Subtree Weight Ratios for Optimal Binary Search Trees

D.S. Hirschberg
L.L. Larmore
M. Molodowitch

University of California, Irvine

Technical Report No. 86-02

January 29, 1986

Subtree Weight Ratios for Optimal Binary Search Trees

D.S. Hirschberg
L.L. Larmore
M. Molodowitch

Abstract

For an optimal binary search tree T with a subtree S(d) at a distance d from the root of T, we study the ratio of the weight of S(d) to the weight of T. The maximum possible value, which we call $\rho(d)$, of the ratio of weights, is found to have an upper bound of $2/F_{d+3}$ where F_i is the ith Fibonacci number. For d = 1, 2, 3, and 4, the bound is shown to be tight. For larger d, the Fibonacci bound gives $\rho(d) = O(\phi^d)$ where $\phi \approx .61803$ is the golden ratio. By giving a particular set of optimal trees, we prove $\rho(d) = \Omega((.58578...)^d)$, and believe a similar proof follows for $\rho(d) = \Omega((.60179...)^d)$. If we include frequencies for unsuccessful searches in the optimal binary search trees, the Fibonacci bound is found to be tight.

1. Introduction

An optimal binary search tree minimizes the expected search time when we are given a fixed set of keys with frequencies $\beta_1, \beta_2, ...\beta_n$ for their occurrence [1]. We refer to the β's as weights of the nodes of the search tree. Melhorn has shown that a tree, that is constructed by equalizing as much as possible the weights of the left and right subtrees, is very near optimal [2]. We consider a related problem: how skewed can an optimal search tree be.

Let T be an optimal binary search tree and S be a subtree with its root at a distance d from the root of T. The weight of T is $W(\mathrm{T}) = \Sigma_{1 \le i \le n} \beta_i$, the weight of S is $W(\mathrm{S}) = \Sigma_{i \in \mathrm{S}} \beta_i$, and the ratio of interest is $\rho(d)$, the maximum possible value for $W(\mathrm{S})/W(\mathrm{T})$.

In Section 2 we give upper bounds for $\rho(d)$, first for the case $d = 1$, where S is the left or right subtree of T, and then for the case of general d. In Section 3, we describe a set of optimal search trees which have, for the cases $d = 1, 2, 3,$ and 4, subtrees that give ratios arbitrarily close to the upper bounds for the ρ's, showing that the bounds are tight. In Section 4, we describe a conjecture as to

---

Authors' address: Department of Information and Computer Science. University of California, Irvine. Irvine, CA 92717.

the asymptotic bounds on $\rho(d)$ and exhibit sets of optimal trees that give ratios close to the conjectured bounds. In Section 5, we examine more general optimal binary search trees, which include frequencies for unsuccessful searches, and find the Fibonacci bounds are tight for these types of trees. In Section 6, we summarize the paper and conclude with some open questions.

2. Upper bound for $\rho(d)$

First we examine the case where $d = 1$. S is then the left or right subtree of T. We use the following conventions for describing components of T:

root(U)	– the root of any tree U
W(U)	– the weight of any tree U
T_L, T_R	– the left and right subtrees of T
T_{LL}, T_{LR}	– the left and right subtrees of *root*(T_L)
T_{RL}, T_{RR}	– the left and right subtrees of *root*(T_R)
β_0	– the weight of *root*(T)
β_L, β_R	– the weights of *root*(T_L), *root*(T_R)

Theorem 1.

If T is an optimal binary search tree, the weight of the left or right subtree must be at most 2/3 the weight of the entire tree.

Proof.

Suppose that $W(T_R) > 2/3\ W(T)$. Root(T_R) has two subtrees, T_{RL} and T_{RR}. There are two possible cases.

Case 1. The weight of T_{RL} is greater than $1/3\ W(T)$. Then make *root*(T_{RL}) the new root of T, using a double left rotation.

Case 2. $\beta_R + W(T_{RR}) > 1/3\ W(T)$. Then make *root*($T_R$) the new root of T, using a single left rotation.

In either case, the new tree has lower expected search time than T, a contradiction to the optimality of T. By symmetry, the same argument holds for the left subtree of T.

For the general case, we need the following lemma.

Lemma 1.

In an optimal binary search tree,

a) $\beta_0 + W(T_R) \geq max\{W(T_{LL}), W(T_{LR})\}$

b) $\beta_0 + W(T_L) \geq max\{W(T_{RR}), W(T_{RL})\}$

Proof.

Parts (a) and (b) are equivalent by symmetry, so we prove part (a). There are two possible cases to consider.

Case 1. Suppose $\beta_0 + W(T_R) < W(T_{LR})$. Perform a double right rotation putting $root(T_{LR})$ as the new root of T, producing a tree with lower expected search time.

Case 2. Suppose $\beta_0 + W(T_R) < W(T_{LL})$. Perform a single right rotation putting $root(T_L)$ as the new root, again producing a tree with lower expected search time.

For both cases we get a contradiction to the optimality of T, so the lemma is proved.

Theorem 2.

For any subtree S *with its root at a distance d from the root of an optimal binary search tree* T, $W(S)/W(T) \leq 2/F_{d+3}$ *where* F_n *is the n-th Fibonacci number* ($F_1 = 1$, $F_2 = 1$, $F_3 = 2$). *Hence* $\rho(d)$ *has an upper bound of* $2/F_{d+3}$.

Proof.

Assume $W(S) = 2W_0$ for some value W_0. Start at $root(S)$ and go up the path to the root of T one level at a time. At each step i, we are at the root of a bigger subtree. Call this subtree T_i, and let β_i be the weight of $root(T_i)$. $root(T_i)$ has another subtree V_i which was not on the path followed. Since every subtree of an optimal tree is also optimal, we can use Lemma 1:

$$\beta_i + W(V_i) \geq W(T_{i-2})$$

But $W(T_i) = W(T_{i-1}) + \beta_i + W(V_i)$, so we obtain the recursive relation

$$W(T_i) \geq W(T_{i-1}) + W(T_{i-2})$$

$W(T_0) = W(S) = 2W_0$ and, by Theorem 1, $\beta_1 + W(V_1) \geq W_0$, so $W(T_1) \geq 3W_0$. We can solve for $W(T_d) = W(T)$,

$$W(\mathrm{T}) \geq W_0 \cdot \mathrm{F}_{d+3}$$

and $W(\mathrm{S})/W(\mathrm{T}) \leq 2/\mathrm{F}_{d+3}$.

