MTR rule produces a tree with an expected access cost that differs from its limiting value by at most one, within  $\lceil n \log n / e \rceil$  references<sup>9</sup>. They also show, however, that this limiting value,  $C(n, \text{MTR}|\mathbf{p})$ , can exceed the *optimal* one by some 40%. We would like to do better.

In the following result we quantify the efficiency of the MOUCS rule in terms of convergence to the optimal tree cost against the length of the request sequence. We show that large trees need even fewer references in phase A than Allen and Munro suggest, and provide expected access cost which is closer to the optimum.

Denote by  $\hat{C}(\text{OPT}|\mathbf{p})$  the expected access cost to the tree built in Phase B for the estimate  $\hat{\mathbf{p}}$  to the RPV  $\mathbf{p}$ .

---

<sup>8</sup>The ratio  $C_i^{(m)}/m$  is a sufficient and unbiased estimator of  $p_i$ . The reason we go beyond it is due to technicalities of the proof of Theorem 5.

<sup>9</sup>The remarkable fact about this value is that it is far smaller than the expected number of references before all records are referenced at least once!

**Theorem 5:** For any RPV  $\mathbf{p}$ ,  $\delta > 0$ ,  $0 < \alpha < 1$ , and sufficiently large  $n$  (which depends on  $\delta$  and  $\alpha$ ),

$$\Pr[\hat{C}(\text{OPT}|\mathbf{p}) - C(\text{OPT}|\mathbf{p}) \geq \delta] \leq \alpha \quad (19)$$

after a sequence of  $m_0$  accesses to the tree, where

$$m_0 = d \frac{n(\lg n)^2}{\delta^2 \alpha}, \quad (20)$$

where  $d$  is a constant.

We use in the proof two lemmas which relate weights (= access probabilities) of subtrees to their position in the optimal BST.

Given a set of records  $R_1, \dots, R_m$ , with weights  $p_1, \dots, p_m$  respectively, let  $T_m$  be an optimal BST made of these records (which may be a subtree of a larger BST). Let  $r, t, B$  be the path from the root  $r$  to some node  $B$  in level 3. Let  $T_1$  and  $T_2$  be the two subtrees under the node  $B$ . We denote by  $T_3$  the subtree rooted in the sibling of  $t$  in  $T_m$  and by  $T_4$  the subtree rooted in the sibling of  $B$  in  $T_m$ . (Examples for two possible structures of the tree  $T_m$  are given in Figure 2 and Figure 3(a) in the Appendix). Let  $P = \sum_{i=1}^m p_i$ , and denote by  $P_t$  and  $P_B$  the weights of the subtrees rooted at  $t, B$  respectively.

**Lemma 4:** Given the tree  $T_m$ , and any  $a \in [0, 1]$

$$\text{either } P_t \leq aP \text{ or } P_B < (1 - a)P. \quad (21)$$

We give the proof in the Appendix. The next lemma will be used to bound from above the maximal level of an item in an optimal BST.

**Lemma 5:** For any set of elements  $R_1, \dots, R_m$  with the corresponding weights  $p_1, \dots, p_m$ , denote by  $l_i$  the level of  $R_i$  in the optimal BST. Then

$$p_i \leq \phi^{l_i-1} \implies l_i \leq 1 + \frac{\log p_i}{\log \phi}, \quad (22)$$

where  $\phi$  is given by  $(\sqrt{5} - 1)/2$ , the familiar golden ratio<sup>10</sup>.

The proof is similar to the proof of Lemma 2 in [26]. We give it in the Appendix.

**Proof of Theorem 5:** Let  $\hat{\mathbf{p}}$  be the estimate for  $\mathbf{p}$  obtained after a sequence of  $m$  references, where  $\hat{\mathbf{p}}(i)$  is defined in (18). Let  $\hat{l}_i$  and  $l_i$  be the levels of  $R_i$  in the optimal trees for  $\hat{\mathbf{p}}$  and  $\mathbf{p}$  respectively. From the optimality of  $\{\hat{l}_i\}$  for  $\{\hat{\mathbf{p}}\}$ , we have that  $\sum_{i=1}^n l_i \hat{p}_i - \sum_{i=1}^n \hat{l}_i \hat{p}_i \geq 0$ . Hence,

$$\begin{aligned} 0 \leq \hat{C}(\text{OPT}|\mathbf{p}) - C(\text{OPT}|\mathbf{p}) &= \sum_{i=1}^n \hat{l}_i p_i - \sum_{i=1}^n l_i p_i \quad (23) \\ &= \left( \sum_{i=1}^n \hat{l}_i p_i - \sum_{i=1}^n \hat{l}_i \hat{p}_i \right) + \left( \sum_{i=1}^n \hat{l}_i \hat{p}_i - \sum_{i=1}^n l_i \hat{p}_i \right) + \left( \sum_{i=1}^n l_i \hat{p}_i - \sum_{i=1}^n l_i p_i \right) \\ &\leq \sum_{i=1}^n \hat{l}_i p_i - \sum_{i=1}^n \hat{l}_i \hat{p}_i + \sum_{i=1}^n l_i \hat{p}_i - \sum_{i=1}^n l_i p_i. \end{aligned}$$

<sup>10</sup>Usually,  $(\sqrt{5} + 1)/2$  is called the golden ratio.

