Assembling Approximately Optimal Binary Search Trees Efficiently Using Arithmetics

Jussi Kujala

Institute of Software Systems, Tampere University of Technology, Finland

Abstract

We introduce a new algorithm for computing an approximately optimal binary search tree with known access probabilities or weights on items. The algorithm is simple to implement and it has two contributions. First, a randomized variant of the algorithm produces a binary search tree with expected performance that improves the previous theoretical guarantees (the performance is dependent on the value of the input random variable). More precisely, if p is the probability of accessing an item, then under expectation the item is found after searching $\lg 1/p + 0.087 + \lg_2(1 + p_{\max})$ nodes, where $p_{\max}$ is the maximal probability among items. The previous best bound was $\lg 1/p + 1$, albeit deterministic. For the optimal tree our upper bound implies a non-constructive performance bound of $H + 0.087 + \lg_2(1 + p_{\max})$, where H is the entropy on the item distribution and the previous bound was $H + 1$. The second contribution of the algorithm is a low cost in I/O models of cost such as the cache-oblivious model, while attaining simultaneously the above bound for the produced tree.

Key words: Data structures, binary search trees, entropy bounds, I/O models, approximation algorithms

1 Introduction

Binary search trees (BSTs) are one of the fundamental data structures in computing. For ordered items, they enable efficient support for such operations as SEARCH, INSERT, DELETE, SUCCESSOR, and PREDECESSOR. More information on BSTs, their operations, and implementation details can be found, e.g., in standard textbooks [5,9].

Email address: jussi.kujala@tut.fi (Jussi Kujala).

Items stored in a BST do not necessarily have an equal status. For example, searches for English words have different frequencies. Hence, it would be desirable to have a BST that stores items with given probabilities or weights such that the *expected cost* of an access is minimal. Such a BST is called an *optimal* BST. Computing an optimal BST is a classic problem for which Knuth [9] gave a well-known algorithm. For n items, the time and space used by this algorithm are both $\Theta(n^2)$. The literature on the subject is vast and a number of $O(n)$ or $O(n \lg n)$ time and space *approximation* algorithms have been suggested [9,6,11,13] with varying assumptions on input and output.

We now give definitions used in this paper. As an input we have ordered items (without loss of generality we set them to $\{1, \ldots, n\}$) and associated probabilities that we denote by p_i for the item i. Additionally we can have an access to an interval between items i and $i+1$. We denote the probability of this event by q_i, and q_0 and q_n are respectively the probabilities of searching for a smaller and larger item than what is stored in the BST. The cost of accessing an item or interval equals its depth in the BST. The depth of an item is the number of items that are accessed while searching for the item and hence the root is at depth one. The depth of an interval is the depth of the parent of the interval.

Most of the approximation algorithms are based on bounding the depth of an item i with $\lg 1/p_i + c$, where lg is a binary logarithm, and c is a constant. For example, Mehlhorn [11] gives an algorithm in which $c = 1$ for items and $c = 2$ for intervals. The bound $\lg 1/p_i + c$ is sufficient for approximating the cost of an optimal BST, because the difference between the expected cost of an optimal BST and entropy $H = \sum_{i=1}^{n} p_i \lg 1/p_i + \sum_{i=0}^{n} q_i \lg 1/q_i$ is relatively small. For example, Alon and Orlitsky [1] proved that the expected cost of the optimal BST (with only internal items) is at least $H - \lg(H+1) - \lg e + 1$.

In Section 2 we introduce an approximation algorithm, AWOBST (arithmetic weight-optimal BST). This algorithm has two contributions.

- Recently the I/O -performance of algorithms has received increased attention, for example resulting in the *cache-oblivious cost model* [7]. In short, in the cache-oblivious cost model we have a memory with a cache which contains lines of size B words each, and we do not know B or the size of the cache. Accessing the cache is free, but if a cache miss occurs then we pay a unit cost for fetching a block of B words from the memory to the cache. The objective of a cache-oblivious algorithm is to minimize the cost in this model. We use the cache-oblivious model, because an algorithm that takes into account the locality of memory accesses should perform well in a problem that is memory intensive.

