Classification via Two-Way Comparisons

MAREK CHROBAK and NEAL E. YOUNG, University of California Riverside, Riverside, CA, United States

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 tests to determine q's class. Such a tree can be faster and smaller than a conventional search tree and smaller than a lookup table (both of which must identify q, not just its class). We give the first polynomial-time algorithm for the problem. The algorithm extends naturally to the setting where each query has multiple allowed classes.

CCS Concepts: • **Theory of computation → Sorting and searching**;

Additional Key Words and Phrases: Data structures, algorithms, optimal search trees, classification

ACM Reference format:
Marek Chrobak and Neal E. Young. 2025. Classification via Two-Way Comparisons. *ACM Trans. Algor.* 21, 2, Article 17 (February 2025), 19 pages.
https://doi.org/10.1145/3709361

1 Introduction

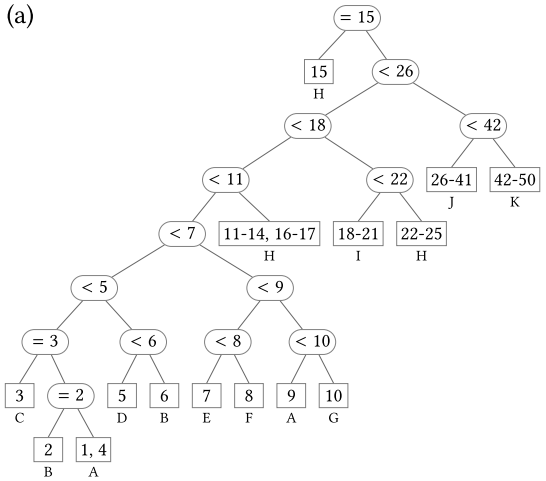
Given a weighted, ordered *query* set Q partitioned into classes, we study the problem of computing a decision tree that, given any query $q \in Q$, uses equality tests (e.g., "$q = 4$?") and less-than tests (e.g., "$q < 7$?") to quickly determine q's class. We call such a tree a **two-way-comparison decision tree (2WDT)**. Figure 1 shows an example. In the special case where each class is a singleton (so identifies the query), we call such a tree a **two-way-comparison search tree (2WST)**. The goal is to find a 2WDT of minimum *cost*, defined as 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 processing query q.

Whereas search trees and lookup tables must identify the query q (or the inter-key interval that q lies in), a decision tree needs only to identify q's class, so can be faster and smaller than a conventional search tree, and smaller than a lookup table. Consequently, decision trees are used in applications such as dispatch trees, which allow compilers and interpreters to quickly resolve method implementations for objects declared with type inheritance [3, 4]. (Each type is assigned a numeric ID via a depth-first search of the inheritance digraph. For each method, its tree maps each type ID to its method resolution.) Chambers and Chen [3, 4] give a heuristic to construct low-cost 2WDTs, but leave open whether minimum-cost 2WDTs can be found in polynomial time.

An extended abstract of this article appears in WADS 2023 [7].
M. Chrobak received partial support from the National Science Foundation Grant CCF-2153723.
Authors' Contact Information: Marek Chrobak (corresponding author), University of California Riverside, Riverside, CA, United States; e-mail: marek@cs.ucr.edu; Neal E. Young, University of California Riverside, Riverside, CA, United States; e-mail: young.neal@gmail.com.

ACM 1549-6333/2025/2-ART17
https://doi.org/10.1145/3709361

(a)



(b)

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 2WDT for the problem instance shown on the right. The instance (but not the tree) is from [3, 4, Figure 6]. Each internal node represents a comparison between the given query and the node's key k: either an equality test, represented as "$=k$," or a less-than test, represented as "$<k$". 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 [3, 4], of cost 2305.

We give the first polynomial-time algorithm to find minimum-cost 2WDTs. It runs in time $O(n^4)$, where $n = |Q|$ is the number of distinct query values. This matches the best run-time known for the special case of 2WSTs. 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 $O(n^3 m)$, where m is the sum of the sizes of the classes.

Related Work. Decision trees of various kinds are ubiquitous in the areas of artificial intelligence, machine learning, and data mining, where they are used for data classification, clustering, and regression (see e.g., [2]). Here we study decision trees for one-dimensional data. Most work on such trees has focussed on search trees. Here is a summary of relevant work on optimal search trees.

The tractability of finding an optimal search tree depends heavily on the kind of tests that the tree may use. The most general case, allowing tests of membership in sets from any given family of subsets of Q, is NP-hard, even if all subsets have size at most three [14], or the family is required to be laminar [15]. 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 [8, §14.5] [17, §6.2.2]. We call such a tree a **three-way-comparison search tree (3WST)**, or 3WST for short (see Figure 2(a)) In a 3WST, the query values that reach any given node form an interval. The possible intervals naturally represent $O(n^2)$ dynamic-programming subproblems, leading to an $O(n^3)$-time algorithm for finding minimum-cost 3WSTs [10]. Knuth reduced the running time to $O(n^2)$ [16].

In practice each three-way comparison is sometimes implemented by doing two two-way tests: a less-than test followed by an equality test. Knuth [17, §6.2.2, Example 33] proposed exploring binary search trees that use these two types of tests directly in any combination, that is, 2WSTs as defined earlier. For the so-called *successful-queries* variant (defined later), assuming the query weights are normalized to sum to 1, there is always a 2WST whose cost exceeds the entropy of the

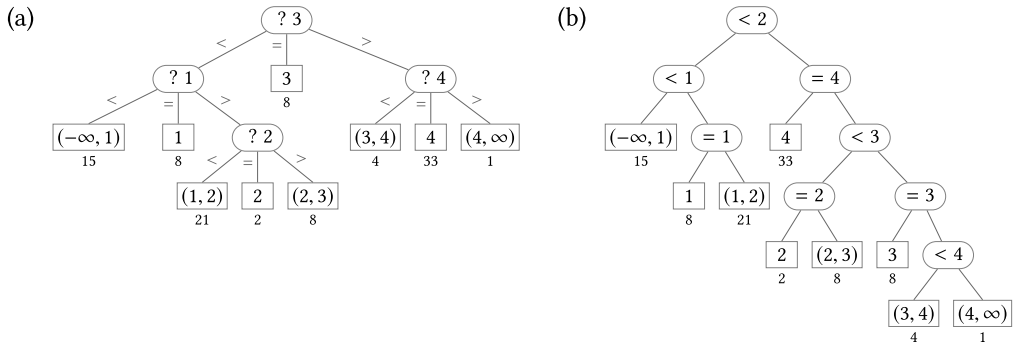


Fig. 2. Tree (a) is a 3WST. Tree (b) is a 2WST for the same instance. The query (or interval of queries) reaching each (rectangular) leaf is within the leaf. The weight of the query (or interval) is below the leaf.

weight distribution by at most 1 [9]. As the entropy is a lower bound on the cost of any binary search tree using arbitrary Boolean tests, this suggests that restricting to less-than and equality tests need not be too costly.

Stand-alone equality tests introduce an algorithmic obstacle not encountered with 3WSTs. Namely, while (analogously to 3WSTs) each node of a 2WST is naturally associated with an interval of queries, not all queries from this interval necessarily reach the node, so the dynamic program for 3WSTs does not extend easily to 2WSTs. This led early works to focus on restricted classes of 2WSTs, namely *median split trees* [19] and *binary split trees* [11, 13, 18]. These, by definition, constrain the use of equality tests so as to sidestep the obstacle they introduce. *Generalized binary split trees* are less restrictive, but the only algorithm proposed to find them [12] is incorrect [6]. Likewise, the recurrence relations underlying the first algorithms proposed to find minimum-cost 2WSTs (which were given without proof [20, 21]) are demonstrably wrong [6].

Spuler conjectured in 1994 that every 2WST instance has an optimal tree with the *heaviest-first* property: namely, in each equality-test node, *the comparison key is the heaviest among keys that reach the node* [21]. In 2002 Anderson et al. proved the conjecture for successful-queries 2WSTs, leading to the first polynomial-time algorithm for that variant [1]. The algorithm runs in $O(n^4)$ time. In 2021, Chrobak et al. simplified their result (in particular, the handling of equal-weight keys, as discussed later) to obtain an $O(n^4)$-time algorithm to find optimal 2WSTs (both variants) [5]. These 2WST algorithms do not extend easily to 2WDTs, because some 2WDT instances have no optimal tree with the heaviest-first property. Figure 3 gives an example.

Our Contributions. The *rotation* operation is a standard tool for studying 2WSTs with only less-than tests (and 3WSTs). Following [1] and [5] we use a generalized rotation that applies to 2WSTs with both types of tests. We generalize it further, to decision trees T such that the test at each internal node u is a test of membership in some set $X_u \subseteq Q$, subject only to the constraint that the collection of such test sets $\{X_u : u \in T\}$ is laminar. For each such node u, the edge to one child is associated with X_u while the edge to the other child is associated with the complement $\overline{X}_u = Q \setminus X_u$. Given any query q, the search for q in T follows the unique root-to-leaf path whose edges' sets all contain q. We call such trees *laminar decision trees*, or LDTs for short (see Section 1.1).