Therefore, for  $\delta > 0$  and  $0 < \alpha < 1$ , it is sufficient to look for the minimal value of  $m$ , call it  $m_0$ , satisfying

$$\Pr \left( \sum_{i=1}^n l_i \hat{p}_i - \sum_{i=1}^n l_i p_i \geq \frac{\delta}{2} \right) \leq \frac{\alpha}{2}, \quad (24)$$

and a similar relation with  $\hat{l}_i$  replacing  $l_i$ . We handle these two cases separately.

- (i) First we find a value  $m_0$  for which inequality (24) is satisfied. Since  $C_i^{(m)}$  has the marginal distribution  $\text{Bin}(m, p_i)$ , we can compute the moments of the estimates  $\hat{p}$ , and find

$$E[\hat{p}_i - p_i] = \frac{(1/n) - p_i}{n^4 + 1},$$

and

$$\begin{aligned} E[(\hat{p}_i - p_i)^2] &= \left( \frac{n^4}{n^4 + 1} \right)^2 \frac{p_i(1-p_i)}{m} + \left( \frac{(1/n) - p_i}{n^4 + 1} \right)^2 \\ &= \frac{p_i(1-p_i)}{m} (1 - O(n^{-4})) + O(n^{-8}), \end{aligned} \quad (25)$$

uniformly in  $p$ ,  $m$  and  $i$ . Using the Bienaymé-Chebyshev inequality (which says that  $\Pr[|X| > t] \leq E[X^2]/t^2$ ), in relation (24), we “solve” for  $m_0$  and find that it is sufficient to have

$$m_0 \geq 8 \frac{\left( \frac{n^4}{n^4 + 1} \right)^2}{\delta^2 \alpha - 5n^{-4}} \left( \sum_{i=1}^n l_i^2 p_i(1-p_i) - \sum_{i=1}^n \sum_{j \neq i} p_i p_j l_i l_j \right). \quad (26)$$

We can simplify the bound by deleting the last double sum in the right-hand side of (26), (which amounts to disregarding the negative covariances between the counters), and simplifying further, by using the bound from Lemma 5, which provides  $l_i \leq 1 + (\log p_i)/(\log \phi)$ , hence it suffices that

$$m_0 \geq 8 \frac{\left( \frac{n^4}{n^4 + 1} \right)^2}{\delta^2 \alpha - 5n^{-4}} \left( \sum_{i=1}^n [1 + (\log p_i)/(\log \phi)]^2 p_i(1-p_i) \right). \quad (27)$$

Finally, to remove the dependence on the RPV, we notice that each of the terms in the sum is at most  $\max_{p \in (0,1)} [1 + (\log p)/(\log \phi)]^2 p(1-p) = 3.115307$  (at  $p = 0.1453$ ), hence the value defined in (20) satisfies the requirement from  $m_0$ ; since in this case we do not need the factor  $\lg^2 n$ , the exact value of the constant is of little importance.

- (ii) We now show that the inequality

$$\Pr \left( \sum_{i=1}^n \hat{l}_i p_i - \sum_{i=1}^n \hat{l}_i \hat{p}_i \geq \frac{\delta}{2} \right) \leq \frac{\alpha}{2} \quad (28)$$

is satisfied with  $m_0$  as given in (20). Define  $Z_i := |\hat{p}_i - p_i|$ , and  $Z$  as their sum,

$$Z := \sum_{i=1}^n Z_i. \quad (29)$$

Note that, by Lemma 5,  $\hat{l}_i \leq 1 + 7.25 \lg n < 8 \lg n$  (for all large  $n$ ); hence, for obtaining a bound on  $m$ , it is sufficient to find a value of  $m$  such that

$$\Pr \left[ \sum_{i=1}^n |\hat{p}_i - p_i| \geq \frac{\delta}{16 \lg n} \right] \leq \frac{\alpha}{2}. \quad (30)$$

Next, we find that

$$\begin{aligned} E[Z] = \sum_{i=1}^n E[Z_i] &\leq \sum_{i=1}^n \sqrt{\frac{p_i(1-p_i)}{m} + \left( \frac{(1/n) - p_i}{n^4 + 1} \right)^2} \\ &= \frac{1}{\sqrt{m}} \sum_{i=1}^n \sqrt{p_i(1-p_i)} (1 + O(n^{-4})) \leq \sqrt{\frac{n}{m}} (1 + O(n^{-4})) \end{aligned}$$

uniformly in  $\mathbf{p}$ . The last inequality derives from  $\mathbf{p} = \mathbf{e}$  being the constrained maximum of the sum  $\sum_{i=1}^n \sqrt{p_i(1-p_i)}$ . We proceed to derive a bound for  $E[Z^2]$ . From (29) we note that

$$E[Z^2] = \sum_{i=1}^n \sum_{j=1}^n E[Z_i Z_j].$$

Now, for any  $1 \leq i, j \leq n$ ,

$$E[Z_i Z_j] \leq \frac{1}{2} (E[Z_i^2] + E[Z_j^2]) \quad \implies \quad E[Z^2] \leq n \sum_{i=1}^n E[Z_i^2].$$

Now, from (25) we have that

$$E[Z^2] \leq n \sum_{i=1}^n \frac{p_i(1-p_i)}{m} (1 + O(n^{-4})) \leq \frac{n}{m} (1 + O(n^{-4})). \quad (31)$$

The desired value of  $m_0$  is obtained by using, as in part (i), the Bienaymé-Chebyshev inequality.  $\blacksquare$

If we were trying to “negotiate” these inequalities for the smallest constant  $d$  in Eq.(20) we could have taken advantage of the very different bounds generated by the two parts, relations (24, 28). The second would be allowed to “use up” nearly all of  $\delta$ , and the first one would have been required to take care of a mere sliver, equal to  $\delta/\lg n$ . However, this is not our goal here; the proof technique we could use is so prodigal, that getting close to the correct functional form is the best we could hope for. As the experimental results suggest, we have not quite achieved that yet.

