ACADEMIC PRESS

Journal of Algorithms 44 (2002) 338–358

Journal of Algorithms

www.academicpress.com

Thresholds and optimal binary comparison search trees ☆

Richard Anderson,[a,*] Sampath Kannan,[b,1] Howard Karloff,[c,d,2] and Richard E. Ladner [a,3]

[a] *Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195, USA*

[b] *AT&T Labs Research and Department of CIS, University of Pennsylvania, Philadelphia, PA 19104, USA*

[c] *AT&T Labs–Research, Room C231, Florham Park, NJ 07932, USA*

[d] *College of Computing, Georgia Institute of Technology, 801 Atlantic Dr., Atlanta, GA 30332-0280, USA*

Received 17 July 2001

Abstract

We present an $O(n^4)$-time algorithm for the following problem: Given a set of items with known access frequencies, find the optimal binary search tree under the realistic assumption that each comparison can only result in a two-way decision: either an equality comparison or a less-than comparisons. This improves the best known result of $O(n^5)$ time, which is based on split tree algorithms. Our algorithm relies on establishing thresholds on the frequency of an item that can occur as an equality comparison at the root of an optimal tree.

☆ A preliminary abbreviated version of this paper appeared in FSTTCS 2001 [1].

* Corresponding author.

E-mail addresses: anderson@cs.washington.edu (R. Anderson), kannan@cis.upenn.edu (S. Kannan), howard@research.att.com (H. Karloff), ladner@cs.washington.edu (R.E. Ladner).

1 Research supported in part by ONR grant N00014-97-1-0505 and NSF grant CCR-9820885.

2 Research supported in part by NSF grant CCR-9732746.

3 Research performed in part while at AT&T Labs–Research, Florham Park, NJ 07932, and supported in part by NSF grant CCR-9732828.

0196-6774/02/$ – see front matter
PII: S0196-6774(02)00203-1

1. Introduction

The *binary search tree* (*BST*) is one of the classic data structures in computer science. One of the fundamental problems in this area is how to build an optimal binary search tree where the items stored in the tree have some observed frequencies of access. In addition, there may be failure frequencies for unsuccessful searches. In the traditional problem each node in the binary search tree is labeled with an item a. In a search for an item x, when the node labeled with a is encountered there are three possible outcomes: $x < a$, $x = a$, and $x > a$. In the first case the search proceeds to the left subtree, in the second case the search ends at the node, and in the third case the search proceeds to the right subtree. A disadvantage of this three-way branching is that it takes two computer instructions to determine the outcome. Knuth [7, Problem 33, p. 457] suggests that it would be interesting to explore an alternative binary tree structure where nodes are labeled with either an equality comparison or a less-than comparison. The resulting tree has two-way branching instead of three-way branching. We call such trees *binary comparison search trees* or *BCST*s for short.

A very simple example demonstrates the benefit of having equality comparisons. Consider the example of three items with weights $(0, 1, 0)$. The optimum with equality has cost 1 while the optimum without equality has cost 2. (We can replace the 0's by arbitrarily small ϵ's to make this example less pathological.) Thus it is interesting to design an algorithm for finding the optimal tree when both types of comparisons are allowed.

Several years ago, Spuler [10] exhibited an $O(n^5)$-time algorithm to find the optimal BCST. His algorithm was based on earlier algorithms to find an optimal split tree [4,6,8]. (One node in a split tree [9] has one less-than comparison and one equality comparison, possibly on different items, leading to 3-way branching.)

In this paper we revisit the problem of finding the optimal BCST in the case of just success frequencies. Using a new approach we show that the optimal BCST can be computed in time $O(n^4)$. The algorithm's correctness depends on our result that if all the frequencies are less than one fourth of the sum of the frequencies then the root of an optimal BCST cannot be an equality comparison.

1.1. Practical motivation

One motivation for our study comes from an increased interest in high performance method dispatching that is required for object-oriented programs. Chambers and Chen [2] describe a *dispatch tree*, with both equality and less-than comparisons, to efficiently look up methods in object-oriented programs. A method is always found so there is no chance of failure. Chambers and Chen employ an interesting and effective heuristic to find a good dispatch tree. They left open the question of how to find an optimal dispatch tree. Dispatch trees are

actually slightly more general than our BCSTs. We focus on the more restricted problem of finding an optimal BCST.

1.2. Traditional binary search trees

There are several efficient algorithms for finding optimal BSTs. A standard $O(n^3)$-time dynamic program can be used, but the time can be improved to $O(n^2)$ by using a clever technique [7]. In the case where the success probabilities are all 0, the running time can be reduced to $O(n \log n)$ by one of two algorithms [3,5]. Note that these latter two algorithms effectively employ only two-way comparisons and hence are "reasonable" from our point of view.

It is interesting to examine the reason why the problem of computing the optimal BCST appears to be more difficult than computing the optimal BST. In a BCST, after a few equality comparisons are done, the resulting subproblem, with its "holes" due to failed equality comparisons, corresponds to anything but an interval $\{i, i+1, i+2, \ldots, j\}$ of indices. The traditional dynamic-programming algorithm relies on the fact that the only subproblems that arise are intervals $\{i, i+1, i+2, \ldots, j\}$; one calculates the cost of an optimal tree for all intervals using dynamic programming. *A priori*, with equality comparisons, one could recursively generate a subproblem corresponding to an arbitrary subset of $\{1, 2, \ldots, n\}$, and there are too many of them to give a polynomial-time algorithm. We will demonstrate that the number of subproblems needed to find the optimal BCST is $O(n^3)$.

However, we show that there is always a tree having only less-than comparisons which has cost at most one more than the optimum when both types of comparisons are allowed.

2. Preliminaries

An input is specified by a sorted sequence $(a_1, a_2, \ldots, a_n)$ with corresponding nonnegative weights $(w_1, w_2, \ldots, w_n)$. A solution is a binary comparison search tree (BCST) which is a binary tree with nodes labeled with either an *equality comparison* of the form $x = a_i$? or a *less-than comparison* of the form $x < a_i$?. For both types of comparisons the two outcomes are "yes" and "no" and we will assume canonically that the left branch of the node representing the comparison corresponds to the "yes" answer. For any BCST T, there is a bijection between the leaves of the tree and the input items. For a tree T, we let $\mathcal{L}(T)$ denote the leaf set of T.

Definition 1. The *weight of a node* v (denoted $w(v)$) in a BCST T is defined as follows. If v is a leaf then $w(v)$ is the weight of the input item labeling v. Otherwise, the *weight of a node* is the sum of the weights of its children. In

other words, the weight of v is the sum of the weights of all leaves which are descendants of v. Similarly, define the *weight of a tree* to be the sum of the weights of its leaves.

Definition 2. The *cost* of a BCST T is defined to be

$$c(T) = \sum_{\ell \in \mathcal{L}(T)} w(\ell)\text{depth}(\ell),$$

where the *depth* of a node is the length of the path from the root to that node. Equivalently

$$c(T) = \sum_{\ell \in V(T) - \mathcal{L}(T)} w(\ell).$$

An *optimal BCST* is one with minimal cost. We need several more definitions and preliminary lemmas leading to our optimal BCST algorithm.

Definition 3. The *depth* of a subtree is the depth of its root.

A technical definition that will be important for our approach is the following one.

Definition 4. The *side-weight* of a node v (denoted $sw(v)$) in a BCST is defined as follows. If v is a leaf, then $sw(v) = 0$. If v is a node representing an equality comparison (henceforth referred to as an *equality node*), $sw(v) = w(a_i)$ where a_i is the input item tested for equality at v. If v is a less-than node with children x and y, then $sw(v) = \min\{w(x), w(y)\}$.

