Efficient Reorganization of Binary Search Trees

Micha Hofri[*]
Dept. of Computer Science
WPI
Worcester MA, 01609
e-mail: hofri@cs.wpi.edu

Hadas Shachnai[†]
Dept. of Computer Science
The Technion – IIT
Haifa, 32000 ISRAEL
e-mail:hadas@cs.technion.ac.il

WPI-CS-TR-98-26

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 elements. 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 we propose achieve a close approximation to the optimal BST, with lower organization costs than any previously studied.

We introduce the *Move Once* (MO) 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 MO 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 MO rule is close to $n(\pi^2/3-2)$ rotations (in a tree with n items). This holds independently of the access probabilities and the number of accesses to the tree.

Next we combine the MO rule with reference counters, one per item, that provide estimates of the reference probabilities. We define the rule MOUCS, and show, that for any δ and $\alpha > 0$, it achieves a cost that approaches the optimum up to a difference of δ with probability higher than $1-\alpha$, within a number of accesses that is proportional to $n/(\alpha\delta^2)$.

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

[*]Part of the work was performed in the Dept. of Computer Science, The University of Houston, Houston Tx 77204-3475, USA.

[‡]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

The *Binary Search Tree* (BST) is commonly used for storing lists of entries that satisfy a total order. The advantage of a tree is that it allows an efficient search in the list. 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 nearly optimal form.

We consider a fixed set of n records in random storage, $L = \{R_1, \ldots, R_n\}$. The record R_i is uniquely identified by the key K_i, for $1 \leq i \leq n$. 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 the a *reference probability vector* (RPV), $\boldsymbol{p} = (p_1, \ldots, p_n)$. Thus, R_i may be requested at any stage with the same 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 is called the *independent reference model* (IRM). Since the RPV and L do not change, 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. This equals its *depth* in the tree.

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 3 below). Different initial-insertion sequences usually result in different trees, with very large range of expected access costs.

The access probabilities listed in the RPV $\boldsymbol{p}$ 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 $\boldsymbol{p}$ unknown, we are reduced to looking at policies that use the accumulating reference history to adapt the tree structure, with the goal of rearranging 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.

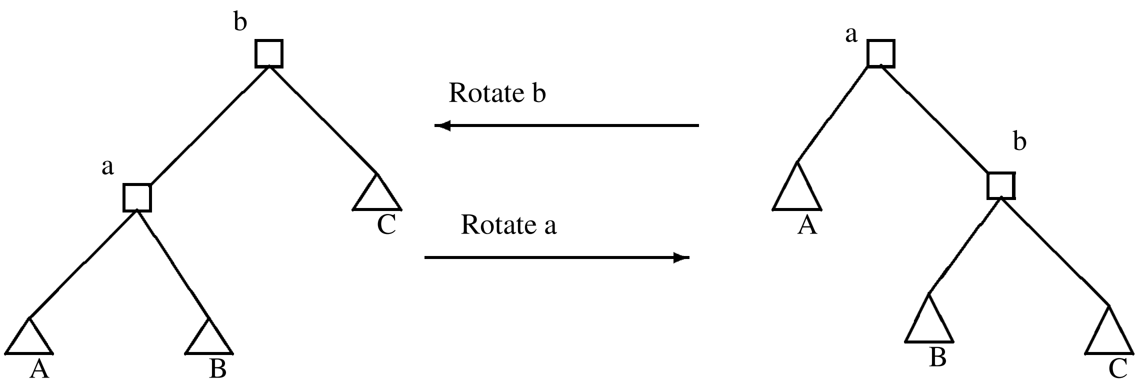

Figure 1: Single left-child and right-child rotations that reflect $(A) < K_a < (B) < K_b < (C)$

The two cost components of key comparisons and rotations are denoted by C and R respectively. A few performance measures which are of interest in this context are:

1. C_m – The access cost following the mth reference (and possible reorganization), $m \geq 0$.
2. C – The asymptotic access cost, especially its expected value, $E[C]$.
3. R – The work done in reorganizing the tree, such as the moments of the total number of rotations.

In addition to the limiting expectation of the random variable C, it is interesting to consider the rate at which C_m approaches this limit: since most 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.