3. The upper bound is tight for depths 1, 2, 3, and 4

We first describe a set of trees, in which $W(\mathrm{S})/W(\mathrm{T})$ comes arbitrarily close to the upper bounds for $d = 1$ and 2. For higher d, the set provides a lower bound for $\rho(d)$ of $(2-\epsilon)/(2^d+1)$ for any small ϵ.

Let $\mathrm{T}(d)$ consist of a complete binary tree of height d in which all the leaves except one are replaced by complete binary trees of height h and the one leaf is replaced by a complete binary tree of height $h+1$ (Figure 1). Let all nodes have unit weight. $\mathrm{T}(d)$ is clearly optimal and, choosing S to be the subtree of height $h+1$ at distance d from the root, we see that

$$W(\mathrm{S})/W(\mathrm{T}(d)) = (2 \cdot 2^{h+1} - 1) \; / \; (\; (2^d+1)2^{h+1} - 1 \;)$$

As we let h grow, the ratio comes arbitrarily close to 2/3 for $d = 1$ and to 2/5 for $d = 2$. For higher d's, the ratio approaches $2/(2^d+1)$.

Using $\mathrm{T}(1)$ and $\mathrm{T}(2)$, we can construct a set of trees T_i recursively, in which $W(\mathrm{S})/W(\mathrm{T}_i)$ comes arbitrarily close to the upper bound for $d = i = 3$ and $d = i = 4$. Let $\mathrm{T}_1 = \mathrm{T}(1)$ and $\mathrm{T}_2 = \mathrm{T}(2)$. For odd (even) $i \geq 3$, the right (left) subtree of $root(\mathrm{T}_i)$ is T_{i-1}. The other subtree of $root(\mathrm{T}_i)$ is a single node with the same weight β_i as $root(\mathrm{T}_i)$, where $\beta_i = .5(\mathrm{W}(\mathrm{T}_{i-2}) + \beta_{i-2})$. Figures 2 and 3 show T_3 and T_4 respectively, and Figure 4 gives the general T_i.

If we let $W_0 = 2^{h+1}$, T_3 has a root with weight $1.5\,W_0$ and a single node with the same weight as the root as the left subtree. The right subtree of T_3 is $\mathrm{T}(2)$ as defined above. Choosing for S the same subtree of height $h+1$ as before, but now at a distance 3 from the root,

$$W(\mathrm{S})/W(\mathrm{T}) = (2\,W_0 - 1)/(8\,W_0 - 1)$$

and, letting h grow, the ratio approaches 2/8. Similarly for T_4, the ratio is

$W(\mathrm{S})/W(\mathrm{T}) = (2\,W_0 - 1)/(13\,W_0 - 1)$ which comes arbitrarily close to 2/13.

We now show that T_3 is optimal. In any rearrangement of the nodes into a search tree minimizing the expected search time, which we will call the cost, the two heavy nodes must stay in the same relative positions on the left side of the tree. Since all other nodes have equal weight, they must be arranged as evenly as possible, approximating a complete tree except on the left side where the two heavy nodes are. Suppose the two heavy nodes are pushed down one level and a node of weight 1 is at the root (Figure 5). The change in cost from that of T_3 is given by

$$\Delta\text{Cost} = +3\,W_0 - (2\,W_0 - 1) - (W_0 - 1) - 2 = 0$$

so the cost is the same as T_3. Pushing the heavy nodes down further results in greater cost, so T_3 is an optimal tree as is the tree in Figure 5.

We show that T_4 is optimal in Section 4.

4. Lower bounds for $\rho(d)$

From Theorem 2 we see that, if $W(T_i)$ obeys the recursive relation

$$W(T_i) = W(T_{i-1}) + W(T_{i-2})$$

then, solving the characteristic equation, gives

$$\lim_{d\to\infty} W(T_d) = O(\ ((1+\sqrt{5})/2)^d\ )$$

so $\rho(d) = O(\ (2/(1+\sqrt{5}))^d\ ) = O(\phi^d)$, where $\phi \approx .61803$ is the golden ratio. We conjecture that $\rho(d) = \Theta(K^d)$ where $K \approx .60179$.

The following stronger version of Lemma 1 suggests why K is probably less than ϕ. The notation conventions are the same as in Section 2.

Lemma 1'

In an optimal binary tree,

a) $\beta_0 + W(T_R) \geq max\{\beta_L + W(T_{LL}), \beta_{LR} + W(T_{LR})\}$

b) $\beta_0 + W(T_L) \geq max\{\beta_R + W(T_{RR}), \beta_{RL} + W(T_{RL})\}$

Proof.

The proof is the same as in Lemma 1.

We can use Lemma 1′ and the same construction as in Theorem 2 of going up the path from *root*(S) to *root*(T) one level at a time. Using the same notation as in Theorem 2 and applying Lemma 1′ at step i,

$$\beta_i + W(V_i) \geq W(T_{i-2}) + \beta_{i-1}$$

if the path from *root*(T_{i-2}) to *root*(T_i) was straight and

$$\beta_i + W(V_i) \geq W(T_{i-2}) + \beta_{i-2}$$

if the path from *root*(T_{i-2}) to *root*(T_i) was bent.

Since $W(T_i) = W(T_{i-1}) + \beta_i + W(V_i)$, we get the following:

$$W(T_i) \geq W(T_{i-1}) + W(T_{i-2}) + \beta_{i-1}$$
$$W(T_i) \geq W(T_{i-1}) + W(T_{i-2}) + \beta_{i-2}$$

the choice depending on whether the path went straight or zigzagged. From this, we see that to get $\rho(d)$, we need to choose the minimum possible weights β_i's for the nodes directly on the path from *root*(S) to *root*(T). In the examples given in Section 3, the relevant β_i's had unit weight, and since $\beta_i / W_0 = 1/2^{h+1} \rightarrow 0$ as $h \rightarrow \infty$, $\rho(d)$ approached the upper bound of $2/F_{d+3}$.