We now prove two lemmas which will be central to the paper.

Lemma 1. *Let S be a sequence of items with associated weights and let S_1, S_2 be a "partition" of S into two complementary subsequences. Let T be a BCST for S. There there are BCSTs T_1 for S_1 and T_2 for S_2 with $c(T_1) + c(T_2) \leqslant c(T)$.*

Proof. Let T be a BCST for S. Let T_1 be obtained from T by removing all leaves which are in S_2 and repeatedly "shorting out" any node with one child. Clearly T_1 is a BCST for S_1. Similarly obtain BCST T_2 for S_2. It is immediate from the (first) definition of the cost of a BCST that $c(T_1) + c(T_2) \leqslant c(T)$ and the lemma follows. □

Note that if T does not have equality comparisons, then neither do T_1 and T_2.

Lemma 2. *Let T be an optimal BCST. If u is the parent of v in T, then $sw(u) \geqslant sw(v)$.*

Proof. Since the side-weight of a leaf is 0, we may assume that neither u nor v is a leaf. There are four cases according to whether each of u, v is a less-than or equality comparison.

Case 1. Both u, v are less-than comparisons. Assume without loss of generality that v is a right child of u. Let T_1 be the subtree rooted at the child of u which is not v. Let T_2, T_3 be the subtrees rooted, respectively, at the left and right children of v. Let α_i denote the weight of T_i, $1 \leqslant i \leqslant 3$. Now $sw(u) = \min\{\alpha_1, \alpha_2 + \alpha_3\}$ and $sw(v) = \min\{\alpha_2, \alpha_3\}$. For a contradiction, assume $sw(u) < sw(v)$, i.e., $\min\{\alpha_1, \alpha_2+\alpha_3\} < \min\{\alpha_2, \alpha_3\}$, which implies that $\alpha_1 < \alpha_2, \alpha_1 < \alpha_3$. Now rotate v upward along the edge to its parent. While T_2 stays at the same depth, T_1 moves down and T_3 moves up, each by one level. The increase in cost is $\alpha_1 - \alpha_3 < 0$. This contradicts the optimality of T.

Case 2. Node u is a less-than comparison $x < a_1?$ and v is an equality comparison $x = a_2?$. Let T_1, of weight, say, α_1, be the subtree rooted at the child of u which is not v. Let $\alpha_2 = w_2$ and let α_3 be the weight of the subtree T_3 rooted at the child of v not corresponding to a_2. We have $sw(u) = \min\{\alpha_1, \alpha_2 + \alpha_3\}$, $sw(v) = \alpha_2$. For a contradiction, assume that $sw(u) < sw(v)$, i.e., $\min\{\alpha_1, \alpha_2 + \alpha_3\} < \alpha_2$. Hence $\alpha_1 < \alpha_2$.

Again, rotate v up along the edge to u, i.e., replace u by the comparison $x = a_2?$. Replace u's right child by $x < a_1?$. From that node's left and right children, respectively, hang T_1 and T_3. Tree T_1 moves down one level, T_3 stays at the same level, yet a_2, of weight α_2, moves up one level. The net increase in cost is $\alpha_1 - \alpha_2 < 0$, contradicting the optimality of T.

Case 3. Both u, v are equality comparisons, say, u with an item a_1 of weight α_1, v with an item a_2 of weight α_2. We have $sw(u) = \alpha_1, sw(v) = \alpha_2$. For a contradiction, assume $\alpha_1 < \alpha_2$. Swap the comparisons in u and v. Node a_1 moves down, a_2 moves up. The increase in cost is $\alpha_1 - \alpha_2 < 0$, a contradiction.

Case 4. Node u is an equality comparison $x = a_1?$ and v is a less-than comparison $x < a_2?$. Let α_1 be the weight of a_1, and let T_2 and T_3 of weight α_2 and α_3, respectively, be the subtrees hanging off v. Once again assume for a contradiction that $sw(u) < sw(v)$; i.e., $\alpha_1 < \min\{\alpha_2, \alpha_3\}$. Rotate v upward and make the comparison $x = a_1?$ at the appropriate child of v. Then exactly one of T_2 and T_3 moves up while a_1 moves down. The increase in cost is again negative, a contradiction. □

Some immediate corollaries of Lemma 2 are the following:

Corollary 3. *If $n > 2$ and the root of an optimal BCST is an equality node, then the item tested for equality must be a largest-weight item. If $n \leqslant 2$, then any BCST without redundant comparisons has the same cost.*

Proof. Suppose that the root is an equality comparison with a_i (so the root has side-weight w_i), yet $w_j > w_i$. Node a_j appears somewhere as a leaf x in the tree. If its parent v is an equality comparison with a_j itself, then its side-weight is $w_j > w_i$, contradicting Lemma 2. If its parent v is an equality comparison $x = a_t?$, $t \neq j$, then $n > 2$ means that v is not the root. Replace the $x = a_t?$ comparison in v by an $x = a_j?$ comparison, getting a contradiction to Lemma 2. If, on the other hand, v is a less-than comparison, replace it by the equality comparison $x = a_j?$, and get a contradiction.

If $n = 2$, an irredundant BCST has a single internal node and any such tree can be seen to have the same cost. If $n = 1$, the BCST consists of a single node and there is only one irredundant BCST. □

Corollary 4. *If there is an item a_m such that $w_m > \frac{1}{2}\sum_{i=1}^{n} w_i$, then there is an optimal BCST with an equality comparison with a_m at the root.*

Proof. If the root comparison is a less-than comparison, then its side-weight, being the minimum of two terms which add up to $W := \sum_{i=1}^{n} w_i$, is at most $W/2$. However, a_m appears as a leaf x somewhere in the tree. If x's parent v is an equality comparison $x = a_m?$, then v is not the root and v's side-weight exceeds $W/2$, contradicting Lemma 2. If x's parent v is an equality comparison $x = a_t?$, $t \neq m$, we can replace it by $x = a_m?$ and get a contradiction. If v is a less-than comparison, then replace it by the comparison $x = a_m?$. If v is the root, we are done. Otherwise, its side-weight is now $a_m > W/2$, which exceeds the side-weight of the root, contradicting Lemma 2. □

We will see later that this factor of $1/2$ can be reduced to $4/9$.

3. Thresholds

Although we have not made the assumption that the weights of the input items sum to 1, in this section it is convenient to speak about weights that have been so normalized. So for the remainder of this section we will make this assumption and we will refer to the weights as probabilities.

Intuitively, if the maximum probability is large, there should be an optimal BCST whose root comparison is an equality comparison, which, by Corollary 3, must be with an item of maximum probability. Analogously, one would expect that if the maximum probability is small, there should not exist an optimal BCST with a root equality comparison. We study in this section the relationship between

the maximum probability and the existence of an optimal BCST with a root equality comparison.

If the maximum is very small, (we will see that) there cannot be an optimal BCST with an equality comparison at the root. Let us define λ to be the supremum of all p such that for any input whose maximum probability is at most p, there is no optimal BCST with an equality comparison at the root. We will prove that if the maximum probability is less than $1/4$, then there is no optimal BCST with an equality comparison at the root (hence, $\lambda \geqslant 1/4$), and there is an instance with maximum probability $1/4$ which has an optimal BCST with a root equality comparison (hence, $\lambda \leqslant 1/4$). So $\lambda = 1/4$.

