ELSEVIER

Theoretical Computer Science 287 (2002) 571–584

Theoretical Computer Science

www.elsevier.com/locate/tcs

A strategy for searching with different access costs

Eduardo Sany Laber *, Ruy Luiz Milidiú, Artur Alves Pessoa

Departamento de Informática, PUC-Rio, Brazil

Received March 2000; accepted May 2001

Abstract

Let us consider an ordered set of keys $A = \{a_1 < \cdots < a_n\}$, where the probability of searching a_i is $1/n$, for $i = 1, \ldots, n$. If the cost of testing each key is similar, then the standard binary search is the strategy with minimum expected access cost. However, if the cost of testing a_i is c_i, for $i = 1, \ldots, n$, then the standard binary search is not necessarily the best strategy.

In this paper, we prove that the expected access cost of an optimal search strategy is bounded above by $4C \ln(n+1)/n$, where $C = \sum_{i=1}^{n} c_i$. Furthermore, we show that this upper bound is asymptotically tight up to constant factors. The proof of this upper bound is constructive and generates a $4\ln(n+1)$-approximated algorithm for constructing near-optimal search strategies. This algorithm runs in $O(n^2)$ time and requires $O(n)$ space, which can be useful for practical cases, since the best known exact algorithm for this problem runs in $O(n^3)$ time and requires $O(n^2)$ space.

1. Introduction

Let us consider an ordered set of keys $A = \{a_1 < \cdots < a_n\}$, a list of strictly positive costs $c_1, \ldots, c_n$ and a list of probabilities $p_1, p_2, \ldots, p_{2n+1}$, where c_i and p_{2i} are, respectively, the access cost of the key a_i and the probability of retrieving a_i. For $i = 1, \ldots, n-1$, p_{2i+1} is the probability of searching an element greater than a_i and smaller than a_{i+1}. The probabilities p_1 and p_{2n+1} are, respectively, the probability of searching an element smaller than a_1 and an element greater than a_n.

Any search strategy for the set A can be represented by a binary search tree (BST) with n nodes and $n+1$ leaves, where each internal node corresponds to a key of A and the ith leaf from left to right corresponds to an unsuccessful search to an element

* Corresponding author. Rua Marques de São Vicente 225, FPLP 11° andar., 2DC, 22453-900, Rio de Janeiro, RJ Brazil.

E-mail addresses: laber@inf.puc-rio.br (E.S. Laber), milidiu@inf.puc-rio.br (R.L. Milidiú), artur@inf.puc.rio.br (A.A Pessoa).

0304-3975/02/$ - see front matter
PII: S0304-3975(01)00262-6

larger than a_{i-1} and smaller than a_i. Let T be a BST and let P_i be the sum of the probabilities of the nodes in the subtree of T rooted by a_i. The expected access cost of the search strategy represented by T is given by

$$EC = \sum_{i=1}^{n} c_i P_i. \tag{1}$$

An optimal BST, is a BST that minimizes EC among all BSTs with n internal nodes and $n+1$ leaves.

The best known algorithm to find an optimal BST is based on dynamic programming and runs in $O(n^3)$, with $O(n^2)$ space requirement [9]. This algorithm is an extension of the algorithm previously proposed by Knight to address the particular case, where only the costs are non-uniform [3]. The case with uniform costs and different access probabilities has been extensively studied [4, 7, 5, 2]. For this case, the best known algorithm to find an optimal BST is due to Knuth [4]. It runs in $O(n^2)$ time, with $O(n^2)$ space requirement.

In this paper, we focus on the case where the costs are non-uniform, all keys (internal nodes) have the same probability q and all unsuccessful searches (leaves) have the same probability p. In this case, the expected access cost of a BST can be rewritten as

$$EC = p\sum_{i=1}^{n} c_i + (p+q)\sum_{i=1}^{n} c_i n(a_i), \tag{2}$$

where $n(a_i)$ denotes the number of internal nodes in the subtree rooted by the node a_i. We can omit the constants p, $\sum_{i=1}^{n} c_i$ and q, since EC is minimized if and only if $\sum_{i=1}^{n} c_i n(a_i)$ is minimized. Hence, all the results in this paper concern to the minimization of the function

$$\sum_{i=1}^{n} c_i n(a_i), \tag{3}$$

which we call the **cost of a BST**. In [3], Knight proposed a simple dynamic programming algorithm to build a BST with minimum cost. His algorithm runs in $O(n^3)$ time, requiring $O(n^2)$ space. Knight has also shown upper and lower bounds on the cost of an optimal BST for the cost structure $c_i = i^t$, where t is a fixed positive constant. This cost structure arises in a filter design problem [10].