**Experimental results:** The procedure we used to derive the bound in (20) suggests that, for most distributions, the stopping point for the execution of MOUCS can be taken significantly lower. Table 3 verifies this for a large

$\alpha \backslash n$	10	20	50	100	200
0.25	0.0748	0.0847	0.0947	0.0947	0.0922
0.1	0.0498	0.0573	0.0474	0.0449	0.0424
0.05	0.0374	0.0324	0.0324	0.0242	0.0224
0.01	0.0125	0.0099	0.0075	0.0049	0.0049

Table 3: The required length for the reorganization process under MOUCS to approach the optimum within a difference  $\delta = 0.25$  with probability higher than  $1 - \alpha$ . The table shows the ratio  $m'/(n/(\delta^2\alpha))$ .

$\delta \backslash n$	50
0.9	0.1720
0.7	0.1346
0.4	0.0847
0.2	0.0573
0.1	0.0374
0.06	0.0249
0.01	0.0049
0.001	0.00005

Table 4: The ratio  $m'/(n/(\delta^2\alpha))$  for  $n = 50$  and  $\alpha = 0.15$ .

set of randomly generated RPVs. For each pair  $(n, \alpha)$ , and  $\delta = 0.25$ , we estimated the stopping point  $m'$  for a set of 500 RPVs of the type we called “random” in Table 1. Table 3 presents the ratio  $m'/(n/(\delta^2\alpha))$  (averaging the 500 values). The results indicate a *linear* dependence on  $n$ . Moreover, for most of the RPVs we tried,  $m'$  was much smaller than  $n/(\delta^2\alpha)$ . Table 3 also suggests that the stopping point depends on  $1/\alpha^y$  for some  $0 < y \leq 0.5$ , whereas we could only prove a bound using  $y = 1$ . Table 4 shows that for fixed values of  $n$  and  $\alpha$ , the ratio  $(n/(\delta^2\alpha))/m'$  is a decreasing function of  $\delta$ . It is more likely then, that the stopping point depends on  $1/\delta$  or even a smaller value rather than  $1/\delta^2$ .

We can summarize our simulation results on the stopping point for MOUCS in the following.

**Conjecture 1:** *For any unknown RPV  $p$ , the expected access cost to a BST rearranged by the MOUCS approaches the optimal average cost within a difference of  $\delta$ , with probability higher than  $1 - \alpha$ , following  $c \cdot n/\delta^x \alpha^y$  references, for some small constant  $c$ ,  $x$  approximately 1, and  $0 < y < 0.5$ .*

## 4.2 The Counter Scheme and Weight Balanced Trees

As the computation of the approximation to the optimal tree requires  $\Theta(n^2)$  steps, there is interest in a more efficient construction of *nearly* optimal BSTs. This holds even when the RPV is known (unlike our statistical scenario), when the truly optimal tree is available. A suitable candidate appears to be the *weight balanced tree*, which is constructed as follows.

**Weight Balancing Rule [14]:** Choose the root so as to equalize the weight of the left and right subtree as much as possible, then proceed similarly recursively on the subtrees.

It is shown in [12] that a weight balanced tree is constructible with time and space complexity  $\Theta(n)$ . Bayer showed in [3] that, for a given RPV  $\mathbf{p}$ , the average access cost to the weight balanced tree, denoted by  $C(\text{WB}|\mathbf{p})$ , satisfies

$$C(\text{WB}|\mathbf{p}) - C(\text{OPT}|\mathbf{p}) \leq \lg H + \lg e + 1 \approx \lg H + 2.44269\dots < 1.45 \ln \ln n + 2.98, \quad (32)$$

where  $H = \sum p_i \lg p_i^{-1}$  is the entropy of the RPV  $\mathbf{p}$ . Since  $H \in [0, \lg n]$ , this bound looks acceptable. An interesting rule, which, like the MOUCS rule, reorders the tree while updating the counters, is based on the near-optimal weight-balanced tree: during the reference sequence the tree is kept weight-balanced as estimated by the counters. (We use the same estimates we used for the MOUCS.) Since the difference between the estimates and the true access probabilities decreases monotonically (in expectation), we conjecture that this rule provides at each stage a closer approximation to the weight-balanced tree which could be constructed if  $\mathbf{p}$  were known.

We show below that for any distribution, the cost of the estimated weight balanced tree approaches within a difference  $\delta$ , with high probability, the cost of the “true” weight-balanced tree (based on the *unknown* RPV  $\mathbf{p}$ ) within a number of accesses given by equation (20). Denote by  $\hat{C}(\text{WB}|\mathbf{p})$  the cost of the balanced tree constructed by the estimate  $\hat{\mathbf{p}}$  for the RPV  $\mathbf{p}$ .