For $d > 4$, we conjecture that the β_i's on the path are no longer negligible, and hence $\rho(d) < 2/F_{d+3}$ for $d > 4$.

In Section 3 we first described a set of trees which gave a lower bound of $(2-\epsilon)/(2^d + 1)$ for $\rho(d)$ so that $\rho(d) = \Omega(2^{-d})$. The set T_i that was described next gives a recursion relation for β_i:

$$\beta_i = \beta_{i-1} + 1.5\beta_{i-2} - .5\beta_{i-3}$$

and we have $\rho(d) = \Omega((.58578...)^d)$.

We now describe a third set of trees which is very similar to the second (Figure 6). In going from T'_{i-1} to T'_i, the change in the weight is the same as before, $W(T'_i) - W(T'_{i-1}) = W(T'_{i-2}) + \beta'_{i-2}$, but now it is equally split between four nodes instead of two. One of the four new nodes is the root of T'_i, the other three nodes form a complete binary tree which is a subtree of *root*(T'_i), and T'_{i-1} is the other subtree.. The weight of a new node is thus $.25(W(T'_{i-2}) + \beta'_{i-2})$ and $\rho(d) = \Omega((.60179...)^d)$. The first two trees, T'_1 and T'_2, are the same for this set as for the other sets.

In the remainder of this section, we prove the optimality of the trees in the second set. For the trees in the third set, we have constructed a proof of optimality using similar arguments. We use extensively a theorem by Knuth [1]: If the inorder of tree T is nodes $A_1,...,A_n$, and A_i is the root of an optimal binary search tree for nodes $A_1,...,A_k$, and A_j is the root for an optimal binary search tree for $A_{k+1},...,A_n$, then the root A_l for $A_1,...,A_n$ satisfies the condition that $i \leq l \leq j$.

Theorem 3.

The binary search trees T_i in the second set, described in Section 3, are optimal.

Proof.

The proof is by strong induction. T_3 was shown to be optimal in Section 3. We also need to show for the basis that T_4 is optimal (Figure 3).

By Knuth's theorem, since T_3 is optimal, there are three possibilities for the root after adding the two heavy nodes C and D.

a) The root is C. This is what we want.

b) The root is B. Then the weight of the right subtree is $10W_0 - 1$, which contradicts Theorem 1.

c) The root is one of the nodes of unit weight. A tree can then be characterized by the levels of nodes B and C.

Case 1) Both B and C are at level 1 (Figure 7). There is no net change in cost from the tree with the root at C.

Case 2) B is at level 2 and C is at level 1 (Figure 8). The change in cost from Case (1) is W_0, which is strictly positive, so that this is not optimal. It is clear that moving B or C down further only increases the cost.

Hence the tree with the root at C (Figure 3) is optimal, as is the tree in Case (1), so that T_4 is optimal.

We now assume that all T_j, for $1 \leq j < i$ (where $i > 4$), are optimal and show that T_i is optimal. In the following, we label the nodes and subtrees of T_i as in Figure 4.

Using Knuth's theorem, there are five possibilities for the root when nodes E and F are added to T_{i-1}.

1) The root is E. This is what we wish to show.

2) The root is B. The subtree containing {nodes C, D, E, F, and the tree T_{i-3}} contradicts Theorem 1.

3) The root is D. The right subtree contains nodes E and F. For the left subtree, we must find the optimal tree for A, B, C, and the nodes in T_{i-3}. If D is added, we have the optimal tree T_{i-1} so that, using Knuth's theorem, the root for an optimal tree which excludes D must be at A or B. However, having the root at A contradicts Theorem 1, so that the root of the left subtree must be B.

   The tree with the root at D is shown in Figure 9. Note that the cost for such a tree must be greater than the cost for the tree T′ shown in Figure 10, where both C and subtree T_{i-3} are at level 2. There is no change in cost from the tree with root E (Figure 4) to T′. Thus the tree with the root at D must cost more than the tree with the root at E.

4) The root is C. The right subtree contains D, E, and F. The left subtree must be optimal and contain A, B, and the nodes in T_{i-3}. Again using Knuth's theorem and the assumption that B is the root of an optimal tree T_{i-1}, the root for the left subtree must be A or B. It cannot be A because of Theorem 1, so the root of the subtree must be B.

   The tree must be the one shown in Figure 11 and there is no difference in cost between it and the tree with the root at E.

5) The root is a node K in T_{i-3}. Let H_0 be the optimal tree formed by the nodes in T_{i-3} before K and H_1 be the optimal tree formed by the nodes in T_{i-3} after K.

   The left subtree consists of A, B, and the nodes in H_0. Again using Knuth's theorem, the root of the left subtree must be A or B, and clearly the tree with B as root has lower cost.

The right subtree consists of C, D, E, F, and the nodes in H_1. There are three possibilities for the root because of Knuth's theorem, shown in Figures 12, 13, and 15. Instead of finding the optimal tree containing C and the nodes of H_1, see that the cost of the tree in Figure 14 is less than that in Figure 13. Comparing costs, we see that both Fig. 14 and Fig. 12 cost more than Fig. 15. Hence the right subtree has E as the root.

The whole tree with the root at K is given in Figure 16. We now rearrange the nodes of T_{i-3} in Figure 4 to make K the root as in Figure 17. Then

Cost(Fig. 16) − Cost(Fig. 4)

$$
\begin{aligned}
&= \text{Cost(Fig. 16)} - \text{Cost(Fig. 17)} + \text{Cost(Fig. 17)} - \text{Cost(Fig. 4)} \\
&= 2\beta_i - 3\beta_K - 2\,W(H_0) - W(H_1) + \Delta_K \\
&= 2\beta_i - 2\beta_K - W(H_0) - W(T_{i-3}) + \Delta_K \\
&= 3\beta_{i-2} - 2\beta_K - W(H_0) + \Delta_K
\end{aligned}
$$

where Δ_K gives the difference in cost in rearranging T_{i-3} to make K the root.

