Classification via Two-Way Comparisons *(Extended Abstract)*

Marek Chrobak(✉) and Neal E. Young

University of California, Riverside, USA
{marek.chrobak,neal.young}@ucr.edu

Abstract. Given a weighted, ordered query set Q and a partition of Q into classes, we study the problem of computing a minimum-cost decision tree that, given any query $q \in Q$, uses equality tests and less-than comparisons to determine the class to which q belongs. Such a tree can be much smaller than a lookup table, and much faster and smaller than a conventional search tree. We give the first polynomial-time algorithm for the problem. The algorithm extends naturally to the setting where each query has multiple allowed classes.

1 Introduction

Given a weighted, ordered *query* set Q partitioned into classes, we study the problem of computing a minimum-cost decision tree that uses equality tests (e.g., "$q = 4$?") and less-than tests (e.g., "$q < 7$?") to quickly determine the class of any given query $q \in Q$. (Here the cost of a tree is the weighted sum of the depths of all queries, where the depth of a given query $q \in Q$ is the number of tests the tree makes when given query q.) We call such a tree a *two-way-comparison decision tree (*2WCDT*)*. See Fig. 1.

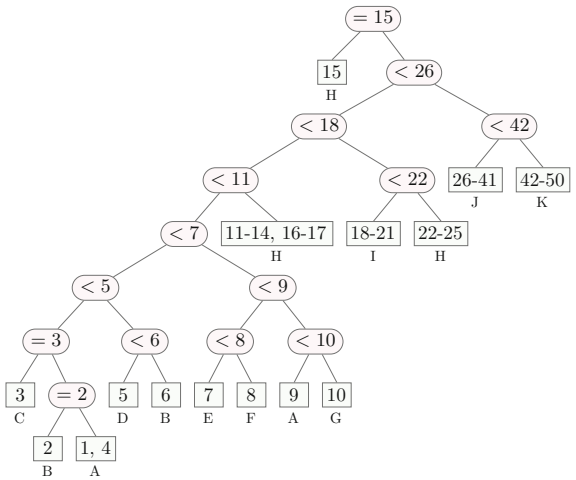
A main use case for 2WCDTs is when the number of classes is small relative to the number of queries. In this case a 2WCDT can be significantly smaller than a lookup table, and, likewise, faster and smaller than a conventional search tree, because a search tree has to identify a given query q (or the inter-key interval that q lies in) whereas a decision tree only has to identify q's class. Because they can be faster and more compact, 2WCDTs are used in applications such as dispatch trees, which allow compilers and interpreters to quickly resolve method implementations for objects declared with type inheritance [2,3]. (Each type is assigned a numeric ID via a depth-first search of the inheritance digraph. For each method, a 2WCDT maps each ID to the appropriate method resolution.)

Chambers and Chen give a heuristic to construct low-cost 2WCDTs, but leave open whether minimum-cost 2WCDTs can be found in polynomial time [2,3]. We give the first polynomial-time algorithm to find minimum-cost 2WCDTs. The algorithm runs in time $O(n^4)$, where $n = |Q|$ is the number of distinct query values,

Most proofs from Sects. 3 and 4 are omitted. See [6] for full proofs.

M. Chrobak—Research partially supported by National Science Foundation grant CCF-2153723.

P. Morin and S. Suri (Eds.): WADS 2023, LNCS 14079, pp. 275–290, 2023.
https://doi.org/10.1007/978-3-031-38906-1_19



q	cls	wt	q	cls	wt	q	cls	wt
1	A	6	18	I	10	35	J	2
2	B	7	19	I	8	36	J	2
3	C	7	20	I	15	37	J	1
4	A	6	21	I	17	38	J	1
5	D	8	22	H	7	39	J	1
6	B	8	23	H	8	40	J	2
7	E	7	24	H	15	41	J	1
8	F	6	25	H	10	42	K	3
9	A	8	26	J	1	43	K	3
10	G	7	27	J	2	44	K	3
11	H	10	28	J	2	45	K	3
12	H	11	29	J	1	46	K	3
13	H	10	30	J	2	47	K	3
14	H	9	31	J	2	48	K	3
15	H	500	32	J	100	49	K	3
16	H	9	33	J	1	50	K	3
17	H	10	34	J	2			

Fig. 1. An optimal two-way-comparison decision tree (2WCDT) for the problem instance shown on the right. The instance (but not the tree) is from [2,3, Figure 6]. Each leaf (rectangle) is labeled with the queries that reach it, and below that with the class for the leaf. The table gives the class and weight of each query $q \in Q = [50] = \{1, 2, \ldots, 50\}$. The tree has cost 2055, about 11% cheaper than the tree from [2,3], of cost 2305.

matching the best time bound known for a special type of 2WCDTs called *two-way-comparison search trees (*2WCSTs*)*, discussed below. The algorithm extends naturally to the setting where each query can belong to multiple classes, any one of which is acceptable as an answer for the query. The extended algorithm runs in time $O(n^3m)$, where m is the sum of the sizes of the classes.

Related Work. Various types of decision trees are ubiquitous in the areas of artificial intelligence, machine learning, and data mining, where they are used for data classification, clustering, and regression.

We study decision trees for one-dimensional data sets. Most work on such trees has focussed on *search trees*, that is, decision trees that must explicitly identify the query or the inter-key interval it lies in (essentially, each class is a singleton). Here is a brief summary of relevant work on such trees. One of our main contributions is to increase the understanding of trees based on two-way comparisons. These are not yet fully understood.

The tractability of finding a minimum-cost search tree depends heavily on the kind of tests that the tree can use. For some kinds of tests, the problem is NP-complete [13]. Early works considered trees in which each test compared the given query value q to some particular comparison key k, with *three* possible outcomes: the query value q is less than, equal to, or greater than k [7, §14.5], [15, §6.2.2] (See Fig. 2(a)). We call such trees *three-way-comparison search trees*, or 3WCSTs for short. In a 3WCST, the query values that reach any given node form an interval. This leads to a natural $O(n^3)$-time dynamic-programming algorithm

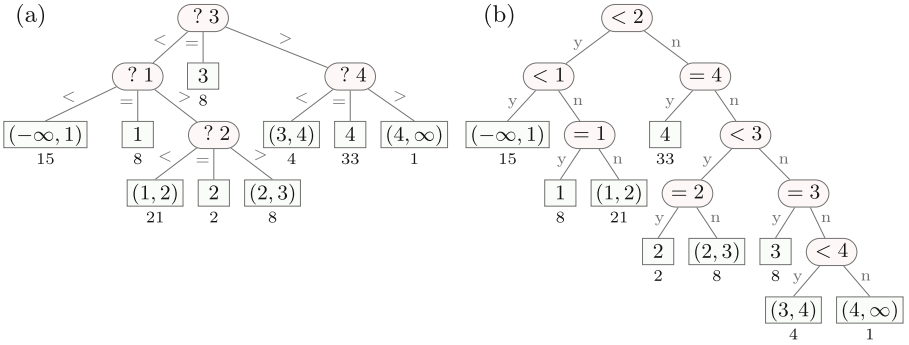


Fig. 2. Tree *(a)* is a three-way-comparison search tree (3WCST). Tree *(b)* is a two-way-comparison search tree (2WCST) for the same instance. The query (or interval of queries) reaching each (rectangular) leaf is within the leaf. The weight of the query (or interval) is below the leaf.

with $O(n^2)$ subproblems for finding minimum-cost 3WCSTs [9]. Knuth reduced the time to $O(n^2)$ [14].