**Theorem 6:** For any  $\delta > 0$  and  $0 < \alpha < 1$ , and for any unknown RPV  $\mathbf{p}$ , and sufficiently large  $n$ , which depends on  $\delta$  and  $\alpha$ ,

$$\Pr(|\hat{C}(\text{WB}|\mathbf{p}) - C(\text{WB}|\mathbf{p})| > \delta) < \alpha, \quad (33)$$

after a sequence of  $m_0$  accesses to the tree, where

$$m_0 = \frac{dn(\lg n)^2}{\delta^2 \alpha}. \quad (34)$$

and  $d \geq 1$  is a constant.

We use in the proof a bound on the structure of these trees that appears identical with the one derived in Lemma 5 for the optimal BST; it was first shown by Mehlhorn in [26]:

**Lemma 6:** ([26]) For any set of elements  $R_1, \dots, R_m$  with the corresponding weights  $p_1, \dots, p_m$ , denote by  $l_i$  the level of  $R_i$  in the weight balanced BST. Then

$$p_i \leq \varphi^{l_i - 1}, \quad (35)$$

with  $\varphi$  as defined in Lemma 5.

**Proof of Theorem 6:** Using the proof of Theorem 5, with  $\hat{C}(\text{OPT}|\mathbf{p})$  and  $C(\text{OPT}|\mathbf{p})$  replaced by  $\hat{C}(\text{WB}|\mathbf{p})$  and  $C(\text{WB}|\mathbf{p})$  respectively, and  $\hat{l}_i, l_i$  denoting the levels of  $R_i$  in the weight balanced trees for  $\hat{\mathbf{p}}, \mathbf{p}$  respectively,  $m_0$  can be taken in that proof (where Lemma 6 is now used in place of Lemma 5). ■

The identical bounds on the structure of these two types of trees suggest at once that they are typically rather close, and that these bounds do not characterize them very tightly.

## 5 Discussion

In this paper we have studied reorganization rules for a BST, where accesses to the tree are generated independently by a fixed unknown distribution. We considered the MOVE\_ONCE rule, which moves each item at most once, when it is first accessed. The MOVE\_ONCE was shown to yield the same expected access cost as the well known MTR rule, which keeps reorganizing the tree. We have also shown that the expected number of rotations under the MOVE\_ONCE rule is at most  $2(n+1)H_n - 4n$ . Our experimental study suggests, that this bound can be tightened, to a bound that is  $O(n)$ . In Eq.(15) we showed that for  $p = e$ , the expected number of rotations under the MOVE\_ONCE is bounded by  $c_1n - c_2 \ln n$ , where  $c_1, c_2 \geq 1$  are small constants. A challenging and interesting problem is to show similar bounds for other distributions.

We considered also the MOUCS rule, which combines the MOVE\_ONCE with the usage of reference counts. We showed that when the distribution is static for sufficiently long durations, then for large trees the MOUCS rule:

- (i) provides an on-going reorganization of the tree which improves the expected access cost and requires a low number of rotations,
- (ii) yields on termination a search tree with access cost which is arbitrarily close to that of the optimal tree, using statistics accumulated from a reference sequence whose length is  $O(n(\lg n)^2)$ .

It is an open challenge to derive a bound on the stopping point of MOUCS that corresponds more closely to the experimental results, as summarized in Conjecture 1. We believe that the discrepancy does not represent a possible worst case, but rather our failure to bound the sums that appear in equation (23) more tightly.

The relative efficiency of the MOVE\_ONCE rule compared to the scheme which keeps the tree weight balanced by the counters is still open. For long access sequences, we would expect the MOVE\_ONCE to be inferior with respect to the total average cost of the sequence, but it will retain its advantage of low reorganization cost.

The main assumption driving the results above is the stationarity of the reference process. While systems may rest unchanged over periods long enough for the analysis to be applicable, they all do change ultimately. It is of interest to extend the results to quasi-stationary systems. Indeed, when the RPV changes slowly over time there is nothing to be gained from the presented approach, but it can be useful for systems with *phase* structure, where each phase is guaranteed to be at least  $k$  references long, for some  $k \gg 1$ , and throughout a single phase the RPV is fixed.

Another issue concerns the cost of computing (which includes the construction of) the optimal tree. The best known algorithm, as presented in Section 2, uses  $\Theta(n^2)$  steps—with a non-trivial coefficient—and has the same space complexity [21]. We showed that MOVE\_ONCE produces a tree with a “nearly optimal” cost—though its shape could differ radically from that of the optimal one. The question of whether a more efficient algorithm is available which uses the structure of the MOVE\_ONCE tree as a starting point is still open.

## Acknowledgments

We would like to thank Shai Ben-David and Mark Wegman for helpful comments on this paper. Two anonymous referees have read the manuscript carefully, uncovered errors, drew our attention to references [6]–[9], and contributed significantly to the accuracy and the clarity of the presentation.

