A BEST POSSIBLE BOUND FOR THE
WEIGHTED PATH LENGTH OF BINARY SEARCH TREES*

KURT MEHLHORN†

Abstract. The weighted path length of optimum binary search trees is bounded above by
\[ \Sigma \beta_i + 2 \Sigma \alpha_j + H \] where \( H \) is the entropy of the frequency distribution, \( \Sigma \beta_i \) is the total weight of the
internal nodes, and \( \Sigma \alpha_j \) is the total weight of the leaves. This bound is best possible. A linear time
algorithm for constructing nearly optimal trees is described.

Key words. binary search tree, complexity, average search time, entropy

One of the popular methods for retrieving information by its “name” is to
store the names in a binary tree. We are given \( n \) names \( B_1, B_2, \ldots, B_n \) and \( 2n + 1 \)
frequencies \( \beta_1, \ldots, \beta_n, \alpha_0, \ldots, \alpha_n \) with \( \Sigma \beta_i + \Sigma \alpha_j = 1 \). Here \( \beta_i \) is the frequency
of encountering name \( B_i \), and \( \alpha_j \) is the frequency of encountering a name which
lies between \( B_j \) and \( B_{j+1} \). \( \alpha_0 \) and \( \alpha_n \) have obvious interpretations [4].

A binary search tree \( T \) for the names \( B_1, B_2, \ldots, B_n \) is a tree with \( n \) interior
nodes (nodes having two sons), which we denote by circles, and \( n + 1 \) leaves, which
we denote by squares. The interior nodes are labeled with the \( B_i \) in increasing
order from left to right and the leaves are labeled with the intervals \( (B_j, B_{j+1}) \) in
increasing order from left to right. Let \( b_i \) be the distance of interior node \( B_i \) from
the root and let \( a_j \) be the distance of leaf \( (B_j, B_{j+1}) \) from the root. To retrieve a
name \( X \), \( b_i + 1 \) comparisons are needed if \( X = B_i \) and \( a_j \) comparisons are required
if \( B_j < X < B_{j+1} \). Therefore we define the weighted path length of tree \( T \) as:
\[ P = \Sigma_{i=1}^{n} \beta_i(b_i + 1) + \Sigma_{j=0}^{n} \alpha_j a_j, \]
It is equal to the expected number of comparisons needed to retrieve a name.

search tree, i.e., a tree with minimal weighted path length. His algorithm operates
in \( O(n^2) \) units of time and \( O(n^2) \) units of space. In [6] we discuss the following
“rule of thumb” for constructing nearly optimal binary search trees: choose the
root so as to equalize the total weight of the left and right subtree as much as
possible, then proceed recursively. The weighted path length of a tree constructed
according to this rule is bounded above by \( 2 + 1.44 \cdot H \), where \( H = \Sigma \beta_i \log (1/\beta_i) + \Sigma \alpha_j \log (1/\alpha_j) \) is the entropy of the frequency distribution. This
bound was recently improved by P. J. Bayer [1] to \( 2 + H \). Here we discuss a
different rule of thumb suggested by [3] and prove the upper bound \( 1 + \Sigma \alpha_j + H \)
for the weighted path length. This bound is best possible.

The rule presented here as well as the rules described in [6] can be
implemented to work in linear time and space ([2]).

* Received by the editors September 24, 1975, and in revised form June 1, 1976.
† Universität des Saarlandes, 66 Saarbrücken, West Germany