How large should the maximum probability be, in order to guarantee the existence of an optimal BCST with a root equality comparison? By Corollary 4, if the maximum probability exceeds $1/2$ then there is a BCST in which the root is an equality comparison. Must there be an optimal BCST with an equality comparison in the root if, instead, the maximum probability is, say, 0.4? Let us define μ to be the infimum of all p such that for any input whose maximum probability is at least p, there is an optimal BCST with an equality comparison at the root. We will prove that if the maximum probability is at least $4/9$, then there is an optimal BCST having an equality comparison at the root (hence, $\mu \leqslant 4/9$), whereas there are instances with maximum probability approaching $3/7$ from below which have no optimal BCST with an equality comparison at the root (hence, $\mu \geqslant 3/7$). So $3/7 \leqslant \mu \leqslant 4/9$.

Later, we will use the fact that $\lambda \geqslant 1/4$ to design a polynomial-time algorithm to find the optimal BCST.

We will use *left* or *right subtree of* T to mean the subtree of T rooted at the left or right child of the root of T.

Theorem 5. *If λ is the supremum over all p such that for any input whose maximum probability is p, there is no optimal BCST with an equality comparison at the root, then $\lambda = 1/4$.*

Proof. First we prove that $\lambda \geqslant 1/4$. Suppose, for a contradiction, that T is an optimal BCST where input item b_0 ($= a_i$ for some i) with weight less than $1/4$ is tested for equality at the root.

Then the side-weight of the root is equal to $w(b_0)$, which is less than $1/4$. By Lemma 2 the side-weight of every other node in the tree is less than $1/4$.

For any internal equality comparison $x = a_i?$ in a node v, let us call the branch leading to the child of v having leaf a_i the *side branch*, and the other branch the *main branch*. For a less-than comparison $x < a_i?$ at node v, we call the branch leading to the child of v of lesser weight the *side branch*, breaking a tie arbitrarily, naming the other branch the *main branch*. The weight of the child along the side branch from v is always $sw(v)$.

Let $r = v_0, v_1, v_2, v_3, \ldots, v_l$ be the nodes, in order, along the unique path from the root to the leaf v_l along main branches. Let s_i, $0 \leqslant i \leqslant l-1$, be the child of v_i on the side branch. Then $sw(v_i) = w(s_i)$. Furthermore, $sw(v_i) \leqslant sw(v_0) < 1/4$, by Lemma 2. Now

$$w(T) = \big[w(s_0) + w(s_1) + w(s_2) + \cdots + w(s_{l-1})\big] + w(v_l).$$

By Corollary 3, $w(v_l) \leqslant w(s_0) = sw(v_0) < 1/4$. Hence, $w(T) \leqslant (l+1)sw(v_0)$. If $l \leqslant 3$, then $w(T) \leqslant 4 \cdot sw(v_0) < 1$, contradicting the fact that $w(T) = 1$. Hence $l \geqslant 4$.

Now let T_1, T_2, T_3 be the trees hanging off the side branches from v_1, v_2, v_3, respectively, and let T_4 be the tree hanging off the main branch from v_3. Note specifically that $w(b_0) \geqslant w(T_1) \geqslant w(T_2) \geqslant w(T_3)$. Figure 1 illustrates this configuration. In this and other figures in this section, we have departed from the usual convention and chosen to depict the side branches as left children and the main branches as right children, since there may be insufficient information to pin down which branch is the "yes" branch.

A convention will be useful in the proof. Since the identities of the items are actually irrelevant—only the weights matter—we assume that $a_i = i$, $i = 1, 2, \ldots, n$. When we speak of weights w_i and w_j for any $i < j$, it is implicit that the first element is smaller than the second. Now each node v_i along the main branch from the root has either the comparison $x = b_i$? or the comparison $x < b_i$?, where $b_i \in \{1, 2, \ldots, n\}$. In either case we will define $b_i \in \{1, 2, 3, \ldots, n\}$ to be the *cut-point* associated with the comparison at v_i. Pictorially, we will represent the cut-points on a number line with a "◯" marking a cut-point corresponding to an equality node and a vertical line marking the cut-point corresponding to a less-than node. Thus our picture has a number line with four cut-points labeled b_0, b_1, b_2, and b_3, corresponding to vertices v_0, v_1, v_2, v_3, respectively.

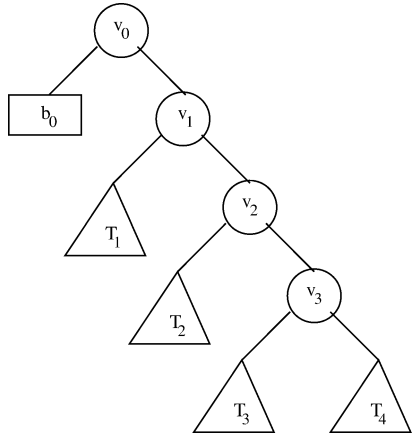

Fig. 1. Tree T expanded along main branches to 4 levels.

Note the following fact: If v_i is a less-than node with comparison $x < b_i$? having cut-point b_i, then for all $j > i$, the cut-points b_j must occur on the same side of b_i, the side which is along the main branch at v_i. In other words, if the main branch corresponds to $x \geqslant b_i$, the "no" branch, then $b_j \geqslant b_i$ for all $j > i$, and if the main branch corresponds to $x < b_i$, then $b_j < b_i$ for all $j > i$. The tree T_i contains only items from the other side of b_i.

Since T_3 and T_4 are symmetrically located in T, we will assume without loss of generality that T_3 contains items to the left of b_3 and T_4 contains b_3 and items to the right.

The idea of the proof is to show that we can rebalance the "skinny" tree shown in Fig. 1 (making the root node a less-than), and reduce its cost, thereby contradicting the optimality of this tree. Since there are four cut-points corresponding to the tree in Fig. 1 it is natural that the balanced tree T' should have at its root a less-than node which splits these cut-points equally. We will call the less-than comparison at the root of T' the *dividing cut*.

There are two main cases.

(1) If the middle two cut-points of T, b_i and b_j, $i, j \in [0, \ldots, 3]$, both correspond to equality nodes, then we will let the dividing cut be $x < b_j$? where $b_j > b_i$.
(2) Otherwise, (at least) one of the middle cut-points corresponds to a less-than comparison $x < b_i$?. In this case, (choose one and) let this less-than comparison be the dividing cut.

Case 1. Note that in this case there are two cut-points to be dealt with on either side of the dividing cut. Let m, p, $0 \leqslant m < p \leqslant 3$, be such that the set of two cut-points to the left of b_j is $\{b_m, b_p\}$. Then we perform the comparison occurring in node v_m of T at the left child of the root of T' and perform the comparison occurring in node v_p of T at the appropriate child of this left child. At the other subtree of this left child, put the optimal BCST for the appropriate set of items. We similarly handle the right side of the dividing cut: Where the rightmost two cut-points are b_q, b_r with $q < r$, do the comparison with b_q and then the comparison with b_r. Intuitively, since $m < p$, $w(T_m) \geqslant w(T_p)$ and we want to have T_m occur higher in T'. This is the reason for the above ordering of comparisons.