## References

- [1] B. Allen, I. Munro, “Self-Organizing Search Trees”, *Jour. of the ACM* **25**, #4, 526–535, 1978.
- [2] M.J. Atallah, S.R. Kosaraju, L.L. Larmore, G.L. Miller, S-H Teng, “Constructing Trees in Parallel”, In *Proc. of the 2nd IEEE Symposium on Parallel Algorithms and Architectures*, 1989.
- [3] P.J. Bayer, “Improved Bounds on the Costs of Optimal and Balanced Search Trees”, Tech. Memo. 69, Proj. MAC M.I.T. Cambridge MA 1975.
- [4] J. Bitner, “Heuristics that Dynamically Organize Data Structures”, *SIAM J. Comput.*, **8**,1, pp. 82-110, 1979.
- [5] A. Boneh, M.Hofri, “The Coupon-Collector Problem Revisited – A Survey of Engineering Problems and Computational Methods.” *Stochastic Models*, **13**, #1 39–66, 1997.
- [6] R.P. Dobrow, “The Move-to-root Rule for Self-organizing Trees with Markov dependent Requests”, *Stochastic Anal. Appl.*, 14, #1, pp. 73–87, 1996.
- [7] G.G. Brown, B.O. Shubert, “On random binary Trees”, *Mathem. Oper. Res.*, **9**, #1, 43–65, 1984.
- [8] R.P. Dobrow, J.A. Fill, “The Move-to-front Rule for Self-organizing Lists with Markov Dependent Requests”, in *Discrete Probability and Algorithms*, Aldous & Diaconis & Spencer & Steele (Eds.), Springer-Verlag, 1995.
- [9] R.P. Dobrow, J.A. Fill, “On the Markov Chain for the Move-to-Root Rule for Binary Search Trees”. *Ann. Appl. Probability*, **5**, 1, pp. 1–19, 1995.
- [10] R.P. Dobrow, J.A. Fill, “Rates of Convergence for the Move-to-Root Markov Chain for Binary Search Trees”. *Ann. Appl. Probability*, **5**, 1, pp. 20–36, 1995.
- [11] J.A. Fill, “On the Distribution of Binary Search Trees under the Random Permutation Model”, *Random Structures & Algorithms*, **8**, 1, pp. 1–25, 1996.
- [12] M. L. Fredman, “Two Applications of a Probabilistic Search Technique: Sorting X+Y and Building Balanced Search Trees”, *7th ACM Symp. on Theory of Computing*, Albuquerque 1975.
- [13] I. Galperin, R. Rivest, “Scapegoat Trees”, In *Proc. of the 4th ACM-SIAM Symposium on Discrete Algorithms*, Austin, TX, January 25–27, 1993.
- [14] R. Güttler, K. Mehlhorn, W. Schneider, “Binary Search Trees: Average and Worst Case Behavior”, *Jour. of Information Processing and Cybernetics*, **16**, 41–61, 1980.
- [15] Hibbard, T.N., “Some Combinatorial Properties of Certain Trees with Applications to Searching and Sorting”, *Jour. of the ACM*, **9**, 13–28, 1962.
- [16] M. Hofri, *Analysis of Algorithms*, Oxford University Press, 1995.

- [17] M. Hofri, H. Shachnai, “Self-Organizing Lists and Independent References – a Statistical Synergy”, *Jour. of Alg.*, **12**, 533–555, 1991.
- [18] M. Hofri, H. Shachnai, “On the Optimality of Counter Scheme for Dynamic Linear Lists”, *Inf. Process. Lett.*, **37**, 175–179, 1991.
- [19] T.C. Hu, K. C. Tan, “Least Upper Bound on the Cost of Optimum Binary Search Trees”, *Acta Informatica*, **1**, 307–310, 1972.
- [20] D.G. Kirkpatrick, T.M. Przytycka, “Parallel Construction of Binary Trees with Almost Optimal Weighted Path Length”, In *Proc. of the 2nd IEEE Symposium on Parallel Algorithms and Architectures*, 1990.
- [21] D.E. Knuth, “Optimum Binary Search Trees”, *Acta Informatica* **1**, 14–25, 1971.
- [22] D.E. Knuth, *The Art of Computer Programming, Vol 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 2nd Edition, 1998.
- [23] T. W. Lai, D. Wood, “Adaptive Heuristics for Binary Search Trees and Constant Linkage Cost”, In *Proc. of the 2nd ACM-SIAM Symposium on Discrete Algorithms*, pp. 72–77, San Francisco, CA, January 28–30, 1991.
- [24] H. M. Mahmoud, “On the Average Internal Path Length of  $m$ -ary Search Trees”, *Acta Informatica* **23**, 111–117, 1986.
- [25] H. M. Mahmoud, *Evolution of Random Search Trees*, John Wiley, New York, 1992.
- [26] K. Mehlhorn, “Nearly Optimal Binary Search Trees”, *Acta Informatica* **5** 287–295, 1975.
- [27] K. Mehlhorn, A. Tsakalidis, “Data Structures”. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Vol A, chapter 6, pp. 301–341, Elsevier, 1990.
- [28] S.K. Park, K.W. Miller: “Random Number Generators: Good Ones Are Hard to Find”, *Commun. ACM* **31**, 1192–1201, 1988. With correspondence “Remarks on Choosing and Implementing Random Number Generators”, *Commun. ACM* **36**, 105–110, 1993.
- [29] D.D. Sleator, R.E. Tarjan, “Self-Adjusting Binary search Trees”, *Jour. of the ACM* **32**, #3, 652–686, 1985.
- [30] D.D. Sleator, R.E. Tarjan, “Amortized Efficiency of List Update and Paging Rules”, *Commun. ACM* **28**,2, pp. 202–208, 1985.