Suppose that, in such a laminar decision tree T, there is an "imbalance" in the tree: for some downward path $u_1 \to u_2 \to \cdots \to u_d$, the sibling u_2' of u_2 is lighter than u_d. (That is, $w(u_2') < w(u_d)$, where $w(u)$, the *weight* of node u, is as usual the total weight of the queries that reach the node.) Then, Theorem 2.1 (Section 2) states that, if T is optimal, *the sets associated with the edges leaving the path* $u_1 \to \cdots \to u_d$ *must be pairwise disjoint.* (The edges leaving the path are

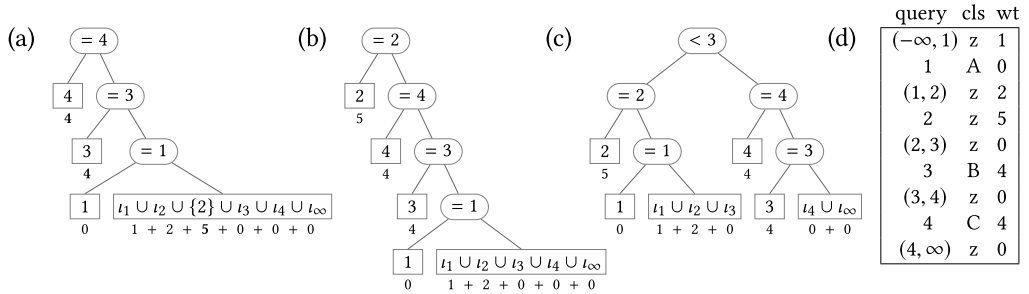


Fig. 3. Three trees for the 2WDT instance shown in (d). The set of queries reaching each (rectangular) leaf is shown within the leaf (to save space, there ι_i denotes the inter-key open interval with right boundary i, e.g. $\iota_1 = (-\infty, 1)$, $\iota_2 = (1, 2)$). The associated weights are below 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. (Note: in our formal model, the inter-key intervals will be represented by virtual non-key queries.)

$\{u_i \to u'_{i+1} : 1 \le i < d\}$, where, for any node u other than the root, u' denotes the sibling of u in T.) This theorem generalizes the key structural theorems of [1] and [5]; in particular, it implies the heaviest-first property for 2WSTs.

Section 3 then proves Theorem 3.1, which strengthens Theorem 2.1 specifically for trees with less-than and equality tests, that is, 2WSTs. Section 4 uses Theorem 3.1 to prove Theorem 4.5, that there is always an optimal tree that is *admissible*. This means roughly that, at each equality-test node $\langle = h \rangle$ in the tree, if the key h is not the heaviest key reaching the node, it must be one of at most three other suitably restricted keys (Definition 4.3). A careful implementation then yields the main result (Theorem 5.1 in Section 5): an $O(n^3 m)$-time dynamic-programming algorithm to find a minimum-cost 2WDT.

The Role of Distinct Key Weights. The discussion above glosses over a secondary technical obstacle for 2WSTs. For 2WST instances whose key weights are *distinct*, the heaviest-first property determines the key of each equality test uniquely, so that the queries that reach any given node in a 2WST (with the property) must form one of $O(n^4)$ predetermined subsets, leading naturally to a dynamic program with $O(n^4)$ subproblems. But this uniqueness is lost when key weights are not distinct. This obstacle turns out to be more challenging than one might expect. Indeed, there are instances with non-distinct weights for which, for every non-empty subset S of Q, there is a 2WST that has the heaviest-first property, and a node u such that the set of queries reaching u is S. One cannot just break ties naively: 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' [5, 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 $O(n^5)$ [11].

Nonetheless, using a perturbation argument Chrobak et al. [5] show that an arbitrary 2WST instance can indeed be handled as if it is a distinct-weights instance just by breaking ties among equal weights in a globally consistent way.We use the same approach here for 2WDTs.

1.1 Definitions

An instance I of the *laminar decision tree* problem (LDT) is specified by a tuple $I = (Q, w, C, \mathcal{F})$, where Q is a finite, totally ordered, non-empty set of *queries*, with each query $q \in Q$ assigned a

weight $w(q) \geq 0$, the set $C \subseteq 2^Q$ is a collection of query *classes* (with each class having a unique identifier), and $\mathcal{F} \subseteq 2^Q \setminus \{\emptyset, Q\}$ is laminar. Call each set $X \in \mathcal{F}$ a *test*, with two *outcomes*: X (the *yes* outcome), and $\overline{X} = Q \setminus X$ (the *no* outcome). Let n and m denote, respectively, $|Q|$ and $\sum_{c \in C} |c|$. A *decision tree* for I is a rooted binary tree T where each non-leaf node u has an associated test $X_u \in \mathcal{F}$, with the edge to one child of u associated with the yes-outcome X_u, and the edge to the other child of u associated with the no-outcome $\overline{X}_u$. Each leaf node u is labeled with (the identifier of) some class $c_u \in C$, which must contain the intersection of the outcomes of the edges along the path from the root to u (this intersection is comprised of those queries $q \in Q$ whose search, as defined next, ends at u).

For each $q \in Q$, the *search for q in T* follows the (unique) root-to-leaf path of edges whose outcomes all contain q. Call this path q's *search path*. Say that *$q$reaches* each node on this path. Call the leaf that q reaches *q's leaf*. Define q's *depth* (in T) to be the depth of q's leaf (equivalently, the number of tests on q's search path). The *cost* of T is the weighted sum of the depths of all queries in Q (where each query $q \in Q$ has weight $w(q)$). A solution for I is a decision tree for I of minimum cost.

A decision tree T is called *irreducible* if, for each node u in T, (i) at least one query in Q reaches u, and (ii) if any class $c \in C$ contains all the queries that reach u, then u is a leaf. Any decision tree can easily be converted into an irreducible tree without increasing its cost, so we generally restrict attention to irreducible trees. As we shall see, in an irreducible tree T, each non-leaf node u has a distinct test X_u and each edge $u \to v$ has a distinct outcome, so, when convenient, we identify each node u with its test X_u and identify each edge $u \to v$ with its outcome (X_u or $\overline{X}_u$).

Note that an LDT 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, it must have the property that each set of queries that cannot be separated by tests in $\mathcal{F}$ (that is, for each test $X \in \mathcal{F}$ either this set is a subset of X or is disjoint with X) must be contained in some class.

An *equality test with key k* is the test (set) $\{k\}$. A *less-than test with key k* is the test (set) $\{q \in Q : q < k\}$. The 2WDT problem is the restriction of LDT to instances in which, for some set $K \subseteq Q$ of *keys*, $\mathcal{F}$ is comprised of the equality and less-than tests whose keys are in K. (It is straightforward to verify that this is a laminar family.) In this context we denote the instance as $I = (Q, w, C, K)$.

The assumption $K \subseteq Q$ is for ease of presentation. Also, we can assume without loss of generality that each query belongs to some class, so $m \geq n = |Q|$ and the input size[1] is $\Theta(n + m) = \Theta(m)$. As discussed at the end of Section 5, although our definition of 2WDTs allows only less-than and equality tests, all results extend easily to the other standard inequality tests.

Successful-Queries Variants. Conventionally, in the *successful-queries* variants of binary search-tree problems, 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. Such queries are called *successful*. In the *standard* variants, the input is augmented with a weight for each open interval between consecutive keys (and before the minimum key and after the maximum key). *Unsuccessful* queries, that is, queries to values within these intervals, are also allowed. They must be answered by returning the interval in which the query falls. Our definition of 2WDTs captures both variants: restricting to $Q = K$ gives the successful-queries variant, while the standard variant can be modeled by adding one non-key query within each open interval to Q.

[1]Note that $\mathcal{F}$, being laminar, can be encoded as a tree in space $O(n)$.

2 Imbalance Theorem for Trees with Laminar Tests

This section states and proves Theorem 2.1 (the imbalance theorem):

Theorem 2.1. *Let T be any optimal, irreducible tree for an LDT instance $I = (Q, w, C, \mathcal{F})$. Let $u_1 \to u_2 \to \cdots \to u_d$ be the downward path from any node u_1 to any proper descendant u_d in T such that $w(u_2') < w(u_d)$. Then the outcomes leaving $u_1 \to \cdots \to u_d$ are pairwise disjoint.*

The outcomes leaving $u_1 \to \cdots \to u_d$ are $\{u_i \to u_{i+1}' : 1 \le i < d\}$. Note that this does not include any outcome out of u_d. Recall that in T each node u is identified with its test set X_u, each edge $u \to v$ is identified with its outcome X_u or $\overline{X}_u$, and u' denotes the sibling of u, unless u is the root.

Intuition. The theorem considers how, in an optimal 2WDT, it can happen that a node (u_d) can be heavier than the sibling (u_2') of some ancestor (u_2). If this happens, then it must be that we can't rotate the node up the tree above its ancestor. The theorem says that this can happen only if the outcomes leaving the path from the ancestor to the node are disjoint.

Here are some examples to build intuition. Let T be as assumed in the theorem. First suppose that each test along the path $u_1 \to \cdots \to u_d$ with $w(u_2') < w(u_d)$ is a less-than test. Then each outcome leaving the path contains either $\min Q$ or $\max Q$, so, by the theorem, at most two edges leave the path (at most one containing $\min Q$, at most one containing $\max Q$). That is, $d \le 3$.

Another consequence: for any downward path $x \to y \to z$ in T, the weight $w(y')$ of the sibling of y is at least $\min\big(w(z), w(z')\big)$. (Otherwise, applying the theorem to the path $x \to y \to z$, and then to the path $x \to y \to z'$, the outcome $x \to y'$ is disjoint from outcomes $y \to z'$ and $y \to z$, so the outcome $x \to y'$ would be empty, contradicting the definition of LDTs.)

Finally, for any equality test $\langle = k\rangle$ in T, for any proper ancestor a of $\langle = k\rangle$, the weight $w(a')$ of the sibling of a (if there is one) is at least $w(k)$. (Otherwise, let p be the parent of a. Let L_k be the yes-child of $\langle = k\rangle$. Then the theorem applies to the path $p \to a \to \cdots \to \langle = k\rangle \to L_k$, so the outcome of $p \to a'$ is disjoint from the outcome of $\langle = k\rangle \to L_k'$, so must be a subset of the outcome of $\langle = k\rangle \to L_k$, i.e., the singleton $\{k\}$. So the outcome $p \to a'$ is either empty, contradicting the definition of 2WDTs, or also $\{k\}$, contradicting the irreducibility of T.) As a special case every equality-test ancestor $\langle = h\rangle$ of $\langle = k\rangle$ satisfies $w(h) \ge w(k)$.

