Optimal Binary Search Trees with Costs Depending on the Access Paths

Jayme L Szwarcfiter[1] Gonzalo Navarro[2]

Joísa de S. Oliveira[3] Ricardo Baeza-Yates[4]

Walter Cunto[5] Nívio Ziviani[6]

August 1998

Key Words: algorithms, binary search trees, path dependent costs.

ABSTRACT

We describe algorithms for constructing optimal binary search trees, in which the access cost to each key depends on the k preceeding keys which were reached in the path to the desired key. Two kinds of optimal trees are considered, namely optimal worst case and weighted average case trees. The time complexities of the corresponding algorithms are $O(n^{k+2})$ and $O(kn^{k+2})$ respectively, while the space complexity is $O(n^{k+1})$, for both cases. The algorithms are based on a convenient decomposition theorem, and on a characterization of sequences of keys which are paths in some binary search tree. In addition, a characterization of sequences of keys which lead to a subtree formed by consecutive keys, in some binary search tree. An exact analysis of the number of steps performed by the algorithms is included.

[1]Universidade Federal do Rio de Janeiro, Instituto de Matemática, NCE and COPPE, Caixa Postal 2324, 20001-970 Rio de Janeiro, RJ, Brasil. E-mail: jayme@nce.ufrj.br

[2]Universidad de Chile, Departamento de Ciência de la Computación. E-mail: gnavarro@dcc.uchile.cl

[3]Universidade Federal do Rio de Janeiro, COPPE, Caixa Postal 68511, Rio de Janeiro, RJ, Brasil. E-mail: joisa@cos.ufrj.br

[4]Universidad de Chile, Departamento de Ciência de la Computación. E-mail: rbaeza@dcc.uchile.cl

[5]Universidad Simon Bolivar and IBM, Caracas, Venezuela. E-mail: wcunto@vnet.ibm.com

[6]Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Belo Horizonte, MG, Brasil. E-mail: nivio@dcc.ufmg.br

1 Introduction

Binary search trees form one of the topics most widely studied in computer science, probably due to its range of application. Its importance can be assessed by reading [3] and [4]. Relevant papers on binary search trees date back to the 50's, while a tutorial on the subject has recently appeared [5].

In this paper we consider the problems of finding optimal binary search trees in which the access cost to each key depends on the k preceeding keys which were reached in the path to the desired key. The classical optimal binary search tree construction by Gilmore and Moore [1] and Knuth [2] corresponds thus to the fundamental case $k = 0$. In this work we are concerned with the values $k \geq 1$. Two kinds of optimal trees are considered, namely optimal worst case and weighted average case trees. The inputs of these problems are a number n of keys, the value k, $1 \leq k < n$, and a cost associated to each possible sequence formed by at most $k+1$ keys, all of them distinct. For the weighted average case minimization problem, each key is additionally given a weight. We remark that the input size grows exponentially with k.

We describe algorithms for solving the two problems above. Their complexities are $O(n^{k+2})$ and $O(k.n^{k+2})$, respectively for minimizing worst case and weighted average case. The algorithms are polynomial in the size of the input.

The optimal binary search tree for $k = 0$ and with uniform key access costs, as considered in [1, 2], is a model for situations in which the keys are in the main memory. Greater values of k and arbitrary access costs could model the cases in which other kind of memories are involved. For example, when all keys are stored in a disk, the access cost to a given key depends on the position on the disk of the key previously accessed. Therefore finding an optimal tree when all keys are stored in a disk would correspond to the case $k = 1$. In this situation, the input size is $O(n^2)$ and the complexity of the proposed algorithm is $O(n^3)$. Besides practical motivations, we believe that some of the concepts presented in this paper are of interest in the study of search trees, in general.

The following are some basic definitions.

A **binary tree** is a rooted tree T in which every node z, other than the root, is labelled **left child** or **right child**, in such a way that any two brothers have different labels. When z has no brothers it is called an **only child**. A **path** of T is a sequence of nodes $z_1, \ldots, z_t$, such that z_q is the father of z_{q+1}. In this case, z_1 is an **ancestor** of z_t, while z_t is a **descendant** of z_1. When $z_1 \neq z_t$ they are called **proper ancestor** and **proper descendant**, respectively. A t-path is a path formed by t nodes. The notation $N(T)$ represents the set of nodes of T. For $z \in N(T)$, the binary tree defined in T by all descendants of z is called the **subtree** of T **rooted** at z, and denoted by $T(z)$. The **left subtree** of z is the binary tree formed in T by the left child of z and all of its descendants. Similarly, define the **right subtree** of z. Represent by $T_L(z)$ and $T_R(z)$ the left and right subtrees of z, respectively. A binary tree defined in T by a subset of $N(T)$ is called a **partial subtree** of T. A **root path** is a path starting at the root of T, while a **root-leaf path** starts at the root and ends at some leaf of T.

Let $\{x_1, \ldots, x_n\}$ be a set of elements called **keys**, $x_q < x_{q+1}$. A **binary search tree** for $\{x_1, \ldots, x_n\}$ is a binary tree T in which $N(T) = \{x_1, \ldots, x_n\}$, with every pair of keys $x_p, x_q \in N(T)$ satisfying: $x_q \in N(T_L(x_q))$ implies $x_q < x_p$, and $x_q \in N(T_R(x_q))$ implies $x_q > x_p$. A **legal path** is a sequence of keys which is a path in some binary search tree.

The described minimization problems are solved by dynamic programming equations. The corresponding decompositions employ the concept of a legal path and that of a (i, j)-legal path. The latter means those legal paths leading to a subtree formed by consecutive keys. We then describe characterizations for both legal and (i, j)-legal paths. The algorithms are obtained by combining the decompositions and the characterizations. The decompositions are presented in Section 2, the characterizations in Section 3, while the algorithms and their analysis are described in 4. Some additional remarks form the last section.