In practice each three-way comparison is often implemented by doing a less-than test followed by an equality test. Knuth [15, §6.2.2, Example 33] proposed exploring binary search trees that use these two tests directly in any combination. Such trees are called *two-way-comparison search trees (*2WCSTs*)* [1]. For the so-called *successful-queries* variant (defined later, after Theorem 2), assuming that the query weights are normalized to sum to 1, there is always a 2WCST whose cost exceeds the entropy of the weight distribution by at most 1 [8]. The entropy is a lower bound on the cost of any binary search tree that uses Boolean tests of any kind. This suggests that restricting to less-than and equality tests need not be too costly [8].

Stand-alone equality tests introduce a technical obstacle not encountered with 3WCSTs. Namely, while (analogously to 3WCSTs) each node of a 2WCST is naturally associated with an interval of queries, not all queries from this interval necessarily reach the node. For this reason the dynamic-programming approach for 3WCSTs does not extend easily to 2WCSTs. This led early works to focus on restricted classes of 2WCSTs, namely *median split trees* [17] and *binary split trees* [10,12,16]. These, by definition, constrain the use of equality tests so as to altogether sidestep the aforementioned technical obstacle. *Generalized binary split trees* are less restrictive, but the only algorithm proposed to find them [11] is incorrect [5]. Similarly, the first algorithms proposed to find minimum-cost 2WCSTs (without restrictions) were given without proofs of correctness [18,19], and the recurrence relations underlying some of those proposed algorithms turned out to be demonstrably wrong [5].

In 1994, Spuler made a conjecture that leads to a natural dynamic program for 2WCSTs. Namely, that every instance admits a minimum-cost 2WCST with the *heaviest-first* property: that is, at any equality-test node $\langle = h \rangle$, *the compari-*

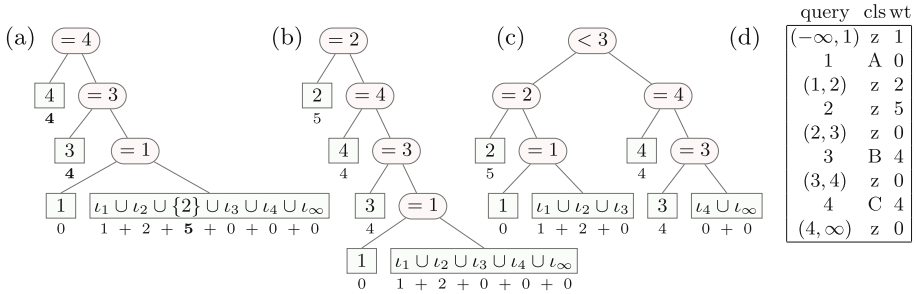


Fig. 3. Three trees for the 2WCDT instance shown in *(d)*. The set of queries reaching each (rectangular) leaf is shown within the leaf (to save space, there ι_i denotes the inter-key open interval with right boundary i, e.g. $\iota_1 = (-\infty, 1)$, $\iota_2 = (1, 2)$). The associated weights are below the leaf. The optimal tree *(a)* has cost 36 and is not heaviest-first. Each heaviest-first tree (e.g. *(b)* of cost 37 or *(c)* of cost 39) is not optimal. These properties also hold if each weight is perturbed to make the weights distinct.

son key h is heaviest among keys reaching the node [19]. In a breakthrough in 2002, Anderson et al. proved the conjecture for the aforementioned successful-queries variant, leading to an $O(n^4)$-time dynamic-programming algorithm to find minimum-cost 2WCSTS for that variant [1]. In 2021, Chrobak et al. simplified their result (in particular, the handling of keys of equal weights, as discussed later) obtaining an $O(n^4)$-time algorithm for finding minimum-cost 2WCSTS [4].

Our Contributions. Unfortunately these 2WCST algorithms don't extend easily to 2WCDTS. The main obstacle is that for some instances (e.g. in Fig. 3) no minimum-cost 2WCDT has the crucial heaviest-first property. To overcome this obstacle we introduce a *splitting* operation (Definition 7), a correctness-preserving local rearrangement of the tree that can be viewed as an extension of the well-studied rotation operation to a more general class of trees, specifically, to trees whose allowed tests induce a laminar set family (Property 1).

We use splitting to identify an appropriate relaxation of the heaviest-first property that we call being *admissible* (Definition 4). Most of the paper is devoted to proving the following theorem:

Theorem 1. *If the instance is feasible, then some optimal tree is admissible.*

Section 3 gives the proof. Along the way it establishes new structural properties of optimal 2WCSTS and 2WCDTS. Section 4 shows how Theorem 1 leads to a suitable dynamic program and our main result:

Theorem 2. *There is an $O(n^3m)$-time algorithm for finding a min-cost* 2WCDT.

Remarks. The presentation above glosses over a secondary technical obstacle for 2WCSTS. For 2WCST instances with *distinct* query weights, the heaviest-first property uniquely determines the key of each equality test, so that the subset of queries that reach any given node in a 2WCST with the heaviest-first

property must be *one of* $O(n^4)$ *predetermined subsets*. This leads to a natural dynamic program with $O(n^4)$ subproblems (See Sect. 3). But this does not hold for instances with non-distinct weights. This obstacle turns out to be more challenging than one might expect. Indeed, there are instances for which, for each of the 2^n subsets S of Q, there is a minimum-cost 2WCST, having the heaviest-first property, with a node u such that the set of queries reaching u is S. It appears that one cannot just break ties arbitrarily: it can be that, for two maximum-weight keys h and h' reaching a given node u, there is an optimal subtree in which u does an equality-test to h, but none in which u does an equality-test to h' [4, Figure 3]. Similar issues arise in finding optimal *binary split trees*—these can be found in time $O(n^4)$ if the instance has distinct weights, while for arbitrary instances the best bound known is $O(n^5)$ [10].