In fact, Theorem 2.1 generalizes the key structural theorems of [1] and [5] for 2WSTs. For instance, the heaviest-first property of 2WSTs follows easily from the above paragraph. Indeed, fix an optimal, irreducible 2WST tree T. Assume without loss of generality that, if the parent of any leaf L_k in T is a test node $\langle = h\rangle$, then $w(h) \ge w(k)$. (Otherwise just change the parent to $\langle = k\rangle$, making L_k the yes-child.) To show that T has the heaviest-first property, consider any test node $\langle = h\rangle$ whose no-subtree has a leaf L_k for a key k. We will show $w(k) \le w(h)$. In the case that L_k is a child of $\langle = h\rangle$, then the previous assumption implies $w(k) \le w(h)$. So assume that L_k is not a child of $\langle = h\rangle$. If the parent of L_k is not already $\langle = k\rangle$, consider replacing that parent by $\langle = k\rangle$, making L_k the yes-child. This preserves optimality and correctness. Now $w(h) \ge w(k)$ follows from the last sentence in the previous paragraph, applied to the (possibly modified) tree.

The Generalized Rotation. Next we lay the groundwork for the proof of Theorem 2.1. Fix an LDT instance $I = (Q, w, C, \mathcal{F})$. Say tests $X, Y \in \mathcal{F}$ are *equivalent* if $X = Y$ or $X = \overline{Y}$. We'll use only the following property of $\mathcal{F}$, which is essentially[2] a restatement of laminarity:

Property 1. *Given two non-equivalent tests $X, Y \in \mathcal{F}$, among the four pairs of outcomes in $\{X, \overline{X}\} \times \{Y, \overline{Y}\}$, exactly one pair are disjoint.*

[2]Property 1 is a-priori weaker than laminarity, but any family $\mathcal{F}$ with Property 1 can be converted into an equivalent laminar family $\mathcal{F}'$ by fixing any element $q_0 \in Q$ and taking $\mathcal{F}' = \{X \in \mathcal{F} : q_0 \notin X\} \cup \{\overline{X} : X \in \mathcal{F}, q_0 \in X\}$.

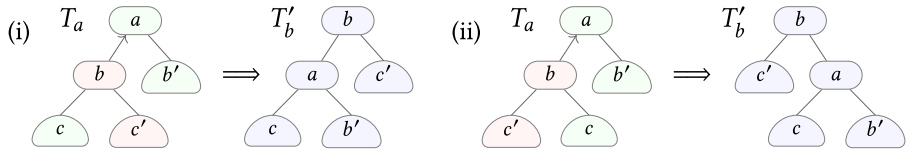


Fig. 4. Rotating a non-root test node b in T moves b (along with its preferred child c' and the subtree rooted at c') above its parent a. Unlike binary search trees, laminar search trees are not inherently ordered. When drawing a rotation in a laminar tree, we draw the first tree T_a using any convenient order, then, when drawing the rotated tree T'_b, order each node's two outcomes the same as they were ordered in T_a. Above, (i) and (ii) are two ways of drawing the exact same rotation. Throughout, u' denotes the sibling of a given node u in the original tree T, in which T_a is a subtree.

Fix an irreducible tree T for I.

Property 2. *Let u and v be distinct non-leaf nodes in an irreducible decision tree T for I. Then (i) the tests at u and v are not equivalent. If u is a proper ancestor of v then (ii) the outcome from u on the path from u to v overlaps with both outcomes from v, while (iii) the other outcome from u (the one leaving the path from u to v) is a subset of one outcome from v, and disjoint from the other outcome from v.*

Proof. Part (ii) follows directly from the irreducibility of T. In the case when u is an ancestor of v, Part (ii) implies both Part (i) and, using Property 1, Part (iii), so we are done in this case. Since Part (i) (non-equivalence) holds when u is an ancestor of v, it also holds when v is an ancestor of u, just by reversing their roles. To finish we show Part (i) when neither is an ancestor of the other.

Suppose for contradiction that u and v are equivalent. Let a be the lowest common ancestor of u and v. Let $a \to b$ and $a \to b'$ be the outcomes from a leading towards u and v, respectively. Part (ii) holds for a and u, so $a \to b$ overlaps both outcomes from u. By the same reasoning (reversing the roles of u and v) outcome $a \to b'$ overlaps both outcomes from v, implying (by the equivalence of v and u) that $a \to b'$ overlaps both outcomes from u. So both outcomes at a overlap both outcomes at u, contradicting Property 1. □

Here is some hopefully mnemonic terminology:

Definition 2.2. Given an outcome $b \to c'$ in T from a non-root node b to child c', let a be the parent of b. Call the sibling b' of b the *uncle* of the child c'. If $b \to c'$ is the outcome at b that is disjoint from the outcome $a \to b'$ from the grandparent to the uncle, say that the child c' and the outcome $b \to c'$ are *preferred* by b.

By Property 2(iii), b has exactly one preferred child and one preferred outcome, which leads to that child. Also, the outcome $a \to b'$ from the grandparent to the uncle is a subset of the non-preferred outcome $b \to c$ at b.

Definition 2.3. Given a non-root test node b, let a be the parent of b. *Rotating b (above a)* replaces the subtree T_a rooted at a in T with the subtree T'_b obtained from T_a as shown in Figure 4, that is, it exchanges the nodes a and b along with the subtrees rooted at their respective children b' and c'.

Next, we show that the rotation operation is correct. To avoid confusion, note that, when considering a sequence of trees derived from T, the notation u' always denotes the sibling of node u *in* T, which is not necessarily the sibling of u in subsequent trees. Likewise, the notation $u \to v$ always denotes the outcome leading from u to v *in* T. The notation $u \xrightarrow{T'} v$ denotes the outcome leading from u to v in some subsequent tree T'.

Observation 1. *Let T' be obtained from T by rotating b up as described above. Then (i) T' is an irreducible decision tree for I, and (ii) the cost of T' is the cost of T plus $w(b') - w(c')$, so, provided T is optimal, $w(b') \geq w(c')$. That is, the preferred child c' cannot be heavier than its uncle b'.*

Proof. *Part (i).* Recall that the queries reaching a node are those in the intersection of all outcomes along the path from the root to the node. We will show that, for each leaf L, this set is the same in T as it is in T'.

If L is not a descendant of a, the path from the root to L does not change. If L is a descendant of c, this path changes but the set of outcomes on this path is the same in T and T'. It remains to consider the cases when L is a descendant of b' or c' in T.

In the case that L is a descendant of b', the only change to the path to L is the addition of the outcome $b \to c$. (In T' that outcome is now $b \xrightarrow{T'} a$.) But the path contains the outcome $a \to b'$, which (being disjoint from the preferred outcome $b \to c'$) is a subset of the non-preferred outcome $b \to c$. So the intersection is unchanged.

Similarly, in the remaining case (L is a descendant of c') the path loses $a \to b$ (which is $a \xrightarrow{T'} c$ in T'). But the path contains the preferred outcome $b \to c'$ which (being disjoint from $a \to b'$) is a subset of $a \to b$. So the intersection is unchanged.

Part (ii). The rotation increases the depth of each descendant of the uncle b' by one, while decreasing the depth of each descendant of the preferred child c' by one, thus increasing the tree cost by $w(b') - w(c')$. □

Each non-root test node has a preferred child, so by Observation 1 if T is optimal each non-root test node has a child that weighs no more than the child's uncle:

Observation 2. *Suppose T is optimal. For any non-root node u with children v and v', $w(u') \geq \min\big(w(v), w(v')\big)$.*

We now prove the theorem.

Proof of Theorem 2.1. Let T, $I = (Q, w, \mathcal{C}, \mathcal{F})$, and $u_1 \to u_2 \to \cdots \to u_d$ be as in the theorem statement, so $w(u'_2) < w(u_d)$. We claim that $w(u'_2) \geq w(u'_3) \geq \cdots \geq w(u'_d)$. Suppose otherwise for contradiction. Fix $j < d$ such that $w(u'_2) \geq w(u'_3) \geq \cdots \geq w(u'_j) < w(u'_{j+1})$. By Observation 2 and $w(u'_j) < w(u'_{j+1})$, it must be that $w(u'_j) \geq w(u_{j+1})$. Using this, the choice of j, and that u_d is a descendant of u_{j+1}, we have $w(u'_2) \geq w(u'_j) \geq w(u_{j+1}) \geq w(u_d)$, contradicting $w(u'_2) < w(u_d)$ and proving the claim.

The claim, and $w(u'_2) < w(u_d)$, and the ancestry relations imply

$$w(u'_d) \leq w(u'_{d-1}) \leq \cdots \leq w(u'_2) < w(u_d) \leq w(u_{d-1}) \leq \cdots \leq w(u_1). \tag{1}$$

Next suppose for contradiction that at least one pair of outcomes leaving the path overlaps. Fix such a pair $u_p \to u'_{p+1}$ and $u_q \to u'_{q+1}$ with $p < q < d$ such that the later outcome $u_q \to u'_{q+1}$ overlaps the earlier outcome $u_p \to u'_{p+1}$, but *is disjoint from* each outcome leaving the path between these two. (Formally, $u_q \to u'_{q+1}$ overlaps $u_p \to u'_{p+1}$ but is disjoint from each $u_i \to u'_{i+1}$ with $p < i < q$. Such a pair must exist. For example, fix any $q < d$ such that there is an earlier outcome leaving the path that overlaps $u_q \to u'_{q+1}$. Then, among the latter, take $u_p \to u'_{p+1}$ to be the one with maximum p.)