## Appendix

**Proof of Lemma 1:** Consider an arbitrary pair of records,  $R_i$  and  $R_j$ . We look for the sufficient and necessary conditions for  $R_i$  to be an ancestor of  $R_j$  in  $T_{MTR}(I)$  and  $T_{MO}(I)$ . The structure of the BST is determined (uniquely) once we specify the ancestor-offspring relation for all pairs of nodes in the tree. The lemma will

be established if we show that  $R_i$  is ancestor of  $R_j$  in  $T_{MO}(I)$ , iff  $R_i$  is ancestor of  $R_j$  in  $T_{MTR}(I^R)$ , for all  $1 \leq i, j \leq n$ .

Recall, that  $K_i, K_j$  are the keys of  $R_i, R_j$  respectively. Without loss of generality, we assume that  $K_i < K_j$ . The *interval set* of the ordered pair  $(i, j)$  includes  $R_i, R_j$  and all other records whose keys lie between  $K_i$  and  $K_j$ . Formally,

$$IS(i, j) = \{R_l \mid K_i \leq K_l \leq K_j\} .$$

We avoid sticky notation by assuming all records were referenced at least once. For sufficiently long reference strings this holds with an arbitrarily high probability. (On the other hand, we should mention that when the RPV is far from uniform we expect to obtain informative reference counts long before the above assumption is satisfied). Whatever the case may be, all the claims here hold also for strings that cover only part of the set of records.

The proof devolves from properties of the rotation operation. Refer to Figure 1. We say that “the rotated node” is the one that gets to a higher (=lower numbered) level. The second node taking part in the rotation is “the lowered node”. For example, the rotated node in the left tree in Figure 1 is  $a$ , and the lowered node is  $b$ . The salient properties are:

- (a) When a node is rotated it continues to be an ancestor to all of its previous offspring, and *becomes* an ancestor to the lowered node and its other subtree. The effect of a sequence of rotations of a single node is cumulative.
- (b) A lowered node *loses* as offspring the rotated node and its left (right) subtree when the rotation is to the right (left).

**Claim 1:** *A necessary and sufficient condition for  $R_i$  to be ancestor of  $R_j$  under the MOVE\_ONCE (MTR) rule, is that  $R_i$  is the first (last) node to be referenced in  $IS(i, j)$ .*

**Proof:** The proof for MOVE\_ONCE is immediate if we consider the subtree in the initial tree that contains  $IS(i, j)$ . References (and the consequent rotations) of records outside of this subtree do not change its structure, but may change its level only. References to records in it which are outside of  $IS(i, j)$ , before  $R_i$  is used, will make them ancestors of the entire  $IS(i, j)$ . Once  $R_i$  is referenced (and rotated as high as necessary) it will be ancestor to all other nodes in  $IS(i, j)$ , and since it will not be lowered again, this relation will be maintained indefinitely. Hence the sufficiency.

For the necessity: If some  $R_k \in IS(i, j)$ ,  $k \neq i, j$  is referenced before  $R_i$ , it will put  $R_i$  and  $R_j$  in its two separate subtrees, again indefinitely. And lastly, if  $R_j$  is the first to be referenced in  $IS(i, j)$  it will be the ancestor of  $R_i$ .

The claim for MTR holds due to the fact that a referenced node is rotated all the way to the root. For sufficiency: at the last reference to  $R_i$  it reaches the root, and all the rest of  $IS(i, j)$  is in its right subtree. Subsequent references to records with lower keys (which are in the left subtree of  $R_i$ ) will leave it as ancestor of all  $IS(i+1, j)$ . References to records with keys higher than  $K_j$ , will get during their sequence of rotations to have  $IS(i+1, j)$  in their left subtrees, and will allow  $R_i$  to retain its ancestry with respect to this set (property (b) above). The necessity is similar to the previous case. A subsequent reference to an intermediate key in  $IS(i, j)$  will place  $R_i$  and  $R_j$  in two disjoint subtrees. ■

The statement of the lemma is now obvious. ■

We remark that similar considerations also allow us to determine conditions under which  $R_i$  ends up as the *immediate* parent of  $R_j$ : in  $T_{MO}(I)$  it is required that  $R_i$  and  $R_j$  were the first two records from  $IS(i, j)$  to

be referenced, in that order, and the same state will be found in  $T_{MTR}(I)$  when  $R_j$  and  $R_i$  were the last two records referenced, in that order, from  $IS(i, j)$ .

**Proof of Lemma 2:** Let  $T_n^{(m-1)}$  be the BST obtained from the initial tree  $T_n$  after  $(m-1)$  references. We assume throughout the proof that  $m$  is incremented whenever a node is requested for the first time (other accesses do not cause any changes in the tree). Suppose that the  $m$ th reference is to  $R_k$ , which is stored in the node  $b$ . Let  $r'$  be the last node on the path from  $b$  to the root of  $T_n^{(m-1)}$ , which was not accessed yet. In other words, among the nodes on that path that were not accessed,  $r'$  has the smallest level (and so, its parent has already been accessed and is fixed in position). Without loss of generality, we assume that  $b$  is in the right subtree of  $r'$ . Denote by  $B, C$  the left and right subtrees of  $b$  respectively; let  $D$  be the left subtree of  $r'$ .