Nonetheless, using a perturbation argument Chrobak et al. show that an arbitrary 2WCST instance can indeed be handled as if it is a distinct-weights instance just by breaking ties among equal weights in an appropriate way [4]. We use the same approach here for 2WCDTS.

Most search-tree problems come in two flavors: the *successful-queries* variant and the *standard* variant. In the former, the input is an ordered set K of weighted keys, each comparison must compare the given query value to a particular key in K, and each query must be a value in K. In the latter, the input is augmented with a weight for each open interval between successive keys. Queries (called *unsuccessful queries*) to values in these intervals are also allowed, and must be answered by returning the interval in which the query falls. Our formal definition of 2WCDTS generalizes both variants.

2 Definitions and Technical Overview

For the remainder of the paper, fix a 2WCDT instance $(Q, w, \mathcal{C}, K)$, where Q is a totally ordered finite set of *queries*, each with a weight $w(q) \geq 0$, the set $\mathcal{C} \subseteq 2^Q$ is a collection of *classes* of queries (where each class has a unique identifier), and $K \subseteq Q$ is the set of *keys*. Let $n = |Q|$ and $m = \sum_{c \in \mathcal{C}} |c|$. The problem is to compute a minimum-cost *two-way-comparison decision tree* (2WCDT, as defined below) for the instance.

To streamline presentation, throughout the paper we restrict attention to the model of decision trees that allows only less-than and equality tests. Our results extend naturally to decision trees that also use other inequality comparisons between queries and keys. See the end of Sect. 4 for details.

Definition 1 (2WCDT)**.** *A* two-way-comparison decision tree (2WCDT) *is a rooted binary tree T where each non-leaf node is a* test *of the form $\langle < k \rangle$ for some $k \in K \backslash \{\min Q\}$, or of the form $\langle = k \rangle$ for some $k \in K$, and the two children of the node are labeled with the two possible test outcomes ("yes" or "no"). Each leaf node is labeled with the identifier of one class in $\mathcal{C}$. This class must contain every query $q \in Q$ whose search (as defined next) ends at the leaf.*

For each $q \in Q$, the search for q in T *starts at the root, then recursively searches for q in the root's yes-subtree if q satisfies the root's test, and otherwise*

in the no-subtree. The search stops at a leaf, called the leaf for q. *The path from the root to this leaf is called* q*'s* search path. *We say that* q reaches *each node on this path, and* q*'s* depth *in* T *is defined as the length of this path (equivalently, the number of comparisons when searching for* q*). The* cost *of* T *is the weighted sum of the depths of all queries in* Q.

A tree T *is called* irreducible *if, for each node* u *in* T*, (i) at least one query in* Q *reaches* u*, and (ii) if some class* $c \in \mathcal{C}$ *contains all the queries that reach* u*, then* u *is a leaf.*

For any $\ell, r \in Q$*, let* $[\ell, r]_Q$ *and* $[\ell, r]_K$ *denote the* query interval $\{q \in Q : \ell \le q \le r\}$ *and* key interval $\{k \in K : \ell \le k \le r\} = K \cap [\ell, r]_Q$.

Allowing K and Q to be specified as we do captures both the successful-queries and standard variants. The successful-queries variant corresponds to the case when $K = Q$. The standard variant is modeled by having one non-key query between every pair of consecutive keys, and before the minimum key and after the maximum key (so $|Q \backslash K| = |K| + 1$). Each such non-key query represents an interval between keys.

For ease of exposition, assume without loss of generality that each query belongs to some class, so $m \ge |Q|$ and the input size is $\Theta(n + m) = \Theta(m)$. Note that the instance is not necessarily feasible, that is, it might not have a decision tree. (To be feasible, in addition to each query belonging to some class, each query interval that contains no keys must be contained in some class.) If the instance is feasible, some optimal tree is irreducible, so we generally restrict attention to irreducible trees. As we shall see, in an irreducible tree any given test is used in at most one node.

Definition 2 (ordering queries by weight). *For any query subset* $R \subseteq Q$ *and integer* $i \ge 0$ *define* $\mathsf{heaviest}_i(R)$ *to contain the* i *heaviest queries in* R *(or all of* R *if* $i \ge |R|$*). For* $q \in Q$*, define* $\mathsf{heavier}(q)$ *to contain the queries (in* Q*) that are heavier than* q*. Define* $\mathsf{lighter}(q)$ *to contain the queries (in* Q*) that are lighter than* q*. Break ties among query weights arbitrarily but consistently throughout.*

Formally, we use the following notation to implement the consistent tie-breaking mentioned above. Fix an ordering of Q by increasing weight, breaking ties arbitrarily. For $q \in Q$ let $\tilde{w}(q)$ denote the rank of q in the sorted order. Throughout, given distinct queries q and q', define q to be lighter than q' if $\tilde{w}(q) < \tilde{w}(q')$ and heavier otherwise ($\tilde{w}(q) > \tilde{w}(q')$). So, for example $\mathsf{heaviest}_i(R)$ contains the last i elements in the ordering of R by increasing $\tilde{w}(q)$. The symbol $\perp$ represents the undefined quantity $\arg\max \emptyset$. Define $\tilde{w}(\perp) = w(\perp) = -\infty$, $\mathsf{heavier}(\perp) = Q$, and $\mathsf{lighter}(\perp) = \emptyset$.

Definition 3 (intervals and holes). *Given any non-empty query subset* $R \subseteq Q$*, call* $[\min R, \max R]_Q$ *the* query interval *of* R*. Define* $k^*(R)$ *to be the heaviest key in* R*, if there is one (that is,* $k^*(R) = \arg\max\{\tilde{w}(k) : k \in K \cap R\}$*). Define also* $\mathsf{holes}(R) = [\min R, \max R]_Q \backslash R$ *to be the set of* holes *in* R*. We say that a hole* $h \in \mathsf{holes}(R)$ *is* light *if* $\tilde{w}(h) < \tilde{w}(k^*(R))$*, and otherwise* heavy.

The set of queries reaching a node u in a tree T is called u's query set*, denoted Q_u. The query interval, and light and heavy holes, for u are defined to be those for u's query set Q_u.*

Overview. Note that each hole $h \in \mathsf{holes}(Q_u)$ at a node u in a tree T must result from a failed equality test $\langle = h \rangle$ at a node v on the path from the root to u in T. In particular, $h \in K$. Further, if the hole is light, then h is not the heaviest key reaching v.

The problem has the following *optimal substructure* property. Any query subset $R \subseteq Q$ naturally defines the subproblem $\pi(R)$ induced by restricting the query set to R (that is, $\pi(R) = (R, w, \mathcal{C}_R, K)$ where $\mathcal{C}_R = \{c \cap R : c \in \mathcal{C}\}$). In any minimum-cost tree T for R, if T is not a leaf, then the yes-subtree and no-subtree of T must be minimum-cost subtrees for their respective subproblems.