The problem of reorganizing a BST to minimize the average access cost has been widely studied. Most of the work we have seen focus either on the asymptotic value of C, or an "amortized" analysis of the cost, which combines C and R, but is limited to considering a worst-case scenario [8, 19, 16]. Typically, such an approach only yields bounds. Such an analysis cannot assess the advantages of methods that rely on properties of the IRM (on the other hand, since it considers the worst case, the bounds it produces apply to more diverse scenarios, in particular – where the IRM assumption does not hold). The results in [4] refer to the case when the elements of the RPV are known only up to a permutation.

Some of the research focused on the situation where the RPV is *known*, with the goal of finding the optimal tree – or achieving a nearly optimal one, with a smaller computational effort. An early outline is given in [15]. A survey of more recent work on balanced BSTs appears in [18]. Recently, parallel algorithms have been considered for construction of optimal and nearly optimal BSTs ([2, 13]).

In this paper we devise and analyze policies which achieve a close approximation to the optimal BST, with lower organization costs than any of the previously studied heuristics.

Section 2 defines some additional notation, and presents the dynamic-programming algorithm that constructs the optimal tree for a known RPV. In Section 3 we discuss the *Move Once* (MO) rule, which achieves the same cost C as the *Move to the Root* (MTR) rule ([1]), 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 the MO or MTR. Our method, which we call *Move Once and Use Counter Scheme* (MOUCS) guarantees, that for any δ and $\alpha > 0$, the tree is dynamically reorganized until the average access cost is within δ 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 linear in n (the size of the tree) and $1/(\alpha\delta^2)$.

2 Preliminaries

Let $C(T_n)$ denote the average access cost to a BST T of n elements, with the access probabilities $p_1, \ldots, p_n$, then, with the root at level 0,

$$C(T_n) = 1 + \sum_{i=1}^{n} p_i \cdot (\text{level}\,(R_i)). \tag{1}$$

Under the IRM, for any set of keys with a given RPV, there exists an optimal *static* BST. We denote by $C(\text{OPT}|\boldsymbol{p})$ 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 [15]). Let $C(i,j)$ be the expected access cost of the optimal subtree that consists of records $R_{i+1}, R_{i+2}, \ldots, R_j$. Then $C(0,n)$ is $C(\mathrm{OPT}|\boldsymbol{p})$, as defined above. We also define $\pi_{i,j} = \sum_{k=i}^{j} p_k$. These costs satisfy the Bellman equations

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

When the access probabilities are *unknown*, a dynamic reorganization of the tree may be used to achieve an approximation of the optimal order. Some of the well known modification rules are studied in [1] and [4]. Various performance measures were considered for this model. With a given reorganization policy B and an unknown RPV $\boldsymbol{p}$, the following costs are used below:

1. The average access cost after the mth reference, $m \ge 0$, denoted by $C_m(\mathrm{B}|\boldsymbol{p})$,

   $$C_m(B|\boldsymbol{p}) = 1 + \sum_{i=1}^{n} p_i \cdot E[\,\mathrm{Level}\,(i)\,|\,B, \boldsymbol{p}], \tag{3}$$

   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 elements were inserted into the tree. As a rule we average over all possible insertion sequences, considering them equiprobable[1].

   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 approaches the optimal tree (or the optimal expected cost only) after a sufficiently long sequence of searches.

2. The expected access cost in the limiting state:

   $$C(B|\boldsymbol{p}) = \lim_{m\to\infty} C_m(B|\boldsymbol{p}).$$

3. The total expected number of rotations induced by an input string, characterized either by its size (e.g. m requests) or by the number of distinct records it references. In particular – by a sequence that contains each record at least once.

3 The Move Once (MO) Rule

3.1 The Average Cost of a Single Access

Allan and Munro [1] analyzed the Move To the Root (MTR) rule in detail, assuming the IRM. 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

[1]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.

the tree (unless it is there already). They showed an upper bound on the ratio $C(\text{MTR}|\boldsymbol{p})/C(\text{OPT}|\boldsymbol{p})$ for any distribution (their Theorem 3.3), and also estimated its rate of convergence (their Theorem 5.1).