There are three possibilities for T_{i-3}: it can be T_2, T_3, or some T_k where $k > 3$. When T_{i-3} is T_2, in the above equation for the difference in costs, $\beta_{i-2} = 1.5\,W_0$, $\beta_K = 1$, and we can explicitly find $-W(H_0) + \Delta_K$ for different K's. For all K's, the difference in costs is positive. Similarly for the case of T_3, we can explicitly show that the difference is positive for all K's. For the last case, we look at the structure of T_{i-3} (Figure 18). K can be any of the four nodes shown or be a node in T_{i-5}. However, we get the minimum in the difference in costs when $2\beta_K + W(H_0)$ is a maximum, so we need only to look at the cases where K is either node M or node N. For node N, the rearranging cost Δ_K is too high. For node M, even assuming $\Delta_K = 0$,

$$
\begin{aligned}
3\beta_{i-2} - 2\beta_K - W(H_0) &= 3\beta_{i-2} - 2\beta_{i-4} - 2\beta_{i-3} - W(T_{i-5}) \\
&= 3\beta_{i-2} - W(T_{i-3})
\end{aligned}
$$

which can be shown to be always positive for $i > 6$. Therefore the cost of trees with the root at a node from T_{i-3} is greater than the cost of the tree with the root at E.

We have now covered all possibilities and shown that the tree with the root at E (Figure 4) is optimal, which proves the theorem.

5. More general optimal binary search trees

In the more general case of an optimal binary search tree, we are also given α_0, α_1,...α_n, where α_i is the frequency of unsuccessful searches for a key between K_i and K_{i+1}. Lemmas 1 and 1′ and Theorems 1 and 2 still hold with the weights of trees now including the weights of the leaves α's, but now we can show that the Fibonacci bounds are tight for all depths.

Let T''_i be the set of binary search trees defined recursively as follows (Figure 19). T''_1 and T''_2 are T(1) and T(2) as defined in Section 3, where the leaves are now the α_i's. T''_i has a root of unit weight, its left subtree is T''_{i-1}, and its right subtree is a leaf of weight $\alpha_n = W(T''_{i-2}) - 1$. Since all the β_i's have unit weight and $\beta_i / W_0 \rightarrow 0$, $\rho(d) \rightarrow 2/F_{d+3}$ as discussed in Section 4.

Theorem 4.

The binary search trees T''_i are optimal.

Proof.

The proof is by induction on i. T''_1 and T''_2 are obviously optimal. We first show T''_3 is optimal (Figure 20). We can characterize any rearrangement of the tree by the level of the rightmost leaf Z with weight $3W_0 - 1$. T''_3 as shown has the leaf at level 1. The lowest cost of a tree with Z at level 2 is the same as that for T''_3, and the cost rises as Z goes down further. Hence T''_3 is optimal.

Now assume T''_{i-1} is optimal. When we add one internal node of unit weight and one leaf of weight $W(T''_{i-2}) - 1$ to the right of T''_{i-1}, by Knuth's theorem there are only two possible choices for an optimal tree: T''_i and Figure 21. We can see that their costs are equal, so that T''_i is optimal, as is Figure 21.

6. Summary and open questions

We studied the behavior of $\rho(d)$, defined as the maximum possible value

for the ratio of the weight of a subtree of an optimal binary search tree to the weight of the entire tree, where the subtree is at a distance d from the root. $\rho(d)$ was shown to have an upper bound of $2/F_{d+3}$, so that $\rho(d) = O((.61803...)^d)$. We described sets of trees giving lower bounds for $\rho(d)$ of $\Omega((.5)^d)$, $\Omega((.58578...)^d)$ and $\Omega((.60179...)^d)$. For $d = 1$, 2, 3, and 4, the upper bound was found to be tight, but for higher d's the question of closing the gap between the two bounds still remains. For optimal binary search trees which include frequencies for unsuccessful searches, the Fibonacci bound was shown to be tight.

References

[1] Knuth, D.E.: Optimum binary search trees. Acta Informatica **1**, 14-25 (1971)

[2] Mehlhorn, K.: Nearly optimal binary search trees. Acta Informatica **5**, 287-295 (1975)