Let $\mathsf{cost}(R)$ be the minimum cost of an irreducible tree for $\pi(R)$. (If R is empty, then $\mathsf{cost}(R) = \infty$, as no tree for R is irreducible.) Then the following recurrence holds:

Observation 1 (recurrence relation). *Fix any $R \subseteq Q$. If $R = \emptyset$, we have $\mathsf{cost}(R) = \infty$. Otherwise, if $(\exists c \in \mathcal{C})\, R \subseteq c$ (that is, R can be handled by a single leaf labeled c), then $\mathsf{cost}(R) = 0$. Otherwise, for any test u, let $(R_u^{\mathsf{yes}}, R_u^{\mathsf{no}})$ be the bipartition of R into those queries that satisfy u and those that don't. Then*

$$\mathsf{cost}(R) = w(R) + \min_u \left(\mathsf{cost}(R_u^{\mathsf{yes}}) + \mathsf{cost}(R_u^{\mathsf{no}}) \right),$$

where the variable u ranges over the allowed tests (per Definition 1) such that R_u^{yes} and R_u^{no} are non-empty. (If there are no such tests then $\mathsf{cost}(R) = \infty$.)

The goal is to compute $\mathsf{cost}(Q)$ using a dynamic program that applies the recurrence in Observation 1 recursively, memoizing results so that for each distinct query set R the subproblem for R is solved at most once. (The algorithm as presented computes only $\mathsf{cost}(Q)$. It can be extended in the standard way to also compute an optimal tree.) The obstacle is that exponentially many distinct subproblems can arise.

Identity Classification Without Equality Tests. For intuition, consider first the variant of our problem in which $\mathcal{C}$ is the *identity classification*, that is $\mathcal{C} = \big\{\{q\} : q \in Q\big\}$, and only less-than tests $\langle < k \rangle$ are allowed (equality tests are not). In the absence of equality tests, there are no holes. When applying the recurrence recursively to $\mathsf{cost}(Q)$, each query set R that arises is a query interval. There are $O(n^2)$ such query intervals, and for each the right-hand side of the recurrence can be evaluated in $O(n)$ time. This yields an $O(n^3)$-time algorithm. This approach mirrors a classical dynamic-programming algorithm for 3WCSTS [9], as discussed in the introduction.

The algorithm extends easily to arbitrary classifications $\mathcal{C}$. Recall that a given query set R can be handled by a leaf (at zero cost) if and only if $R \subseteq c$ for some $c \in \mathcal{C}$. This condition can be checked in constant time given (ℓ, r) such that $R = [\ell, r]_Q$ (after an appropriate precomputation, e.g., for each ℓ, precompute the maximum r for which the condition holds).

Identity Classification with Equality Tests Allowed. Next consider the variant when $\mathcal{C}$ is the identity classification but both kinds of tests are allowed. This is essentially the problem of computing a minimum-cost 2WCST. In this variant, each node u in a tree T has query set $Q_u = [\min Q_u, \max Q_u]_Q \setminus \mathsf{holes}(Q_u)$. Applying the recurrence naively to $\mathsf{cost}(Q)$ can yield exponentially many subproblems because $\mathsf{holes}(Q_u)$ can be almost any subset of $[\min Q_u, \max Q_u]_Q$. However, as mentioned in Sect. 1, it is known that some optimal tree T has the *heaviest-first* property [1,4]: for each node u in T that does an equality test $\langle = h \rangle$, the test key h is the heaviest key reaching u. (Our tie-breaking scheme makes h unique.) In such a tree there are no light holes. That is, the hole set of any given node u is the set of heavy holes at u:

$$\mathsf{holes}(Q_u) = [\min Q_u, \max Q_u]_K \cap \mathsf{heavier}(k^*(Q_u)).$$

(Note that, by the definition of $k^*(Q_u)$, no keys heavier than $k^*(Q_u)$ reach u, so the set $[\min Q_u, \max Q_u]_K \cap \mathsf{heavier}(k^*(Q_u))$ contains exactly the heavy holes at u.)

A non-empty query set R without light holes is determined by the triple $(\min R, \max R, k^*(R))$, so there are $O(n^3)$ query sets without light holes. This leads naturally to an $O(n^4)$-time algorithm for instances with distinct weights [1,4]. (Specifically, redefine $\mathsf{cost}(R)$ to be the minimum cost of any *heaviest-first*, irreducible tree for $\pi(R)$. Then $\mathsf{cost}(R) = \infty$ if R has at least one light hole. Add this case as a base case to the recurrence. Apply the modified recurrence recursively to calculate $\mathsf{cost}(Q)$. Then the number of distinct non-trivial subproblems that arise is $O(n^3)$, and each can be solved in $O(n)$ time.)

Allowing Equality Tests and An Arbitrary Classification. The existing results for 2WCSTs don't extend to 2WCDTs because, as shown in Fig. 3, there are 2WCDT instances with distinct weights for which no optimal tree is heaviest-first. But, in some sense, the example in Fig. 3 is as bad as it gets. There is an optimal tree in which an appropriate relaxation of the heaviest-first property holds, namely, that each node's query set is *admissible*. Roughly, this means that there are at most three light holes, and the light holes must be taken heaviest first from those keys that don't belong to some class $c \in \mathcal{C}$ that contains k^* (the heaviest key reaching the node). Here's the formal definition:

Definition 4 (admissible). *Consider any query subset $R \subseteq Q$. The set R is called* admissible *if it is non-empty and the set of light holes in R is either empty or has the form*

$$\mathsf{heaviest}_b([\min R, \max R]_K \cap \mathsf{lighter}(k^*(R)) \setminus c)$$

for some $b \in [3]$ and $c \in \mathcal{C}$ such that $k^*(R) \in c$. A tree (or subtree) T is called admissible *if the query set of each node in T is admissible.*

Above (and within any mathematical expression), for any integer i, the notation $[i]$ denotes the set $\{1, 2, \ldots, i\}$.

To gain some intuition note that, by definition, for any query set R its holes must be in $[\min R, \max R]_K$, and its light holes must be in $\mathsf{lighter}(k^*(R))$. For the algorithm, roughly, we redefine $\mathsf{cost}(R) = \infty$ if R is not admissible, add a corresponding base case to the recurrence and then recursively apply the modified recurrence to compute $\mathsf{cost}(Q)$. Each admissible query set R with no light holes is determined by the triple $(\min R, \max R, k^*(R))$. Per Definition 4, each admissible query set R with at least one light hole is determined by a triple $(\min R, \max R, k^*(R), b, c)$, where $(b, c) \in [3] \times \mathcal{C}$ with $k^*(R) \in c$. It follows that there are $O(n^3 + n^2 m) = O(n^2 m)$ admissible query subsets, so that, in the recursive evaluation of $\mathsf{cost}(Q)$, $O(n^2 m)$ distinct non-trivial subproblems arise. These are solvable in total time $O(n^3 m)$. Section 4 gives the detailed proof.