The time and space requirements of the algorithm proposed by Knight makes it prohibitive for large values of n. Motivated by this fact, we consider alternatives for constructing near-optimal BSTs. Here, we present the Ratio algorithm. Ratio is a $4\ln(n+1)$-approximated algorithm that runs in $O(nH_R)$ time, with $O(n)$ space requirement, where H_R is the height of the tree T_R obtained at the end of the algorithm. Since $H_R < n$, then Ratio runs in $O(n^2)$ time. Let $C = \sum_{i=1}^{n} c_i$. The analysis of Ratio shows that $4C\ln(1+n)$ is an upper bound on the cost of T_R, and as a consequence, an upper bound on the cost of an optimal BST. Moreover, this upper bound is asymptotically tight up to constant factors in the sense that there are some cost structures such that the cost of an optimal BST is $\Omega(C\log n)$ [3].

We also study some combinatorial properties of optimal BSTs. We present a necessary condition for a BST to be an optimal search tree. Based on this condition, we give a non-trivial upper bound on the height of an optimal BST. We argue that the height of an optimal BST is $O(\sqrt{n})$ for practical cases. Finally, we report some experimental results comparing the performance of Ratio to other algorithms proposed in the literature. These experiments give some evidence that Ratio has a much better performance than that analytically predicted.

1.1. Related problems and other applications

In general, non-uniform costs may appear in any kind of binary identification procedure. As an example, consider the problem of searching for a defect in a pipeline [6]. The pipeline test problem consists of a finite number of segments linked together from left to right which can be tested for defects only at a link. The test of a link establishes whether the defect is in a segment to the left or to the right of the link. If the test of link i, for $i = 1,\ldots,n$, has cost c_i, then it can be useful to determine a test strategy with minimum expected cost.

The filter design problem described by Steiglitz and Parks [10] requires the solution of the following problem. For each positive integer k between two prescribed limits there is an associated LP (linear programming) of complexity k, the feasibility of which is not known. When the LP is feasible for some value of k, then all larger values of k have feasible associated problems. It is required that we find the smallest k whose associated LP is feasible. In this case, the cost of probe an integer k is the time required to solve a LP of complexity k.

Non-uniform costs also appear on searching in tapes [11], Hard-Disks and CD-ROMs [9, 8]. In these applications, the access cost of a key is not fixed since it depends on the last accessed key.

1.2. Paper organization

This paper is divided as follows. In Section 2, we present the Ratio algorithm and an upper bound on the cost of optimal BSTs. In Section 3, we show a necessary condition for a BST to be an optimal BST and we give an upper bound on the height of an optimal BST. In Section 4, we report some experiments comparing Ratio to other algorithms proposed in the literature. In Section 5, we comment our findings and discuss some future research.

2. The ratio algorithm

In this section, we prove that the cost of an optimal BST is bounded above by $4C\ln(1+n)$. It must be observed that we do not assume anything concerning the costs. Our upper bound is based on the analysis of the Ratio algorithm presented in Fig. 1.

```
Ratio Algorithm;
root ← Root(1,n);
Function Root (i,m): node;
 If i ⩽ m then
    k* ← i;
    For k = i to m do
        If  c_k / min{k − i + 1, m − k + 1} ⩽ c_{k*} / min{k* − i + 1, m − k* + 1}
        then k* ← k;
    a_{k*}.left ← Root(i, k* − 1) ; a_{k*}.right ← Root(k* + 1, m);
    Return a_{k*};
Else
    Return null;
```

Fig. 1. The Ratio algorithm.

Ratio uses a top-down approach combined with a simple rule to select the root of the BST for the current key interval. If the current key interval is $[a_i,\ldots,a_m]$, then the key a_k^* that minimizes

$$c_{k^*}/\min\{k^*-i+1, m-k^*+1\}$$

is selected to be the root of the subtree for the current interval. After selecting the root, Ratio recursively constructs subtrees for the key intervals $[a_i,\ldots,a_{k^*-1}]$ and $[a_{k^*+1},\ldots,a_m]$. The root of the subtree for $[a_i,\ldots,a_{k^*-1}]$ becomes the left child of a_k^* and the root of the subtree for $[a_{k^*+1},\ldots,a_m]$ becomes the right child of a_k^*. In the pseudo-code, $a_k^*.left$ and $a_k^*.right$ are, respectively, the left and the right child of the node a_k^*. If a node a_k^* does not have a left (right) child, then $null$ is assigned to $a_k^*.left(right)$.

Basically, Ratio attempts to balance two goals when it selects the root of the subtree for the current interval :

(i) select the key that splits the interval into two halves; and

(ii) select the key with minimum cost.