Now, as illustrated in Figure 5(a) and (b), rotate u_q up the sub-path $u_p \to u_{p+1} \to \cdots \to u_q$, ancestor by ancestor, just until u_q becomes the parent of u_p. That is, let T^{q-1} be the initial tree T, then, for each $i \leftarrow q-1, q-2, \ldots, p$ in decreasing order, let the next tree T^i be obtained from the

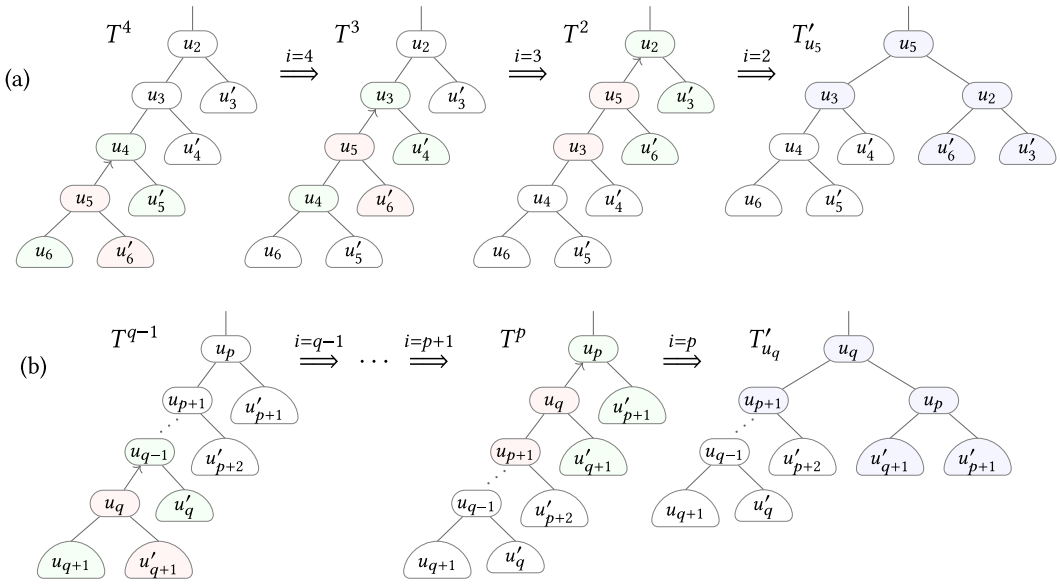


Fig. 5. The sequence of rotations in the proof of Theorem 2.1. The drawing orders the initial tree $T^{q-1} = T$ so the path $u_p \to \cdots \to u_q$ lies on the left spine. The case $(p, q) = (2, 5)$ is shown in (a). For the general case, (b) shows the first and last two trees in the sequence. In each rotation except the last, the preferred outcome of u_q is $u_q \to u'_{q+1}$. The preferred outcome is drawn to the right, so the rotation is of the form shown in Figure 4(i). It moves u_q (and the preferred outcome $u_q \to u'_{q+1}$) above u_i. Finally, in the last rotation, the preferred outcome of u_q is $u_q \to u_{q+1}$. The preferred outcome is drawn to the left, so the rotation is of the form shown in Figure 4(ii). This rotation moves the root u_p down and out of the path.

previous tree T^{i+1} by rotating u_q above u_i. In each tree T^i except the last, the parent of u_q is u_i. The final tree T^{p+1} is obtained from T^p by rotating u_q above u_p.

For each rotation except the last (each $i > p$), by the choice of q and p, the outcome leaving u_q that is disjoint from $u_i \to u'_{i+1}$ is $u_q \to u'_{q+1}$ (in both the original tree T and the current tree T^i). So $u_q \to u'_{q+1}$ is the preferred outcome for this rotation, and the rotation is as illustrated in Figure 5(a) and (b). The preferred outcome is drawn to the right, so takes the form shown in Figure 4(i). It moves u_q (and the preferred outcome $u_q \to u'_{q+1}$) above u_i. Thus, just before the final rotation, the tree (T^p) is as shown in Figure 5(a) and (b), with u_q (and the preferred outcome $u_q \to u_{q+1}$) just below u_p. (The tree T^p could also be obtained directly from T by just deleting the three edges in $u_{q-1} \to u_q \to u_{q+1}$ and $u_p \to u_{p+1}$and replacing them by the three edges $u_{q-1} \to u_{q+1}$ and $u_p \to u_q \to u_{p-1}$.) The final rotation then rotates u_q above u_p. By the choice of p, the preferred outcome at u_q for this rotation is $u_q \to u_{q+1}$ (in T; in the current tree T^p this outcome is $u_q \xrightarrow{T^p} u_{p+1}$). So the rotation is as illustrated on the right of Figure 5(a) and (b), where the preferred outcome is drawn as the *left* outcome of u_q, so is drawn in the form shown in Figure 4(ii). This rotation moves u_p down and out of the path.

By inspection of the first and last trees in Figure 5(b), rotating u_q (with u'_{q+1}) all the way up the path and then rotating u_p out of the path in the final rotation changes the leaf depths as follows. The depths of descendants of u_{q+1} decrease by one, as they lose the ancestor u_p. The depths of descendants of u'_{q+1} decrease by $q - p - 1 \geq 0$, as they lose ancestors $u_{p+1}, \ldots, u_{q-1}$. The depths of descendants of u'_{p+1} increase by one, as they gain the ancestor u_q, which is rotated above them. The depths of other leaves in the subtree T_{u_q} don't change, as they gain ancestor u_q but lose u_p.

Hence, the increase in cost is at most $w(u'_{p+1}) - w(u_{q+1})$. From the optimality of T it follows that $w(u'_{p+1}) \geq w(u_{q+1})$, contradicting (1) and proving Theorem 2.1. □

3 Structural Theorem for 2WDTs

This section proves Theorem 3.1, below, which is an intermediate step towards proving the existence of an admissible tree. The proof uses Theorem 2.1, a "bisection" operation (a generalization of the rotation operation), and specific properties of inequality and equality tests. The example in Figure 3 may be helpful in developing intuition for the theorem.

Let T be an arbitrary irreducible tree for an arbitrary 2WDT instance (Q, w, C, K). Recall that, since we are working with classification rather than search, the leaf L_k for a key k may have additional queries in its query set.

Theorem 3.1. *Suppose the instance has distinct weights and T is optimal. Consider any equality-test node $\langle = h\rangle$ in T and a key k with $w(k) > w(h)$ reaching this node. Then (i) a search for h from the no-child of $\langle = h\rangle$ would end at the leaf L_k for k, and (ii) the path from $\langle = h\rangle$ to L_k has at most four nodes (including $\langle = h\rangle$ and L_k). (iii) Also, h is not in the class that T assigns to k.*

Proof. Let $u_1 \to u_2 \to \cdots \to u_d$ be the path from $\langle = h\rangle$ to L_k. As usual, the outcomes leaving the path are $\{u_i \to u'_{i+1} : 1 \leq i < d\}$. So u_1 is $\langle = h\rangle$, while u'_2 is the leaf for h, and u_d is L_k. As $w(u_d) = w(L_k) \geq w(k) > w(h) = w(u'_2)$, the imbalance theorem (Theorem 2.1) applies to the path. The theorem implies the following observation:

Observation 3. *The outcomes leaving the path are pairwise disjoint.*

The yes-outcome $u_1 \to u'_2$ of $\langle = h\rangle$ leaves the path, so by Observation 3 that outcome, that is, $\{h\}$, is disjoint with all other outcomes leaving the path. Hence, a search for h starting from the no-child u_2 of $\langle = h\rangle$ would not leave the path, so would end at L_k. This proves Part (i) of the theorem.

To prove Part (iii), suppose for contradiction that h is in the class that T assigns to k. Then, in the case $d = 2$, we could replace the node $\langle = h\rangle$ by a leaf labeled with the class assigned by T to k, contradicting irreducibility. So assume $d \geq 3$. By Part (i) of the theorem, changing the test key at $\langle = h\rangle$ to k (and relabeling u'_2 with a class containing k) would give a correct tree, while decreasing the cost by $(w(k) - w(h))(d - 2)$. By assumption $w(k) > w(h)$, so $(w(k) - w(h))(d - 2) > 0$, and thus the modification would give a correct tree strictly cheaper than T, contradicting the optimality of T.

The rest of this section proves Part (ii), that is, that d is at most 4. Assume for contradiction that $d \geq 5$. We prove two independent lemmas.

Lemma 3.2. $2\,w(h) < w(u_3)$.

Proof. Consider inserting a new equality-test $\langle = k\rangle$ above u_3, that is, replacing T_{u_3} by a new equality test $\langle = k\rangle$ whose yes-child is a new leaf labeled with any answer that k accepts, and whose no-subtree is a copy of T_{u_3}. This increases the search depth of every query reaching u_3, except key k, by 1. It decreases the search depth of k by at least 1. Thus, the increase in cost is at most $(w(u_3) - w(k)) - w(k)$. With the optimality of T this implies $w(u_3) \geq 2\,w(k) > 2\,w(h)$. □

Let $k_1 \leq k_2 \leq k_3 \leq k_4$ be the comparison keys of the four tests in u_1, u_2, u_3, and u_4, sorted into non-decreasing order. Next we consider "bisecting" the subtree T_{u_1} by introducing test node $\langle < k_3\rangle$ as a new root and adjusting the rest of the tree appropriately.

Lemma 3.3. *Among the four outcomes $u_i \to u'_{i+1}$ (with $1 \leq i \leq 4$) leaving the path, two are disjoint with the yes-outcome of $\langle < k_3\rangle$, while the other two are disjoint with the no-outcome of $\langle < k_3\rangle$.*

Proof. We will show that k_3 has the desired property.