2 The decompositions

Let $k \geq 1$ be a given integer value and $\{x_1, \ldots, x_n\}$ a set of keys, $x_q < x_{q+1}$. For each x_q and legal path $y_1, \ldots, y_t$, where $1 \leq t \leq k+1$ and $x_q = y_t$, it is given a real non-negative **key cost** $c(y_1, ..., y_t)$ of y_t **relative** to $y_1, \ldots y_t$. It corresponds to the cost of reaching y_t through the path $y_1, ..., y_t$. In adddition, each key x_q is given a non-negative real **weight** $w(x_q)$. For a legal path $y_1, \ldots, y_m$, define its **path cost** as

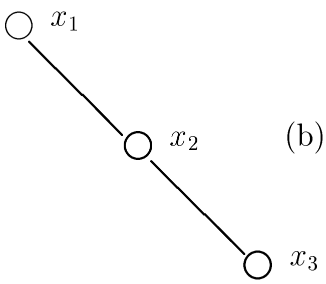
$$C(y_1, \ldots, y_m) = \sum_{1 \leq q \leq m} c(y_{\max\{1, q-k\}}, \ldots, y_q) \tag{1}$$

Let T be a binary search tree for $\{x_1, \ldots, x_n\}$. Denote by x_q^* the root path to key x_q. The values $max_{1 \leq q \leq n}\{C(x_q^*)\}$ and $\sum_{1 \leq q \leq n} w(x_q).C(x_q^*)$ are called **worst case tree cost** and **weighted average case tree cost**, respectively. The question consists of finding the tree T which minimizes one of these two above costs, as desired. A minimizing tree is called **optimal**.

Observe that subtrees of an optimal tree are not necessarily optimal, for any $k > 0$. Consider the example having $k = 1$, $n = 3$, with key costs as given by figure 1(a) and having all weights equal to 1.

legal paths	x_1	x_2	x_3	x_1x_2	x_1x_3	x_2x_1	x_2x_3	x_3x_1	x_3x_2
key costs	0	0	0	0	2	3	2	3	1

(a)



(b)

Figure 1

The tree of figure 1(b) is both worst and average case optimal, but $T(x_2)$ is not optimal in any case. However, special kinds of partial subtrees are optimal, making it possible to solve our minimization problems by conveniently decomposing them into smaller subproblems. We need more notation.

First, introduce k additional keys $\{x_{n+1}, \ldots, x_{n+k}\}$, called **dummy keys**, also satisfying $x_q < x_{q+1}$, $n \leq q < n+k$. Each of these keys has weight 1. The key costs relative to paths containing dummy keys are defined as follows. Let $y_1, \ldots, y_t$ be a legal path having at least one dummy key, $1 \leq t \leq k+1$. Then

$$c(y_1, \ldots, y_t) = \begin{cases} 0, & \text{when } y_1, \ldots, y_t \text{ are all dummy keys} \qquad (2) \\ c(y_q, \ldots y_t), & \text{when } \exists\, q > 1 \text{ such that } y_1, \ldots, y_{q-1} \text{ are} \\ & \text{dummy keys, but } y_q, \ldots y_t \text{ are not} \qquad (3) \\ \infty, & \text{otherwise} \qquad (4) \end{cases}$$

Denote $X = \{x_1, \ldots, x_{n+k}\}$, $X_i^- = \{x_1, \ldots, x_i\}$, $X_i^+ = \{x_{i+1}, \ldots, x_{n+k}\}$, $X_{ij} = \{x_{i+1}, \ldots, x_j\}$ and $W_{ij} = \sum_{i<q\leq j} w(x_q)$.

Let i, j be a pair of integers, $0 \leq i \leq j \leq n$. A path $y_1, \ldots, y_k$ is **(i,j)-legal** when there exists a binary search tree T having node set X containing the path $y_1, \ldots y_k$ and such that either $i = j$ and y_k is a leaf of T, or y_k has a child $x_\ell \in X_{ij}$ satisfying $N(T(x_\ell)) = X_{ij}$.

A **root extension** of a binary search tree T is a root path $y_1, \ldots, y_t$, such that each y_q has at most one child in T, $1 \leq q \leq t$.

Let $y_1, \ldots, y_k$ be a (i,j)-legal path. Denote by $T_{ij}(y_1, \ldots, y_k)$ an optimal binary search tree having root extension $y_1, \ldots, y_k$ and node set $X_{ij} \cup \{y_1, \ldots, y_k\}$. Represent by $C_{ij}(y_1, \ldots, y_k)$ the (optimal) tree cost of $T_{ij}(y_1, \ldots, y_k)$. In terms of this notation, a solution to the stated minimization problems is the subtree of $T_{0n}(x_{n+k}, x_{n+k-1}, \ldots, x_{n+1})$, having as root the child of x_{n+1}.

The following are decomposition theorems for the computation of the costs of the above optimal trees. For each (i,j)-legal path $y_1, \ldots, y_k$, Theorems 1 and 2 compute $C_{ij}(y_1,\ldots,y_k)$, for the worst and average case cost criteria, respectively. The theorems decompose $T_{ij}(y_1,\ldots,y_k)$ into the optimal trees $T_{i,\ell-1}(y_2,\ldots,y_k,x_\ell)$ and $T_{\ell,j}(y_2,\ldots,y_k,x_\ell)$, where x_ℓ is the child of y_k in $T_{ij}(y_1,\ldots,y_k)$. See figure 2.

Theorem 1 (Worst case minimization):

$$C_{ij}(y_1,\ldots,y_k) = \begin{cases} C(y_1,\ldots,y_k), \text{when } i=j. \text{ Otherwise,} & (5) \\ \\ min_{i<\ell\le j}\{max\{C_{i,\ell-1}(y_2,\ldots,y_k,x_\ell), C_{\ell j}(y_2,\ldots,y_k,x_\ell)\}+ & \\ \qquad +C(y_1,\ldots,y_k,x_\ell)-C(y_2,\ldots,y_k,x_\ell)\}, & (6) \end{cases}$$

for all $1\le i\le j\le n$ and (i,j)-legal k-paths $y_1,\ldots,y_k$, $k\ge 1$.