235
We describe and analyze an approximation algorithm. The algorithm constructs binary search trees in a top-down fashion. It uses bisection on the set
\[
\left\{ s_i; s_i = \sum_{p=0}^{i-1} (\alpha_p + \beta_p) + \beta_i + \frac{\alpha_i}{2} \quad \text{and} \quad 0 \leq i \leq n \right\},
\]
i.e., the root is determined such that \( s_{k-1} \leq \frac{i}{2} \) and \( s_k \geq \frac{i}{2} \). It then proceeds recursively on the subsets \( \{ s_i; i \leq k - 1 \} \) and \( \{ s_i; i \geq k \} \). In the definition of the \( s_i \)'s we assumed \( \beta_0 = 0 \) for ease of writing. The main program

\begin{verbatim}
begin
    let \( s_i \leftarrow \sum_{p=0}^{i-1} (\alpha_p + \beta_p) + \beta_i + \alpha_i/2 \) for \( 0 \leq i \leq n \);
    construct-tree (0, n, 0, 1)
end
\end{verbatim}

uses the recursive procedure construct-tree:

\begin{verbatim}
procedure construct-tree (i, j, cut, l);
    comment we assume that the actual parameters of any call of construct-tree satisfy the following conditions.
    (1) \( i \) and \( j \) are integers with \( 0 \leq i < j \leq n \),
    (2) \( l \) is an integer with \( l \geq 1 \),
    (3) \( cut = \sum_{p=1}^{l-1} x_p 2^{-p} \) with \( x_p \in \{0, 1\} \) for all \( p \),
    (4) \( cut \leq s_i \leq s_j \leq cut + 2^{-l+1} \).
    A call construct-tree (\( i, j, --, --, \)) will construct a binary search tree for the nodes \( \lceil i \rceil, \ldots, \lceil j \rceil \) and the leaves \( \lfloor i \rfloor, \ldots, \lfloor j \rfloor \);
    begin
        if \( i + 1 = j \) (Case A)
            then return the tree shown in Fig. 1.
        else comment we determine the root so as to bisect the interval \( (cut, cut + 2^{-l+1}) \)
            begin
                determine \( k \) such that
                (5) \( i < k \leq j \)
                (6) \( k = i + 1 \) or \( s_{k-1} \leq cut + 2^{-l} \)
                (7) \( k = j \) or \( s_k \geq cut + 2^{-l} \)
                comment \( k \) exists because the actual parameters are supposed to satisfy condition (4);
                if \( k = i + 1 \) (Case B)
                    then return the tree shown in Fig. 2;
                if \( k = j \) (Case C)
                    then return the tree shown in Fig. 3;
                if \( i + 1 < k < j \) (Case D)
                    then return the tree shown in Fig. 4;
            end
        end
    end procedure construct-tree;
\end{verbatim}
Lemma. The approximation algorithm constructs a binary search tree whose weighted path length $P_{\text{approx}}$ is bounded above by $1 + \sum \alpha_i + H$.

Proof. We state several simple facts.

Fact 1. If the actual parameters of a call construct-tree $(i, j, \text{cut}, l)$ satisfy conditions (1) to (4) and $i + 1 \neq j$, then a $k$ satisfying conditions (5) to (7) exists and the actual parameters of the recursive calls of construct-tree initiated by this call again satisfy conditions (1) to (4).

Proof. Assume that the parameters satisfy conditions (1) to (4) and that $i + 1 \neq j$. In particular, $\text{cut} \leq s_j \leq \text{cut} + 2^{-i+1}$. Suppose, that there is no $k$, $i < k \leq j$, with $s_{k-1} \leq \text{cut} + 2^{-l}$ and $s_k \geq \text{cut} + 2^{-l}$. Then either for all $k$, $i < k \leq j$, $s_k < \text{cut} + 2^{-l}$ or for all $k$, $i < k \leq j$, $s_k > \text{cut} + 2^{-l}$. In the first case $k = j$ satisfies (6) and (7), in the
second case $k = i + 1$ satisfies (6) and (7). This shows that $k$ always exists. It
remains to show that the parameters of the recursive calls satisfy again (1) and (4).
This follows immediately from the fact that $k$ satisfies (5) to (7) and that $i + 1 \neq j$
and hence $s_k \geq \text{cut} + 2^{-l}$ in Case B and $s_{k-1} \leq \text{cut} + 2^{-l}$ in Case C. Q.E.D.

**FACT 2.** The actual parameters of every call of construct-tree satisfy conditions (1) to (4) (if the arguments of the top-level call do).