Note that in this case, the dividing cut introduces a new split and thereby fractures one of the subtrees in T. Note also that the subtree fractured must be T_3 or T_4. To see this, note that in order for T_i to be fractured, v_i must be a less-than comparison (otherwise T_i consists of a single node) and hence b_i must be one of the two end cut-points. If b_i is the leftmost cut-point and $i < 3$, then since b_3 occurs to the right of b_i, T_i must be to the left and is not fractured. A similar argument holds if b_i is the rightmost cut-point. Whichever of T_3 or T_4 is fractured, we will let S_1 and S_2 denote the subtrees for the two pieces obtained

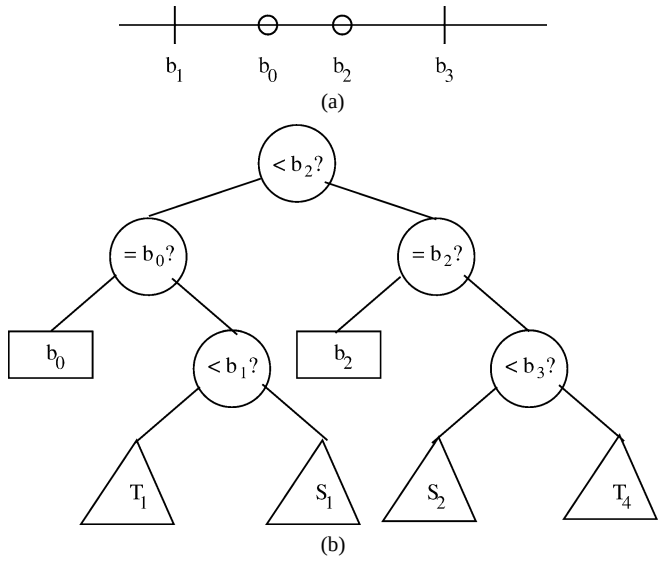

Fig. 2. Example scenario where the middle two cut-points are equalities and the resulting tree T'.

using Lemma 1. Figure 2 shows one example scenario that falls within this case, and the resulting T'.

We will compare the costs of T and T' by looking at the changes in the depths of (the leaf corresponding to) b_0 and the roots of $T_1, T_2,$ and T_3. The depth of b_0 goes from 1 to 2 since the comparison $x = b_0$? will be done at a child of the root of T'. There are two cases for the change in the depth of T_1. If b_1 occurs on the same side of the dividing cut as b_0, then the depth of the root of T_1 goes from 2 to 3. Otherwise, it remains 2. In the former case, the depth of T_2 goes from 3 to 2, while in the latter case, the depth of T_2 remains 3. Since $w(T_1) \geqslant w(T_2)$, the worst possible improvement is when the depth of T_1 goes from 2 to 3. Note that the depth of all three pieces that constitute the original T_3 and T_4 goes from 4 to 3.

Using Lemma 1 for the fractured subtree,

$$\begin{aligned} c(T') - c(T) &\leqslant w(b_0) + \big[w(T_1) - w(T_2)\big] - w(T_3) - w(T_4) \\ &= w(b_0) + w(T_1) - \big(1 - w(b_0) - w(T_1)\big) \\ &= 2\big(w(b_0) + w(T_1)\big) - 1. \end{aligned}$$

Since $w(T_1) \leqslant w(b_0) < 1/4$, the bound above is negative, showing that the cost of T' is less than the cost of T and contradicting the assumption that T is optimal.

Case 2. Since one of the four cut-points of T has already been dealt with as the dividing cut, we will have two cut-points to take care of on one side of the dividing cut and only one to take care of on the other. Just as in Case 1, the two cut-points

on one side of the dividing cut are sequenced in the same order in T' as in T. The subtrees T_i, $i = 1, \ldots, 4$, will not be fractured in this case since the cuts we will use in T' are the same as the cuts used in T.

Consider first the situation where the dividing cut corresponds to cut-point b_1. Since b_2 and b_3 are on the same side of b_1, b_0 must be the lone cut-point on one side and b_2 and b_3 must be on the other side. In the resulting tree T', the depth of b_0 goes from 1 to 2, the depth of T_1 stays at 2, the depth of T_2 goes from 3 to 2 and the depths of T_3 and T_4 go from 4 to 3. Thus $c(T') - c(T) \leqslant w(b_0) - w(T_2) - w(T_3) - w(T_4) < 0$, a contradiction.

Next consider the case where the dividing cut corresponds to cut-point b_2. An example of this situation when b_3 is the lone cut-point on one side of the dividing cut is shown in Fig. 3. In this case, the depths of b_0 and T_1 increase by 1, the depth of T_2 is unchanged and the depths of T_3 and T_4 decrease by 2. Since $w(T_3) + w(T_4)$ is at least $1/4$ and $w(b_0) + w(T_1)$ is less than $1/2$, the net change in cost is negative, again contradicting the optimality of T. If the lone cut-point on one side of the dividing cut is either b_0 or b_1, then in the resulting T' the depth of b_0 increases by 1, the depth of T_1 is unchanged and the depths of T_2, T_3 and T_4 decrease by 1, again yielding a net reduction in cost.

Finally consider the situation where the dividing cut corresponds to cut-point b_3. If b_2 is the sole cut on one side of the dividing cut, then the depths of b_0 and T_1 increase by 1, the depth of T_2 drops by 1 and the depths of T_3 and T_4 drop by at least 1 giving a net reduction in cost. If b_0 or b_1 is the sole cut-point on

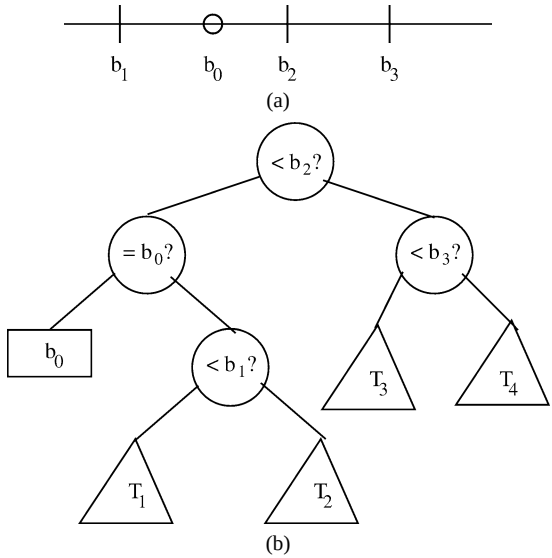

Fig. 3. Example scenario where b_2 is the dividing cut and the resulting tree T'.

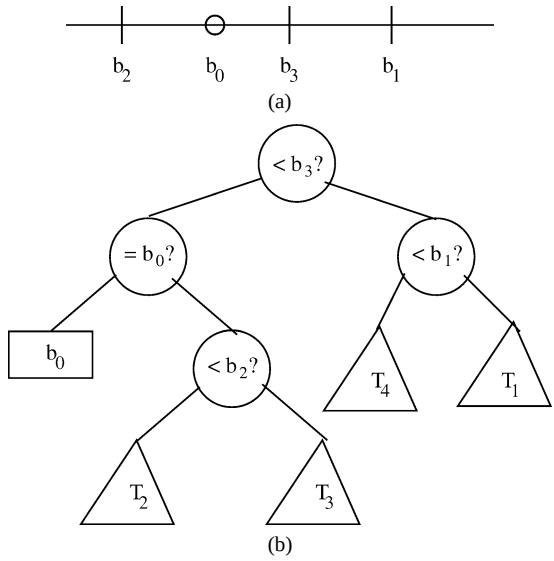

Fig. 4. Example scenario where b_3 is the dividing cut and the resulting tree T'.

one side of the dividing cut, then the depth of b_0 increases by 1, the depths of T_1 and T_2 are unchanged and the depths of T_3 and T_4 drop by at least 1, again giving us the result. Figure 4 depicts one case of this situation.

To prove that $\lambda \leqslant 1/4$, we consider a six-item example $(a_1, a_2, \ldots, a_6)$ with weights $(1/4, 0, 1/4, 1/4, 0, 1/4)$. By a case analysis, an optimal BCST has cost $5/2$ and at least one optimal tree has root equality comparison $x = a_3$?. □

Theorem 6. *If μ is the infimum of all p such that for any input whose maximum probability is at least p, there is an optimal BCST with an equality comparison at the root, then* $3/7 \leqslant \mu \leqslant 4/9$.