  Previous algorithms for computing a weighted BST have not been analysed in an I/O-model; usually the items are processed from the root towards the

leaves, which implies that the I/O cost is not low. There is one exception we are aware of: the cache-oblivious algorithm of Brodal and Fagerberg [3] attains the bound of $2(\lceil \lg 1/p_i \rceil + 1)$ for the depth of the item i. However, this is far from the bound of $\lg 1/p_i + 1$ that other algorithms attain, and in data structures such difference is significant. We analyse AWOBST in the cache-oblivious model and show that it causes $O(n/B)$ cache misses, while it also simultaneously attains the $\lg 1/p_i + 1$ bound. Additionally, the cost is $O(n)$ in the traditional unit cost model.

- Mehlhorn has shown that if only H and $\sum_{i=0}^{n} q_i$ are known then the *best possible* upper bound for the expected cost of an optimal BST is $H + 1 + \sum_{i=0}^{n} q_i$ [11]. There is a non-constant gap between the lower bound in [1] and this upper bound, because H and $\sum_{i=0}^{n} q_i$ do not completely characterize the cost that an optimal BST must pay. However, for the special case of the alphabetic search tree, where $\sum_{i=0}^{n} q_i = 1$, there are bounds that depend on other information, for example $H + 2 - q_0 - q_n$ [13], but for the general problem there are no such bounds. The closest bound is for a related problem where the items are not necessarily stored in key order, i.e., they are words of a code that is not prefix-free. This bound is $H + 1 - p_{\max} \lg 1/p_{\max}$ [2], where $p_{\max} \leq 0.5$ is a maximal probability on items.

  We give an improved bound for the general problem; a randomized variant of AWOBST has an expected (taken over randomness in the algorithm) upper bound $H + 0.087 + \lg(1 + (p+q)_{\max}) + \sum_{i=0}^{n} q_i$ for the cost. Here $(p+q)_{\max}$ is the maximal probability over pairs of an adjacent item and an interval, i.e., maximum over pairs of either $q_{i-1} + p_i$ or $p_i + q_i$. Our bound essentially changes the 1 in Mehlhorn's bound to $0.087 + \lg(1 + (p+q)_{\max})$. For the special case when $(p+q)_{\max} < 2/3$ we also give a better bound $H + 0.503$. If $\sum_{i=1}^{n} p_i = 1$ the bound is $H + 0.087 + \lg(1 + p_{\max})$, and for $\sum_{i=0}^{n} q_i = 1$ the bound is $H + 1.087 + \lg(1 + q_{\max})$. Note that these bounds are deterministic bounds on the optimal BST.

The structure of the BST produced by AWOBST is different from the other algorithms cited, although there are very close similarities between the rule that AWOBST uses and what the others use, for example the ones in [11,10]. Most closely the structure of the BST bears a resemblance to the code words assembled in the arithmetic coding [12]. The algorithm itself is quite simple and we have implemented it on integer weights (rational weights reduce to the integer weights).

In the rest of this paper we consider a situation where the probabilities q_i are zero, i.e., no accesses to intervals. We make this assumption only to simplify our presentation and it does not restrict the problem we solve, because we can do a reduction. In the reduction we redistribute the probability mass on the leaves to probabilities of the internal (possibly dummy) items and the BST is constructed with these probabilities. More precisely, assign modified probabilities $p'_i = p_i + q_{i-1}/2 + q_i/2$ to internal items and construct a BST on

Table 1
Computing an arithmetic weight-optimal BST (AWOBST)

Input: a sorted array of items $\{1, \ldots, n\}$, each having a probability or "weight" $p_i > 0$.
Output: a BST in which each item i is assigned a depth of $\lg 1/p_i + 1$ or less. Recall that the root is at depth one.
Assumptions: the algorithm has an auxiliary stack which contains roots of temporary BSTs and is initially empty. By stack[top] we refer to the top of the stack and the item below it by stack[top-1] (read only access). A depth-restriction constant is attached to each item in the stack, and ".d" is used to refer to this; e.g., the depth-restriction constant for the item at the top of the stack is stack[top].d. The function FIRST_DIFF_BIT returns the position of the first differing bit in its arguments (from the more significant part and with position one being the first fractional bit). An operation automatically fails, if it does not have enough operands. Nodes remain in the memory allocated from the stack until explicitly stated otherwise.