Then, by the definition of the MOVE ONCE rule, when  $R_k$  completes the required sequence of rotations, i.e., becomes repositioned at node  $r'$ , each element  $R_i$ ,  $i \neq k$ , satisfies one of the following:

- (i) If  $R_i \in C$ , then  $l_i^{(m)} < l_i^{(m-1)}$ . In addition,  $b$  is now a referenced ancestor of  $R_i$ ; therefore  $\gamma_i^{(m)} < \gamma_i^{(m-1)}$ .
- (ii) If  $R_i \in B$ , then  $l_i^{(m)} = l_i^{(m-1)}$ , and  $b$  is a referenced ancestor of  $R_i$ , therefore  $\gamma_i^{(m)} = \gamma_i^{(m-1)} - 1$ .
- (iii) If  $R_i \notin B, C$ , but  $R_i$  was in the right subtree of  $r'$  in  $T_n^{(m-1)}$ , then  $l_i^{(m)} = l_i^{(m-1)} + 1$ . Again, since  $b$  has become a referenced ancestor of  $R_i$ , we get that  $\gamma_i^{(m)} = \gamma_i^{(m-1)}$ .
- (iv) If  $R_i \in D$ , then (as in (iii)) we have  $\gamma_i^{(m)} = \gamma_i^{(m-1)}$ .
- (v) If  $R_i$  is not in the subtree rooted in  $r'$  in  $T_n^{(m-1)}$ , then clearly,  $l_i^{(m)} = l_i^{(m-1)}$  and  $\gamma_i^{(m)} = \gamma_i^{(m-1)}$ . ■

**Proof of Lemma 3:** The shape of a BST  $T_n$  is uniquely defined by the preorder enumeration sequence of its nodes, denoted by  $PO(T_n)$ . The uniqueness follows from the fact that  $PO(T_n)$  is one of the (usually many) insertion sequences that generates  $T_n$ . This in turn is evident from the recursive definition of the enumeration: it is an empty sequence for an empty tree, and otherwise

$$PO(T_n) = ( \text{root}(T_n), PO(\text{Left}(T_n)), PO(\text{Right}(T_n)) )$$

where  $\text{Left}(T_n)$  and  $\text{Right}(T_n)$  are the left- and right-subtrees of the root of  $T_n$ .

We now observe that when we rotate a node to the root, the BST changes, and its PO sequence then changes too, but it changes in a disciplined way. Specifically, if we rotate  $R_I$ , the relative order of the entries in  $PO(T_n)$  which are smaller than  $I$  is unchanged, and likewise for the relative order of the terms larger than  $I$ : all that may happen is that the interleaving of the two subsequences is different. The proof is by induction on the initial level of  $R_I$ . Clearly the rotation of the root is null, and no change is affected. Let us look at a rotation from level 2 to the root. Consider the two trees in Figure 1, let  $I = a$  and denote them, from left to right, by  $T_L$  and  $T_R$ . Their preorder sequences are

$$PO(T_L) = ( b, a, PO(A), PO(B), PO(C) ), \quad PO(T_R) = ( a, PO(A), b, PO(B), PO(C) ),$$

where the claimed invariance is manifest. The same holds for the reverse rotation, where  $I = b$ . A (single) rotation of a node in level  $k$  is a rotation to the root of a level- $(k-1)$  subtree, and no changes in the preorder sequence of the tree occur except those that are limited to this level- $(k-1)$  subtree, and they follow the above

scheme. The invariance is similarly preserved during the entire sequence of rotations needed to bring  $R_I$  to the root. Hence the observation.

Now for the claim of the Lemma: We consider a fixed value of  $I$ , the  $I - 1$  smallest keys in the tree (which will form the left subtree of the root when the entire rotation is done), and look at all  $n!$  insertion sequences and BSTs they generate. Since the order of the smallest  $I - 1$  keys in the preorder sequence of each tree is preserved during the complete rotation of  $R_I$ , from whatever original level it happened to land at, all the way to the root, we find in the final  $n!$  trees the same uniform distribution of permutations of these keys (with each permutation is repeated  $(n - I + 1)!$  times), as when a  $T_{I-1}$  is created with a uniform distribution over the insertion sequences. ■