The MTR rule is on the one hand more attractive than the MTF for a linear list, since the limiting value of its access cost can be shown to be closer to the optimal cost; on the other hand, the R component of its cost is even more pronounced than with a list, where any rearrangement uses the same time; here, moving a record to the root uses the same number of rotations as the number of steps to reach the record in the first place. Hence it makes sense to look for rules that use less expensive modifications. In [10] 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 finite time. 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 (*MO*).

Allan and Munro [1] consider a similar rule, calling it the First Request Rule, and show 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* BST *be referenced according to the* IRM *with the* RPV $\boldsymbol{p}$. *The rules* MTR *and* MO *have the same expected access costs for the mth request, for any* $m \geq 1$.

We use in the proof the following result.

Lemma 2: 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

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

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

We give a detailed proof 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 may seem surprising, since usually the rules construct for the *same* input string two entirely different trees. The important difference is that MO uses far fewer rotations than MTR, and moreover, the latter churns its tree indefinitely, whereas MO rests after a time which has a finite expectation. □

Hence we can use the results in [1] for the average cost under MTR to state the following theorem.

Theorem 3: ([1]): *The expected access cost to a* BST *reorganized with the* MO *policy, after m references, is given by*

$$C_m(\text{MO}|\boldsymbol{p}) = 1 + \sum_{1 \leq i < j \leq n} \left[\frac{2p_i p_j}{\pi_{i,j}} + (1-\pi_{i,j})^m \left(\frac{p_i + p_j}{j-i+1} - \frac{2p_i p_j}{\pi_{i,j}} \right) \right] \tag{4}$$

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. If it were created by the same applications which gave rise to the RPV we used above, then its initial form has the asymptotic distribution induced by the MO policy. This has then the limiting expected value

$$C(\text{MO}|\boldsymbol{p}) = 1 + \sum_{1\le i<j\le n} \frac{2p_ip_j}{\pi_{i,j}} \ , \tag{5}$$

and the MO policy calls for no more modifications.

3.2 The Expected Number of Rotations

The variable R_n, the total number of rotations the MO policy requires to organize a BST of size n (that is created by a random sequence) till all records have been accessed at least once, characterizes the cost of implementing this policy.

The size of such a sequence is known as the length of the Coupon Collector Search. This process is discussed in detail in [5]. Its length depends critically on the RPV; the expected value of the length is smallest when $\boldsymbol{p}$ is uniform, and equals then nH_n (H_n is the nth harmonic number, and approaches $\ln n$ asymptotically).

While the number of rotations for the first few references would be typically in $O(\log n)$, we should expect most subsequent references to require few rotations, if any.

The number of rotations, R_n for a tree of size n depends on two distributions. One generates the insertion sequence that creates the initial tree, and the second governs subsequent accesses – this is the above RPV. In fact, it would be more accurate to say it depends on the relation between the two distributions. This general statement has an exception: if the first distribution is uniform – every permutation of the records is equally likely to serve as the insertion sequence – then R_n does *not* depend on the access RPV. The reason is apparent from the equation we derive now. Consider the first reference. It addresses some record R_I, where I is the position of that record in the total order of the keys. Since the tree was created with the uniform distribution, then regardless of the values of I and of the access probability p_I, the depth of the record R_I is distributed as D_n (= the depth of a randomly selected node in a randomly constructed BST) – independently of I. Hence we may assume that the variable I, sampled according to the access RPV, is uniformly distributed on $[1,n]$. D_n is also the number of rotations that bring it to the root. Once this is done it will have two subtrees of sizes $I-1$ and $n-I$, and again, their structure is that of randomly created BSTs. The MO policy translates to independent reorganization of the subtrees; we find then

$$R_n = D_n + R_{I-1} + R_{n-I}. \tag{6}$$

The statistics of D_n are well known; it satisfies a recursion even simpler than (6): $D_n = 1 + D_{I-1} + D_{n-I}$. From this it is easy to obtain its *probability generating function* (PGF), the expected value and its variance, respectively:

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

We use $H_n^{(2)}$ to denote the nth second-order harmonic number, that converges promptly to $\pi^2/6$.