In fact, Ratio assigns penalties (multiplicative factors) to the keys. The keys that are closer to the extremes of the current interval receive larger penalties than those that are closer to the median of the current interval. As an example, if the current interval is $[a_3,a_4,a_5,a_6,a_7,a_8]$, then a_3 and a_8 receive penalty $1=1/\min\{3-3+1, 8-3+1\}=1/\min\{8-3+1, 8-8+1\}$, a_4 and a_7 receive penalty $\frac{1}{2}$ and a_4 and a_5 receive penalty $\frac{1}{3}$. Ratio selects the key with minimum cost, accounting the penalties, to be the root of the search interval.

2.1. Cost analysis

In this section we analyze the cost of the tree produced by the Ratio algorithm. In order to bound this cost, we need the following proposition.

Proposition 1. *If the key a_{k^*} is selected by Ratio to be the root of the key interval $[a_1,\ldots,a_n]$, then*

$$c_{k^*} \leqslant \frac{4\min\{k^*, n-k^*+1\}C}{n(n+2)}.$$

Proof. Since a_{k^*} is selected, then we have the following inequalities:

$$c_i \geqslant \frac{c_{k^*} \times \min\{i, n-i+1\}}{\min\{k^*, n-k^*+1\}} \quad \text{for } i = 1,\ldots,n.$$

By adding the inequalities above in i, we obtain that

$$\begin{aligned}\sum_{i=1}^{n} c_i &\geqslant \sum_{i=1}^{\lceil n/2\rceil} \frac{c_{k^*} \times i}{\min\{k^*, n-k^*+1\}} + \sum_{i=\lceil n/2\rceil+1}^{n} \frac{c_{k^*} \times (n-i+1)}{\min\{k^*, n-k^*+1\}} \\ &\geqslant \frac{c_{k^*}(n+2)n}{4\min\{k^*, n-k^*+1\}}, \end{aligned} \tag{4}$$

since for even n,

$$\sum_{i=1}^{n/2} i + \sum_{i=n/2+1}^{n} n-i+1 = \frac{(n+2)n}{4},$$

and for odd n,

$$\sum_{i=1}^{(n+1)/2} i + \sum_{i=(n+1)/2+1}^{n} n-i+1 = \frac{(n+1)^2}{4} > \frac{(n+2)n}{4}.$$

Since $\sum_{i=1}^{n} c_j = C$, then it follows from (4) that

$$c_{k^*} \leqslant \frac{4\min\{k^*, n-k^*+1\}C}{n(n+2)}. \qquad \square$$

Theorem 2. *The cost of the tree T_{R} constructed by the Ratio algorithm for the key interval $[a_1,\ldots,a_n]$ is bounded above by $4C\ln(n+1)$.*

Proof. If $n=1$, the result holds, since $4c_1 \ln 2 > c_1$. Now, we assume that the result holds for all key subintervals $[a_i,\ldots,a_m]$ with $m-i+1<n$ and we prove that the result also holds for the key interval $[a_1,\ldots,a_n]$.

Let us assume that the key a_{k^*} is selected by Ratio to be the root of the tree T_{R} constructed for the key interval $[a_1,\ldots,a_n]$. Moreover, we assume without loss of generality that $k^* \leqslant \lceil n/2\rceil$. The case where $k^* > \lceil n/2\rceil$ is symmetric. Hence, the cost $c(T_{\mathrm{R}})$ of the tree T_{R} is given by

$$c(T_{\mathrm{R}}) = nc_{k^*} + c(T(1,k^*-1)) + c(T(k^*+1,n)), \tag{5}$$

where $c(T(1,k^*-1))$ and $c(T(k^*+1,n))$ are, respectively, the costs of the subtrees constructed by the Ratio algorithm for the key intervals $[a_1,\ldots,a_{k^*-1}]$ and $[a_{k^*+1},\ldots,a_n]$.

If $C_1 = \sum_{i=1}^{k^*-1} c_i$ and $C_2 = \sum_{i=k^*+1}^{n} c_i$, then it follows from (5) and from the inductive hypothesis that

$$c(T_R) \leqslant nc_{k^*} + 4C_1 \ln(k^*) + 4C_2 \ln(n - k^* + 1).$$

Since $\ln(k^*) \leqslant \ln(n - k^* + 1)$ for $1 \leqslant k^* \leqslant \lceil n/2 \rceil$ and $C_1 + C_2 < C$, then we obtain that

$$c(T_R) \leqslant nc_{k^*} + 4(C_1 + C_2)\ln(n - k^* + 1) < nc_{k^*} + 4C \ln(n - k^* + 1).$$

On the other hand, since $k^* \leqslant \lceil n/2 \rceil$, it follows from Proposition 1 that $c_{k^*} \leqslant (4k^*C)/n$
$(n + 2)$. Hence,

$$c(T_R) < \frac{4k^*C}{n+2} + 4C \ln(n - k^* + 1).$$

Let $f(k^*) = 4k^*C/(n+2) + 4C\ln(n - k^* + 1)$. Since the derivative of f is negative in the interval $[0, n]$, then the value of f decreases in this interval. Hence, $f(k^*)$ reaches its maximum in the interval $[0, \lceil n/2 \rceil]$ when $k^* = 0$, and as a consequence we have that

$$c(T_R) < \frac{4k^*C}{n+2} + 4C \ln(n - k^* + 1) \leqslant 4C \ln(n + 1),$$

what establishes the theorem. □

Now, we present two corollaries of the previous result.