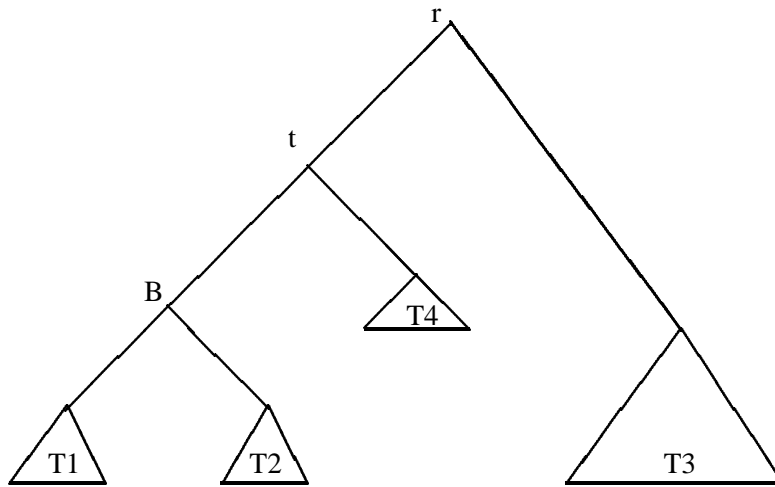


Figure 2: An optimal tree  $T$

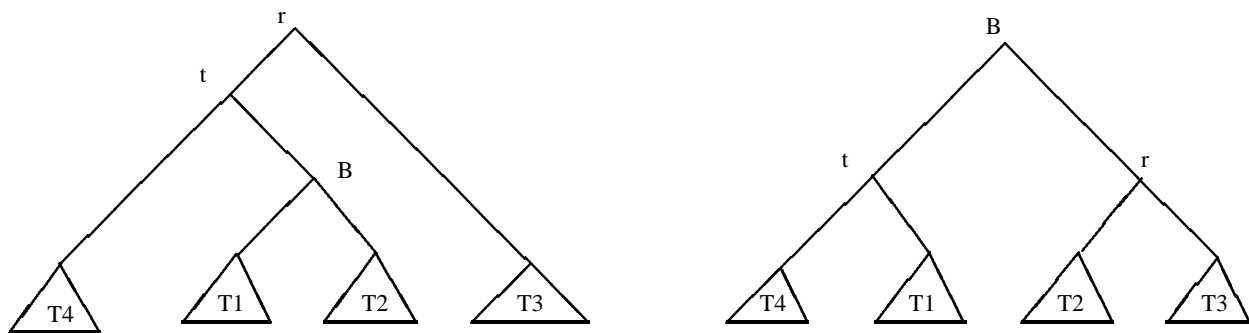


Figure 3: (a)  $T$

(b)  $T'$

**Proof of Lemma 4:** If  $P_t \leq aP$  we are done. Otherwise we consider the case where

$$P_t > aP. \tag{36}$$

There are two geometrically different cases:

1. If  $B$  is in the left subtree of  $t$  (as in Figure 2), then by the optimality of  $T$ , rotating  $t$  to the root would result in a possibly non-optimal tree. Note, that after this rotation, any node in the subtree  $t \cup B \cup T_1 \cup T_2$  gets closer to the root, namely, decreases its level by 1; for each node in  $r \cup T_3$  the level increases by 1. In other words, such a rotation would add to the expected access cost the value  $p_r + P_3 - (P_B + p_t)$ . From the optimality of  $T$ , it follows that

$$p_t + P_B \leq p_r + P_3.$$

Therefore, since  $P_t = p_t + P_B + P_4$ ,

$$aP < p_t + P_B + P_4 \leq p_r + P_3 + P_4.$$

Hence,

$$P_B = P - (p_r + P_3 + P_4 + p_t) \leq P - (p_r + P_3 + P_4) < P(1 - a).$$

2. If  $B$  is in the right subtree of  $t$ , as shown in Figure 3(a), then the expected access cost to  $T$  is at most the expected access cost to  $T'$  (as given in Figure 3(b), and obtained by rotating  $B$  twice). Thus

$$p_r + P_3 \geq 2p_B + P_1 + P_2 > P_B$$

and since  $p_r + P_3 = P - P_t$ , using (36) we find

$$P_B < P_r + P_3 = P - P_t < P(1 - a).$$

(Observe, that the case where  $t$  is the root of the right subtree of  $r$  is symmetric). ■

**Proof of Lemma 5:** Let  $T$  be an optimal BST, and  $B$  an internal node at level  $b$ . Let its weight be given by  $p_B$ , and denote by  $w_b$  the weight of the subtree rooted at  $B$ . We need to show that  $p_B \leq \phi^{b-1}$ , and we shall prove the usually stronger claim, that

$$w_b \leq \phi^{b-1}. \quad (37)$$

Then, since  $p_B \leq w_b$ , the proof will be complete.

First, note that (37) holds trivially for  $b = 1$ , thus we need to show the claim for  $b \geq 2$ . Let  $(\text{root} = B_{j_1}, \dots, B_{j_b} = B)$  be the order of the nodes on the path from the root to  $B$ , and let  $w_i$  be the weight of the subtree rooted at  $B_{j_i}$ . We will use Lemma 4, and its structure suggests that we show by induction the following: for all  $1 \leq i \leq b$ ,

$$\text{either } w_{i-1} \leq \phi^{i-1} \text{ or } w_i \leq \phi^i. \quad (38)$$

**Base:** For  $i = 1$  the *or* part, and for  $i = 2$  the *either* part of the claim are immediate from the fact that the weight of a tree is at most 1.

**Induction Step:** For  $i > 2$ , from the induction hypothesis we have

$$\text{either } w_{i-2} \leq \phi^{i-2} \text{ or } w_{i-1} \leq \phi^{i-1}.$$

If  $w_{i-1} \leq \phi^{i-1}$  then we are done. Otherwise, we use Lemma 4 with  $a = \phi$ , for the subtree rooted at  $B_{j_{i-2}}$ , and since  $w_{i-2} \leq \phi^{i-2}$ , we get that either

$$w_{i-1} \leq \phi w_{i-2} \leq \phi^{i-1}$$

or, since  $1 - \varphi = \varphi^2$ ,

$$w_i \leq \varphi^2 w_{i-2} \leq \varphi^i.$$

Finally, following the last step, we observe that trivially

$$w_b = \min(w_b, w_{b-1}) \leq \varphi^{b-1},$$

hence Eq.(37) holds. ■