Theorem 2 (Weighted average case minimization):

$$C_{ij}(y_1,\ldots,y_k) = \begin{cases} \sum_{1\le t\le k} w(y_t).C(y_1,\ldots,y_t), \text{when } i=j. \text{ Otherwise,} & (7) \\ \\ C_{i,\ell-1}(y_2,\ldots,y_k,x_\ell)+C_{\ell j}(y_2,\ldots,y_k,x_\ell)+ & \\ +W_{ij}.C(y_1,\ldots,y_k,x_\ell)-[W_{ij}+w(x_\ell)].C(y_2,\ldots,y_k,x_\ell)+ & \\ +\sum_{1<t\le k} w(y_t).[C(y_1,\ldots,y_t)-2.C(y_2,\ldots,y_t)]+ & \\ +w(y_1).C(y_1), & (8) \end{cases}$$

for all $1\le i\le j\le n$ and (i,j)-legal k-paths $y_1,\ldots,y_k$, $k\ge 1$.

PROOF OF THEOREMS 1 and 2:

By hypothesis, $y_1, \ldots, y_k$ is a (i,j)-legal k-path, $k \geq 1$ and $0 \leq i \leq j \leq n$. If $i = j$ it is simple to verify that (5) and (7) are correct. Let $i < j$ and $i < \ell \leq j$. First, we show that $y_2, \ldots, y_k, x_\ell$ is $(i, \ell-1)$-legal. Since $y_1, \ldots, y_k$ is (i,j)-legal, there exists a binary search tree T such that $N(T) = X$, while $y_1, \ldots, y_k$ is a path of it, with $x_\ell \in X_{ij}$ a child of y_k and $N(T(x_\ell)) = X_{ij}$. Hence T contains the path $y_2, \ldots, y_k, x_\ell$. The latter implies that $y_2, \ldots, y_k, x_\ell$ is $(i, \ell-1)$-legal whenever $X_{i,\ell-1} = \emptyset$. Consider now $X_{i,\ell-1} \neq \emptyset$. In this case, x_ℓ has a left child $x_m \in X_{i,\ell-1}$ in T and $N(T(x_m)) = X_{i,\ell-1}$. Consequently, $y_2, \ldots, y_k, x_\ell$ is a $(i, \ell-1)$-legal k-path. Analogously, $y_2, \ldots, y_k, x_\ell$ is a (ℓ, j)-legal k-path.

Also, by hypothesis, $T_{ij}(y_1, \ldots, y_k)$ is an optimal binary search tree having root extension $y_1, \ldots, y_k$ and node set $X_{ij} \cup \{y_1, \ldots, y_k\}$. Call tree $T_{ij}(y_1, \ldots, y_k)$ as T' and let C' be its cost. Denote by T'' and T''' the partial subtrees of T' with root y_2, containing $X_{i,\ell-1}$ and $X_{\ell j}$ respectively, each of them containing additionally the path $y_2, \ldots, y_k, x_\ell$. That is, T'' and T''' are binary search trees with root extension $y_2, \ldots, y_k, x_\ell$ and node sets $X_{i,\ell-1} \cup \{y_2, \ldots, y_k, x_\ell\}$ and $X_{\ell j} \cup \{y_2, \ldots, y_k, x_\ell\}$, respectively. Since $y_2, \ldots, y_k, x_\ell$ is both $(i, \ell-1)$-legal and (ℓ, j)-legal it follows that T'' and T''' are exactly the kind of trees are needed for the construction of the subproblems. Denote by C'', C''' the costs of T'' and T''', respectively.

WORST CASE MINIMIZATION:

Let $z \in N(T')$ be a key of T' with maximum access cost through the root path $z_1, \ldots, z_q$ to it, where $z = z_q$. That is $C' = C(z_1, \ldots, z_q)$. Since all costs are non-negative, we can assume that z_q is a leaf of T'. Hence, $z_q \in X_{ij} \setminus \{x_\ell\}$. Assume that $z_q \in X_{i,\ell-1}$. Then $z_2, \ldots, z_q$ is a root path to z_q in T''. Compare $C(z_1, \ldots, z_q)$ with $C(z_2, \ldots, z_q)$. The contribution of each key of $X_{i,\ell-1}$ to these costs is the same in both cases. However, x_ℓ contributes with $c(y_1, \ldots, y_k, x_\ell)$ to the first cost, while with $c(y_2, \ldots, y_k, x_\ell)$ to the second. Similarly, for $1 \leq a \leq k$, y_a contributes with $c(y_1, \ldots, y_a)$ to $C(z_1, \ldots, z_q)$ and with $c(y_2, \ldots, y_a)$ to $C(z_2, \ldots, z_q)$, except that y_1 does not contribute to $C(z_2, \ldots, z_q)$. That is, $C' = C(z_2, \ldots, z_q) + C(y_1, \ldots, y_k, x_\ell) -$

$C(y_2, \ldots, y_k, x_\ell)$. Note that $z_2, \ldots, z_q$ must be a root path of maximal access cost in T''. Otherwise, if there exists a key z_t such that the cost of the root path in T''' to z_t is greater than that to z_q, then $C' > C(z_1, \ldots, z_q)$, a contradiction. By the same argument it follows that $C'' \geq C'''$. Hence $C'' = C(z_2, \ldots, z_q)$. Since C' is minimal, it follows that C'' must be so. That is, $C'' = T_{i,\ell-1}(y_2, \ldots, y_k, x_\ell)$. Similarly, $C''' = T_{\ell j}(y_2, \ldots, y_k, x_\ell)$ and $C''' \geq C''$ when $x_q \in X_{\ell j}$. Equation(6) follows.

WEIGHTED AVERAGE CASE MINIMIZATION:

The contribution of each key $z \in N(T')$ to the cost of T'' is $w(z).C(z^*)$, where z^* is the root path to z. Compare C' with $C'' + C'''$. For each $z \neq y_1$, let $d(z)$ be the difference between the contributions of key z to C' and $C'' + C'''$. The following are clear. When $z \in X_{i,\ell-1} \cup X_{\ell j}$ the difference is $d(z) = w(z).[C(y_1, \ldots, y_k, x_\ell) - C(y_2, \ldots, y_k, x_\ell)]$. For $z = x_\ell$, we have $d(x_\ell) = w(x_\ell).[C(y_1, \ldots, y_k, x_\ell) - 2.C(y_2, \ldots, y_k, x_\ell)]$.