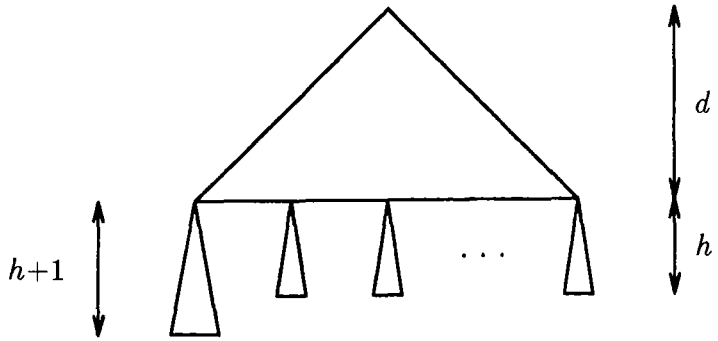

Figure 1. $\mathrm{T}(d)$

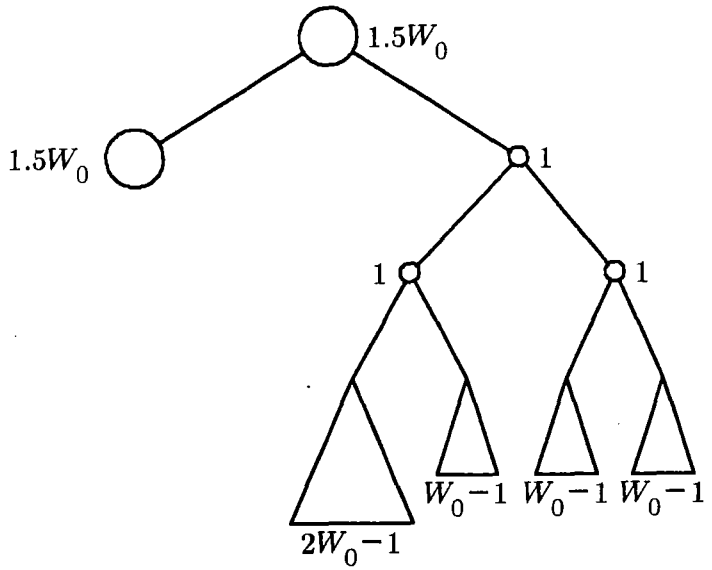

Figure 2. T_3 for $d = 3$

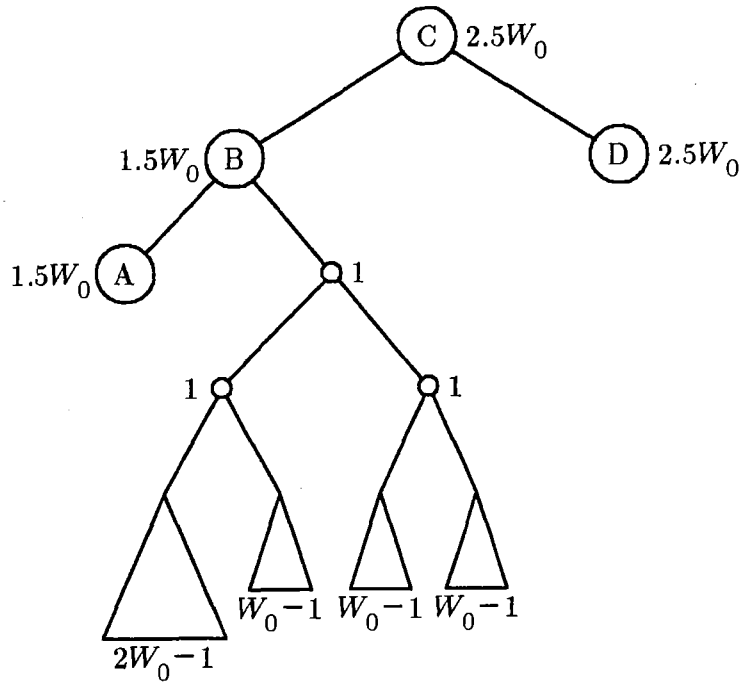

Figure 3. T_4 for $d = 4$

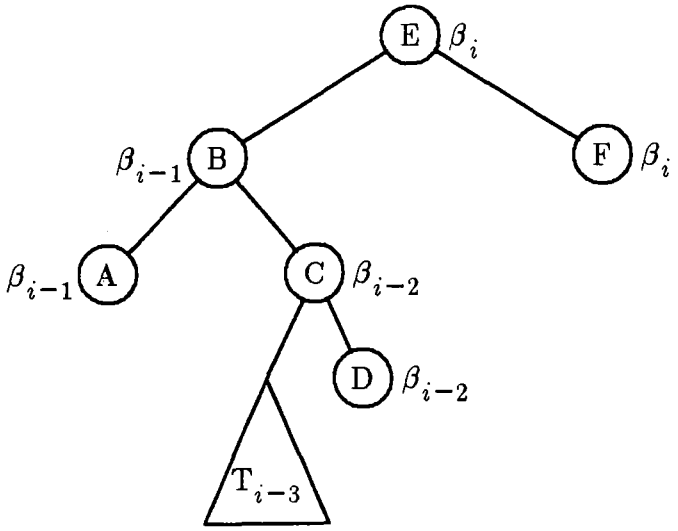

Figure 4. T_i for second set

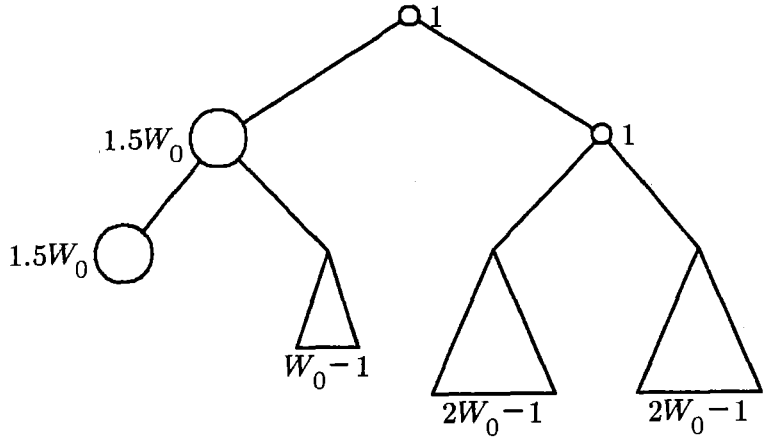

Figure 5. Alternate T for $d = 3$