3 Some Optimal Tree is Admissible

This section proves Theorem 1: if the instance is feasible, then some optimal tree is admissible. Along the way we establish quite a bit more about the structure of optimal trees. Section 3.1 introduces the aforementioned *splitting* operation (Definition 7), a correctness-preserving local rearrangement of the tree that extends the well-studied rotation operation to trees whose allowed tests induce a laminar family (Property 1). (These trees subsume 2WCDTS and 2WCSTS.) Splitting is used to prove two weight bounds (Lemmas 3 and 4), which are used in turn to prove the main structural theorem (Theorem 3). The theorem says that if one child of a node u_1 is lighter than some descendant u_d of the other child, then the path from u_1 to u_d must have a highly restricted structure. (Roughly, for any two distinct nodes u_i and u_j along the path, the outcome from u_i that remains on the path is *consistent*, per Definition 5, with both outcomes leaving u_j.) Theorem 3 holds for any class of trees whose allowed tests induce a laminar family. Lemmas 5 and 6 use Theorem 3 to prove Theorem 1 for distinct-weights instances. A perturbation argument then extends the result to all instances.

We start with some general terminology for how pairs of tests can relate. Recall that $(Q, w, \mathcal{C}, K)$ is a problem instance with at least one correct tree. In any such tree, each edge $u \to v$ from a node to its child corresponds to one of the two possible outcomes of the test at u. We use $u \to v$ to denote both the edge and the associated outcome at u. For example, if u is $\langle < 3 \rangle$, and v is the no-child of u, then outcome $u \to v$ means the queried value is at least 3.

Definition 5. *Two such outcomes $u \to v$ and $x \to y$ are called* consistent *if Q contains a query value that satisfies them both. Otherwise they are* inconsistent.

Two tests are said to be equivalent *if either for all $q \in Q$ the tests give the same outcome for q, or for all $q \in Q$ the tests give opposite outcomes for q.*

For example, assume $Q = [4]$. The yes-outcome of $\langle < 3 \rangle$ is inconsistent with the yes-outcome of $\langle = 4 \rangle$ and with the no-outcome of $\langle < 4 \rangle$, but is consistent with both outcomes of $\langle = 2 \rangle$, and with both outcomes of $\langle < 2 \rangle$. The tests $\langle < 4 \rangle$ and $\langle = 4 \rangle$ are equivalent.

Most of the proof requires only the following property of tests:

Property 1 (laminarity). *Let u and x be test nodes. (i) If u and x do non-equivalent tests, then, among the four pairs of outcomes between the two nodes, exactly one pair is inconsistent, while the other three pairs are consistent. Formally, let $u \to v$, $u \to v'$, $x \to y$, and $x \to y'$ be the two outcomes from u and the two outcomes from x. Then exactly one pair in $\{u \to v, u \to v'\} \times \{x \to y, x \to y'\}$ is inconsistent. (ii) If u and x do equivalent tests, each outcome at u is consistent with a distinct outcome at x.*

Property 1 is easily verified. (Note that, by the definition of 2WCDTS in Sect. 2, and assuming there is more than one test, each outcome of each test is satisfied by at least one query in Q.) We call Property 1 *laminarity* because it is equivalent to the laminarity of the collection of sets that has, for each possible test, one set containing the queries that satisfy the test. In our case this laminar collection is

$$\big\{\{q \in Q : q < k\} : k \in K \backslash \{\min Q\}\big\} \cup \big\{\{q\} : q \in K\big\}.$$

As an example, consider the query set $Q = [4]$. Then the yes-outcome of $\langle < 3\rangle$ and the yes-outcome of $\langle = 4\rangle$ are inconsistent, while every other pair of outcomes is consistent; e.g., the yes-outcome of $\langle < 3\rangle$ and the no-outcome of $\langle = 4\rangle$ are consistent, as they are both satisfied by the query value 2.

Throughout most of the rest of this section (including Sects. 3.1 and 3.2), fix T to be an arbitrary irreducible tree.

Property 2. *(i) In T, if u is a proper ancestor of a test node v then the outcome of u leading to v is consistent with both outcomes at v, and the other outcome of u is consistent with exactly one outcome at v. (ii) No two nodes in T are equivalent.*

Proof. The irreducibility of T implies the first part of (i) (that the outcome of u leading to v is consistent with both outcomes at v). So Property 1(ii) implies that u and v are not equivalent. Then Property 1(i) implies the second part of (i) (that the outcome at u leading away from v is consistent with exactly one outcome at v.) To justify Property 2(ii), let x and y be two different test nodes in T. We have already established that if one of x, y is an ancestor of the other then they cannot be equivalent. In the remaining case, let u be the lowest common ancestor of x and y. Using (i) twice, the outcome at u leading to x is consistent with both outcomes at x but is consistent with exactly one of the outcomes at y. So x and y cannot be equivalent. □

3.1 Two Weight Bounds, via Splitting

This section defines splitting and proves the weight bounds (Lemmas 3 and 4).

Definition 6 (x-consistent path). *Let u be a node in T, T_u the subtree of T rooted at u, and x any allowed test (not necessarily in T). The* x-consistent path from u *is the maximal downward path from u in T_u such that each outcome along this path is consistent with both outcomes at x.*

The x-consistent path from u is unique because (by laminarity) at most one outcome out of any given node is consistent with both outcomes at x. In the case that T_u contains a node $\tilde{x}$ that is equivalent to x, the x-consistent path from u is the path from u to $\tilde{x}$ (using here the irreducibility of T and that neither outcome at $\tilde{x}$ is consistent with both outcomes at x). In the case that T_u contains no such node $\tilde{x}$, this x-consistent path from u ends at a leaf.

Fix a node u in T and a test node x, not necessarily in T. Informally, *splitting T_u around x* replaces subtree T_u of T by the subtree T'_x obtained by the following process: initialize T'_x to a subtree with root x, whose yes- and no-subtrees are each a copy of T_u, then splice out each redundant test (that is, each test w such that one of the outcomes at w is inconsistent with the outcome at x that leads to w, implying that every query reaching w satisfies the other outcome at w). The resulting subtree T'_x has a particular structure that we'll need to use. The formal definition, below, makes this structure explicit.

This construction and the proofs below use notation $u_1 \to u_2 \to \cdots \to u_j$ for a downward path in T, and use u'_i to denote the sibling of u_i, so each edge $u_i \to u'_{i+1}$ leaves the path.