The latter follows from the fact that x_ℓ contributes twice to $C'' + C'''$. For $z \in \{y_2, \ldots, y_k\}$, the value is $d(z) = w(z).[C(y_1, \ldots, y_z) - 2.C(y_2, \ldots, y_z)]$, by this same reason. Finally, y_1 contributes with $w(y_1).C(y_1)$ to C' and nothing to $C'' + C'''$. The difference $d(z)$ in all cases does not depend on the costs of the trees. Hence C'' and C''' are minimal, otherwise C' would not be. That is, $C'' = C_{i,\ell-1}(y_2, \ldots, y_k, x_\ell)$ and $C''' = C_{\ell j}(y_2, \ldots, y_k, x_\ell)$ By writing C' in terms of $C'' + C'''$, as above mentioned, equation(8) is obtained. $\square$

3 Characterizing legal paths

In this section we describe characterizations for legal and (i, j)-legal paths. That is, for sequences of keys which are paths in some binary tree, and which lead to subtrees formed by consecutive keys, respectively. The following definition is useful.

Let $Y \subset X$. An ordering $y_1, \ldots, y_m$ of the keys of Y is called **min-max** when each y_q is either minimal or maximal in $\{y_q, \ldots, y_m\}$. In this case, label each y_q, $1 \leq q \leq m$, as **min** or **max**, respectively.

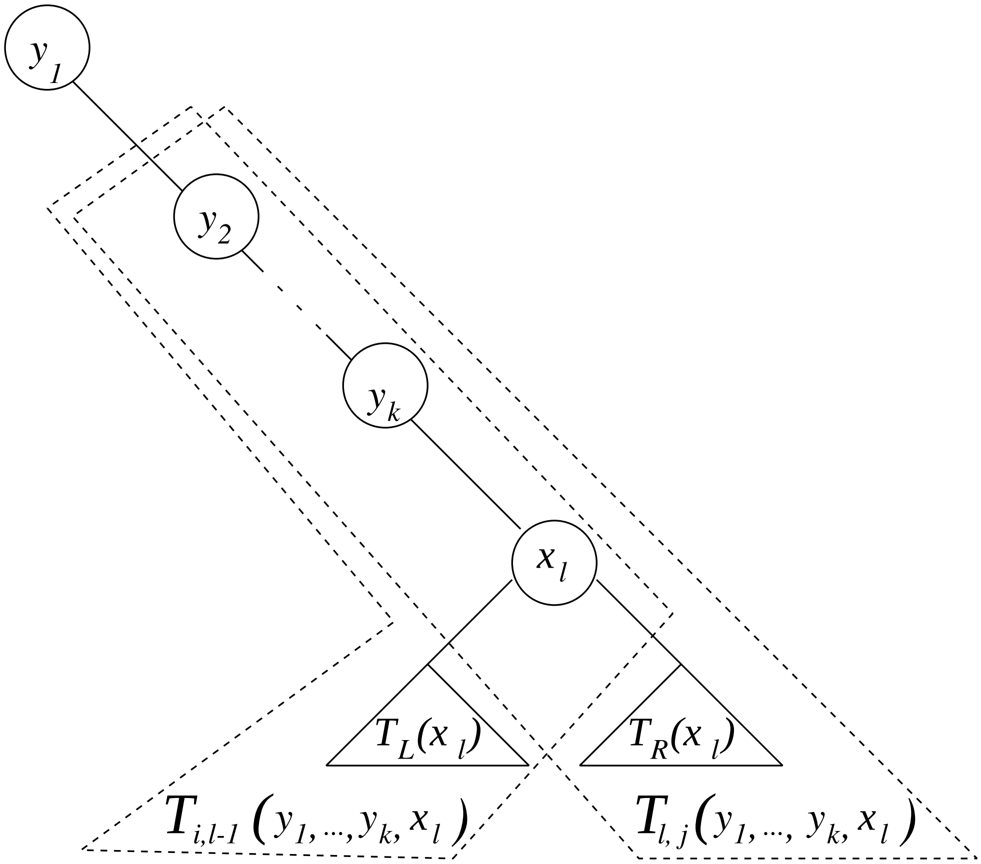


Figure 2: The decomposition of $T_{ij}(y_1, \ldots, y_k)$

The following characterizes legal paths.

Theorem 3: A path is legal if and only if it is a min-max ordering.

Proof: Let $y_1, \ldots, y_m$ be a legal path. Then there exists a binary search tree T, such that $y_1, \ldots, y_m$ is a path of T. If it is not a min-max ordering there exists a key y_i which is neither the minimal nor the maximal of $\{y_i, y_{i+1}, \ldots, y_m\}$, $i \leq m-2$. If y_{i+1} is a left child in T then $y_i > y_{i+1}, \ldots, y_m$, impliying that y_i is a max key. Similarly, y_{i+1} can not be a right child, because it would imply that y_i is a min key. The contradiction implies that $y_1, \ldots, y_m$ is a min-max ordering.

Conversely, let $y_1, \ldots, y_m$ be a min-max ordering. We construct a binary tree T such that $y_1, \ldots, y_m$ is a path of it. For each i, $1 < i \le m$, let y_i be either the left or right child of y_{i-1} in T, according to whether y_i is a min or max key, respectively. It follows that T is a binary search tree. Consequently, $y_1, \ldots, y_m$ is a legal path. □

The following ordering is also of interest.

For $Y \subset X$ and $0 \le i < j \le n+k$, an ordering Y' of Y is called (i,j)-**inc-dec** when

- $Y \subset X_i^- \cup X_j^+$,
- the keys of $Y \cap X_i^-$ are in increasing ordering in Y', while those of $Y \cap X_j^+$ are in decreasing ordering.
- $Y \cap X_i^- \neq \emptyset \Longrightarrow x_i \in Y$, and
  $Y \cap X_j^+ \neq \emptyset \Longrightarrow x_{j+1} \in Y$

Lemma 1: A (i,j)-inc-dec ordering is necessarily a min-max ordering.