Corollary 3. *The cost of an optimal search tree for the key interval* $[a_1, \ldots, a_n]$ *is bounded above by* $4C \ln(n + 1)$.

Proof. It follows from Theorem 2 and from the fact that the cost of an optimal BST is no greater than the cost of the tree produced by Ratio algorithm. □

Corollary 4. *Ratio is a* $4 \ln(n + 1)$*-approximated algorithm.*

Proof. It follows from Theorem 2 and from the fact that the cost of an optimal BST is $\Omega(C)$. □

The upper bound $4C \ln(n + 1)$ is asymptotically tight up to constant factors in the sense that the cost of an optimal BST, for some cost structures, is $\Omega(C \log n)$. As an example, for the cost structure $c_i = i$, Knight [3] proved that the cost of an optimal BST is at least $(n + 1)^2[\log(n + 1)/2 - 1]$.

Although there are some cost structures such that the cost of an optimal BST is $\Omega(C \log n)$, there are others where the cost of an optimal BST is $O(C)$. As an example, the cost structure $c_i = 2^i$, for $i = 1, \ldots, n$. In this case, the tree constructed by Ratio is totally unbalanced, that is, the only node at level i, for $i = 1, \ldots, n$, is the key a_i. It is

easy to verify that the cost of this tree is $O(C)$. It would be interesting to determine if there is a constant K such that $c(T_R)/c(T^*) \leqslant K$ for any given cost structure, where $c(T_R)$ and $c(T^*)$ are, respectively, the cost of the tree produced by Ratio and the cost of an optimal BST. So far, we do not know how to answer this question.

2.2. Time analysis

Clearly, the Ratio algorithm runs in $O(nH_R)$ time, where H_R is the height of the tree obtained at the end of the algorithm. Since $H_R \leqslant n$, then the time complexity of Ratio is $O(n^2)$.

3. Combinatorial properties

In this section, we show some combinatorial properties of the optimal search trees. First, we show a necessary condition for a given key to be the root of an optimal search tree. After that, we give a necessary condition for the height of an optimal search tree to exceed H when $H > 2\sqrt{ne} \approx 3.297\sqrt{n}$.

3.1. Optimality condition

Theorem 5. *Let T^* be an optimal search tree for the key interval $[a_i,\ldots,a_m]$. Furthermore, let $n^*(a_h)$ be the number of descendants of the key a_h in T^* for $h=i,\ldots,m$. If a_k is the root of T^*, then we have that*:

(i) $c_j \geqslant \frac{(j-i+1)c_k}{m-i+1-n^*(a_j)}$, for $j=i,\ldots,k-1$.

(ii) $c_l \geqslant \frac{(m-l+1)c_k}{m-i+1-n^*(a_l)}$, for $l=k+1,\ldots,m$.

Proof. The idea of the proof is to show that if either (i) or (ii) does not hold, then the cost of the tree T^* can be improved. We only give a proof that condition (i) must hold, since the proof for condition (ii) is similar.

Let us assume that a_k is the root of the optimal search tree T^* for the key interval $[a_i,\ldots,a_m]$. Now, let $j<k$ and let T be the tree obtained after the application of the procedure below.

Procedure MoveToRoot

1. $T \leftarrow T^*$

2. While a_j **is not the root of the tree** T, **obtain a new tree** T **through a rotation involving** a_j **and its parent**

Fig. 2(a) shows a tree T^* and Fig. 2(b) shows the tree T obtained by applying the procedure MoveToRoot.

We can observe two facts:

(a) If $a_{j'}$ is not an ancestor of node a_j in T^*, then its number of descendants does not modify from T^* to T.

(b) If $a_{j'}$ is an ancestor of node a_j in T^* and $j' \neq j$, then its number of descendants in T^* is greater than or equal its number of descendants in T.