Suppose at least two of the four tests in u_1, u_2, u_3, and u_4 are inequality tests, say $\langle < k_i \rangle$ and $\langle < k_j \rangle$ with $i < j$. Then (using $k_i \leq k_j$) the yes-outcome of $\langle < k_i \rangle$ and the no-outcome of $\langle < k_j \rangle$ are disjoint. By Property 2 all other pairs of outcomes between the two nodes overlap. So, by Observation 3, *if there are two less-than tests* $\langle < k_i \rangle$ *and* $\langle < k_j \rangle$ *in* $\{u_1, u_2, u_3, u_4\}$ *with* $i < j$*, then the outcomes leaving the path from* $\langle < k_i \rangle$ *and* $\langle < k_j \rangle$ *are, respectively, the yes-outcome and the no-outcome.*

By the preceding sentence, $\{u_1, u_2, u_3, u_4\}$ contains at most two less-than tests, and therefore at least two equality tests. The yes-outcome of any equality test u_i is disjoint with some outcome of any u_j, so by Property 2 the no-outcome of u_i overlaps both outcomes of any u_j with $j \neq i$, and by Observation 3 *the outcomes leaving the path from the (at least two) equality tests are yes-outcomes.*

Suppose for contradiction that the yes-outcome of some less-than test $\langle < k_j \rangle$ with $k_j \neq k_1$ leaves the path. By the conclusion of the second-to-last paragraph above, and by $k_1 < k_j$, the test with key k_1 cannot be a less-than test, so must be $\langle = k_1 \rangle$. But then the yes-outcome of $\langle = k_1 \rangle$ overlaps the yes-outcome of $\langle < k_j \rangle$, contradicting Observation 3. So *if a yes-outcome leaves the path from any less-than test, the test's key is* k_1. By symmetric reasoning, *if a no-outcome leaves the path from any less-than test, the test's key is* k_4.

It follows that any inequality test in $\{u_1, u_2, u_3, u_4\}$ must be in u_1 and/or u_4, implying that u_2 and u_3 do equality tests, so $k_2 < k_3$.

For all $q \geq k_3$, none of the following hold: $q = k_1$, $q = k_2$ (using here $k_2 < k_3$), or $q < k_2$. So the no-outcome of $\langle < k_3 \rangle$ is disjoint with the yes-outcomes of $\langle = k_1 \rangle$, $\langle = k_2 \rangle$, and $\langle < k_1 \rangle$. By the conclusions of the preceding paragraphs, these include all outcomes that leave the path from the nodes with keys k_1 and k_2. Similarly, for all $q < k_3$, none of the following hold: $q = k_3$, $q = k_4$, or $q \geq k_4$, so the yes-outcome of $\langle < k_3 \rangle$ is disjoint with all outcomes that leave the path from the nodes with keys k_3 and k_4. This proves Lemma 3.3. □

Returning to the proof of Theorem 3.1(ii), consider replacing T_{u_1} in T by the subtree T' obtained by *bisecting* T_{u_1} around the new node $\langle < k_3 \rangle$, in the following two steps (shown in Figure 6). First, make a subtree with root $\langle < k_3 \rangle$, whose yes- and no-subtrees are each a copy of T_{u_1}. (Note that this subtree is a correct replacement for T_{u_1}.) For each outcome $u_i \to u'_{i+1}$ ($1 \leq i \leq 4$) that leaves the path $u_1 \to \cdots \to u_5$, per Lemma 3.3, the outcome is disjoint with either the yes-outcome or the no-outcome of $\langle < k_3 \rangle$. If the outcome $u_i \to u'_{i+1}$ is disjoint with the yes-outcome, *splice it out* from the yes-copy of T_{u_1}. Otherwise (it is disjoint with the no-outcome) splice it out from the no-copy of T_{u_1}.

Specifically, to *splice out* the copy of $u_i \to u'_{i+1}$ means to remove that copy of u_i and the subtree rooted at its child u'_{i+1} by replacing the subtree rooted at u_i by the subtree rooted at the current sibling of u'_{i+1} (the other child of u_i), as happens in Figure 6. The outcome from $\langle < k_3 \rangle$ that leads towards this copy of u_i is disjoint with the deleted outcome $u_i \to u'_{i+1}$, so every search that reached the (now spliced out) copy of u_i continued through the sibling, so splicing out this copy of $u_i \to u'_{i+1}$ preserves correctness.

By Lemma 3.3, two of the four outcomes are spliced out of the yes-copy of T_{u_1}, while the other two are spliced out of the no-copy, so the tree T' obtained by bisecting T_{u_1} around $\langle < k_3 \rangle$ has one of the three forms shown in Figure 7(a), (b), or (c). Note that T' contains two copies of the subtree T_{u_5} rooted at u_5, so is not (in general) irreducible. However, it is still correct.

Now, we consider two cases, each reaching the desired contradiction.

Case 1: The tree T' *has the form in Figure 7(a).* By inspection, the replacement increases the cost by $w(u'_3) + w(u'_2) - w(u_5) - w(u'_5) - w(u'_4) = w(u'_3) + w(u'_2) - w(u_3)$. By the optimality of T this

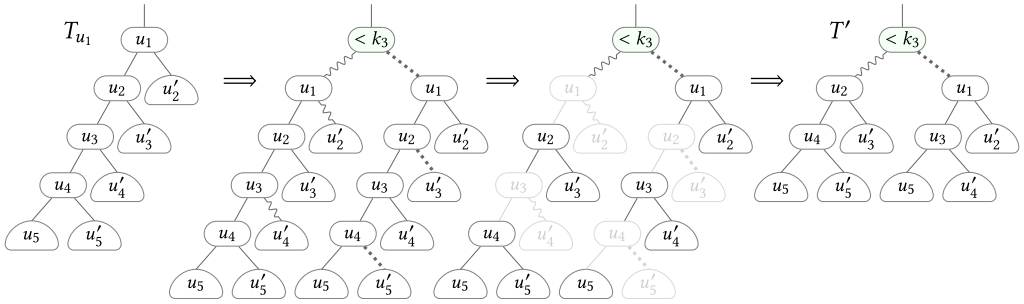


Fig. 6. Bisecting T_{u_1} around $\langle < k_3 \rangle$ makes a new root $\langle < k_3 \rangle$, makes each of its subtrees (yes and no) a copy of T_{u_1}, and then, for each outcome $u_i \to u'_{i+1}$ leaving the path, splices out whichever copy of $u_i \to u'_{i+1}$ is disjoint with the outcome of $\langle < k_3 \rangle$ that leads to that copy. (In the example here the squiggly outcomes are pairwise disjoint and the dashed outcomes are pairwise disjoint.) The outcomes on $u_1 \to \cdots \to u_5$ are drawn to the left.

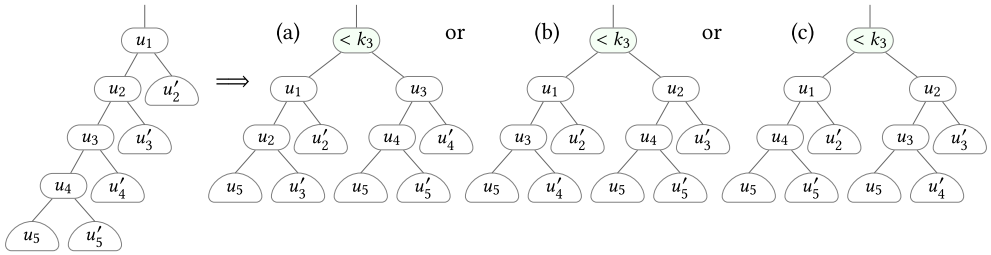


Fig. 7. Bisecting T_{u_1} around $\langle < k_3 \rangle$ yields a tree with one of forms (a), (b), or (c). The outcomes on the path $u_1 \to \cdots \to u_d$ are drawn to the left, as is the outcome $\langle < k_3 \rangle \to u_1$.

is non-negative. With Lemma 3.2 and $w(h) = w(u'_2)$ this implies $w(u'_3) \geq w(u_3) - w(u'_2) > w(u'_2)$. But then, by Theorem 2.1 applied to the path $u_1 \to u_2 \to u'_3$, the outcomes $u_1 \to u'_2$ and $u_2 \to u_3$ (leaving that path) are disjoint. By Observation 3, outcomes $u_1 \to u'_2$ and $u_2 \to u'_3$ are also disjoint, contradicting Property 2 for u_1 and u_2.

Case 2: The tree T' has one of the forms in Figure 7(b) or (c). By inspection, either replacement increases the cost by $w(u'_2) - w(u_5) - w(u'_5) = w(u'_2) - w(u_4)$. With the optimality of T this implies $w(u'_2) \geq w(u_4)$, which implies $w(h) \geq w(k)$, contradicting $w(k) > w(h)$. This proves Theorem 3.1. □

4 Some Optimal Tree Is Admissible

This section defines *admissible* (Definition 4.3), then proves that some optimal tree is admissible (Theorem 4.5). As mentioned in the introduction, we first handle the case when all weights are distinct (Lemma 4.4) then use a perturbation argument to extend to the general case. The perturbation argument requires a globally consistent tie-breaking for equal-weight keys.

Let T be any irreducible tree for a feasible 2WDT instance $I = (Q, w, C, K)$.