Proof: Label the keys of $Y \cap X_i^-$ as min, and as max those of $Y \cap X_j^+$. □

The next theorem characterizes (i,j)-legal paths.

Theorem 4: For $i = j$ a path is (i,j)-legal if and only if it is a min-max ordering. For $i < j$, a path is (i,j)-legal if and only if it is a (i,j)-inc-dec ordering.

Proof: When $i = j$ the results follows from Theorem 3. Let $i < j$. By hypothesis, $y_1, \ldots, y_m$ is a (i,j)-legal path. Then there is a binary search tree T, having X as its node set, where $y_1, \ldots, y_m$ is a path of it, y_m the father of some $x_\ell \in X_{ij}$ and the subtree $T(x_\ell)$ contains exactly the keys of X_{ij}. Let $Y = \{y_1, \ldots, y_m\}$. First, clearly $Y \subset X_i^- \cup X_j^+$. Second, suppose there exists a key $y_q \in X_i^- \cap Y$ such that $y_q > y_{q+1}$ for some $1 \le q < m$. Since T is a binary search tree, it follows that y_{q+1} is a key of the left subtree of y_q. Since y_q is an ancestor of y_m, we know that x_ℓ also belongs to this subtree, contradicting $x_\ell > y_q$, implied by $y_q \in X_i^-$. Hence no such q can exist.

Consequently, the keys of $X_i^- \cap Y$ are in decreasing ordering in $y_1, \ldots, y_m$. Similarly, we prove that those of $X_j^+ \cap Y$ form a decreasing ordering. Third, suppose that $X_i^- \cap Y \neq \emptyset$ and $x_i \notin Y$. Denote by y_t the maximal key of $X_i^- \cap Y$. Then $y_t < x_i$. We try to locate key x_i in T. Suppose that x_i is a descendant of y_t. Then x_i belongs to the right subtree R of y_t. Consequently, $T(x_\ell)$ is also in R. If $t = m$ then $x_i \in T(x_\ell)$, a contradiction. When $t < m$ we know that $y_t, \ldots, y_m$ is a path of R. Because T is a binary search tree and the maximality of y_t in X_i^- it follows that $y_{t+1}, \ldots, y_m \in X_j^+$. Consequently, because the keys of $X_j^+ \cap Y$ are in decreasing ordering in $y_1, \ldots, y_m$, we conclude that y_{t+1} is a right child, but $y_{t+2}, \ldots, y_m, x_\ell$ are all left children. Because $x_i < y_{t+1}, \ldots, y_m, x_\ell$ it follows that x_i must belong to $T(x_\ell)$. The latter contradicts again the fact that $T(x_\ell)$ contains exactly X_{ij}. Hence x_i is not a descendant of y_t. Neither can x_i be an ancestor of y_t. Because in this case, y_t belongs to the left subtree L of x_i, implying that $x_\ell > x_i$ belongs to L, a contradiction. The remaining possibility is that x_i is neither a descendant nor an ancestor of y_t. In this case, let z be the nearest common ancestor of x_i and y_t. Denote by L and R the left and right subtrees of z, respectively. If x_i is in L then y_t must be in R, contradicting $y_t < x_i$. The other case is x_i in R and y_t in L, making it impossible the assumption $x_i < x_{i+1}$. Therefore the alternative that x_i is neither a descendant nor an ancestor of y_t can also not occur. Consequently, $X_i^- \cap Y \neq \emptyset$ implies $x_i \in Y$. The proof that $X_j^+ \cap Y \neq \emptyset$ implies $x_{j+1} \in Y$ is similar.

Conversely, suppose that $y_1, \ldots, y_m$ is a (i,j)-inc-dec ordering, $0 \leq i < j \leq n+k$. We construct a tree T' as follows. It contains the path $y_1, \ldots, y_m$. T' also contains a subtree $T'(x_\ell)$, having an arbitrary root $x_\ell \in X_{ij}$, and satisfying the following property: $T'(x_\ell)$ is a binary search tree containing exactly the keys of X_{ij}. Finally, make x_ℓ the left or right child of y_m, according to whether $y_m \in X_j^+$ or $y_m \in X_i^-$, respectively. The construction of T' is completed. Let $Y = \{y_1, \ldots, y_m\}$. Since $y_1, \ldots, y_m$ is a (i,j)-inc-dec ordering, it follows that $Y \cap X_{ij} = \emptyset$. Hence the path $y_1, \ldots, y_m$ and $T(x_\ell)$ are disjoint. The latter assures that T' is indeed a binary tree. We show that it is a binary search tree. Let z_1, z_2 be keys of T', z_1 belonging to the left subtree L of z_2. Consider the possibilities:

Case 1: $z_1, z_2 \in Y$

Since $y_1, \ldots, y_m$ is a (i,j)-inc-dec ordering, by Lemma 1 it is a min-max ordering. By Theorem 3 it must be a legal path. Hence z_1 being in L implies $z_1 < z_2$.

Case 2: $z_1 \in X_{ij}$ and $z_2 \in Y$
Suppose $z_2 = y_m$. Then x_ℓ must be the left child of y_m. By the construction of T', we conclude that $z_2 \in X_j^+$. Hence $z_1 < z_2$. Suppose now $z_2 \neq y_m$. By Case 1, we conclude that $y_m < z_2$. Suppose $y_m \in X_j^+$. Then $z_1 < y_m$, implying $z_1 < z_2$. Alternatively, consider $y_m \in X_i^-$. In this case, if $z_2 \in X_i^-$ then z_2, y_m must appear in increasing ordering, because $y_1, \ldots, y_m$ is a (i,j)-inc-dec ordering. Hence $z_2 < y_m$, a contradiction. Consequently, $z_2 \in X_j^+$. That is, $z_1 < z_2$.

Case 3: $z_1 \in Y$ and $z_2 \in X_{ij}$
This case can not occur, because it implies that z_2 is a descendant of z_1. This contradicts z_1 belonging to L.

Case 4: $z_1, z_2 \in X_{ij}$
Since $T(x_\ell)$ is a binary search tree, z_1 being in L implies $z_1 < z_2$.