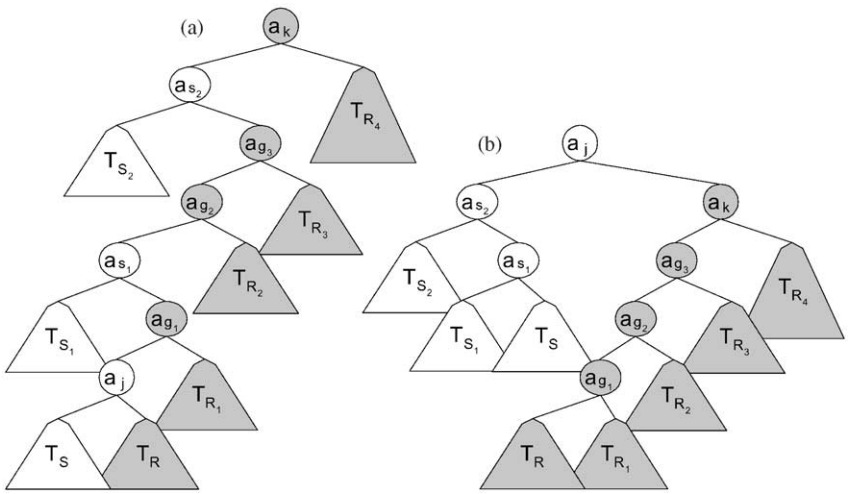

Fig. 2. (a) shows a tree T^* with root a_k and (b) shows the tree T obtained by applying the procedure MoveToRoot. Every node and every subtree that contains nodes with index greater than j is shadowed in both figures.

Fact (a) holds since the subtree rooted at $a_{j'}$ is not modified due to the rotations. On the other hand, fact (b) holds since every descendant of $a_{j'}$ in T is also a descendant of $a_{j'}$ in T^*.

Given a key a_h, let $n(a_h)$ be the number of descendants of node a_h in the tree T. We have that $n(a_j)=(m-i+1)$, $n^*(a_k)=(m-i+1)$ and $n(a_k)=(m-j)$. Furthermore, it follows from (a) and (b) that $n(a_h)\leqslant n^*(a_h)$ for $h\neq j,k$. Hence, we have that

$$\begin{aligned} c(T)-c(T^*) &= \sum_{h=i}^{m} c_h n(a_h) - \sum_{h=i}^{m} c_h n^*(a_h) \\ &\leqslant (m-i+1-n^*(a_j))c_j - (j-i+1)c_k. \end{aligned}$$

As a consequence, we must have $c_j \geqslant ((j-i+1)c_k)/(m-i+1-n^*(a_j))$, otherwise we would have $c(T)<c(T^*)$, which contradicts the optimality of T^*. □

We point out that this result can be easily generalized for the case where the keys have different access probabilities.

3.2. Bounding the height of an optimal search tree

Now, we give a necessary condition for the height of an optimal search to exceed H when $H>2\sqrt{ne}\approx 3.297\sqrt{n}$. A consequence of this result is that the height of optimal search trees for practical cases must be $O(\sqrt{n})$, otherwise there must be costs differing by an enormous factor as $f^{\sqrt{n}}$, where f is $\omega(1)$. For example, if $H=6.6\sqrt{n}$, then there are two keys differing by a factor greater than $2^{\sqrt{n}}$.

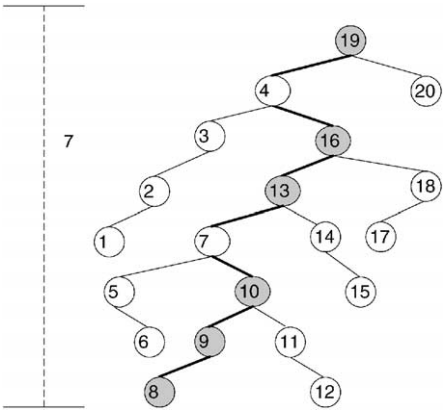

Fig. 3. The picture shows a BST for the key interval $[a_1,\ldots,a_{20}]$. The bold edges define the path P and the shadowed nodes are the nodes that belong to the set L.

Theorem 6. *If the height of the optimal search tree for the key interval* $[a_1,\ldots,a_n]$ *is* H, *with* $H > 2\sqrt{ne} \approx 3.297\sqrt{n}$, *then there are two keys* a_i *and* a_j *such that*

$$\frac{c_i}{c_j} > \sqrt{\pi H}\quad\left(\frac{H}{2\sqrt{ne}}\right)^H.$$

Proof. Let us consider an optimal search tree T^* with height H and let P be a path from some leaf to the root of T^* that contains $H+1$ nodes. Then, let $a_{j_0},\ldots,a_{j_H}$ be the nodes of P, where a_{j_k} is the unique node of P at level $H-k$ in T^*. Now, we construct two subsets of the nodes of P, the first called L and the second called R. The set L is given by all nodes a_{j_k} such that $a_{j_{k-1}}$ is a left child of a_{j_k}, while R is given by all nodes a_{j_k} such that $a_{j_{k-1}}$ is a right child of a_{j_k}. Furthermore, both L and R contain a_{j_0}.