Definition 7 (splitting). Splitting T_u around x *yields the subtree T'_x produced by the following process. Let $u_1 \to u_2 \to \cdots \to u_d$ be the x-consistent path from $u = u_1$, as defined in Definition 6. Initialize T'_x to have root x, with yes- and no-subtrees, denoted T_u^{yes} and T_u^{no}, each a copy of T_u.*

For each outcome $\alpha \in \{\mathsf{yes}, \mathsf{no}\}$ at x, modify T_u^{α} within T'_x as follows. For each $i \in [d-1]$, if outcome $u_i \to u'_{i+1}$ is inconsistent with the α-outcome at x, within T_u^{α}, delete node u_i and the subtree $T_{u'_{i+1}}$, making u_{i+1} the child of the current parent of u_i in place of u_i. For $i = d$, if u_d is a leaf, stop. Otherwise (u_d is a test node), let $u_d \to y'$ be the outcome at u_d that is inconsistent with the α-outcome at x. Within T_u^{α}, delete node u_d and the subtree $T_{y'}$, making the other child y of u_d the child of the current parent of u_d in place of u_d.

Note that, for each $\alpha \in \{\mathsf{yes}, \mathsf{no}\}$, by the definition of the x-consistent path from u and Property 1 (laminarity), outcome $u_i \to u'_{i+1}$ is inconsistent with exactly one outcome at x. Also, if u_d is a test node then it must be equivalent to x, so exactly one outcome at u_d is inconsistent with the α-outcome at x. (See Lemmas 1 and 2 in the full paper [6] for a more detailed characterization of the result of splitting.) Figs. 4 and 5 give examples of splitting. In Fig. 4, $d = 4$ and x is a test node in T_u (in fact $x = u_4$). In Fig. 5, x is a new node (not equivalent to any node in T_u, where $u = u_1$), $d = 5$ and u_5 is a leaf.

The proofs of the following weight bounds take advantage of laminarity. Specifically, as T is irreducible, Property 2(i) implies that if u_i is a proper ancestor of u_j then outcome $u_i \to u'_{i+1}$ is consistent with one outcome at u_j and inconsistent with the other.

Lemma 3. *Suppose T is optimal. Let $u_1 \to \cdots \to u_{j+1}$ be any downward path in T. For $1 \le i \le j-1$, let δ_i be the number of ancestors u_s of u_i on the path such that outcomes $u_s \to u'_{s+1}$ and $u_i \to u'_{i+1}$ are consistent with opposite outcomes*

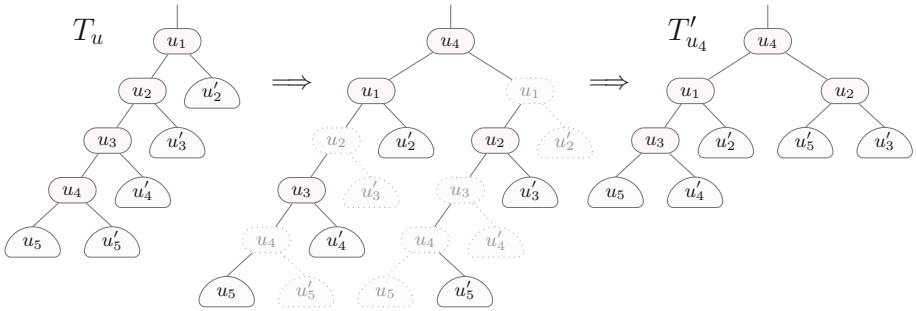


Fig. 4. Splitting a subtree T_u (where $u = u_1$) around descendant u_4. The figures in this section draw T_u by drawing u_i and u'_i as the left and right children of their parent u_{i-1}, so that the u_4-consistent path from u_1 is drawn as a prefix of the left spine. Each rounded half-circle represents a subtree, labeled with its root. Here outcomes $u_1 \to u'_2$ and $u_3 \to u'_4$ are consistent with the outcome $u_4 \to u_5$ at u_4 while outcome $u_2 \to u'_3$ is consistent with the other outcome $u_4 \to u'_5$. In the notation of Lemma 3 (taking $j = 4$) $\delta_2 = \delta_3 = 1$ and $\beta = 2$, and the lemma gives the bound $w(u'_2) \geq w(u_5) + 2w(u'_5)$.

at u_j. Let β be the number of ancestors u_s of u_{j-1} whose outcome $u_s \to u'_{s+1}$ is consistent with outcome $u_j \to u_{j+1}$ (so $0 \leq \beta \leq j-1$). Then

$$w(u'_2) \;\geq\; (j-1-\beta)\, w(u_{j+1}) + \beta w(u'_{j+1}) + \textstyle\sum_{i=3}^{j}(\delta_{i-1}-1)w(u'_i).$$

Lemma 4. *Suppose T is optimal. Let x be any test node, not necessarily in T. Let $u_1 \to \cdots \to u_{j+1}$ be a prefix of the x-consistent path from u_1. For $1 \leq i \leq j-1$, let δ_i be the number of ancestors u_s of u_i on the path such that outcomes $u_s \to u'_{s+1}$ and $u_i \to u'_{i+1}$ are consistent with opposite outcomes at u_j. Let β' be the number of ancestors u_s of u_j whose outcome $u_s \to u'_{s+1}$ is consistent with the yes-outcome of x (so $0 \leq \beta' \leq j$). Then*

$$w(u'_2) \;\geq\; \min(j-1-\beta', \beta'-1)w(u_j) + \textstyle\sum_{i=3}^{j}(\delta_{i-1}-1)w(u'_i).$$

The proofs (in the full paper) show that if the claimed inequalities are not met, then splitting the subtree T_u around u_j (for Lemma 3) or x (for Lemma 4) would give a cheaper optimal tree.

3.2 Structural Theorem

As in the previous section, for any downward path $u_1 \to u_2 \to \cdots \to u_j$, the sibling of u_i is denoted u'_i (for $2 \leq i \leq j$).