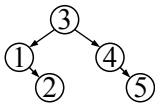
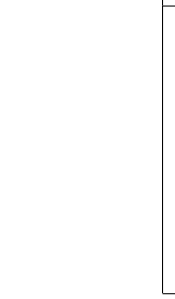
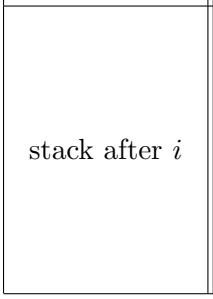
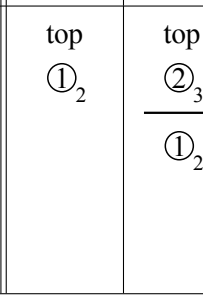
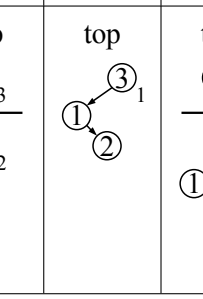
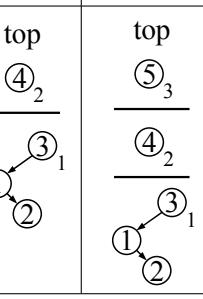
(1) Initialize $S := 0$.

(2) **For** each item i in order of key values:
   (a) **If** i is the last item **then** set $d_i :=$ FIRST_DIFF_BIT$(S, 0.111\ldots)$,
   else set $d_i :=$ FIRST_DIFF_BIT$(S, S + p_i)$.
   (b) **While** $d_i <$ stack[top-1].d $<$ stack[top].d **do**
   pop nodes $T_1 :=$ stack[top-1] and $T_2 :=$ stack[top],
   place the node T_2 to memory,
   link T_2 as the right child of T_1, and
   push T_1 to the stack.
   (c) **If** $d_i <$ stack[top].d **then**
   pop node $T :=$ stack[top],
   place the node T to memory, and
   link T as a left child of item i.
   (d) Push the item i with d_i to the stack.
   (e) Set $S := S + p_i$.

(3) Link all trees still in the stack, i.e., set the right child of each BST, not including the top of the stack, to the BST that is one step closer to the top of the stack.

the internal items according to these modified probabilities. Observe that an interval is a child of an adjacent internal item and so its cost is the maximum of their depths. Also, the maximum of the log-probabilities of the adjacent internal items is bounded: $\max\left(\lg 1/p'_i, \lg 1/p'_{i+1}\right) \leq \lg 1/q_i + 1$. The bounds given in this introduction are obtained from this reduction.

Table 2
An illustration of a building process with five items, the probabilities are given in binary. Note that the stack contains only the roots of the corresponding BSTs, although for clarity we give full BSTs. The depth-restriction constant for each tree in the stack is given in the sub-script to the root node.

item i	1	2	3	4	5	final tree
probability p_i	0.01	0.001	0.001	0.01	0.01	
$\sum_{j=1}^{i} p_j$	0.01	0.011	0.100	0.11	0.111 ...	
d_i	2	3	1	2	3	
stack after i						

2 Description of the Algorithm

The algorithm AWOBST is given in Table 1. We now explain its basic idea. Let $p_1, \ldots, p_n > 0$ be the probabilities on items $1, \ldots, n$. Assume without loss of generality that the key value equals the item number. Let $S_0 = 0$ and let $S_i = S_{i-1} + p_i$. Assume that $S_n = 0.1111\ldots$ in binary. Give an item i a priority that is the first bit where S_i and S_{i-1} differ and define that the first fractional bit is given a priority of one. Create a BST that is in heap order according to the priorities on the items.