From the above cases, we can conclude that z_1 belonging to $T_L(z_2)$ implies that $z_1 < z_2$, for any $z_1, z_2 \in Y \cup X_{ij}$. Similarly, it can be proved that z_1 belonging to $T_R(z_2)$ implies $z_1 > z_2$. Consequently, T' is a binary search tree containing the keys $N(T') = Y \cup X_{ij}$. Let $X' = X \setminus N(T')$. We now include in T' each key of X', as follows. If $Y \cap X_i^- = \emptyset$ and $i > 0$ then include $x_i \in X'$ in T' so as y_1 becomes the right child of x_i. Similarly, if $Y \cap X_j^+ = \emptyset$ and $j < n + k$ then $x_{j+1} \in X'$ is included in T' in such a way that y_1 is the left child of x_{j+1}. Note that the above two conditions can not occur simultaneouly. Next, for each key of X' not yet included in the tree, include it according to the rules of binary search tree insertion. Let T be the final tree so obtained. Since T' is a binary search tree, T is so. Also, T' is a partial subtree of T. Clearly $N(T) = X$ and $y_1, \ldots, y_k$ is a path of T'. In order to show that $y_1, \ldots, y_m$ is (i,j)-legal, it remains to prove that $T(x_\ell) = T'(x_\ell)$. Suppose the contrary. Then $T(x_\ell)$ contains some key $z \in X'$. Suppose $z \in X_i^-$. The following alternatives exist.

Case 1: $Y \cap X_i^- \neq \emptyset$
By the definition of (i,j)-inc-dec ordering, it follows that $x_i \in Y$. That is, x_i is a proper ancestor of x_ℓ in T. Hence, $z \neq x_i$. Since x_i is the maximal

key of X_i^-, it follows $z < x_i$. Then the binary search tree insertion procedure would not include z in the right subtree of x_i. On the other hand, x_ℓ belongs to the right subtree of x_i, as $x_i < x_\ell$. Hence $z \notin N(T(x_\ell))$.

Case 2: $Y \cap X_i^- = \emptyset$
If $i = 0$ then $X_i^- = \emptyset$, contradicting $z \in X_i^-$. When $i > 0$, y_1 is the right child of x_i, by the construction of T. Hence $z < x_i$, implying that the binary search tree insertion again could not include z in $T_R(x_i)$. However $x_\ell \in N(T_R(x_i))$. That is, $z \notin N(T(x_\ell))$.

Consequently, $z \in X_i^-$ implies that z is not in $T(x_\ell)$. Similarly, we prove that $z \in X_j^+$ also implies that z can not be in $T(x_\ell)$. Therefore $T(x_\ell)$ is formed exactly by the keys of X_{ij}. Hence $y_1, \ldots, y_m$ is a (i, j)-legal path, completing the proof of Theorem 4. $\square$

4 The algorithms and analysis

The algorithms for finding optimal worst case and weighted average case binary search trees can now be described.

The input consists of an integer $k > 0$, a set $\{x_1, \ldots, x_n\}$ of keys, $x_q < x_{q+1}$, and a key cost $c(y_1, \ldots, y_t)$ for each legal t-path, $1 \leq t \leq k + 1$. Alternatively, the input can consist of a function which enables to compute the key costs $c(y_1, \ldots, y_t)$, whenever needed. In the latter case we assume that this computation can be done in constant time. In addition, in the weighted average case problem each key x_q is also given a non-negative weight $w(x_q)$.

The algorithms start by defining the dummy keys $\{x_{n+1}, \ldots, x_{n+k}\}$. Using $(2) - (4)$, compute the key costs $c(y_1, \ldots, y_t)$, for each legal t-path $y_1, \ldots y_t$ with at least one dummy key, $1 \leq t \leq k+1$. Define $w(x_q) = 1$ for each $n+1 \leq q \leq n+k$. For each (i, j)-legal t-path $y_1, \ldots, y_t$ and $0 \leq i \leq j \leq n$, compute $C_{ij}(y_1, \ldots, y_t)$ by $(5) - (6)$ and $(7) - (8)$, respectively for the worst case and weighted average case problems. The final solution is $C_{0n}(x_{n+k}, \ldots, x_{n+1})$.

Below, we compute the exact number of steps performed by the worst case minimization algorithm.

To count the total amount of work to do, consider that each different subinterval (i, j) of the array reached through a different legal path must be processed. To process such interval, we must consider all its positions from i to j, and compute the worst-case or expected-case cost at each position. To compute such cost, we need the cost of some subintervals. Given that those subintervals are already computed, we work $O(|j-i|)$ to solve the subinterval (i, j) given a previous access history of length k. Hence, what we have to compute is the sum of $|j - i|$ for all $i \leq j$ for all access histories of length k which lead to the subinterval (i, j).

Rethink the access history in this way: instead of considering a sequence of y_q min-max values, consider that the interval to work on, initially $[1, n]$, is reduced k times, by either incrementing its left limit (min value) or decrementing its right limit (max value). Hence, we have a sequence of increments and a sequence of decrements, where the sum of the steps is k. We can identify the access history with the pair of sequences (accounting also for the form in which they are mixed). When the k steps are done, we work in time proportional to the size of the interval left. See Figure 3.

We use generating functions to count the total amount of work.

The function to be used has three variables z, x, w. Let the variable z count the total size of the array (n), x count the total number of accesses (k) and w the total amount of work. Our generating function is thus

$$F(z, x, w) \quad = \sum_{n,k,r \geq 0} W_{n,k,r} z^n x^k w^r$$

such that in an array of n elements there are $W_{n,k,r}$ different histories of k steps which lead to an interval of size r (which costs $O(r)$). Therefore the total amount of work is the coefficient of $z^n x^k$ in the function

$$g(z, x) \quad = \quad \frac{\delta F}{\delta w}(z, x, 1) \quad = \sum_{n,k,r \geq 0} r W_{n,k,r} z^n x^k$$

This is correct, since $r W_{n,k,r}$ is the total amount of work to do on an array of size n and histories of length k.