Proof. We begin with a proof of the upper bound. We prove by induction on $n \geqslant 2$ (n being the number of items) that if there is some weight which is at least $4/9$ of the sum of all weights, then there is an optimal BCST whose root is an equality comparison. By Corollary 3, this equality comparison must be with a largest-weight item.

For $n = 2, 3$, there is an optimal BCST with an equality comparison at the root. To see this, note that in these cases, for any inequality comparison that yields useful information, there is an equality comparison that yields the same information. Also, an optimal BCST cannot perform a comparison which does not yield useful information. Fix $n \geqslant 4$ and assume that for any list of at least 2 and no more than $n - 1$ items having a weight which is at least $4/9$ of the sum of all its weights, there is an optimal BCST with a root equality comparison. For a

contradiction, assume that there is an instance L of items $a_1, \ldots, a_n$ with weights $w_1, \ldots, w_n$, respectively, with largest weight at least $4/9$ of the sum of its weights, for which no optimal BCST has a root equality comparison. Let T be an optimal BCST for L. If the first comparison is $x < a_2?$ or $x < a_n?$, we can replace it by $x = a_1?$ or $x = a_n?$, respectively, a contradiction. So we may assume that the root comparison is $x < a_r?$, where $3 \leqslant r \leqslant n-1$. By symmetry, we can assume without loss of generality that an instance of the largest-weight item is to the left of the root less-than comparison. The left subtree T_L has between 2 and $n-2$ items, and has an item of weight at least $(4/9)w(T) \geqslant (4/9)w(T_L)$. So by the inductive hypothesis we can replace the left subtree by one which has as its root an equality comparison $x = a_m?$, where $w_m \geqslant (4/9)w(T)$ is the largest weight.

Let T_1 be the subtree rooted at the right child of the equality comparison $x = a_m?$ and let T_2 be the right subtree of the less-than comparison $x < a_r?$ at the root of T. By Lemma 2, $w(T_2) \geqslant w_m$. Note that T_2 must consist of more than one node, since there are at least two elements in the right subtree of the root.

There are two cases to consider, depending on whether T_2 is a tree with an equality comparison at the root, or a tree with a less-than comparison at the root. In each case we will reach a contradiction by showing that there is a tree T' with $c(T') \leqslant c(T)$ having root equality comparison $x = a_m?$.

Case 1. If T_2 has a root equality comparison $x = a_i?$, then let T_3 be the right subtree of T_2. Define tree T' as follows. Tree T' has root equality comparison $x = a_m?$, the right subtree of which has root equality comparison $x = a_i?$, the right subtree of which, in turn, has root less-than comparison $x < a_r?$. The trees T_1 and T_3 are the left and right subtrees of the comparison $x < a_r?$, respectively. Hence, $c(T') = 3 - 2w_m - w_i + c(T_1) + c(T_3)$. By Theorem 5, $w_i \geqslant w(T_2)/4$. Since $w(T_2) \geqslant w_m$, we have $c(T') \leqslant 3 - (9/4)w_m + c(T_1) + c(T_3)$. On the other hand, $c(T) = 2 + c(T_1) + c(T_3)$. Since $w_m \geqslant 4/9$, $c(T') \leqslant c(T)$.

Case 2. If T_2 has a root less-than comparison $x < a_i?$, then let T_3 and T_4 be the left and right subtrees of T_2, respectively. We consider two cases depending on whether $w(T_4) \geqslant w(T_2)/4$.

Case 2.1. Suppose $w(T_4) \geqslant w(T_2)/4$; then we form T' by a sequence of two rotations. First, rotate the equality comparison $x = a_m?$ to be the root. Next, rotate the less-than comparison $x < a_i?$ to be the right child of the root. As a result, T' has root $x = a_m?$, the right subtree of which has root $x < a_i?$, the left subtree of which in turn has root $x < a_r?$. We have $c(T') = 3 - 2w_m - w(T_4) + c(T_1) + c(T_3) + c(T_4)$. Since $w(T_4) \geqslant w(T_2)/4 \geqslant w_m/4$, we have

$$c(T') \leqslant 3 - (9/4)w_m + c(T_1) + c(T_3) + c(T_4).$$

On the other hand,

$$c(T) = 2 + c(T_1) + c(T_3) + c(T_4).$$

As in Case 1, $w_m \geqslant 4/9$ implies $c(T') \leqslant c(T)$.

Case 2.2. Suppose that $w(T_4) < w(T_2)/4$; then $w(T_3) \geqslant (3/4)w(T_2)$. We again consider two cases depending on whether the root of T_3 has an equality or less-than comparison. We do not have to consider the case that T_3 is a single node (a leaf) because that would imply that T_2 is equivalent to a tree whose root is an equality comparison, and that case was covered earlier in Case 1.

Case 2.2.1. Suppose the root of T_3 is a less-than comparison $x < a_j?$. Let T_5 and T_6 be the subtrees of T_3 rooted at the left and right children, respectively, of the root of T_3. We form T' by a series of three rotations. First, rotate the equality comparison $x = a_m?$ up to the root. Next, doubly rotate the less-than comparison $x < a_j$ to be the right child of the root. This results in a tree where the roots of the four trees $T_1, T_5, T_6,$ and T_4 are all at depth 3 in T'. We have

$$c(T') = 3 - 2w_m + c(T_1) + c(T_4) + c(T_5) + c(T_6).$$

We also have

$$c(T) = 2 + w(T_3) + c(T_1) + c(T_4) + c(T_5) + c(T_6).$$

Since $w(T_4) < w(T_2)/4$, we have $w(T_3) \geqslant (3/4)w(T_2) \geqslant (3/4)w_m$. Hence,

$$c(T) \geqslant 2 + (3/4)w_m + c(T_1) + c(T_4) + c(T_5) + c(T_6).$$

Because $w_m \geqslant 4/9 \geqslant 4/11$, we have $c(T') \leqslant c(T)$.

Case 2.2.2. Suppose the root of T_3 is an equality comparison $x = a_j?$. We let T_5 be the subtree rooted at the right child of T_3. The tree T_5 is not a leaf because if it were then the root of T_3 could be replaced by a less-than comparison which is covered in the previous Case 2.2.1. We now use Lemma 1 to partition T_5 into two trees T_6 and T_7 where the less-than comparison $x < a_{j+1}?$ is used to partition T_5, with T_6 having the smaller items. The construction of T' is similar for the two symmetric cases, depending on whether $w(T_6) \leqslant w(T_5)/2$ or $w(T_7) \leqslant w(T_5)/2$. We consider just the former case. In this case T' has the equality comparison $x = a_m?$ at the root and the right subtree has the new less-than comparison $x < a_{j+1}?$ at its root, whose left and right children have comparisons $x < a_r?$ and $x < a_i?$, respectively. Finally, the right subtree of the subtree rooted at the comparison $x < a_r$ has as its root the equality comparison $x = a_j?$. The final result is that in T', the roots of the trees T_1, T_4, and T_7 are at depth 3, and of T_6, at depth 4. We have

$$c(T') = 3 - 2w_m + w_j + w(T_6) + c(T_1) + c(T_4) + c(T_6) + c(T_7).$$

For T to be optimal, Lemma 2 implies that $w(T_4) \geqslant w_j$. Using this fact, together with

$$w(T_6) \leqslant w(T_5)/2, \qquad w(T_3) = w(T_5) + w_j,$$
$$w(T_2) = w(T_3) + w(T_4) \quad \text{and} \quad c(T_6) + c(T_7) \leqslant c(T_5)$$

(by Lemma 1), we have

$$c(T') \leqslant 3 - 2w_m + w(T_2)/2 + c(T_1) + c(T_4) + c(T_5).$$

The cost of T, on the other hand, is

$$c(T) = 2 + w(T_3) + c(T_1) + c(T_4) + c(T_5).$$

In this case, $w(T_3) \geqslant (3/4)w(T_2)$. Hence,

$$c(T) \geqslant 2 + (3/4)w(T_2) + c(T_1) + c(T_4) + c(T_5).$$

Hence,

$$c(T) - c(T') \geqslant (1/4)w(T_2) + 2w_m - 1 \geqslant (9/4)w_m - 1,$$

where the second inequality follows from the fact that $w(T_2) \geqslant w_m$. Hence, $c(T') \leqslant c(T)$ because $w_m \geqslant 4/9$.