Theorem 3. *Suppose T is optimal. Let $u_1 \to u_2 \to \cdots \to u_d$ be any downward path in T (not necessarily starting at the root) such that $w(u'_2) < w(u_d)$. Then, for all different nodes u_i, u_j on the path, with $i, j < d$, both outcomes at u_i are consistent with outcome $u_j \to u_{j+1}$.*

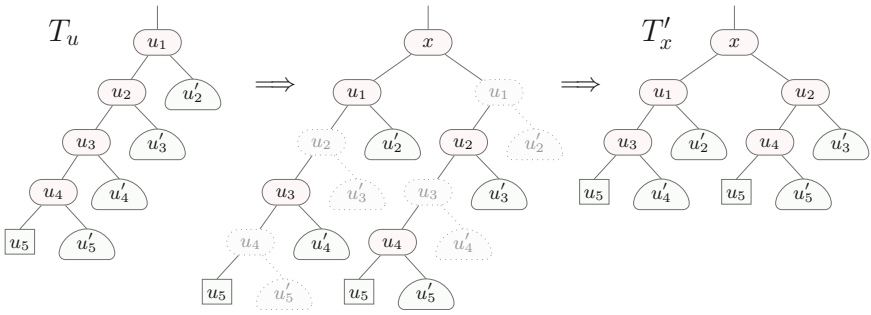


Fig. 5. Splitting a subtree T_u (where $u = u_1$) around a new node x (not equivalent to any node in T_u). The x-consistent path from u_1 is $u_1 \to \cdots \to u_5$. Here $u_1 \to u_2'$ and $u_3 \to u_4'$ are consistent with the left outcome at x, while $u_2 \to u_3'$ and $u_4 \to u_5'$ are consistent with the right outcome. In the notation of Lemma 4 (taking $j = 4$) $\delta_2 = \delta_3 = 1$ and $\beta' = 2$. The lemma gives the bound $w(u_2') \geq w(u_4)$.

For intuition, suppose a node u in T does an equality test $\langle = h \rangle$, and, in the no-subtree of u, some node x has $w(x) > w(h)$. By the theorem, then, the query value $q = h$ satisfies all outcomes along the path from the no-child of u to x.

The only property of the admissible tests that Theorem 3 relies on is laminarity. The proof (in the full paper) uses Lemma 3 twice, and a careful induction.

3.3 Proof of Theorem 1 (Some Optimal Tree is Admissible)

The proofs above rely only on laminarity. The proofs below use the particular structure of less-than and equality tests, and the properties of u-consistent paths. In particular, when x is an equality test, say x is $\langle = h \rangle$, the x-consistent path from u is the path that a search for h would take if started at u.

Lemma 5. *Suppose the instance has distinct weights and T is optimal. Consider any equality-test node $\langle = h \rangle$ and a key k with $w(k) > w(h)$ reaching this node. Then a search for h from the no-child of $\langle = h \rangle$ would end at the leaf L_k for k, and the path from $\langle = h \rangle$ to L_k has at most four nodes (including $\langle = h \rangle$ and L_k). Also, h is not in the class that T assigns to k.*

The proof (in the full paper) has several parts. First it applies Theorem 3 to the path from $\langle = h \rangle$ to L_k to show that, for any two different test nodes u_i, u_j along the path, both outcomes at u_i are consistent with $u_j \to u_{j+1}$. This (with $i = 1$) implies that a search for h starting at u_2 ends at L_k. A local-exchange argument shows h is not in the class that T assigns to k. That the path has at most four nodes then follows from Lemma 4 and local-exchange arguments.

Lemma 6. *If the instance has distinct weights, every irreducible optimal tree is admissible.*

The lemma follows from a careful application of Lemma 5 and the definition of admissible trees. Here is the full proof.

Proof. Let T be any irreducible optimal tree. Consider any node u in T. To prove the lemma we show that u's query set is admissible. If Q_u has no light holes, then we are done, so assume otherwise. Let $k^* = k^*(Q_u)$ be the heaviest key reaching u. Let $H_u = \mathsf{holes}(Q_u) \cap \mathsf{lighter}(k^*)$ be the set of light holes at u and $b = |H_u|$. Let c be the class that T assigns to k^* and $S = [\min Q_u, \max Q_u]_K \cap \mathsf{lighter}(k^*) \backslash c$. We want to show $H_u = \mathsf{heaviest}_b(S)$ and $b \in [3]$.

First we show $H_u \subseteq S$. By definition, $H_u \subseteq [\min Q_u, \max Q_u]_K \cap \mathsf{lighter}(k^*)$. For any light hole $h \in H_u$, key k^* is heavier than h and reaches the ancestor $\langle = h \rangle$ of u. Applying Lemma 5 to that ancestor, hole h is not in c. It follows that $H_u \subseteq S$.

Next we show $H_u = \mathsf{heaviest}_b(S)$. Suppose otherwise for contradiction. That is, there are $k \in S \backslash H_u \subseteq Q_u$ and $h \in H_u$ such that k is heavier than h. Keys k^* and k reach the ancestor $\langle = h \rangle$ of u. Applying Lemma 5 (twice) to that ancestor, the search path for h starting from the no-child of $\langle = h \rangle$ ends both at L_{k^*} and at the leaf L_k for k. So $L_k = L_{k^*}$, which implies that k is in c, contradicting $k \in S$. Therefore $H_u = \mathsf{heaviest}_b(S)$.

Finally, we show that $b \leq 3$. Let $h \in H_u$ be the light hole whose test node $\langle = h \rangle$ is closest to the root. Key k^* reaches $\langle = h \rangle$ and weighs more than h. Applying Lemma 5 to $\langle = h \rangle$ and key k^*, the path from $\langle = h \rangle$ to L_{k^*} has at most four nodes (including the leaf). Each light hole has a unique equality-test node on that path. So (using that u is on this path) there are at most three light holes in Q_u. □

Finally we prove Theorem 1:

Theorem 1. *If the instance is feasible, then some optimal tree is admissible.*

The proof (in the full paper) is a somewhat subtle perturbation argument, showing (informally) that every instance I is "arbitrarily close" to a distinct-weights instance I^* that shares the same set of admissible trees, and such that any optimal tree for I^* is also optimal for I. By Lemma 6, I^* has an optimal tree that is admissible, which is therefore also optimal and admissible for I.

4 Algorithm

This section proves Theorem 2, that the problem admits an $O(n^3 m)$-time algorithm. The input is an arbitrary 2WCDT instance $(Q, w, \mathcal{C}, K)$. In this section, for any $R \subseteq Q$ redefine $\mathsf{cost}(R)$ to be the minimum cost of any *admissible* tree for the subproblem $\pi(R) = (R, w, \mathcal{C}, K)$ obtained by restricting the query set to R. (Take $\mathsf{cost}(R) = \infty$ if there is no admissible tree for $\pi(R)$.) The algorithm returns $\mathsf{cost}(Q)$, the minimum cost of any admissible tree for $(Q, w, \mathcal{C}, K)$. By Theorem 1, this equals the minimum cost of any tree.