As an example, Fig. 3 shows a BST with height 7. The number inside each node corresponds to the key index. The path P is defined by the nodes that touch a bold edge. The shadowed nodes are the nodes of L. In this case, L is given by the keys $a_8, a_9, a_{10}, a_{13}, a_{16}, a_{19}$ and R is given by the keys a_8, a_7, a_4.

We assume without loss of generality that $|L| \geqslant |R|$. Since $|L| + |R| = H + 2$, it follows that $|L| \geqslant 1 + \lceil H/2 \rceil$. Then, let $a_{i_1}, a_{i_2},\ldots,a_{i_{|L|}}$ be the keys of L, with $i_j < i_{j+1}$, for $j = 1,\ldots,|L|-1$.

Now, for $j = 1,\ldots,|L|$, let $a_{\ell_{j-1}}$ be the leftmost descendant of the subtree of T^* rooted at a_{i_j}. Hence, a_{i_j} is the root of the key interval $[a_{\ell_{j-1}},\ldots,a_{i_{j+1}-1}]$, for $j = 1,\ldots,|L|$. For the correctness of the last statement, we define $i_{|L|+1} = n+1$. Since T^* is an optimal search tree and a_{i_j} is descendant of $a_{i_{j+1}}$, it follows from Theorem 5 that

$$c_{i_j} \geqslant \frac{c_{i_{j+1}}(i_j - \ell_j + 1)}{(i_{j+2} - i_{j+1}) + (\ell_{j-1} - \ell_j)} \quad \text{for } j = 1,\ldots,|L|-1.$$

By multiplying the set of inequalities above, we obtain that

$$\frac{c_{i_1}}{c_{i_{|L|}}} \geqslant \prod_{j=1}^{|L|-1} \frac{i_j - \ell_j + 1}{(i_{j+2} - i_{j+1}) + (\ell_{j-1} - \ell_j)}. \tag{6}$$

Hence, in order to give a lower bound on $c_{i_1}/c_{i_{|L|}}$, we must minimize the right hand side of (6) constrained to $1 \leqslant \ell_{|L|-1} \leqslant \cdots \leqslant \ell_0 \leqslant i_1 < \cdots < i_{|L|+1} = n+1$.

First, we have that

$$\prod_{j=1}^{|L|-1} (i_j - \ell_j + 1) \geqslant (|L|-1)! \tag{7}$$

On the other hand, since

$$\sum_{j=1}^{|L|-1} (i_{j+2} - i_{j+1}) + (\ell_{j-1} - \ell_j) \leqslant n,$$

it follows from means inequality that

$$\prod_{j=1}^{|L|-1} (i_{j+2} - i_{j+1}) + (\ell_{j-1} - \ell_j) \leqslant \left(\frac{n}{|L|-1}\right)^{|L|-1}. \tag{8}$$

From (6)–(8), we obtain that

$$\frac{c_{i_1}}{c_{i_{|L|}}} \geqslant \prod_{j=1}^{|L|-1} \frac{i_j - \ell_j + 1}{(i_{j+2} - i_{j+1}) + (\ell_{j-1} - \ell_j)} \geqslant (|L|-1)! \quad \left(\frac{|L|-1}{n}\right)^{|L|-1}.$$

It follows from Stirling approximation that

$$(|L|-1)! > \sqrt{2\pi(|L|-1)} \quad \left(\frac{|L|-1}{e}\right)^{|L|-1}.$$

From the two previous inequalities, we obtain that

$$\frac{c_{i_1}}{c_{i_{|L|}}} > \sqrt{2\pi(|L|-1)} \quad \left(\frac{|L|-1}{\sqrt{ne}}\right)^{2|L|-2}.$$

Since $|L|-1 \geqslant \lceil H/2 \rceil$, then

$$\frac{c_{i_1}}{c_{i_{|L|}}} > \sqrt{\pi H} \quad \left(\frac{H}{2\sqrt{ne}}\right)^{2|L|-2}.$$

If $H > 2\sqrt{ne}$, then we have that

$$\frac{c_{i_1}}{c_{i_{|L|}}} > \sqrt{\pi H} \quad \left(\frac{H}{2\sqrt{ne}}\right)^{H}. \qquad \square$$

4. Experimental results

In this section, we report some experiments obtained by comparing the performance of the Ratio algorithm with the optimal search strategy and with other strategies proposed in the literature. We consider four different algorithms:

Opt: the optimal algorithm proposed in [3];

Ratio: the algorithm presented in this paper;

BinS: a standard binary search, that is, a strategy represented by a complete binary search tree;

Small: a greedy algorithm proposed in [9] that always choose the node with minimum cost to be the root of the current key interval.