Definition 4.1 (*Ordering Queries by Weight*). For any query subset $R \subseteq Q$ and integer $i \geq 0$ define $\text{heaviest}_i(R)$ to contain the i heaviest queries in R (or all of R if $i \geq |R|$). For $q \in Q$, define $\text{heavier}(q)$ to contain the queries (in Q) that are heavier than q. Define $\text{lighter}(q)$ 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 tie-breaking mentioned above. Fix an ordering of Q by increasing weight, breaking ties in favor of queries that are smaller in the linear ordering of Q. (This particular tie-breaking rule is only for concreteness. Any consistent rule would work.) For $q \in Q$ let $\tilde{w}(q)$ denote the rank of q in this 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 4.2 (*Intervals and Holes*). 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* $\{k \in K : \ell \le k \le r\} = K \cap [\ell, r]_Q$.

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 $\mathsf{holes}(R) = [\min R, \max R]_Q \setminus R$ 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*, and 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. Write $w(u)$ as a shorthand for $w(Q_u)$, where $w(R) = \sum_{q \in R} w(q)$ denotes the total weight in the query set $R \subseteq Q$.

If R contains no keys then $k^*(R)$ is $\perp$ (undefined), so $\tilde{w}(k^*(R))$ is $-\infty$ and R has no light holes.

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 an ancestor v of u in T, so $h \in K$. The hole is light if any heavier key (and therefore $k^*(Q_u)$) reaches u. For example, in the optimal tree in Figure 3(a) (in which $K = [4]$) the query set $Q_{\langle=1\rangle}$ of node $\langle = 1 \rangle$ has light holes 3 and 4. These are lighter than the heaviest key $k^*(Q_{\langle=1\rangle}) = 2$ reaching $\langle = 1 \rangle$, but (not coincidentally, as we shall soon see) are the two heaviest in the node's key interval *minus* 2*'s class*. The light holes in the query set of $\langle = 1 \rangle$'s (right) no-child are 1, 3, and 4, which are the three heaviest in the node's key interval minus 2's class. The query sets of the nodes in the trees in Figure 3(b) and (c) have no light holes, but these trees are not optimal.

Definition 4.3 (*Admissible*). A non-empty query subset $R \subseteq Q$ is *admissible* if the set of light holes in R is 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 $c \in C$ such that $k^*(R) \in c$. (Throughout, for $i \in \mathbb{N}$, let $[i]$ denote $\{1, 2, \ldots, i\}$.)

The tree T (or any subtree) is *admissible* if all its nodes have admissible query sets.

By definition, the holes of any query set R lie in R's key interval $[\min R, \max R]_K$, and its light holes are those lighter than $k^*(R)$, the heaviest key in R.

We next show Lemma 4.4. Here is the intuition. We need to constrain how the heaviest-first property can fail at a node u in T. One way the property can fail (as illustrated in Figure 3(a)), is that there is a single class c that contains all of Q_u except for a few scattered keys, so that the optimal tree can use equality tests to pull out these "stragglers," then use a single leaf (labelled with c) to handle the rest. These stragglers can include a few keys lighter than $k^*(u)$, whose removal creates light holes, violating the heaviest-first property.

In fact, the proof shows that the path from u to such a leaf can have length at most four. (The path may have up to two less-than tests.) The lemma states that if Q_u fails to be heaviest first (that is, Q_u has light holes), it will still be admissible: for some $b \in [3]$ and some class c that can be assigned to $k^*(Q_u)$, the light holes must be the b heaviest keys in R's interval that are lighter than $k^*(Q_u)$ and are not in c. We can think of this as the heaviest-first property being preserved with respect to the keys *minus those in c*, with the restriction that at most three keys from c can be

exempted from being holes in this way. (This restriction to $O(1)$ keys is helpful for efficiency.) As we see later, the number of possible admissible query sets will turn out to be small enough to yield an efficient dynamic program.

As an exercise, consider the instance with query set $Q = [8]$, with classes and weights as specified in the table below, and key set $K = \{2, 3, 4, 5, 7\}$. (Keys are underlined.)

query	1	$\underline{2}$	$\underline{3}$	$\underline{4}$	$\underline{5}$	6	$\underline{7}$	8
classes	C	A, B	B, D	A, C	B	C	C, D	B
weight	10	13	67	49	27	58	38	12

For the subset $R_1 = \{1, 2, 4, 8\}$, we have $k^*(R_1) = 4 \in A \cap C$. Subset R_1 has three holes: a heavy hole 3 and two light holes 5, 7. In the above definition, choose A for the class c of $k^*(R_1)$. Then 5 and 7 are the two heaviest keys in $[\min R_1, \max R_1]_K \cap \mathsf{lighter}(k^*(R_1)) \setminus A$. So R_1 is admissible. For the subset $R_2 = \{2, 3, 6, 8\}$, we have $k^*(R_2) = 3$, and three holes 4, 5 and 7, all light. Both classes (B and D) that contain $k^*(R_2)$ also contain one of the light holes, so R_2 is not admissible.

Perhaps counterintuitively, the admissibility of a set R is *not* determined solely by the subinstance naively defined by R. (This instance is $I_R = (R, w_R, C_R, K_R)$, where w_R is w restricted to R, while C_R is $\{c \cap R : c \in C\} \setminus \{\emptyset\}$, and K_R is $K \cap R$.) Admissibility of R also depends on its set of light holes, in $K \setminus R$. This will be important for the implementation.

Lemma 4.4. *If the instance has distinct weights and T is optimal, then T is admissible.*

Proof. 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. Let $b = |H_u|$. Let c be the class that T assigns to k^* and $S = [\min Q_u, \max Q_u]_K \cap \mathsf{lighter}(k^*) \setminus c$. 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 Theorem 3.1 to that ancestor, hole h is not in c. It follows that $H_u \subseteq S$.

Next (recalling $b = |H_u|$) we show $H_u = \mathsf{heaviest}_b(S)$. Suppose otherwise for contradiction. That is, there are $k \in S \setminus H_u \subseteq Q_u$ 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 Theorem 3.1 (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 $b \le 3$. 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 Theorem 3.1 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. □

Now, we use a perturbation argument to extend Lemma 4.4 to the general case. Recall that "feasible" means the instance has a correct tree. As discussed in Section 1, not all instances do.

Theorem 4.5. *If the instance is feasible, then some optimal tree is admissible.*

Proof. Assume the instance $I = (Q, w, C, K)$ is feasible. Recall that $\tilde{w}(q)$ is the rank of q in the sorting of Q by weight, breaking ties consistently, as defined at the start of the section.

Let $I^* = (Q, w^*, C, K)$ be an instance obtained from I by perturbing the query weights infinitesimally so that (i) the perturbed weights are distinct and (ii) sorting Q by w^* gives the same order as sorting by $\tilde{w}$. Specifically, take $w^*(q) = w(q) + \delta\, \tilde{w}(q)$, for δ such that $0 < \delta < \epsilon/n^3$, where $\epsilon > 0$ is

the minimum of two quantities: the minimum absolute difference between any two distinct weights and the minimum absolute difference in cost between any two irreducible trees with distinct costs, using here that there are finitely many irreducible trees. Recall also that $\tilde{w}(q) \in [n]$.

The concept of tree irreducibility (defined in Section 1.1) is independent of the weight function (w or $\tilde{w}$). So the sets of irreducible trees for I and for I^* are the same.

Let T^* be an optimal, irreducible tree for I^* (so also irreducible for I). Applying Lemma 4.4 to T^* and I^*, tree T^* is admissible for I^*. By inspection of Definition 4.3, whether T^* is admissible for an instance depends only on T^* and the (tie-broken) ordering of the queries by weight. Since these orderings are the same in I and I^*, the tree T^* is admissible for I if and only if it is admissible for I^*.

To finish we observe that T^* is also optimal for I. For any tree T', let $\text{cost}(T')$ and $\text{cost}^*(T')$ denote the costs of T'under weight functions w (for I) and w^* (for I^*), respectively. Recall that earlier we fixed T to be an irreducible tree for I. Assume that T is also optimal for I. Then

$$\text{cost}(T^*) \;\le\; \text{cost}^*(T^*) \;\le\; \text{cost}^*(T) \le \text{cost}(T) + n^3\delta \;<\; \text{cost}(T) + \epsilon.$$

So by the choice of ϵ we have $\text{cost}(T^*) \le \text{cost}(T)$. Therefore T^* is optimal for I as well. □

5 Algorithm

This section proves the main result:

Theorem 5.1. *There is an $O(n^3m)$-time algorithm for finding a minimum-cost 2WDT.*

Proof. Fix the input, an arbitrary 2WDT instance $I = (Q, w, C, K)$. Let $\mathcal{A}$ denote the set of admissible query subsets of Q (per Definition 4.3). For any $R \in \mathcal{A}$, if R is contained in some class, then the tree for R consists of a single leaf (labeled with some such class). Otherwise an admissible tree for R consists of any root u whose test partitions R into $(R_u^{\text{yes}}, R_u^{\text{no}})$ (the bipartition of R into those values that satisfy u and those that don't), with u's yes-subtree being any admissible tree for R_u^{yes} and u's no-subtree being any admissible tree for R_u^{no}. So, defining $\text{cost}_{\mathcal{A}}(R)$ to be the minimum cost of any subtree for R that is admissible for I,[3] the following recurrence holds:

Recurrence 1. *For any $R \in \mathcal{A}$,*

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

where u ranges over the allowed tests (defined in Section 1.1) for which R_u^{yes} and R_u^{no} are in $\mathcal{A}$ (that is, admissible). If there are no such tests the minimum is infinite.

The algorithm returns $\text{cost}_{\mathcal{A}}(Q)$, the minimum cost of any admissible tree for $I = (Q, w, C, K)$. By Theorem 4.5, this equals the minimum cost of any tree for I, so the algorithm is correct. Next we describe how to achieve the desired running time.