**Proof.** The proof is by induction, Fact 1 and the observation that the actual parameters of the top-level call construct-tree $(0, n, 0, 1)$ satisfy (1) to (4). Q.E.D.

We say that node $i$ (leaf $k$ resp.) is constructed by the call construct-tree $(i, j, \text{cut}, l)$ if $h = j$ ($h = i$ or $h = j$) and Case A is taken or if $h = i + 1$ ($h = i$) and Case B is taken or if $h = j$ ($h = j$) and Case C is taken or if $h = k$ and Case D is taken. Let $b_i$ be the depth of node $i$ and let $a_j$ be the depth of leaf $j$ in the tree returned by the call construct-tree $(0, n, 0, 1)$.

**FACT 3.** If node $i$ (leaf $j$) is constructed by the call construct-tree $(i, j, \text{cut}, l)$, then $b_i + 1 = l$ ($a_j = l$).

**Proof.** The proof is by induction on $l$.

**FACT 4.** If node $i$ (leaf $j$) is constructed by the call construct-tree $(i, j, \text{cut}, l)$, then $b_i \geq 2^{l+1}$ ($a_j \geq 2^{l+1}$).

**Proof.** The actual parameters of the call satisfy condition (4) by Fact 2. Thus

$$2^{l+1} \geq s_i - s_j = (\alpha_i + \alpha_j)/2 + \beta_{i+1} + \alpha_{i+1} + \cdots + \beta_j$$

$$\geq b_i \text{ (resp. } a_j/s_j\text{).}$$ Q.E.D.

**FACT 5.** The weighted path length $P_{\text{approx}}$ of the tree constructed by the approximation algorithm is bounded above by $\sum b_i + \sum \alpha_j + H$.

**Proof.**

$$P_{\text{approx}} = \sum b_i (b_i + 1) + \sum \alpha_j a_j$$

$$\leq \sum b_i (\log (1/\beta_i) + 1) + \sum \alpha_j (\log (1/\alpha_j) + 2)$$

$$\leq \sum b_i + 2 \cdot \sum \alpha_j + H.$$ Q.E.D.

**THEOREM.** Let $\alpha_0, \beta_1, \alpha_1, \cdots, \beta_n, \alpha_n$ be any frequency distribution, let $P_{\text{opt}}$ be the weighted path length of the optimum binary search tree for this distribution, let $P_{\text{approx}}$ be the weighted path length of the tree constructed by the approximation algorithm, and let $H = -\sum \beta_i \log \beta_i - \sum \alpha_j \log \alpha_j$ be the entropy of the frequency distribution. Then

$$P_{\text{opt}} \leq P_{\text{approx}} \leq \sum b_i + 2 \cdot \sum \alpha_j + H.$$ 

Furthermore, this upper bound is the best possible in the following sense: if

$$c_1 \sum \beta_i + c_2 \sum \alpha_i + c_3 \cdot H$$

is an upper bound for $P_{\text{opt}}$, then $c_1 \geq 1$, $c_2 \geq 2$, and $c_3 \geq 1$.

**Proof.** The first part of the theorem follows from the preceding lemma. The second part is proven by exhibiting suitable frequency distributions:

- $c_1 \geq 1$: Take $n = 1$, $\alpha_0 = \alpha_1 = 0$ and $\beta_1 = 1$.
- $c_2 \geq 2$: Take $n = 2$, $\alpha_0 = \alpha_2 = \beta_1 = \beta_2 = 0$, $\alpha_1 = 1$.
- $c_3 \geq 1$: Take $n = 2^k - 1$, $\beta_i = 0$ for all $i$ and $\alpha_j = 2^{-k}$ for all $j$.
It is easy to see that the complete binary tree is the optimal binary search tree for this distribution. Thus

\[
H = \log (n + 1) = k = \sum_{\text{leaves}} \left(\frac{1}{2^k}\right) \cdot k = P_{\text{opt}}. 
\]

Q.E.D.

REFERENCES