We observe that Small was devised for a text retrieval application described in [9], and it has a good performance only if the following assumption holds: given an index k, with $k \in \{i,\ldots,j\}$, the probability of a_k being the key with minimum cost in the key interval $[a_i,\ldots,a_j]$ is $1/(j-i+1)$.

We use two kinds of cost structures in our experiments. The first one, is the structure that arises in the filter design problem described in [3]. The cost c_i is given by i^t, where t is a fixed positive constant. The second one is a random structure of costs. The costs are generated by choosing c_i, randomly, in the interval $[1,\ldots,c^*]$, where c^* is the maximum number of different costs. We consider this cost structure, because it satisfies the assumption required by the Small algorithm. In every experiment, the cost of the produced tree was normalized through a division by $\sum_{i=1}^{n} c_i$.

4.1. The filter design cost structure

Table 1 presents the cost of the BSTs obtained by the algorithms for the cost structure $c_i = i^t$, with $t = 1,2,3$ and $n = 50,200,500$. The cost obtained by Opt is presented as an absolute value, while the cost of the trees obtained by the other algorithms are presented as relative errors with respect to the optimal one. Observe that Ratio and BinS achieved very good results. The relative error of these algorithms with respect to the optimal one was smaller than 4% in all the cases. In this table, we omit the results of Small, since the assumption required by this algorithm does not hold for the filter design cost structure. In fact, Small produces very poor results in this case.

Table 1
Cost of the BSTs constructed by OPT, Ratio and BinS, for the cost structure $c_i = i^t$, with $t = 1,2,3$ and $n = 50,200,500$

t	1			2			3		
n	50	200	500	50	200	500	50	200	500
Opt	4.75	6.66	7.98	4.39	6.27	7.56	4.09	5.94	7.22
Ratio (%)	2.3	1.5	0.3	1.8	1.2	0.5	2.7	0.8	1.5
BinS (%)	1.7	1.5	0.1	3.4	2.7	1.3	3.7	2.9	1.6

Table 2
Heights of the trees produced by Opt, Ratio and BinS for the cost structure $c_i = i^t$, with $t = 1, 2, 3$ and $n = 50, 200, 500$

t	1			2			3		
n	50	200	500	50	200	500	50	200	500
Opt	5	7	8	7	9	11	8	10	12
Ratio	6	8	9	10	14	16	12	17	21
BinS	5	7	8	5	7	8	5	7	8

Table 3
Average cost of the BSTs constructed by OPT, Ratio, BinS and Small for the random cost structure

c^*		2			5			100			1000		
n		50	200	500	50	200	500	50	200	500	50	200	500
Opt	Average	3.82	5.07	5.94	3.01	3.67	4.12	2.52	2.67	2.74	2.49	2.61	2.67
	StdDev	0.09	0.06	0.05	0.12	0.07	0.04	0.2	0.12	0.08	0.2	0.14	0.09
Ratio (%)	Average	0.6	0.57	0.51	0.68	0.75	0.73	0.5	0.55	0.51	0.49	0.47	0.48
	StdDev	0.52	0.19	0.12	0.55	0.27	0.16	0.57	0.3	0.17	0.62	0.26	0.19
Small (%)	Average	1.04	0.89	0.78	3.06	2.58	2.54	9.38	9.68	9.77	9.56	10.45	10.71
	StdDev	0.61	0.26	0.16	1.41	0.72	0.42	4.1	2.1	1.39	4.12	2.48	1.65
BinS (%)	Average	27	33	35	62	83	95	95	154	191	93	158	204
	StdDev	10.8	8.8	8	18	15	16	36	33	29	31	35	31

Table 2 presents the height of the trees obtained at the end of each experiment. Observe that the results suggest that the height of the tree produced by Ratio for the cost structure $c_i = i^t$ is $O(t \log n)$.

4.2. The random cost structure

Table 3 presents the results obtained for the random cost structure. We report experiments for c^* equal to $2, 5$, 100 and 1000. In order to obtain more stable results, we generated one hundred distinct cost structures for each pair (n, c^*). Hence, each cost reported in the third line of Table 3 represents the average cost of one hundred optimal BSTs constructed for the corresponding random cost structures. The fifth, seventh and ninth line indicate the average relative error of the BSTs generated by Ratio, Small and BinS, respectively. We also include an estimation for the standard deviation (StdDev) of each reported average. We make the following observations:

1. The expected cost of an optimal BST decreases when the number of different costs increases;
2. Ratio was the best non-optimal algorithm, obtaining excellent results in all experiments. Its relative error was smaller than 1% in all cases;