AWOBST efficiently performs this in a way resembling the construction of the Cartesian tree in [8]. Moreover, each item is assigned to a depth less or equal to the given priority. Note that the depth of the root is one. For example, if $S_{i-1} = 0.10011$ in binary, and $p_i = 0.00110$, then $S_i = S_{i-1} + p_i = 0.11001$ and hence the item i is placed to a depth of two or one in the final BST. We also give an example of the building process in Table 2. Let us discuss how the item 3 is processed. The depth-restriction constant for the item 3 is 1, which is less than the one for the item 2 at the top of the stack. Hence, we know that the item 2 should be in the subtree rooted at the item 3. Before we link the item 2 to the item 3, we must make sure that all items in the left subtree of the item 3 will be linked to it. Thus, we go through the stack and notice that the item 2 should be linked to the item 1 before linking them to the item 3.

3 Analysis of AWOBST

This section contains an analysis of AWOBST. The analysis is divided to three steps. First we show that for depth-restriction constants d_i it holds that $d_i \leq \lg 1/p_i + 1$. Then we show that AWOBST is correct. Finally, we demonstrate that in theory a simple random modification to AWOBST gives an expected bound to depth-restriction constants that is $\lg 1/p_i + 0.087 + \lg(1 + p_{\max})$.

3.1 Depth-restriction Constants are Logarithmic

In the description of AWOBST, each item i is assigned a *depth-restriction* constant d_i. This d_i is the position of the most significant changing bit between $\sum_{j=1}^{i-1} p_j$ and $\sum_{j=1}^{i} p_j$. These two sums differ by p_i and hence,

$$\frac{p_i}{2} \leq 2^{-d_i} \iff d_i \leq \lg \frac{1}{p_i} + 1,$$

because d_i is at least the position of the first set bit of p_i.

3.2 AWOBST Produces a BST with Depth-restriction Constants

We give the following bound:

Theorem 1. AWOBST places an item i with probability p_i to a depth of at most $d_i \leq \lg 1/p_i + 1$.

Proof. The claim follows from the invariant B below, because the root of the final BST has a depth restriction constant d_r of one.

- A Each item i in the stack has a smaller depth-restriction constant d_i than the one closer to the top of the stack.
- B Each item i in any BST rooted at a node in the stack is at any time during the execution found in its BST at a depth less or equal to $d_i - d_r + 1$, where r is the root of that BST.

Proof of invariant A. We use an induction on the items. Empty stack satisfies the invariant. Assume an induction hypothesis that the current stack conforms to the claim. Let there be an item i which causes the stack to violate the induction assumption. Although the while loop can modify the stack, it can not violate the induction hypothesis, because of the condition of the while loop. So when we push the violating item i to the stack during the step (d),

there is a BST with a root k in the stack such that $d_k = d_i$; depth-restriction constants larger than d_i are not a problem in the stack, because the while loop and the if clause during steps (b) and (c) take care of them and values smaller than d_i do not violate the induction hypothesis.

We observe the following fact: if for items i and k it holds that $d_i = d_k$ then there is an item j such that $k < j < i$ and $d_j < d_i = d_k$. This is true because for each two changes in the same bit in the sum of probabilities, a more significant bit must also change. Hence, an item j is in the stack between items i and k, because items are handled in key order, and thus there is a contradiction of the induction hypothesis which stated that depth-restriction constants are larger near the top of the stack.

Proof of invariant B. We do a similar induction as with the invariant A. Assume a stack that is conforming. Using the invariant A it is easy to see that each link operation on BSTs, whether in the while loop or in the if clause, maintain the induction hypothesis. More precisely, each increase in the depth of any item is countered by a smaller depth restriction of the new root. □

3.3 Randomized Analysis Gives a Better Bound

Careful examination of the proof leading to Theorem 1 reveals that the bound $\lg 1/p_i + 1$ is tight only in a malign situation. The depth is at most the first changing bit between binary numbers $\sum_{j=1}^{i-1} p_j$ and $\sum_{j=1}^{i} p_j$ and hence the worst case is attained when the first bit is as far from $\lg 1/p_i$ as possible. This happens when the binary number p_i is of form $0.0\ldots01111\ldots1$, which intuitively is not very common so we should get a better bound. We formalize this intuition in the following theorem, where we randomize bits of p_is by adding two random dummy items 0 and $n+1$.