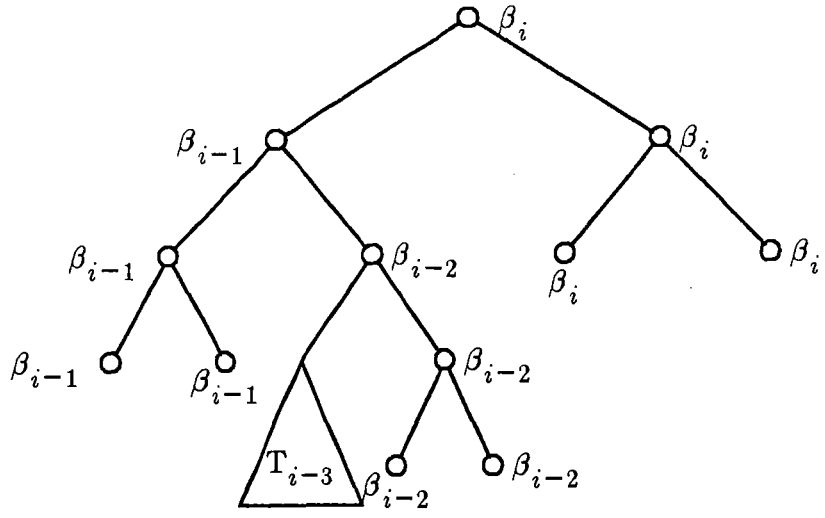

Figure 6. T'_i for third set

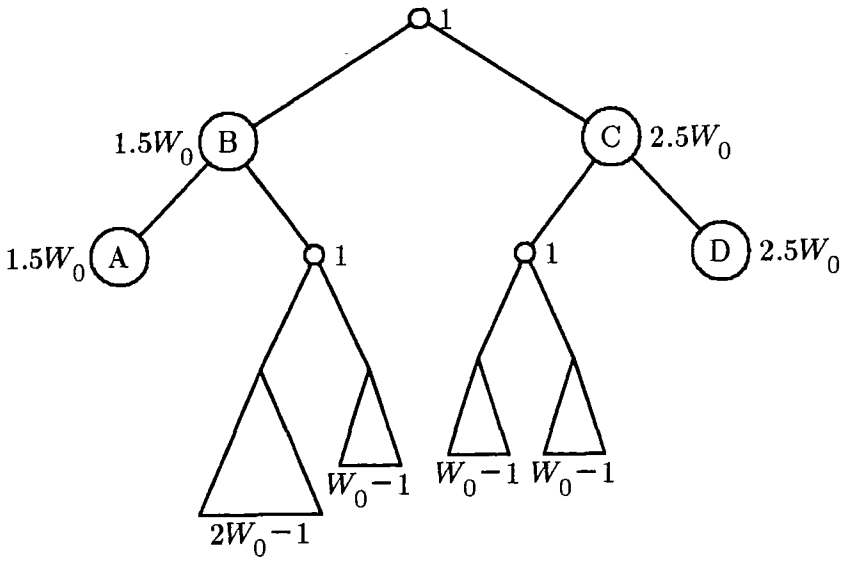

Figure 7. Case (1)

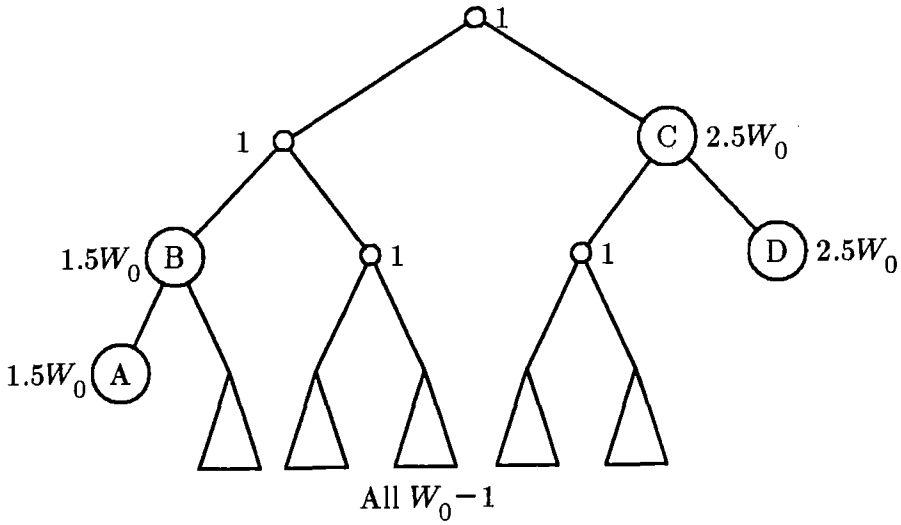

Figure 8. Case (2)

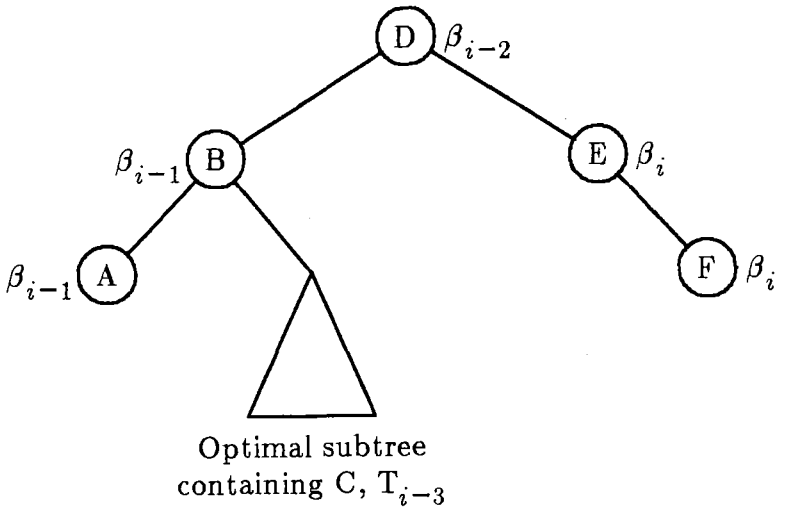

Figure 9. Root at D

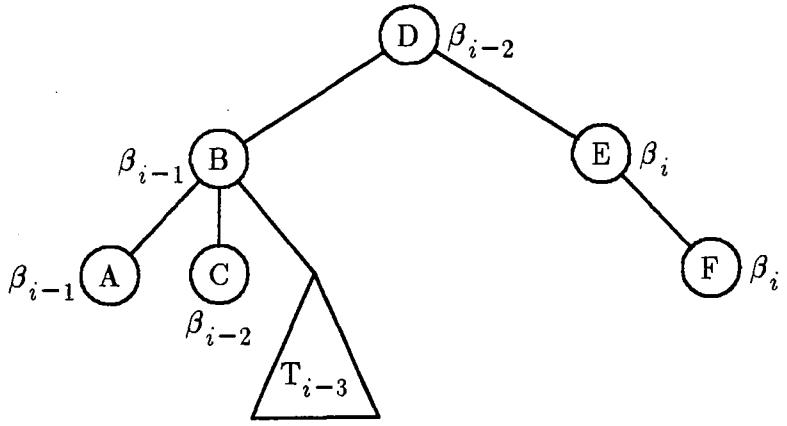

Figure 10. Lower cost tree with root at D
(Not a binary search tree)