Table 4
Average height of the BSTs constructed by OPT, Ratio, Small and BinS for the random cost structure

c^*	2			5			100			1000		
n	50	200	500	50	200	500	50	200	500	50	200	500
Opt	5.8	7.96	9.05	6.09	8.37	10.01	6.79	9.72	11.52	7	9.93	11.85
Ratio	5.9	8	9.11	6.12	8.44	10.03	6.88	9.79	11.66	7.07	10.17	12.16
Small	6.02	8.1	9.87	7.16	9.7	11.5	9.63	13.9	16.4	9.7	14.66	18.42
BinS	5	7	8	5	7	8	5	7	8	5	7	8

3. Small obtained good results in all cases, since its maximum relative error was about 10%. The experiments suggest that its relative error does not depend on the number of keys and that it increases with the number of different costs;
4. BinS obtained poor results. In fact, it is not difficult to show that the expected cost of the tree obtained by BinS is $\Theta(C \log n)$ [9], which is $\Theta(\log n)$ in our case, due to the normalization. Hence, due to the observation 1, its relative error with respect to Opt increases when the number of different costs increases.

Finally, Table 4 presents the average height of the BSTs produced for the random cost structure. We observe that the average height of the trees produced by Ratio was smaller than $1.5 \log n$ in all experiments, which suggests that Ratio runs in $O(n \log n)$ expected time for a random cost structure. This conjecture is motivated by the following observation. Given two indexes k and k', with $|k - n/2| < |k' - n/2|$, the probability of Ratio selecting the key a_k, as the root of the key interval $[a_1, \ldots, a_n]$, is not smaller than the probability of selecting the key $a_{k'}$. Hence, the BST produced by Ratio is more likely to be balanced than a random binary search tree, which has expected height $O(\log n)$ ([1, Chapter 13]) .

Comparing all the experiments, we realize that Ratio got a satisfactory result, with relative error smaller than 3% in all cases. Nevertheless, we must observe that we have just considered two cost structures and we do not have a strong (constant) approximation result for Ratio algorithm.

5. Conclusion

In this paper we introduced the Ratio algorithm, an approximated algorithm for constructing binary search trees with minimum expected access cost over uniform probabilities. The algorithm runs in $O(n^2)$ time and requires $O(n)$ space. The analysis of Ratio shows that $4C \ln(n + 1)$ is an upper bound on the cost of the optimal binary search strategy.

The Ratio algorithm presented here can be useful since the fastest known exact method to construct minimum cost binary search trees runs in $O(n^3)$ time, requiring $O(n^2)$ space. As a future work, we intend to extend our techniques to handle the case where the access probabilities are also non-uniform.

Acknowledgements

We would like to thank Prof. Gonzalo Navarro that introduced us this problem and to Prof. Ricardo Baeza-Yates for some helpful comments. Furthermore, we would like to thank the referees for the careful revision.

References

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, 6th Edition, MIT Press and McGraw-Hill Book Company, Cambridge, MA; New York, 1992.
[2] R. De Prisco, A. De Santis, On binary search trees, Inform. Process. Lett. 45 (1993) 249–253.
[3] W.J. Knight, Search in an ordered array having variable probe cost, SIAM J. Comput. 17 (1988) 1203–1214.
[4] D.E. Knuth, in: Optimum Binary Search Trees, Acta Informatica, Vol. 1, Springer, Heidelberg, FRG and NewYork NY, USA, 1971.
[5] L.L. Larmore, A subquadratic algorithm for constructing approximately optimal binary search trees, J. Algorithms 8 (1987) 579–591.
[6] M.J. Lipman, J. Abrahams, Minimum average cost testing for partially order, IEEE Trans. Inform. Theory 41 (1995) 287–291.
[7] K. Mehlhorn, Nearly optimal binary search trees, Acta Informatica 5 (1975) 287–295.
[8] R.L. Milidiú, A.A. Pessoa, E.S. Laber, R. Renteria, Fast calculation of optimal strategies for searching with non-uniform costs, in: P. Fuente (Ed.), Proc. SPIRE 2000, A Coruña, Spain, September 2000, pp. 229–235.
[9] G. Navarro, E. Barbosa, R. Baeza-Yates, W. Cunto, N. Ziviani, Binary searching with non-uniform costs and its application to text retrieval, Algorithmica 27 (2000) 145–169.
[10] K. Steiglitz, T.W. Parks, What is the filter-design problem, in: B.W. Dickenson (Ed.), Proc. Princeton Conference on Information Sciences and Systems, New Jersey, November 1986.
[11] M. Wachs, On an efficient dynamic programming technique of f f yao, J. Algorithms 10 (1989) 518–530.