Let us return to rotations: Taking expected values (with respect to the entire access sequence used by the MO policy) of equation (6) we derive the first-order difference equation

$$r_{n+1} \equiv E[R_{n+1}] = d_{n+1} - \frac{n}{n+1} d_n + \frac{n+2}{n+1} r_n, \tag{7}$$

which has the immediate solution

$$r_n = 2n(H_n^{(2)} - 1) - 2H_n + 2H_n^{(2)}. \tag{8}$$

For not-too-small n, a good approximation of r_n is given by $r_n \approx n(\pi^2/3 - 2) - 2\log n + 2.3950$.
The total expected number of rotations *per record* in the tree is then less than 1.3.
The PGF of R_n does not appear to be easy to obtain. Even the variance $v_n \equiv V[R_n]$, which satisfies a relation very much like (7), does not seem to have a useful closed form representation. An asymptotic estimate is obtainable, though the explicit form is very complicated. It boils down to

$$v_n \approx 1.165381n - 4\log^2 n - 10.617725\log n + O(1). \tag{9}$$

The main information this result provides is that for large trees the distribution is very tightly centered at the mean value.

4 Reference Counters and Approximately Optimal Trees

The glaring 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 unless tree reorganization is "free," there appears to be no such simple rule for BSTs that results in an optimal structure.

4.1 The Counter Scheme and Dynamic Programming

In [11] the CS was shown to be optimal *for linear lists*, not only asymptotically, but also for every *finite* reference string. It is tempting to search such a rule for BSTs. This does not seem to exist. In particular, the so-called "monotonic BST", which like the CS keeps records with higher counters closer to the root, will usually fail to result in the optimal structure, for the same reason that using a *known* RPV to structure a BST monotonically fails to reproduce the tree which is computed by the Dynamic Programming (DP) 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 $n/\log n$, [17]).

But all is not lost. We can combine the MO and the CS with the DP algorithm as follows. The counters provide estimates for the RPV which could be used in DP to produce a tree which is the optimal one for the estimates (but would usually be in fact suboptimal). The computation takes a (constant) time in $\Theta(n^2)$; this is non-trivial for a large tree. Hence we would like to do it once, and to 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 the MO phase. The criterion must connect the total number of references (possibly with some information about the estimated RPV) and the nearness of the sub-optimal 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 MO and also compile reference counters, C_i for R_i, during a total of m_0 references. A suitable value for m_0 is shown below.
Phase B: use $\{C_i/m_0\}$ as estimates for the RPV; compute the "ostensibly optimal" BST using them as input for the DP algorithm, restructure the tree accordingly and stop reorganizing.

For the access cost during Phase A we have an explicit, if cumbersome result in equation (4).
Allen and Munro use equation (4) 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[2]. They also show, however, that this limiting value, $C(\text{MTR}|\boldsymbol{p})$, 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 vs. the length of the request sequence. We shall see that large trees need even less references in phase A than Allen and Munro suggest, and provide expected access cost which is close to the optimum.

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

Theorem 4: *For any unknown* RPV $\boldsymbol{p}$, $\delta > 0$ *and* $\alpha < 1$,

$$Prob\left(\left|\hat{C}(\text{OPT}|\boldsymbol{p}) - C(\text{OPT}|\boldsymbol{p})\right| > \delta\right) < \alpha \tag{10}$$

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

$$m_0 = \frac{5.235n}{\delta^2\alpha} \ , \tag{11}$$

Note: Below we compare this m_0 with m', the number of references found empirically to be needed to approach the optimal cost to the desired level.

We use in the proof the following lemmas, which relate weights (= access probabilities) of subtrees to their position in the optimal BST. We remind the reader that the level of a node was defined as its distance from the root.

Lemma 5: *Let T be an optimal* BST *as given in Figure 2 (possibly a subtree of the complete* BST*). Node B is at level 2, and P is the weight of T. Let p_B, p_t, p_r denote the weights of the nodes B, t, r respectively, P_t is the*

[2]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!