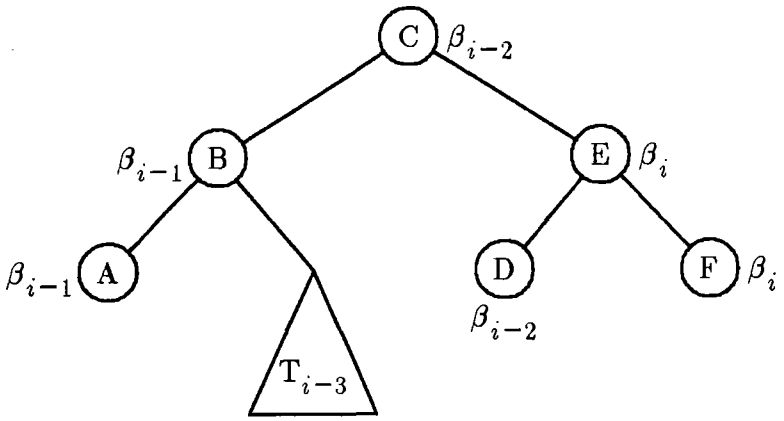

Figure 11. Root at C

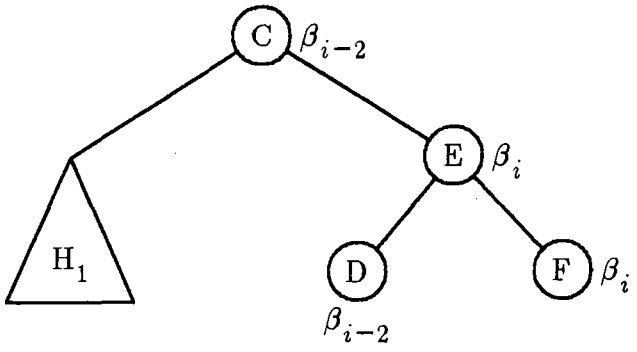

Figure 12. Possible right subtree with root at C

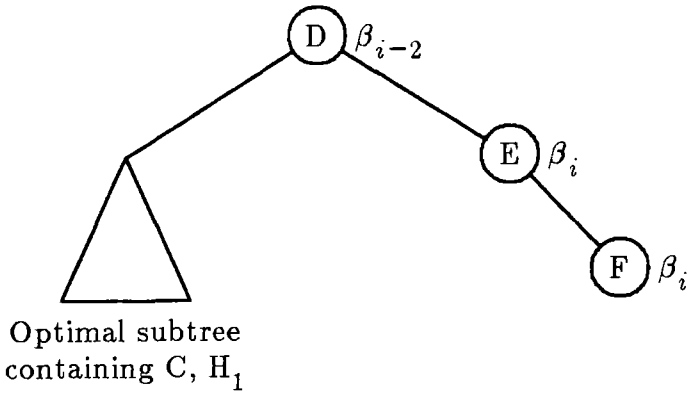

Figure 13. Possible right subtree with root at D

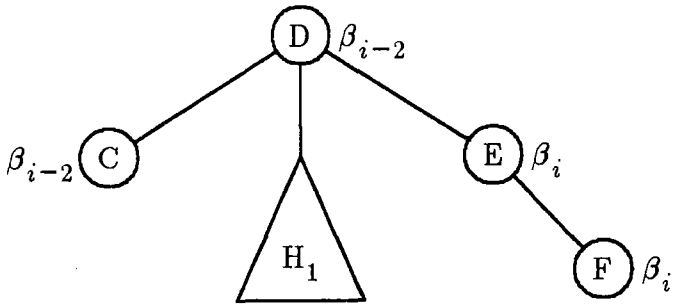

Figure 14. Lower cost subtree with root at D
(Not a binary search tree)