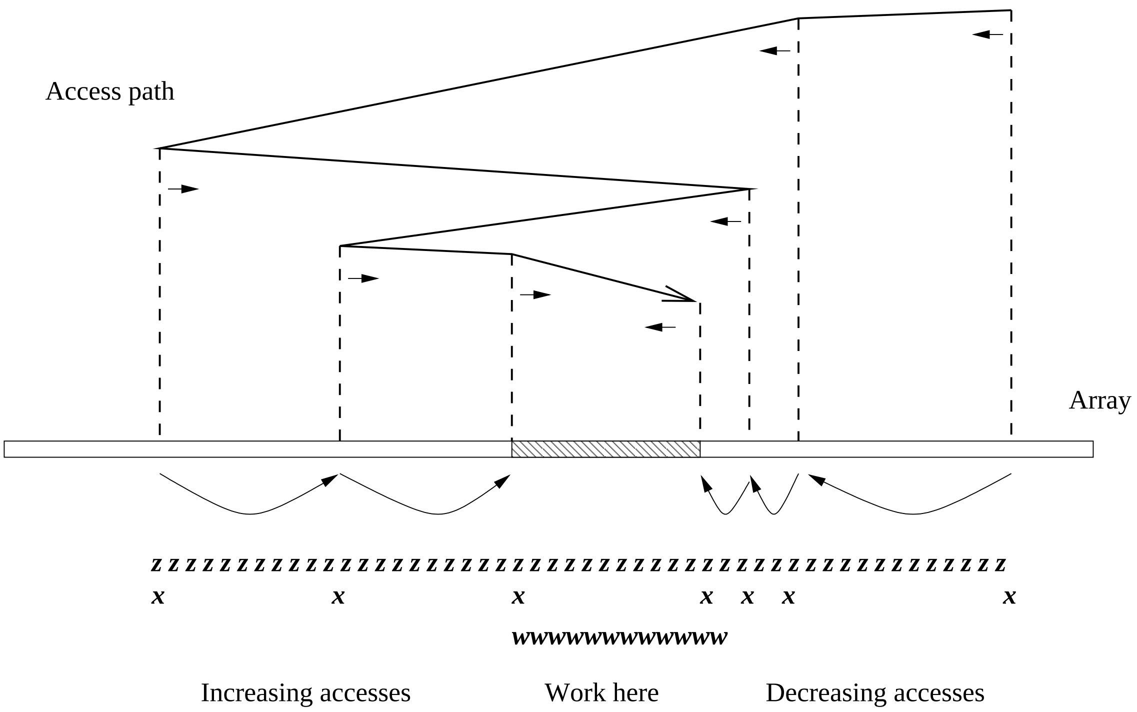


Figure 3: Interpreting legal paths. Variables z, x and w correspond to the quantities to be counted.

There are two important cases here. First, if an interval has increasing and decreasing components, then we do not have to perform a different computation for all the possible original subintervals. For instance, suppose that $n = 100$ and $k = 2$. The access history given by [25,75] for example, that yields the subinterval [26,74] to work on does not depend on the original interval [1,100]. The final subinterval [26,74] would not need to be recomputed. If, on the other hand, both accesses at 25 and 75 are increasing then the final subinterval is [76,100], which certainly depends on the initial interval [1,100]. Hence, we must sum over all access histories with no regard to the initial subintervals, except for those which have only increasing or only decreasing components.

To keep count of the size of the array (in z) and the number of steps (in x) at the same time, we consider the number of elements "skipped" in the consecutive increments (see Figure 3). A single increasing step is represented by the function

$$I(z,x) \quad = \quad \frac{xz}{1-z} \quad = \quad xz + xz^2 + xz^3 + ...$$

that is, one access is performed (x) after skipping over one or more elements of the array (z's). There is at least one element, which is the array element compared. A sequence of zero or more increasing accesses is represented by

$$I^*(z,x) \quad = \quad \frac{1}{1-I(z,x)} \quad = \quad 1 + I(z,x) + I(z,x)^2 + I(z,x)^3 + ...$$

and the same formulas hold for $D(z,x) = I(z,x)$ and $D^*(z,x) = I^*(z,x)$. A sequence of intermingled increasing and decreasing accesses corresponds to

$$ID^*(z,x) = \frac{1}{1-(I(z,x)+D(z,x))}$$

and the final sequence of elements of the set where we have to work is represented by

$$\frac{1}{1-wz} \quad = \quad 1 + wz + w^2z^2 + w^3z^3 + ...$$

(where we count one unit of work in w and one element of the array in z). On the other hand, a sequence of elements where we do not have to work is simply $1/(1-z) = 1 + z + z^2 + ...$

We are now ready to state the general formula. Since we are disregarding the initial and final ends of the array, we represent the sequence of accesses just by $ID^*(z,x)$. However, for the case of only increasing or decreasing elements we have to subtract what we have added and replace it by a formula that allows to consider all the possible initial right extremes (for I) and all possible initial left extremes (for D). In the case of increments (the decrements are similar), this is obtained by subtracting $I^*(z,x)$ from $ID^*(z,x)$ and then adding $I^*(z,x)/(1-z)$, since this allows to add an arbitrary number of z's to the right, accounting for all possible positions of the sequence inside the array.

Finally, after a sequence of increasing and decreasing steps, there is a final central segment on which we work. Since we sum z's along all this process, we have in z the length of the resulting array. We add an x per step so we have in x the number of steps. Finally, we have in w the amount of work. At the end, we select those processes which turn out to have n elements (z^n), and k steps (x^k). The formula is

$$\begin{aligned} F(z,x,w) &= \Big(ID^*(x,y) - I^*(z,x) + I^*(z,x)\frac{1}{1-z} - D^*(z,x) + \\ &\quad + D^*(z,x)\frac{1}{1-z}\Big)\frac{1}{1-wz} \end{aligned}$$

which is equal to

$$F(z,x,w) = \left(\frac{1}{1-\frac{2xz}{1-z}} + \frac{2z/(1-z)}{1-\frac{xz}{1-z}}\right)\frac{1}{1-wz}$$

We derive the above formula with respect to w and evaluate it at $w=1$, to obtain

$$g(z,x) = \frac{z}{(1-z)^2}\left(\frac{1}{1-\frac{2xz}{1-z}} + \frac{2z/(1-z)}{1-\frac{xz}{1-z}}\right)$$