The algorithm computes $\mathsf{cost}(Q)$ by using memoized recursion on the following recurrence relation:

Recurrence 1. *For any $R \subseteq Q$,*

$$\mathsf{cost}(R) = \begin{cases} \infty & (R \notin \mathcal{A}) \\ 0 & (R \in \mathcal{A} \wedge (\exists c \in \mathcal{C})\, R \subseteq c) \\ w(R) + \min_u \left(\mathsf{cost}(R_u^{\mathsf{yes}}) + \mathsf{cost}(R_u^{\mathsf{no}})\right), & (\textit{otherwise}) \end{cases}$$

where above $\mathcal{A}$ denotes the set of admissible query subsets of Q (per Definition 4), $(R_u^{\mathsf{yes}}, R_u^{\mathsf{no}})$ is the bipartition of R into those values that satisfy test u and those that don't, and u ranges over the allowed tests (per Definition 1) such that R_u^{yes} and R_u^{no} are admissible. (If there are no such tests then the minimum is infinite.)

There are $O(n^2 m)$ admissible query sets. (Indeed, for any admissible set R, if R has no light holes it is determined by the triple $(\min R, \max R, k^*(R))$. Otherwise, per Definition 4, R is determined by a tuple $(\min R, \max R, k^*(R), b, c)$, where $(b, c) \in [3] \times \mathcal{C}$ with $k^*(R) \in c$.) So $O(n^2 m)$ subproblems arise in recursively evaluating $\mathsf{cost}(Q)$. To finish we describe how to evaluate the right-hand side of Recurrence 1 for a given R in $O(n)$ amortized time.

Assume (by renaming elements in Q in a preprocessing step) that $Q = [n]$. Given a non-empty query set $R \subseteq Q$, define the *signature* of R to be

$$\tau(R) = (\min R, \max R, k^*(R), H(R)),$$

where $H(R) = \mathsf{holes}(R) \cap \mathsf{lighter}(k^*(R))$ is the set of light holes in R.

For any R, its signature is easily computable in $O(n)$ time (for example, bucket-sort R and then enumerate the hole set $[\ell, r]_Q \setminus R$ to find $H(R)$). Each signature is in the set

$$\mathcal{S} = Q \times Q \times (K \cup \{\perp\}) \times 2^Q$$

of *potential signatures*. Conversely, given any potential signature $t = (\ell, r, k, H') \in \mathcal{S}$, the set $\tau^{-1}(t)$ with signature t, if any, is unique and computable from t in $O(n)$ time. (Specifically, $\tau^{-1}(t)$ is $Q_t = [\ell, r]_Q \setminus ((K \cap \mathsf{heavier}(k)) \cup H')$ provided Q_t is non-empty and has signature t.)

Lemma 7. *After an $O(n^3 m)$-time preprocessing step, given the signature $\tau(R)$ of $R \in \mathcal{A}$, the right-hand of Recurrence 1 is computable in amortized time $O(n)$.*

The proof of the lemma is in the full paper. Theorem 2 follows.

Extending the Algorithm to Other Inequality Tests. Our model considers decision trees that use less-than and equality tests. Allowing the negations of these tests is a trivial extension. (E.g., every greater-than-or-equal test $\langle \geq k \rangle$ is equivalent by swapping the children to the less-than test $\langle < k \rangle$.) We note without proof that our results also extend easily to the model that allows less-than-or-equal tests (of the form $\langle \leq k \rangle$). The proof of Theorem 3 requires only a minor adjustment: such tests need to be taken into account when proving the first claim in the proof of Lemma 5; the extended algorithm then allows such tests in Recurrence 1.

Acknowledgements. Thanks to Mordecai Golin and Ian Munro for introducing us to the problem and for useful discussions.

References

1. Anderson, R., Kannan, S., Karloff, H., Ladner, R.E.: Thresholds and optimal binary comparison search trees. J. Algorithms **44**, 338–358 (2002). https://doi.org/10.1016/S0196-6774(02)00203-1
2. Chambers, C., Chen, W.: Efficient multiple and predicated dispatching. In: Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA 1999), Denver, Colorado, USA, 1–5 November 1999, pp. 238–255 (1999)
3. Chambers, C., Chen, W.: Efficient multiple and predicated dispatching. SIGPLAN Not. **34**(10), 238–255 (1999). https://doi.org/10.1145/320385.320407
4. Chrobak, M., Golin, M., Munro, J.I., Young, N.E.: A simple algorithm for optimal search trees with two-way comparisons. ACM Trans. Algorithms **18**(1), 2:1–2:11 (2021). https://doi.org/10.1145/3477910
5. Chrobak, M., Golin, M., Munro, J.I., Young, N.E.: On Huang and Wong's algorithm for generalized binary split trees. Acta Informatica **59**(6), 687–708 (2022). https://doi.org/10.1007/s00236-021-00411-z
6. Chrobak, M., Young, N.E.: Classification via two-way comparisons (2023). https://doi.org/10.48550/ARXIV.2302.09692
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 4th edn. The MIT Press, Cambridge (2022)
8. Dagan, Y., Filmus, Y., Gabizon, A., Moran, S.: Twenty (simple) questions. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, 19–23 June 2017, pp. 9–21 (2017). https://doi.org/10.1145/3055399.3055422
9. Gilbert, E., Moore, E.: Variable-length binary encodings. Bell Syst. Tech. J. **38**(4), 933–967 (1959). https://doi.org/10.1002/j.1538-7305.1959.tb01583.x
10. Hester, J.H., Hirschberg, D.S., Huang, S.H., Wong, C.K.: Faster construction of optimal binary split trees. J. Algorithms **7**, 412–424 (1986). https://doi.org/10.1016/0196-6774(86)90031-3
11. Huang, S.H.S., Wong, C.K.: Generalized binary split trees. Acta Informatica **21**(1), 113–123 (1984). https://doi.org/10.1007/BF00289143
12. Huang, S.H.S., Wong, C.K.: Optimal binary split trees. J. Algorithms **5**, 69–79 (1984). https://doi.org/10.1016/0196-6774(84)90041-5
13. Hyafil, L., Rivest, R.L.: Constructing optimal binary decision trees is NP-complete. Inf. Process. Lett. **5**(1), 15–17 (1976). https://doi.org/10.1016/0020-0190(76)90095-8
14. Knuth, D.E.: Optimum binary search trees. Acta Informatica **1**, 14–25 (1971). https://doi.org/10.1007/BF00264289
15. Knuth, D.E.: The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd edn. Addison-Wesley Publishing Company, Redwood (1998)
16. Perl, Y.: Optimum split trees. J. Algorithms **5**, 367–374 (1984). https://doi.org/10.1016/0196-6774(84)90017-8
17. Sheil, B.A.: Median split trees: a fast lookup technique for frequently occurring keys. Commun. ACM **21**, 947–958 (1978). https://doi.org/10.1145/359642.359653
18. Spuler, D.: Optimal search trees using two-way key comparisons. Acta Informatica **31**(8), 729–740 (1994). https://doi.org/10.1007/BF01178732
19. Spuler, D.A.: Optimal search trees using two-way key comparisons. Ph.D. thesis, James Cook University (1994)