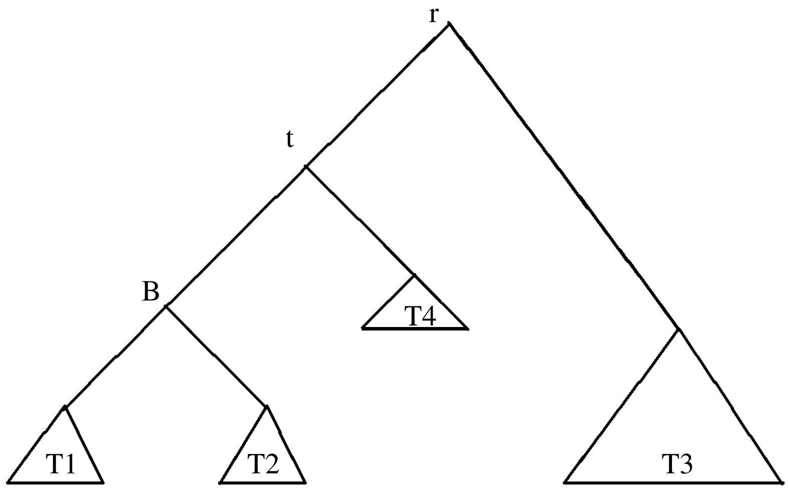

Figure 2: The optimal tree T

weight of the tree rooted at t, and P_B is the weight of the tree rooted at B, and for $1 \le i \le 4$, P_i are the weights of the subtrees T_i, then $\forall\, a \in [0,1]$

$$\textit{either } P_t \le aP \;\; \textit{or} \;\; P_B < (1-a)P. \tag{12}$$

We give the proof in the Appendix. Lemma 6 improves the bound on the maximal level of an item in an optimal BST as given in [9].

Lemma 6: *For any* RPV $p_1, \ldots, p_n$ *corresponding to the records* $R_1, \ldots, R_n$*, such that* $S = \sum_{i=1}^{n} p_i \le 1$*, if* L_i *is the level of* R_i *in an optimal* BST *(where the level of the root is* 0*), then*

$$p_i \le \varphi^{L_i - 1}. \tag{13}$$

where φ *is given by* $(\sqrt{5}-1)/2$*, the celebrated golden ratio.*

Proof: Using Lemma 5, the proof is similar to the proof of Lemma 2 in [17]. In addition we get an *a-fortiori* bound (read: possibly poor) by replacing the weight of a subtree by that of its root. ■

$\alpha \backslash n$	10	20	50	100	200
0.25	0.0141	0.0164	0.01768	0.01816	0.01766
0.1	0.00961	0.0108	0.00895	0.00843	0.00807
0.05	0.00715	0.0064	0.00538	0.00462	0.004405
0.01	0.00223	0.0017	0.00132	0.00109	0.000971

Table 1: 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'/m_0).

Proof of Theorem 4: Let $\hat{p}$ be the estimate for $\boldsymbol{p}$, obtained after a sequence of m references, and let $\hat{l}_i, l_i$ be the levels of R_i in the optimal trees for $\hat{p}, \boldsymbol{p}$ respectively. Using the optimality of $\{\hat{l}_i\}$ for $\{\hat{p}_i\}$,

$$\left|\hat{C}(\mathrm{OPT}|\boldsymbol{p}) - C(\mathrm{OPT}|\boldsymbol{p})\right| = \left|\sum_{i=1}^{n}\hat{l}_i p_i - \sum_{i=1}^{n} l_i p_i\right| \tag{14}$$

$$\leq \left|\sum_{i=1}^{n}\hat{l}_i p_i - \sum_{i=1}^{n}\hat{l}_i \hat{p}_i\right| + \left|\sum_{i=1}^{n} l_i \hat{p}_i - \sum_{i=1}^{n} l_i p_i\right| . \tag{15}$$

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

$$\mathrm{Prob}\left(\left|\sum_{i=1}^{n} l_i \hat{p}_i - \sum_{i=1}^{n} l_i p_i\right| > \frac{\delta}{2}\right) < \alpha, \tag{16}$$

and a similar relation with $\hat{l}_i$ replacing l_i (for which the *same* m would suffice, since the $\hat{l}_i$ are the optimal levels for the estimated probabilities $\hat{p}_i$). Now, since $C_i^{(m)}$ has the marginal distribution $\mathrm{Bin}(m, p_i)$, we can compute the moments of the estimates $\hat{p}$. Using the Chebyshev inequality in relation (16) we solve for m and have

$$m \leq 4\frac{\sum_{i=1}^{n} l_i^2 p_i(1-p_i) - \sum_{i=1}^{n}\sum_{j\neq i} p_i p_j l_i l_j}{\delta^2\alpha} \leq \frac{4}{\delta^2\alpha}\sum_{i=1}^{n} l_i^2 p_i(1-p_i), \tag{17}$$

where the last inequality amounts to neglecting the (negative) covariances between the counters. From Lemma 6 we have $l_i \leq 1 + \log p_i / \log\varphi$, hence

$$m \leq \frac{4}{\delta^2\alpha}\sum_{i=1}^{n}(1 + \log p_i/\log\varphi)^2 p_i(1-p_i). \tag{18}$$