We conclude with a proof of the lower bound of $3/7$ on μ. Consider the weights $(3/7 - 4\epsilon,\ 1/7 + \epsilon,\ 0,\ 1/7 + \epsilon,\ 1/7 + \epsilon,\ 0,\ 1/7 + \epsilon)$ for a 7-item example $(a_1, a_2, \ldots, a_7)$. By a careful case analysis, it can be shown that the *unique* optimal BCST, with cost $17/7 + 3\epsilon$ for all sufficiently small positive ϵ, has root comparison $x < a_3$?. By contrast the lowest cost tree with root comparison $x = a_1$? has cost $17/7 + 10\epsilon$. □

4. The $O(n^4)$-time algorithm

In this section we give an $O(n^4)$-time, dynamic programming-based algorithm for finding an optimal BCST on n items. The algorithm relies heavily on the fact that the initial comparison cannot be $x = a_i$? unless a_i has the maximum weight and that weight is at least $1/4$ of the sum of all weights. Since the identities of the items $a_1, a_2, \ldots, a_n$ are irrelevant, the input consists of a sequence $(w_1, w_2, \ldots, w_n)$ of nonnegative weights, assumed integral in this section. Obviously, rational weights can be made integral by scaling by a common denominator.

Our algorithm will compute the optimal cost for each of at most $16n^3$ subproblems, computing each one in $O(n)$ time; as is traditional in dynamic-programming algorithms, we will only compute the optimal cost, leaving it up to the reader to find the actual optimal tree. The notation $S = \langle i_1, i_2, i_3, \ldots, i_r \rangle$ with $1 \leqslant i_1 < i_2 < \cdots < i_r \leqslant n$ or $S = \{i_1, i_2, i_3, \ldots, i_r\}$ denotes the subproblem of finding the optimal BCST for the items numbered $i_1, i_2, i_3, \ldots, i_r$, with associated respective weights $w_{i_1}, w_{i_2}, w_{i_3}, \ldots, w_{i_r}$.

We will compute the optimal cost of each "valid" subproblem:

Definition 5. Let S be a nonempty subset of $\{1, 2, \ldots, n\}$. Let $i = \min S$, $j = \max S$ (possibly $i = j$), and $M = \max_{l \in S} w_l$. S is *valid* if and only if it satisfies these two conditions:

(1) S is closed (strictly) downward in the interval $[i, j]$ in that S contains every $l \in [i, j]$ such that $w_l < M$.
(2) Let $T = \{l \mid i \leqslant l \leqslant j\colon\ w_l = M\}$. Then either $|T| \leqslant 4$ or $T \subseteq S$.

That cryptic last condition says that if the largest weight M in positions in S occurs in the *original* list at least five times among positions $i, i+1, i+2, \ldots, j$, then the indices of all such occurrences must appear in S. If M occurs at most four times in $(w_i, w_{i+1}, w_{i+2}, \ldots, w_j)$, then S is free to contain any nonempty subset of those positions.

First, a simple lemma.

Lemma 7. *The number of valid sets is at most* $16n^3$.

Proof. Consider a valid set S. Since S is nonempty, define $i = \min S$ and $j = \max S$. Let $M = \max_{i \in S} w_l$, the largest weight in S. Because S is closed downward, S must contain all l in $[i, j]$ such that $w_l < M$. If M appears at least five times in $(w_i, w_{i+1}, w_{i+2}, \ldots, w_j)$, then S contains all such occurrences. Otherwise, S contains any nonempty subset of the at-most-four occurrences. It follows that there are at most $16n^3$ valid sets, since for each triple (i, j, M), there are at most 16 valid sets characterized by that triple. □

We will compute the optimal cost of valid sets in order of increasing size of valid sets, starting with valid sets of size one. To do so, we need a simple name for each valid set. Lemma 7 motivates our naming scheme. Give valid set S a unique name (i, j, k, v) as follows. Let $i = \min S$, $j = \max S$. Let $M = \max_{l \in S} w_l$ and let $k = \min\{l \mid l \in S,\ w_l = M\}$ (the leftmost position of an M in S). Let $i \leqslant j_1 < j_2 < \cdots < j_d \leqslant j$ be all the positions in which M appears as a weight of an item in $[i, j]$ in the original list. Then $v = *$ if $d \geqslant 5$, in which case S contains all the positions in $[i, j]$ corresponding to weight M. Otherwise, $v \in \{0, 1\}^d$ with $v_t = 1$ if and only if $j_t \in S$. It is easy in $O(n)$ time to convert from the name of a valid set to an enumeration of its items or *vice versa*.

The first step of the algorithm is to enumerate all valid sets, via their names, to calculate the size of each. We prepare, for each $w \in \{1, 2, \ldots, n\}$, a list of the names of all valid sets of size w. We prepare an array $cost(i, j, k, v)$, to contain, at termination, the optimal cost of the corresponding valid set. This takes $O(n^4)$ time. The cost of the optimal BCST for all n items is the value of the *cost* entry for the valid set $\{1, 2, \ldots, n\}$.

We initialize $cost(i, j, k, v)$ to 0 for all valid sets of size 1, and otherwise we initialize $cost(i, j, k, v)$ to $+\infty$. Then in increasing order by w, we calculate the

optimal cost of each valid set S of size w as follows. In $O(n)$ time, we enumerate the items $\langle i_1, i_2, \ldots, i_w\rangle$ of S in increasing order and calculate the weight W of S.

We consider first the possibility that the root comparison in some optimal BCST for S is a less-than comparison. To do so, we must consider the subproblems $\langle i_1\rangle$ and $\langle i_2, i_3, i_4, \ldots, i_w\rangle$ (associated with $x < a_{i_2}$?), $\langle i_1, i_2\rangle$ and $\langle i_3, i_4, i_5, \ldots, i_w\rangle$ (associated with $x < a_{i_3}$?), $\langle i_1, i_2, i_3\rangle$ and $\langle i_4, i_5, i_6, \ldots, i_w\rangle$ (associated with $x < a_{i_4}$?), $\ldots$, and $\langle i_1, i_2, i_3, \ldots, i_{w-1}\rangle$ and $\langle i_w\rangle$ (associated with $x < a_{i_w}$?). The key point is that each of these subproblems is valid—see Lemma 8—and smaller than S, so we already know the optimal cost of each. Furthermore, in $O(n)$ time in total, one can construct the names of *all* of them (Lemma 10). For each $t = 1, 2, 3, \ldots, w-1$, let C_1 denote the cost of the left subproblem $\langle i_1, i_2, \ldots, i_t\rangle$ and let C_2 be the cost of the right subproblem $\langle i_{t+1}, i_{t+2}, \ldots, i_w\rangle$. We now replace $cost(i, j, k, v)$ by the cost $W + C_1 + C_2$ of the optimal tree rooted at $x < a_{i_{t+1}}$?, if it is smaller than $cost(i, j, k, v)$.