Theorem 2. Choose a positive integer m such that $p_{\max} < 2r$ for $r = 1/(2^m - 1)$. Use a dummy item 0 and set p_0 to a uniform random number on $[0, r)$. Let another dummy item $n+1$ have a probability $p_{n+1} = r - p_0$. Then set p'_i to $p_i/(1+r)$ so that the values p'_i sum to one and form a probability distribution. If AWOBST is given an input with probabilities p'_i then the *expected* depth of any item i is limited by $\lg 1/p_i + 0.087 + \lg(1+r)$.

Proof. From the definition of the constant r it follows that p'_0 is uniformly distributed in $[0, 2^{-m})$ and the binary form of p'_0 is

$$p'_0 = 0.\underbrace{0\ldots0}_{m} r_1 r_2 \ldots,$$

where r_j are uniform random bits. The binary representation $0.b'_1 b'_2 \ldots$ of

$\sum_{j=0}^{i-1} p'_j$ has random bits at positions following the first set bit of p'_i. This follows from the fact that each such bit b'_l is of the form $(b_l \textbf{ xor } r_{l-m} \textbf{ xor } c_l)$, where b_l is the corresponding bit in the sum $\sum_{j=1}^{i-1} p'_j$, r_{l-m} is a random bit from p'_0, and c_l is the carry bit of the summation which is independently random from r_{l-m}. Note that $m < l$, because we assumed that $p_i < 2r$, so the $p'_{\max} < 2r/(1+r) = 1/2^{m-1}$ is upper bounded:

$$p_{\max} \leq 0.\underbrace{0\ldots01}_{m}11111\ldots.$$

Therefore, we can write $\sum_{j=0}^{i-1} p'_j$ in the following binary form:

$$\sum_{j=0}^{i-1} p'_j = 0.b'_1b'_2\cdots = 0.\underbrace{z_1\ldots z_{k-1}}_{\text{arbitrary bits}} y_1y_2\cdots \geq 2^{-k}\ y,$$

where $y = 0.y_1y_2\ldots$ is a uniform random number in $[0,1)$. Also, write p'_i in binary as

$$p'_i = 0.\underbrace{0\ldots0}_{k-1}1x_1x_2x_3\cdots = 2^{-k}\ x,$$

where x is a binary number $1.x_1x_2x_3\ldots$ in $[1,2)$ and x_j are arbitrary bits. We can now upper bound the position of the first changing bit between $\sum_{j=0}^{i-1} p'_j$ and $\sum_{j=0}^{i} p'_j$ with the position of the first set bit of the binary number $2^{-k}\ y+2^{-k}\ x$. This is because the arbitrary bits from z_1 to z_{k-1} can only decrease the position of the first changing bit.

Subtracting $\lg 1/p'_i$ from the position k of the first set bit in p'_i yields $\lg x$, i.e., $k = \lg 1/p'_i + \lg x$. If $x+y \geq 2$ then the carry bit makes the position of the first set bit in $2^{-k}\ y + 2^{-k}\ x$ better than k and we have a gain of at least $\lg x - 1$ in comparison to $\lg 1/p_i$. Otherwise we have a regret of $\lg x$. Let $p(x+y<2)$ be the probability that $x+y<2$. We observe that $p(x+y<2) = 2-x$, because $x \in [1,2)$ and y was random in $[0,1)$. Hence the expected regret with respect to $\lg 1/p'_i$ is:

$$p(x+y<2)\lg x + p(x+y\geq 2)(\lg x - 1) = 1 - x + \lg x \leq \lg\left(\frac{2}{e\lg e}\right) < 0.087.$$

In addition, we had a rescaling of probabilities by $1/(1+r)$, which implies

$$\lg\frac{1}{p'_i} = \lg\frac{1+r}{p_i} = \lg\frac{1}{p_i} + \lg(1+r).$$

The claim follows. □

For $r = 1/3$ we obtain the following corollary:

Corollary 3. Theorem 2 applied to $r = 1/3$ produces a BST with expected depth of $\lg 1/p_i + 0.503$ or less for the item i if $p_{\max} < 2/3$.