Each of the terms in the sum is at most $\max_{p_i\in(0,1)}(1 + \log p_i/\log\varphi)^2 p_i(1-p_i) = 1.33371\ldots$ (at $p_i \approx 0.071$), yielding the bound in (11). ■

$\delta \setminus n$	50
0.9	0.032794
0.7	0.025901
0.4	0.016375
0.2	0.010796
0.1	0.006905
0.06	0.004883
0.01	0.001184
0.001	0.000072

Table 2: The ratio m'/m_0 for $n = 50$ and $\alpha = 0.15$.

Note: The procedure we used to derive the bound in (11) suggests that for most distributions, the stopping point for the execution of MOUCS is significantly lower. Table 1 verifies this for a large set of randomly generated RPVs: We computed the access probabilities from a vector $(x_1,\ldots,x_n)$, such that $x_i \sim U(0,1)$, and $p_i = x_i/\sum_j x_j,\ 1 \le i \le n$.

We estimated the stopping point m' for a set of 500 RPVs, for each pair (n,α), with $\delta = 0.25$. Table 1 presents the ratio m'/m_0 (using the average of the 500 values). The results are consistent with the linear dependence of the bound on n. However, for most of the RPVs we tried, the constant was evidently much smaller. In addition, Table 1 suggests that the stopping point depends on $1/\alpha^y$ for some $0 < y \le 0.5$, whereas we could only proved a bound using $y = 1$. Table 2 shows that for fixed values of n and α, the ratio m_0/m' is a decreasing function of δ. 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 empirical results on the stopping point for MOUCS in the following

Conjecture 7 : *For any unknown* RPV $\boldsymbol{p}$*, the expected access cost to a* BST *rearranged by the* MOUCS *approaches the optimal average cost within a difference of* δ*, with probability higher than* $1-\alpha$*, following* $c \cdot n/\delta^x\alpha^y$ *references, for some small constant* $c < 1$*,* 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, we are interested 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 ([9]): *Choose the root so as to equalize the weight of the left and right subtree as much as possible, then proceed similarly on the subtrees.*

It is shown in [7], that a weight balanced tree is constructible with time and space complexity in $\Theta(n)$.

Bayer shows in [3], that for a given RPV $\boldsymbol{p}$, the average access cost to the weight balanced tree, denoted by $C(\text{WB}|\bar{p})$ satisfies

$$C(\text{WB}|\boldsymbol{p}) - C(\text{OPT}|\boldsymbol{p}) \le \lg H + \lg e + 1 \approx \lg H + 2.4427... < 1.45 \ln\ln n + 2.45\,, \tag{19}$$

where $H = \sum p_i \lg p_i^{-1}$ is the entropy of the RPV $\boldsymbol{p}$. Since $H \in [0, \lg n]$, this bound looks acceptable. We describe in the next section the usage of a scheme which keeps the tree weight-balanced, in terms of the counters of the keys. Its performance is given by the following Theorem. Denote by $\hat{C}(\text{WB}|\bar{p})$ the cost of the balanced tree constructed by the estimate $\hat{p}$ for the rpv $\boldsymbol{p}$.