Now we deal with the possibility that the root comparison in problem (i, j, k, v) is an equality comparison. We use the index k to find the largest weight M in $\{w_i \mid i \in S\}$. In $O(n)$ time, we find all indices j such that $j \in S$ and $w_j = M$ and simultaneously sum the weights corresponding to positions in S. If M is less than a quarter of the sum, then we know that the root comparison cannot be an equality comparison, so we move on to the next valid set. Otherwise, M occurs at most four times in $\{w_i \mid i \in S\}$. For each of the at most four indices i_t with $w_{i_t} = M$ in S, generate the subproblem $S - \{i_t\}$, which is valid (Lemma 9), generate its name, look up its optimal cost C, and replace $cost(i, j, k, v)$ by $W + C$ if it is smaller. All of this can be done in $O(n)$ time for each t. This concludes the description of the algorithm.

It is clear that the algorithm runs in $O(n^4)$ time. Furthermore, assuming that *cost* is correct for smaller sets, the construction of the algorithm ensures that the final value of $cost(i, j, k, v)$ is an upper bound on the true optimal cost. The fact that the optimal tree must begin either with a less-than comparison or with an equality comparison on an item which is simultaneously the max and of at least one fourth the total weight, ensures that $cost(i, j, k, v)$ is a lower bound. Hence the algorithm is correct.

Lemma 8. *Suppose* $S = \langle i_1, i_2, \ldots, i_w\rangle$ *is valid. Then* $S' = \langle i_1, i_2, \ldots, i_t\rangle$ *and* $S'' = \langle i_{t+1}, i_{t+2}, \ldots, i_w\rangle$ *are valid for all* t, $1 \leqslant t \leqslant w-1$.

Proof. We focus on S'. Because S is closed downward on $[i_1, i_w]$, S' is closed downward on $[i_1, i_t]$.

Let $M = \max\{w_{i_j} \mid 1 \leqslant j \leqslant w\}$, $M' = \max\{w_{i_j} \mid 1 \leqslant j \leqslant t\}$. Let $T = \{l \mid i_1 \leqslant l \leqslant i_w\colon\ w_l = M\}$ and $T' = \{l \mid i_1 \leqslant l \leqslant i_t\colon\ w_l = M'\}$.

If $M' < M$, then $S' = \{l \mid i_1 \leqslant l \leqslant i_t\colon\ w_l \leqslant M'\}$, $T' \subseteq S'$, and S' is valid.

So we may assume that $M' = M$. Because S is valid, either $|T| \leqslant 4$ or $T \subseteq S$. If $|T| \leqslant 4$, then since $T' \subseteq T$, $|T'| \leqslant 4$ and S' is valid. If $T \subseteq S$, then $T' = T \cap [i_1, i_t] \subseteq S \cap [i_1, i_t] = S'$. So $T' \subseteq S'$ and S' is valid.

The proof for S'' is analogous and is omitted. □

Lemma 9. *Suppose $S = \langle i_1, \ldots, i_w \rangle$ is valid, $|S| \geqslant 2$, $M = \max\{w_{i_j} \mid 1 \leqslant j \leqslant w\}$, $|\{l \mid i_1 \leqslant l \leqslant i_w\colon\ w_l = M\}| \leqslant 4$, and $1 < s < w$ is such that $w_{i_s} = M$. Then $S' = S - \{i_s\}$ is valid.*

Proof. Let $T = \{l \mid i_1 \leqslant l \leqslant i_w\colon\ w_l = M\}$, $M' = \max\{w_l \mid l \in S'\}$, and $T' = \{l \mid i_1 \leqslant l \leqslant i_w\colon\ w_l = M'\}$. If $M' = M$, then $T' \subseteq T$ and hence $|T'| \leqslant 4$; since S is closed downward, so is S' and hence S' is valid.

If $M' < M$, then $T = \{i_s\}$. $S = \{l \mid i_1 \leqslant l \leqslant i_w\colon\ w_l < M\} \cup \{i_s\}$. Therefore $S' = \{l \mid i_1 \leqslant l \leqslant i_w\colon\ w_l < M\}$. Therefore $S' = \{l \mid i_1 \leqslant l \leqslant i_w\colon\ w_l \leqslant M'\}$. S' is closed downward on $[i_1, i_w]$. Also $T' \subseteq S'$. Hence S' is valid. □

Lemma 10. *There is an $O(n)$-time algorithm that takes a valid set $\langle i_1, i_2, \ldots, i_w \rangle$ with $1 \leqslant i_1 < i_2 < \cdots < i_w \leqslant n$ as input and calculates the names of the subproblems $\langle i_1, i_2, \ldots, i_t \rangle$ and $\langle i_{t+1}, i_{t+2}, \ldots, i_w \rangle$ for all t (all of which are valid by Lemma* 8).

Proof. Let $S = \langle i_1, i_2, \ldots, i_w \rangle$ be our valid set in sorted order. In time $O(n)$ we can compute the name (i_1, i_w, i_k, v) of S. Let $M = w_{i_k}$ be the maximum weight of an item of S. If $v \neq *$, then, in the same time bound, compute $j_1 < j_2 < \cdots < j_d$ such that $d \leqslant 4$ and $\{j_1, j_2, \ldots, j_d\} = \{j\colon\ i_1 \leqslant j \leqslant i_w \text{ and } w_j = M\}$.

We will employ a running prefix computation on the sequence $\langle i_1, i_2, \ldots, i_w \rangle$ to compute for each subproblem $\langle i_1, i_2, \ldots, i_t \rangle$ the maximum weight M_t encountered so far, the first occurrence i_{t_1} of this weight, and a tag T_t such that either (1) $T_t = *$ if more than four occurrences of this weight have been encountered, or (2) T_t is the set $\{i_{t_1}, i_{t_2}, \ldots, i_{t_e}\}$ of all occurrences with this weight, where $e \leqslant 4$.

With this information we can, in constant time, compute the name of the subproblem $\langle i_1, i_2, \ldots, i_t \rangle$. There are two cases to consider:

(i) $M_t < M$ or $v = *$; and
(ii) $M_t = M$ and $v \neq *$.

If $M_t < M$ or $v = *$, then by the validity of $\langle i_1, i_2, \ldots, i_w \rangle$, all the indices in the interval $[i_1, i_t]$ with weight M_t are in the set $\langle i_1, i_2, \ldots, i_t \rangle$. If $T_t = *$ then the name of this subproblem is $(i_1, i_t, i_{t_1}, *)$ and if $T_t \neq *$ then the name is $(i_1, i_t, i_{t_1}, 1^e)$ (1^e is the vector of all 1's of length e).

The more interesting case is the one in which $M_t = M$ and $v \neq *$. It must be the case that $T_t \neq *$ also. The name of the subproblem is (i_1, i_t, i_{t_1}, v') where v'

is a vector of length $e' = \max\{f \mid j_f \leqslant i_t\}$. Given T_t and $\{j_1, \ldots, j_{e'}\}$, the vector v' is defined by $v'_i = 1$ if and only if $j_i \in T_t$.

We also need to compute the names of all the subproblems $\langle i_{t+1}, i_{t+2}, \ldots, i_w\rangle$. This is done in a similar way except via a running suffix computation on the sequence $\langle i_1, i_2, \ldots, i_w\rangle$. □

5. Comparison with other models

In this section we compare BCSTs with BCSTs with only less-than comparisons and with BSTs.

5.1. BCSTs with only less-than comparisons

As we saw in the introduction, the 3-item example with weights $(0, 1, 0)$ has optimal cost 1 when equality and less-than comparisons are allowed but optimal cost 2 when only less-than comparisons are allowed. The following theorem demonstrates that this is the worst possible case: the gain in using equality comparisons is very limited. We assume that the weights are normalized so that they sum to 1.