There are $O(n^2m)$ 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 4.3, R is determined by a tuple $(\min R, \max R, k^*(R), b, c)$, where $(b, c) \in [3] \times C$ with $k^*(R) \in c$.) So $O(n^2m)$ subproblems arise in recursively evaluating $\text{cost}_{\mathcal{A}}(Q)$. To achieve the desired time bound, it suffices to evaluate the right-hand side of Recurrence 1 for any given $R \in \mathcal{A}$ in $O(n)$ amortized time. Next we describe how to do this.

[3] An observant reader may notice that it can be that $\text{cost}_{\mathcal{A}}(R) > \text{cost}(R)$ (the minimum cost of any tree for T), but if so R cannot actually occur as the query set of any node in an optimal tree.

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 $H(R) = \text{holes}(R) \cap \text{lighter}(k^*(R))$ 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 equal to the query set $Q_{(t)} = [\ell, r]_Q \setminus ((K \cap \text{heavier}(k)) \cup H')$, provided that $Q_{(t)}$ is non-empty and has signature $\tau(Q_{(t)}) = t$; otherwise $\tau^{-1}(t)$ is undefined. (In general, the signature of $Q_{(t)}$ may be different from t; for example we may have $k \notin [\ell, r]_Q$, or one of ℓ, r may be in H'.)

To finish the proof we prove Lemma 5.2:

Theorem 5.2. *After an $O(n^3m)$-time preprocessing step, given the signature $\tau(R)$ of $R \in \mathcal{A}$, the right-hand of Recurrence 1 is computable in amortized time $O(n)$.*

Proof. Note that the admissible sets can be enumerated in $O(n^3m)$ time as follows. First do the $O(n^3)$ admissible sets without light holes: for each $(\ell, r, k) \in Q \times Q \times (K \cup \{\perp\})$, output $\tau^{-1}(\ell, r, k, \emptyset)$ if it exists. Next do the $O(n^2m)$ admissible sets with at least one light hole, following Definition 4.3: for each $(\ell, r, k, b, c) \in Q \times Q \times K \times [3] \times C$ with $k \in c$, letting $H' = \text{heaviest}_b([\ell, r]_K \cap \text{lighter}(k) \setminus c)$, if H' is well-defined then output $\tau^{-1}(\ell, r, k, H')$ if it exists.

The preprocessing step initializes the dictionary for admissible query subsets and identifies the leaves. Here are the details.

Initialize a dictionary D holding a record $D[\tau(R)]$ for each set R in $\mathcal{A}$. To be able to determine whether a given query set R is in $\mathcal{A}$, and to store information (including the memoized cost) for each admissible set R, build a dictionary D that holds a record $D[\tau(R)]$ for each $R \in \mathcal{A}$, indexed by the signature $\tau(R)$. For now, assume the dictionary D supports constant-time access to the record $D[\tau(R)]$ for each $R \in \mathcal{A}$ given the signature $\tau(R)$ of R. (We describe a suitable implementation later.) Initialize D to hold an empty record $D[\tau(R)]$ for each $R \in \mathcal{A}$ by enumerating all $R \in \mathcal{A}$ as described above. This takes $O(n^3m)$ time.

Identify the leaves. To identify the sets $R \in \mathcal{A}$ that are leaves (that is, such that $(\exists c \in C) R \subseteq c$) in $O(n^3m)$ time, for each triple $(\ell, r, k) \in Q \times Q \times (K \cup \{\perp\})$, do the following two steps.

(1) Let $\mathcal{R} \subseteq \mathcal{A}$ contain the admissible sets R such that $\tau(R) = (\ell, r, k, H')$ for some H'. Assume $\mathcal{R}$ is non-empty (otherwise move on to the next triple). Let $R_\emptyset$ be the set with signature $(\ell, r, k, \emptyset)$, so that each $R \in \mathcal{R}$ is a subset of $R_\emptyset$ and can be written as $R_\emptyset \setminus H(R)$. Let C_ℓ contain the classes $c \in C$ such that $\ell \in c$. Observe that $|\mathcal{R}| \le 4|C_\ell|$, because $R_\emptyset$ is unique for the triple (ℓ, r, k), and then each $R \in \mathcal{R}$ is determined from $R_\emptyset$ by the class $c \in C$ and the number $b \in [3]$ of light holes, per Definition 4.3.

(2) Each set $R \in \mathcal{R}$ contains ℓ, so R is a leaf if and only if $R \subseteq c$ for some $c \in C_\ell$. The condition $R \subseteq c$ is equivalent to $R_\emptyset \setminus H(R) \subseteq c$, which is equivalent to $R_\emptyset \setminus c \subseteq H(R)$. So, any given set $R \in \mathcal{R}$ is a leaf if and only if some subset of $H(R)$equals $R_\emptyset \setminus c$ for some $c \in C_\ell$. Identify all such R in time $O(n|\mathcal{R}| + n|C_\ell|)$. (Recalling that $|H(R)| \le 3$ for each $R \in \mathcal{R}$, this is straightforward. One way is to construct the collection $\mathcal{H} = \bigcup_{R \in \mathcal{R}} 2^{H(R)}$ of subsets of the light-hole sets. Order the elements within each subset in $\mathcal{H}$ by increasing value, then radix sort $\mathcal{H}$ into lexicographic order. Do the same for the collection $\mathcal{L} = \{R_\emptyset \setminus c : c \in C_\ell, |R_\emptyset \setminus c| \le 3\}$.

Then merge the two collections to find the elements common to both. A given $R \in \mathcal{R}$ is a leaf if and only if some subset of $H(R)$ in $\mathcal{H}$ also occurs in $\mathcal{L}$.)

As noted above, we have $|\mathcal{R}| \leq 4|C_\ell|$, so the time spent above on a given triple (ℓ, r, k) is $O(n|\mathcal{R}| + n|C_\ell|) = O(n|C_\ell|)$. Summing over all triples (ℓ, r, k), the total time is $O(n^2 \sum_{\ell \in Q} n|C_\ell|) = O(n^3 m)$.

In $O(n^3 m)$ time, identify the $O(n^2 m)$ leaves $R \in \mathcal{A}$ as described above. For each, record in its entry $D[\tau(R)]$ that R is a leaf and that $\text{cost}_{\mathcal{A}}(R) = 0$.

How to implement Recurrence 1. Next, we describe how to compute $\text{cost}_{\mathcal{A}}(R)$, given the signature $\tau(R) = (\ell, r, k, H')$ of any set $R \in \mathcal{A}$, in $O(n)$ time.

If the record $D[\tau(R)]$ already holds a memoized cost for R, then we are done, so assume otherwise. (This implies that R is not a leaf.) In $O(n)$ time, build R from $\tau(R)$ and calculate the sum $w(R)$. Let $R = (q_1, q_2, \ldots, q_z)$ be R in increasing order, computed using bucket sort. For every possible test node u, precompute the signatures $\tau(R_u^{\text{yes}})$ and $\tau(R_u^{\text{no}})$. Do this in two $O(n)$-time stages: one stage for all possible less-than tests, the other stage for all possible equality tests:

Stage 1: Precompute the pair of signatures $\tau(R^{\text{yes}}_{\langle<h\rangle})$ and $\tau(R^{\text{no}}_{\langle<h\rangle})$ for every $h \in K$ as follows:

1.1. For $i \in \{0, 1, \ldots, z\}$, define $R_i = (q_1, q_2, \ldots, q_i)$ and $\overline{R}_i = (q_{i+1}, q_{i+2}, \ldots, q_z)$. For $h \in Q = [n]$, define $i(h)$ to be the index such that $R^{\text{yes}}_{\langle<h\rangle}$ is $R_{i(h)}$ and $R^{\text{no}}_{\langle<h\rangle}$ is $\overline{R}_{i(h)}$. Precompute $i(h)$ for all $h \in Q$ in $O(n)$ total time. (Note that $i(h) = \max\{0\} \cup \{i \in [z] : q_i < h\}$. Take $i(1) = 0$, then for $i \leftarrow 2, 3, \ldots, n$ take $i(h) = i(h-1) + 1$ if $q_{i(h-1)+1} < h$; otherwise take $i(h) = i(h-1)$.)

1.2. Compute $k^*(R_i)$ and $k^*(\overline{R_i})$ for all i. (These are the heaviest keys in R_i and in $\overline{R_i}$, respectively. First take $k^*(R_0) = \perp$, then, for $i \leftarrow 1, \ldots, z$, take $k^*(R_i) = q_i$ if $q_i \in K$ and q_i is heavier than $k^*(R_{i-1})$, and otherwise $k^*(R_i) = k^*(R_{i-1})$. Take $k^*(\overline{R_i}) = k^*(R)$ if $i < z$ and $k^*(R) \geq q_{i+1}$; otherwise take $k^*(\overline{R_i}) = \perp$.)

1.3. Compute the light-hole sets $H(R_i) = \{h \in H' : h \leq q_i\}$ and $H(\overline{R_i}) = \{h \in H' : h \geq q_{i+1}\}$. (Each such set can be computed in constant time from H', as $|H'| \leq 3$.)

1.4. Finally, enumerate all $h \in K$. For each, compute the pair of signatures $\tau(R^{\text{yes}}_{\langle<h\rangle})$ and $\tau(R^{\text{no}}_{\langle<h\rangle})$, using $R^{\text{yes}}_{\langle<h\rangle} = R_{i(h)}$, $R^{\text{no}}_{\langle<h\rangle} = \overline{R}_{i(h)}$, and (for $i = i(h)$), $\tau(R_i) = (q_1, q_i, k^*(R_i), H(R_i))$ and $\tau(\overline{R_i}) = (q_{i+1}, q_z, k^*(\overline{R}_i), H(\overline{R}_i))$. (Given the results of the previous three steps, this takes constant time per h.)