Theorem 8: *For any* $\delta > 0$ *and* $0 < \alpha < 1$*, and for any unknown* RPV $\boldsymbol{p}$*,*

$$Prob\,(\left|\hat{C}(\text{WB}|\boldsymbol{p}) - C(\text{WB}|\boldsymbol{p})\right| > \delta) < \alpha, \tag{20}$$

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

$$m_0 = \frac{5.235n}{\delta^2\alpha}. \tag{21}$$

We use in the proof a bound on the structure of these trees that appears identical with the one derived in Lemma 6 for the optimal BST; it was first shown by Mehlhorn in [17]:

Lemma 9: ([17]) *For any* RPV $p_1, \ldots, p_n$ *corresponding to the records* $R_1, \ldots, R_n$*, such that* $S = \sum_{i=1}^{n} p_i \leq 1$*, if* L_i *is the level of* R_i *in the weight balanced tree (where the level of the root is* 0*), then*

$$p_i \leq \varphi^{L_i - 1}\,, \tag{22}$$

with φ *as defined in Lemma 6.*

Proof of Theorem 8: Using the proof of Theorem 4, with $\hat{C}(\text{OPT}|\boldsymbol{p})$ and $C(\text{OPT}|\boldsymbol{p})$ replaced by $\hat{C}(\text{WB}|\boldsymbol{p})$ and $C(\text{WB}|\boldsymbol{p})$ respectively, and $\hat{l}_i, l_i$ denoting the levels of R_i in the weight balanced trees for $\hat{p}$,$\boldsymbol{p}$ respectively, m_0 satisfies equation (17). The bound is obtained by using Lemma 9. ■

The identical bounds on the structure of those 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

We have studied reorganization rules for a BST, where accesses to the tree are generated independently by a fixed unknown distribution. We showed that when the distribution is static for sufficiently long durations, 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 with length which is linear in the number of elements (for relatively large n the length of this sequence is comparable with, or smaller than the expected number of references till all records are touched once).

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 6. It is our belief that the discrepancy does not represent a possible worst case, but rather our failure to bound the sums that appear in equation (14) more tightly.

A different, interesting rule, which also 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. 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 $\boldsymbol{p}$ were known.

In fact, we have shown that for any distribution, the cost of the estimated weight balanced tree approaches—as close as we wish—the cost of the "true" weight-balanced tree (based on the *unknown* RPV $\boldsymbol{p}$) within a number of accesses that is linear in n.

The relative efficiency of the MO 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 MO 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. 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. During each phase, guaranteed to be at least K references long, and average M, the RPV is fixed. Consider the following scheme:

1. Keep reference counters and store their values every K accesses. The tree is reorganized so as to keep it weight-balanced by the counters. Call a sequence of K requests a *segment*.

2. After every segment, test for the hypothesis that the counters accumulated in the last two segments where generated by identical distributions. If the hypothesis is rejected, reset the counters.

This opens the door to a large number of statistical inference problems, that we expect to address in a forthcoming paper.

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 [14]. We showed that MO produces a tree with a "nearly optimal" cost – though its shape could differ radically from that of the optimal one. The question, whether a more efficient algorithm is available which uses the structure of the MO 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.

References

[1] B. Allen, I. Munro, "Self-Organizing Search Trees", *JACM* **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." Purdue University, Department of Computer Science, CSD-TR-952, February 1990.

[6] W. Feller, *An Introduction to Probability Theory and its Applications* John Willey, New York, 1968.

[7] 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.

[8] I. Galperin, R. Rivest, "Scapegoat Trees", In *Proc. of the 4th ACM-SIAM Symposium on Discrete Algorithms*, Austin, TX, January 25-27, 1993.

[9] R. Güttler, K. Mehlhorn, W. Schneider, "Binary Search Trees: Average and Worst Case Behavior", *Jour. of Information Processing and Cybernetics*, **16**, 41–61, (1980).

[10] M. Hofri, H. Shachnai, "Self-Organizing Lists and Independent References – a Statistical Synergy", *Jour. of Alg.*, **12**, 533-555, (1991).

[11] M. Hofri, H. Shachnai, "On the Optimality of Counter Scheme for Dynamic Linear Lists", *Inf. Process. Lett.*, **37**, 175–179, (1991).

[12] T.C. Hu, K. C. Tan, "Least Upper Bound on the Cost of Optimum Binary Search Trees", *Acta Informatica*, **1**, 307-310, (1972).

[13] 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).

[14] D.E. Knuth, "Optimum Binary Search Trees", *Acta Informatica* **1**, 14–25,1971.

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

[16] 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.

[17] K. Mehlhorn, "Nearly Optimal Binary Search Trees", *Acta Informatica* **5** 287–295, (1975).