Theorem 11. *If T is a BCST in which the weights sum to* 1*, then there is a BCST T' that uses only less-than comparisons such that $c(T') \leqslant c(T) + 1$.*

Proof. Let $(a_1, \ldots, a_n)$ with weights $(w_1, \ldots, w_n)$ be an input. There are two stages in the construction of T'. In the first stage we remove all equality comparisons whose right subtree is a single item. This is done with no increase in cost. In the second stage we iteratively remove each terminal equality comparison, until there are none remaining. A *terminal equality comparison* is one for which there is no equality comparison anywhere in the right subtree of the comparison. The cost of the removal of the equality comparison $x = a_i?$ can be bounded by w_i. Hence, the cost of removing all the equality comparisons is bounded by the sum of all the weights w_i with $x = a_i?$ as an equality comparison in T. This sum is less than or equal to 1.

Stage 1. Suppose $x = a_i?$ is an equality comparison with a_j as the single item in its right subtree. At no cost, replace the equality comparison with $x < a_{\max\{i,j\}}?$.

At the end of Stage 1, every terminal equality comparison has as the root of its right subtree a less-than comparison (and not a leaf). This is maintained as an invariant throughout Stage 2.

Stage 2. Suppose $x = a_i?$ is a terminal equality comparison and the root of its right subtree is the less-than comparison $x < a_j?$. Let T_1 and T_2 be the left and

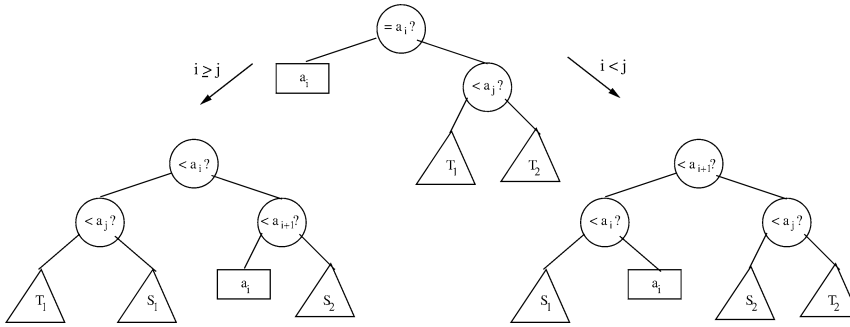

Fig. 5. Removal of equality comparisons in Stage 2 of Theorem 11.

right subtrees of the comparison $x < a_j?$. There are two similar cases to consider depending on whether $i < j$ or $i \geqslant j$.

We first consider the case when $i < j$. In this case, using the construction of Lemma 1, we partition T_1 into T_3 and T_4 where T_3 has all the items in T_1 that are less than a_{i+1}. As noted in the proof of Lemma 1, because T_1 has no equality comparisons, neither will T_3 or T_4. We then replace the *pair* of comparisons $x = a_i?$ and $x < a_j?$ with three less-than comparisons $x < a_{i+1}?$ at the root, with left subtree root $x < a_i?$ and right subtree root $x < a_j?$. The left and right subtrees of $x < a_i?$ are T_3 and a_i, respectively. The left and right subtrees of $x < a_j?$ are T_4 and T_2, respectively. The gain in cost is $c(T_3) + c(T_4) - c(T_1) + w_i$. Since $c(T_3) + c(T_4) \leqslant c(T_1)$, the gain is at most w_i. See Fig. 5.

The case in which $i \geqslant j$ is similar except that we partition T_2 (instead of T_1), using the comparison $x < a_i?$, into T_3, T_4, where T_3 has all the items that are less than a_i and T_4 has all the items that at least a_{i+1}. We then replace the pair of comparisons $x = a_i?$ and $x < a_j?$ with three less-than comparisons: $x < a_i?$ at the root, with left subtree root $x < a_j?$ and right subtree root $x < a_{i+1}?$. The grandchildren, from left to right, are T_1, T_3, a_i, T_4. Again the gain in cost is at most w_i. Again, see Fig. 5. □

5.2. BSTs with only failure weights

There is a natural correspondence between BCSTs with only less-than comparisons and BSTs with only failure weights. Normally, a BST has three-way branching, but when the success weights are zero, the BST exhibits two-way branching. The main difference between a BCST with only less-than comparisons and a BST with only failure weights is that one has success weights and one has failure weights. It turns out that this difference is just superficial. Let $(a_1, \ldots, a_n)$ with weights $(w_1, \ldots, w_n)$ be a BCST input. We construct the following BST instance: $(a_2, \ldots, a_n)$ with all success weights zero, and failure

weights $(w_1, \ldots, w_n)$. Because the success weights are zero, an optimal BST can be interpreted as an optimal BCST in which only less-than comparisons are allowed. The interpretation is as follows: A leaf in the BST indicating all items less than a_2 is interpreted as a leaf labeled a_1 in the BCST, a leaf in the BST indicating all items greater than a_n is interpreted as a leaf labeled a_n in the BCST, and a leaf in the BST indicating all the items in the open interval (a_i, a_{i+1}) for $1 \leqslant i < n$ is interpreted as a leaf labeled a_i in the BCST.

As a consequence of this correspondence, we conclude that any solution to the optimal BST problem where the success weights are zero yields a minimum cost BCST, among the BCSTs which are allowed to use only less-than comparisons, and *vice versa*. Thus, the $O(n \log n)$-time algorithms [3,5] can be used to find an optimal BCST that uses only less-than comparisons. By Theorem 11 such a tree would have cost within 1 of the cost of the optimal BCST that uses both equality and less-than comparisons. Allowing equality comparisons moves the known upper bounds on the complexity of computing the optimal tree from $O(n \log n)$ to $O(n^4)$.

Acknowledgment

Thanks to the anonymous referees for several corrections and suggestions for improvements.

References

[1] R. Anderson, S. Kannan, H. Karloff, R.E. Ladner, Thresholds and optimal binary comparison search trees, in: R. Hariharan, M. Mukund, V. Vinay (Eds.), Proc. Found. Software Technology and Theor. Comput. Sci. (FSTTCS), Bangalore India, in: Lecture Notes in Comput. Sci., Vol. 2245, Springer-Verlag, December 2001, pp. 83–95.
[2] C. Chambers, W. Chen, Efficient multiple and predicate dispatching, in: Proceedings of the 1999 ACM Conference on Object-Oriented Programming Languages, Systems, and Applications (OOPSLA '99), November, 1999.
[3] A.M. Garsia, M.L. Wachs, A new algorithm for minimum cost binary trees, SIAM J. Comput. 6 (1977) 622–642.
[4] J.H. Hester, D.S. Hirschberg, S.-H.S. Huang, C.K. Wong, Faster construction of optimal binary split trees, J. Algorithms 7 (1986) 412–424.
[5] T.C. Hu, A.C. Tucker, Optimal computer-search trees, and variable length alphabetic codes, SIAM J. Appl. Math. 21 (1971) 514–532.
[6] S.-H.S. Huang, C.K. Wong, Optimal binary split trees, J. Algorithms 5 (1984) 69–79.
[7] D.E. Knuth, Sorting and Searching, 2nd edition, in: The Art of Computer Programming, Vol. 3, Addison–Wesley, Reading, MA, 1998.
[8] Y. Perl, Optimum split trees, J. Algorithms 5 (1984) 367–374.
[9] B.A. Sheil, Median split trees: A fast lookup technique for frequently occurring keys, Comm. ACM 21 (1978) 947–958.
[10] D. Spuler, Optimal search trees using two-way key comparisons, Acta Inform. 32 (1994) 729–740.