An item with probability more than 2/3 should always be placed to the root, which we can prove by first assuming the opposite and then deriving a contradiction. Hence, the restriction $p_{\max} < 2/3$ matters only little when comparing performances of BSTs, because we know the root of the optimal BST and new bounds can be derived on the sub-trees.

Although Theorem 2 and Corollary 3 do not give a *deterministic* bound, due to linearity of expectation, they apply to some variant, and thus to the optimal BST (i.e., that obtained using Knuth's method). Hence, we obtain the following corollary.

Corollary 4. The cost of an optimal BST is upper bounded by $H + 0.087 + \lg(1 + p_{\max})$, where H is entropy of the item probabilities.

4 Resources

In I/O-models AWOBST touches each item in the input once in the key order and also uses a stack. The number of **push** and **pop** operations is limited by the number of edges in the final BST, which is $n-1$. Additionally, the nodes of the resulting BST are placed to the memory when their parent is set and this modifies information only in the parent which is a temporary variable. Hence there are $O(n/B)$ cache operations if the cache contains at least five lines with length B, one for the input, two for the stack, one for the output, and one for the temporary variables. Of course, the input must be in a format where we can efficiently iterate the items, such as an array. We do not consider how to make the constructed BST perform well in the cache-oblivious model, as this is a separate problem, but note that if necessary we can achieve this using the standard time-forward processing technique [4].

An argument for $O(n)$ cost in the unit cost model follows similarly after noting that the FIRST_DIFF_BIT (a, b) for integers is the first set bit in a **xor** b. In theory the first set bit can be computed as a unit cost operation and in practice many hardware platforms (for example x86 and ARM) implement it as an instruction for integers. FIRST_DIFF_BIT for floating point values reduces to the integer version of the problem.

5 Conclusions

We gave an algorithm which produces a binary search tree with guarantees of $1 + \lg 1/(\text{probability})$ for the depth of any item with known probability. Furthermore, the bound can be expected to be better for a typical input

which implies that the entropy is a tighter measure for the cost of an optimal BST than previously thought. Running time of AWOBST is linear in unit cost model and it causes few I/O-operations.

References

[1] N. Alon, A. Orlitsky, A lower bound on the expected length of one-to-one codes, IEEE Transactions on Information Theory 40 (5) (1994) 1670–1672.

[2] C. Blundo, R. D. Prisco, New bounds on the expected length of one-to-one codes, IEEE Transactions on Information Theory 42 (1) (1996) 246–250.

[3] G. S. Brodal, R. Fagerberg, Cache-oblivious string dictionaries, in: SODA, ACM Press, New York, NY, USA, 2006.

[4] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, J. S. Vitter, External-memory graph algorithms, in: SODA, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd ed., MIT Press, Cambridge, MA, USA, 2001.

[6] M. L. Fredman, Two applications of a probabilistic search technique: Sorting x+y and building balanced search trees, in: STOC, ACM Press, New York, NY, USA, 1975.

[7] M. Frigo, C. E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, in: FOCS, IEEE Computer Society, Los Alamitos, CA, USA, 1999.

[8] H. N. Gabow, J. L. Bentley, R. E. Tarjan, Scaling and related techniques for geometry problems, in: STOC, ACM Press, New York, NY, USA, 1984.

[9] D. E. Knuth, The Art of Computer Programming, vol. 3, 2nd ed., Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1978, 426–454.

[10] K. Mehlhorn, Nearly optimal binary search trees, Acta Informatica 5 (1971) 287–296.

[11] K. Mehlhorn, A best possible bound for the weighted path length of binary search trees., SIAM Journal on Computing 6 (2) (1977) 235–239.

[12] J. Rissanen, G. Langdon, Arithmetic coding, IBM Journal of Research and Development 23 (1979) 149–162.

[13] R. Yeung, Alphabetic codes revisited, IEEE Transactions on Information Theory 37 (1991) 564–572.