[18] K. Mehlhorn, A. Tsakalidis, "Data Structures". In J. van Leeuwen, editor, *Algorithms and Complexity*, Vol A, chapter 6, pp. 301-341, Elsevier, 1990.

[19] D.D. Sleator, R.E. Tarjan, "Self-Adjusting Binary search Trees", *JACM* **32**, #3, 652–686 (1985).

[20] 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 2: Consider an arbitrary pair of records, R_i and R_j. We look at the sufficient and necessary conditions for R_i to be an ancestor of R_j in $T_{MTR}(I)$ and $T_{MO}(I)$ and call them $C_{MTR}(i,j)$ and $C_{MO}(i,j)$ respectively. The structure of the BST is determined (uniquely) once we specify the ancestor-offspring relation for all pairs of nodes in the tree. Therefore our claim will be established if we show that a string I satisfies $C_{MO}(i,j)$ for all i and j iff $C_{MTR}(i,j)$ holds in I^R for all pairs. The concept of *interval set* is useful for this discussion. Such a set comprises two records and all other records with keys that lie between them. Let $K_i < K_j$, then the corresponding set is denoted by $IS(i,j)$. Throughout the discussion below we assume w.n.l.g. that $K_i < K_j$ so that the first one of the pair of sets $IS(i,j)$ and $IS(j,i)$ is non-empty. 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 shall 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". The salient properties are:

(a) When a node is rotated it continues to be an ancestor to all of its previous offspring, and *becomes* so 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 subtree when the rotation is to the right (or the right one, when the rotation is to the left).

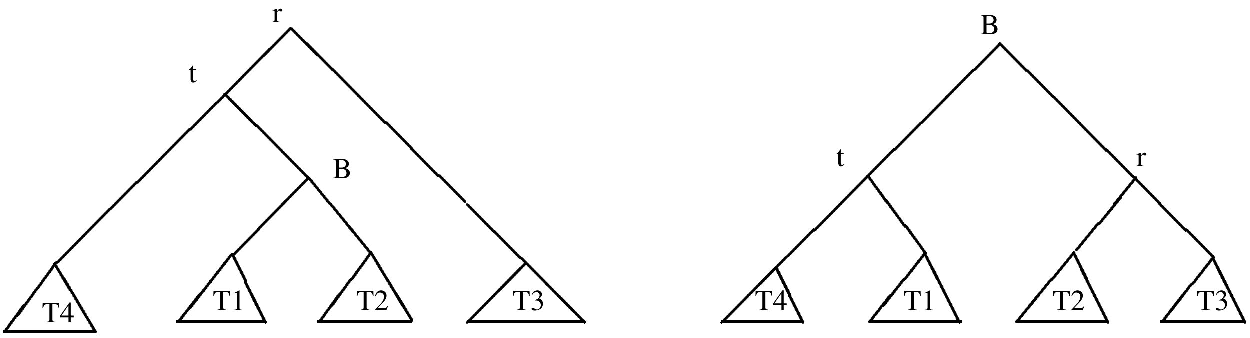

Figure 3: (a) T (b) T'

Claim 10: *(i) $C_{MO}(i,j)$ is: R_i is the first node to be referenced in $IS(i,j)$.*
(ii) $C_{MTR}(i,j)$ is: R_i is the last node to be referenced in $IS(i,j)$.

Proof: The proof of (i) 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.

Part (ii) is 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 5: If $P_t \le aP$ we are done. Otherwise we consider the case where

$$P_t > aP\ . \tag{23}$$

Without loss of generality, we assume that $P = 1$. 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, i.e.

   $$p_t + P_B \le p_r + P_3\ , \tag{24}$$

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

   $$a < p_t + P_B + P_4 \le p_r + P_3 + P_4\ , \tag{25}$$

   Hence

   $$P_B = 1 - (p_r + P_3 + P_4 + p_t) \le 1 - (p_r + P_3 + P_4) < 1 - a\ . \tag{26}$$

2. 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 average access cost to T' (as given in Figure 3(b), and obtained by rotating B twice), thus

   $$p_r + P_3 \ge 2p_B + P_1 + P_2 > P_B \tag{27}$$

   and since $p_r + P_3 = 1 - P_t$, using (23) we find

   $$P_B \le 1 - P_t < 1 - a. \tag{28}$$

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