To find the coefficient that corresponds to x^k in $g(z,x)$, notice that the coefficient for $1/(1-ax)$ is a^k. Hence

$$g_k(z) = \frac{z}{(1-z)^2}\left(\frac{2^k z^k}{(1-z)^k} + \frac{2z^{k+1}}{(1-z)^{k+1}}\right)$$

and to obtain the coefficient that corresponds to z^n in $g_k(z)$, notice that the coefficient of $1/(1-z)^{m+1}$ is $\binom{n+m}{m}$, and that the coefficient of z^n in $zf(z)$ is that of z^{n+1} in $f(z)$. Consequently, the total amount of work is exactly

$$g_{k,n} \quad = \quad 2^k \binom{n}{k+1} \; + \; 2 \binom{n}{k+2}$$

which for instance shows that for $k = 1$ the amount of work is $g_{1,n} = n^3/3 - n/3$. To obtain a more easy to handle formula we can simplify the combinatorials and conclude that the cost is

$$g_{k,n} \quad = \quad \left(\frac{2^k n^{k+1}}{(k+1)!} + \frac{2n^{k+2}}{(k+2)!} \right) \; (1 + O(k/n)) \quad \leq \quad n^{k+2}$$

In fact, we should consider also the access paths with less than k elements, since in the initial accesses we do not have the full history. This is obtained by summing up the above values for k from zero to its maximum values. The result is still upper bounded by n^{k+2}.

Notice that we have left aside the case of zero-length sequences, where both ends of the initial subinterval must be considered (not only the rightmost or leftmost). Because of this the analysis does not apply to $k = 0$, which gives

$$\frac{1}{1-z} \, \frac{1}{1-wz} \, \frac{1}{1-z}$$

i.e. $g_{0,n} = n^3/6 + n^2/2 + n/3$.

We consider space now. We have to store one cell for each different access path. To count the number of access paths, we modify the above analysis so that we count now the number of paths in w. This time, the central subinterval left is not represented by $1/(1-wz)$, but by $w/(1-z)$ instead (i.e. we count only one w which represents the interval). We have now

$$F'(z,x,w) \quad = \quad \left(\frac{1}{1 - \frac{2xz}{1-z}} + \frac{2z/(1-z)}{1 - \frac{xz}{1-z}} \right) \frac{w}{1-z}$$

and hence

$$g'(z,x) \quad = \quad \frac{1}{1-z} \, \left(\frac{1}{1 - \frac{2xz}{1-z}} + \frac{2z/(1-z)}{1 - \frac{xz}{1-z}} \right)$$

which gives

$$g'_{k,n} \quad = \quad 2^k\binom{n}{k} \; + \; 2\binom{n}{k+1}$$

and this can be simplified to

$$g'_{k,n} \quad = \quad \left(\frac{2^k n^k}{k!} + \frac{2n^{k+1}}{(k+1)!}\right) \; (1 + O(k/n)) \quad \leq \quad n^{k+1}$$

which again is kept unchanged if we add up also the histories of length less than k.

Hence, the total time complexity is $O(n^{k+2})$ and the space complexity is $O(n^{k+1})$.

The analysis of the weighted average case minimization algorithm is similar. The corresponding time and space complexities are $O(kn^{k+2})$ and $O(n^{k+1})$.

5 Conclusions

We have described algorithms for finding optimal binary search trees for a given set $\{x_1, \ldots, x_n\}$ of keys when the cost of each key x_q depends on the $(k+1)-$path leading to x_q. The parameter k is a given arbitrary integer in the range $1 \leq k < n$. The optimality refers to a tree having either minimal worst case or weighted average case cost. The algorithms are polynomial in the size of the input, although they are exponential in k. Their complexities are $O(n^{k+2})$ and $O(k.n^{k+2})$, respectively for the worst and weighted average cases. It should be noted that these complexities are $O(n)$ and $O(k.n)$ times the input size.

The algorithms make use of additional dummy keys $\{x_{n+1}, \ldots, x_{n+k}\}$, with costs accordingly defined. It is simple to modify the algorithms to avoid computations with dummy keys. An idea is to impose that whenever x_p and x_q are dummy keys and x_p is a proper ancestor of x_q then $p > q$.

The *monotonicity principle* by Knuth [2] made it possible to decrease the number of iterations for constructing an optimal binary search tree from

$O(n^3)$ to $O(n^2)$. Unfortunately, the principle does not hold for $k > 0$, as shown by the following example. Let $\{x_1, \ldots, x_{k+2}\}$ be the given set of keys, all with unit weights and costs defined as:

$$c(x_{k+1}, \ldots, x_1) = c(x_{k+1}, \ldots, x_2) = \ldots = c(x_{k+1}) = 0,$$

$$c(x_1, \ldots, x_k, x_{k+2}) = c(x_1, \ldots, x_k) = \ldots = c(x_1) = 0,$$

$$c(x_2, \ldots, x_k, x_{k+2}, x_{k+1}) = 0,$$

while any other key cost is equal to 1. The solution of both minimization problems for the keys $\{x_1, \ldots, x_{k+1}\}$ is the tree formed by the single path $x_{k+1}, \ldots, x_1$. When adding the key x_{k+2}, the optimal tree for $\{x_1, \ldots, x_{k+2}\}$ is the path $x_1, \ldots, x_k, x_{k+2}, x_{k+1}$, meaning that the principle does not apply for $k > 0$. In fact, it does not hold also for $k = 0$ and non uniform key costs.

Finally, we mention that gap weights can be introduced following the techniques introduced in [2].

References

[1] E. N. Gilbert and E. F. Moore, Variable-lenght binary encoding, *Bell System Tech. J.* **38** (1959), pp. 933-968.

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

[3] D. E. Knuth, *The Art of Computer Programming 1: Fundamental Algorithm*, Addison-Wesley, Reading, MA, 1968, 2nd ed. 1973.

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

[5] S. V. Nagaraj, Optimal binary search trees, *Theoretical Computer Science* **188** (1997), pp. 1-44.