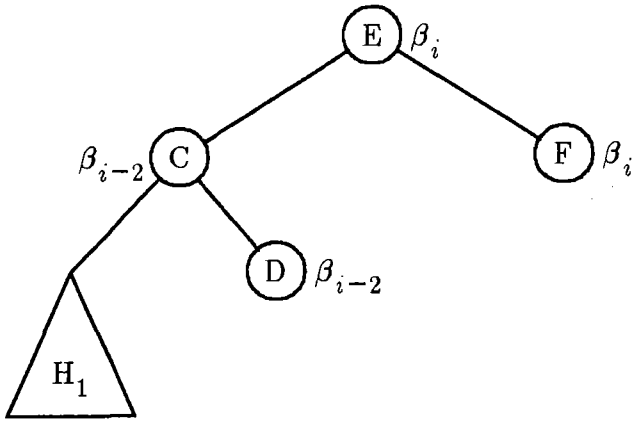

Figure 15. Optimal subtree with root at E

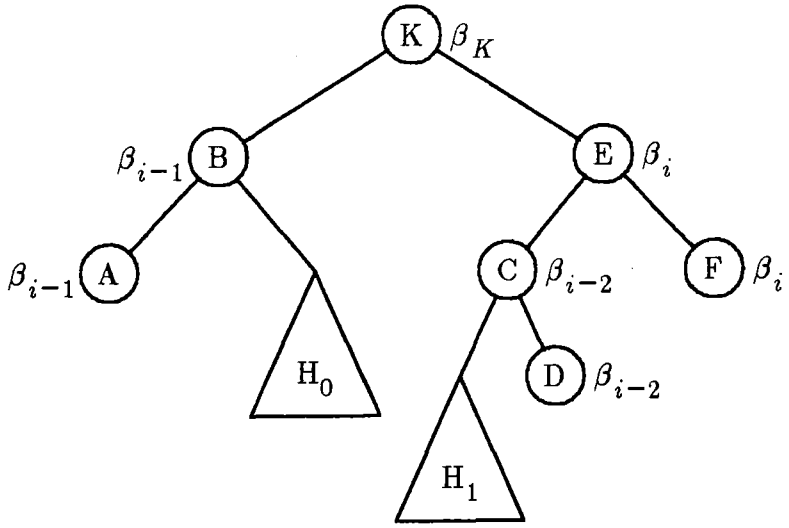

Figure 16. Root at K, a node in T_{i-3}

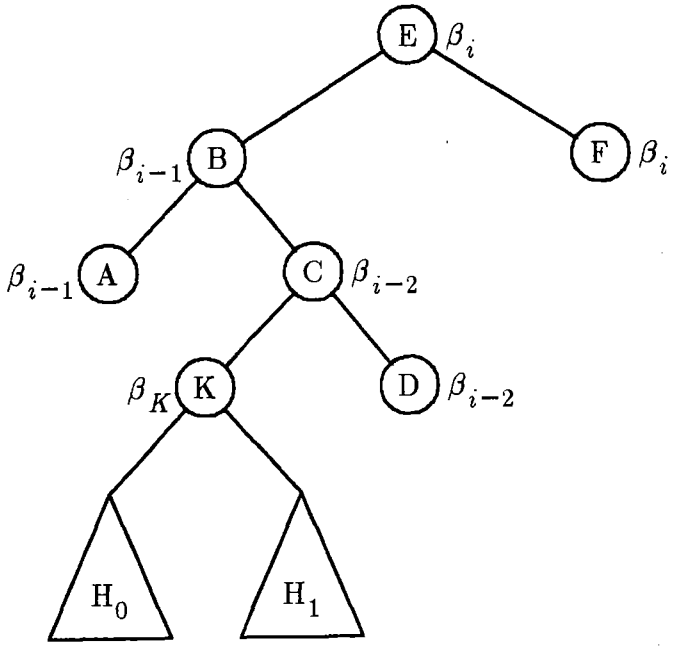

Figure 17. Subtree T_{i-3} replaced by a subtree with K as root

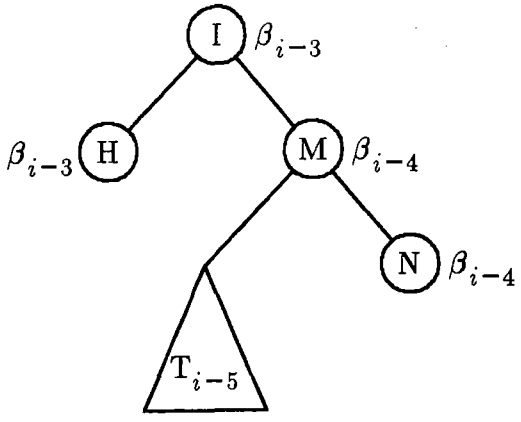

Figure 18. Expansion of T_{i-3}

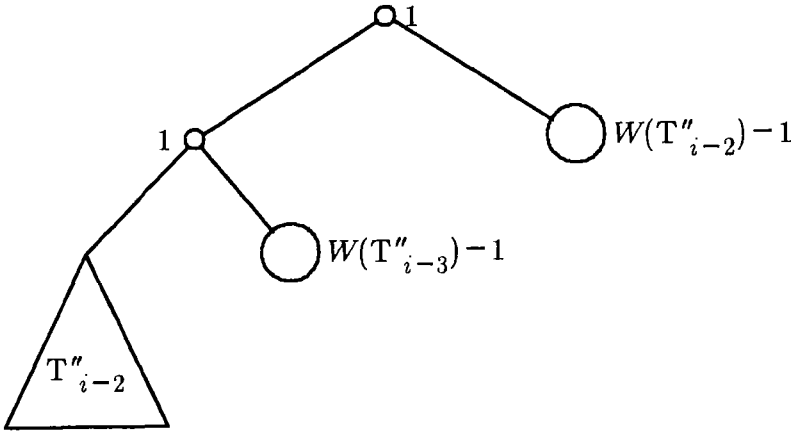

Figure 19. T''_i

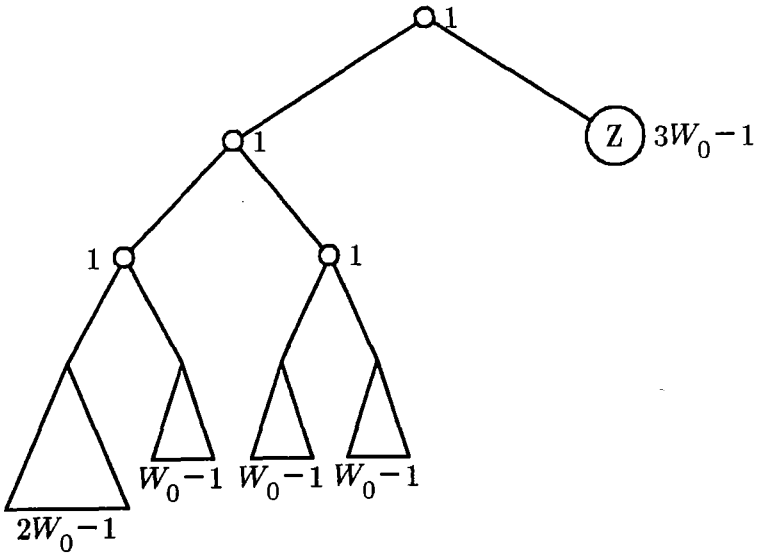

Figure 20. T''_3

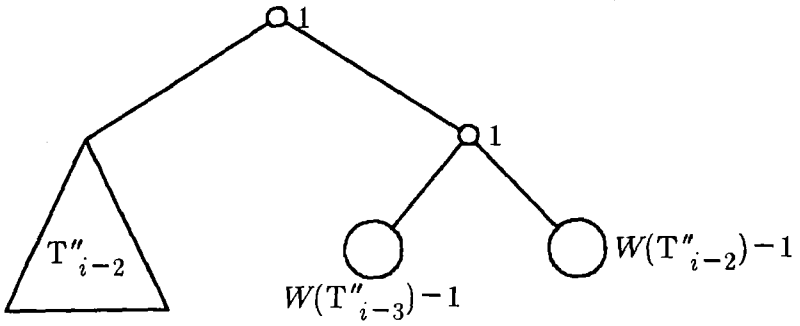

Figure 21. Another optimal rearrangement of T''_i