Stage 2: Precompute the pair of signatures $\tau(R^{\text{yes}}_{\langle=h\rangle})$ and $\tau(R^{\text{no}}_{\langle=h\rangle})$ for every $h \in K \cap R$. For each such h, we have $R^{\text{yes}}_{\langle=h\rangle} = \{h\}$, so $\tau(R^{\text{yes}}_{\langle=h\rangle}) = (h, h, h, \emptyset)$, and $\tau(R^{\text{no}}_{\langle=h\rangle})$ can be computed as follows:

2.1. If $h \notin \{\min R, \max R, k^*(R)\}$ (using that $|R| \geq 2$, as R is not a leaf, so $R^{\text{no}}_{\langle=h\rangle} \neq \emptyset$) the signature $\tau(R^{\text{no}}_{\langle=h\rangle})$ is $(\min R, \max R, k^*(R), H' \cup \{h\})$, which (as $|H'| \leq 3$) is computable from $\tau(R)$ in constant time.

2.2. Otherwise (h is one of the three values in $\{\min R, \max R, k^*(R)\}$), using $R^{\text{no}}_{\langle=h\rangle} = R \setminus \{h\}$, explicitly compute $R^{\text{no}}_{\langle=h\rangle}$ and its signature in $O(n)$ time.

Finally, for each pair of signatures $\tau(R_u^{\text{yes}})$ and $\tau(R_u^{\text{no}})$ enumerated above (in Step 1.4 or Stage 2), check whether R_u^{yes} and R_u^{no} are admissible (by checking, in constant time, whether their signatures have entries in D). If so, compute the values of $\text{cost}_{\mathcal{A}}(R_u^{\text{yes}})$ and $\text{cost}_{\mathcal{A}}(R_u^{\text{no}})$ recursively from their signatures. Then, for $\text{cost}_{\mathcal{A}}(R)$, returns (and memoize in $D[\tau(R)]$) the value from the recurrence, namely $w(R) + \min_u(\text{cost}_{\mathcal{A}}(R_u^{\text{yes}}) + \text{cost}_{\mathcal{A}}(R_u^{\text{no}}))$, with the minimum taken over all such u.

In this way, for each $R \in \mathcal{A}$, the time to evaluate the right-hand side of the recurrence is $O(n)$. There are $O(n^2m)$ sets in $\mathcal{A}$, so the total time is $O(n^3m)$.

How to implement the dictionary D. For each admissible query set $R \in \mathcal{A}$, the set $H(R)$ of light holes has size at most three. It follows that the signature $\tau(R) = (\ell, r, k, H(R))$ has size $O(1)$ and one way to implement the dictionary D (to support constant-time lookup) is to use a hash table with universal hashing. Then the algorithm uses space $O(n^2m)$, but is randomized. If a deterministic implementation is needed, one can implement the dictionary by storing an $n \times n \times n$ matrix M of buckets It follows that the signature $\tau(R) = (\ell, r, k, H(R))$ has size $O(1)$ such that a given bucket $M[\ell, r, k]$ holds the records for the admissible query sets R with signatures of the form $\tau(R) = (\ell, r, k, H')$ for some H'. Organize the records in this bucket using a trie (prefix tree) of depth 3 keyed by the (sorted) keys in H'. This still supports constant-time access, but increases the space to $O(n^3m)$. More generally, for any $d \geq 1$, one can represent each element $k \in [n]$ within each set H' as a sequence of $\lceil \log_2(n)/d \rceil d$-bit words, then use a trie with alphabet $\{0, 1, \ldots, 2^d - 1\}$ and depth at most $3\lceil \log_2(n)/d \rceil$. Then space is $\Theta(2^d n^2 m)$ while the access time is $\Theta(\log(n)/d)$. For example, we can take $d = \lceil \epsilon \log_2 n \rceil$ for any constant ϵ to achieve space $O(n^{2+\epsilon}m)$ and access time $\Theta(1/\epsilon) = \Theta(1)$. Or we can take $d = 1$ and achieve space $O(n^2m)$ and access time $\Theta(\log n)$, increasing the total time to $O(n^3 m \log n)$. □

Per Lemma 5.2, the preprocessing takes time $O(n^3m)$, and for each of $O(n^2m)$ sets $R \in \mathcal{A}$ Recurrence 1 can be evaluated in time $O(n)$. This proves Theorem 5.1. □

Remarks. In the common case that C partitions Q, each query $q \in Q$ is contained in just one class $c \in C$ (so $m = n$ and the algorithm runs in time $O(n^4)$), and then the algorithm can be implemented to use space $O(n^2m) = O(n^3)$. To do this, in the above implementation of the dictionary using a matrix M of buckets, each bucket $M[\ell, r, k]$ stores the records of at most four sets, so no prefix tree is needed to achieve constant access time and space.

We note without proof that there is a deterministic variant of the algorithm that uses space $O(n^2m)$ and time $O(n^3m)$. This variant is more complicated, so we chose not to present it.

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 $\langle \leq k \rangle$). Such tests only need to be accounted for in the proof of Theorem 3.1; the extended algorithm then allows such tests in Recurrence 1.

Acknowledgments

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

References

[1] R. Anderson, S. Kannan, H. Karloff, and R. E. Ladner. 2002. Thresholds and optimal binary comparison search trees. *Journal of Algorithms* 44 (2002), 338–358. DOI: https://doi.org/10.1016/S0196-6774(02)00203-1

[2] Dimitris Bertsimas and Jack Dunn. 2017. Optimal classification trees. *Machine Learning* 106, 7 (Jul. 2017), 1039–1082. DOI: https://doi.org/10.1007/s10994-017-5633-9

[3] C. Chambers and W. Chen. 1999a. Efficient multiple and predicated dispatching. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA '99)*, 238–255.

[4] C. Chambers and W. Chen. 1999b. Efficient multiple and predicated dispatching. *ACM SIGPLAN Notices* 34, 10 (Oct. 1999), 238–255. DOI: https://doi.org/10.1145/320385.320407

[5] Marek Chrobak, Mordecai Golin, J. Ian Munro, and Neal E. Young. 2021. A simple algorithm for optimal search trees with Two-Way comparisons. *ACM Transactions on Algorithms* 18, 1 (Dec. 2021), 2:1–2:11. DOI: https://doi.org/10.1145/3477910

[6] Marek Chrobak, Mordecai Golin, J. Ian Munro, and Neal E. Young. 2022. On Huang and Wong's algorithm for generalized binary split trees. *Acta Informatica* 59, 6 (Dec. 2022), 687–708. DOI: https://doi.org/10.1007/s00236-021-00411-z

[7] Marek Chrobak and Neal E. Young. 2023. Classification via two-way comparisons (extended abstract). In *Proceedings of the 18th International Symposium on Algorithms and Data Structures* (WADS '23). Pat Morin and Subhash Suri (Eds.), Lecture Notes in Computer Science, Vol. 14079, Springer, 275–290. DOI: https://doi.org/10.1007/978-3-031-38906-1_19

[8] Thomas H. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. 2022. *Introduction to Algorithms* (fourth ed.). The MIT Press, Cambridge, MA.

[9] Y. Dagan, Y. Filmus, A. Gabizon, and S. Moran. 2017. Twenty (simple) questions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC '17)*, 9–21. DOI: https://doi.org/10.1145/3055399.3055422

[10] E. N. Gilbert and E. F. Moore. 1959. Variable-length binary encodings. *The Bell System Technical Journal* 38, 4 (Jul. 1959), 933–967. DOI: https://doi.org/10.1002/j.1538-7305.1959.tb01583.x

[11] J. H. Hester, D. S. Hirschberg, S. H. Huang, and C. K. Wong. 1986. Faster construction of optimal binary split trees. *Journal of Algorithms* 7 (1986), 412–424. DOI: https://doi.org/10.1016/0196-6774(86)90031-3

[12] S-H. S. Huang and C. K. Wong. 1984. Generalized binary split trees. *Acta Informatica* 21, 1 (1984), 113–123. DOI: https://doi.org/10.1007/BF00289143

[13] S-H. S. Huang and C. K. Wong. 1984. Optimal binary split trees. *Journal of Algorithms* 5 (1984), 69–79. DOI: https://doi.org/10.1016/0196-6774(84)90041-5

[14] Laurent Hyafil and Ronald L. Rivest. 1976. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters* 5, 1 (May 1976), 15–17. DOI: https://doi.org/10.1016/0020-0190(76)90095-8

[15] Tobias Jacobs, Ferdinando Cicalese, Eduardo Laber, and Marco Molinaro. 2010. On the complexity of searching in trees: Average-case minimization. In *Automata, Languages and Programming*. Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis (Eds.), Lecture Notes in Computer Science, Springer, Berlin, 527–539. DOI: https://doi.org/10.1016/j.tcs.2011.08.042

[16] D. E. Knuth. 1971. Optimum binary search trees. *Acta Informatica* 1 (1971), 14–25. DOI: https://doi.org/10.1007/BF00264289

[17] D. E. Knuth. 1998. *The Art of Computer Programming*, Vol. 3. Sorting and Searching (2nd ed.). Addison-Wesley Publishing Company, Redwood City, CA.

[18] Y. Perl. 1984. Optimum split trees. *Journal of Algorithms* 5 (1984), 367–374. DOI: https://doi.org/10.1016/0196-6774(84)90017-8

[19] B. A. Sheil. 1978. Median split trees: A fast lookup technique for frequently occurring keys. *Communications of the ACM* 21 (1978), 947–958. DOI: https://doi.org/10.1145/359642.359653

[20] D. Spuler. 1994. Optimal search trees using two-way key comparisons. *Acta Informatica* 31, 8 (1994), 729–740. DOI: https://doi.org/10.1007/BF01178732

[21] D. A. Spuler. 1994. *Optimal Search Trees using Two-Way Key Comparisons*. Ph.D. Dissertation. James Cook University.

Received 17 May 2023; revised 1 August 2024; accepted 8 December 2024