Complexity of Fault Tolerant Query Complexity

Ramita Maharjan Thomas Watson

University of Memphis

November 28, 2022

Abstract

In the model of fault tolerant decision trees introduced by Kenyon and Yao, there is a known upper bound E on the total number of queries that may be faulty (i.e., get the wrong bit). We consider this computational problem: Given as input the truth table of a function $f\colon \{0,1\}^n \to \{0,1\}$ and a value of E, find the minimum possible height (worst-case number of queries) of any decision tree that computes f while tolerating up to E many faults. We design an algorithm for this problem that runs in time $\widetilde{O}\big(\binom{n+E}{E} \cdot (2E+3)^n\big)$, which is polynomial in the size of the truth table when E is a constant. This generalizes a standard algorithm for the non-fault tolerant setting.

1 Introduction

In practice, the fields of program synthesis (for software) and logic synthesis (for hardware) are about automating the design of computational processes: Given an input-output specification, generate an efficient implementation that meets the specification.

Complexity theory's version of synthesis is known as "meta-complexity" or "complexity of complexity": Given the truth table of a boolean function f, compute f's complexity—the cost of an optimal algorithm for f in some model of computation. Upper and lower bounds are known on the complexity of computing the complexity of a function (given its truth table) for various models, including circuits [KC00, AD17, AGvM+18, All20, ILO20, Ila20], formulas [AKRR03, AHM+06, Ila20, Ila21], branching programs [FS90, AKRR03, Rav13], communication protocols [KW09, ILO20, HIL21], and decision trees [GLR99, Aar03, Rav13].

We focus on decision trees. A decision tree computes a function $f\colon \{0,1\}^n \to \{0,1\}$ by adaptively querying (reading) individual bits of the input. A decision tree's height (cost) is the maximum over all inputs of the number of queries made on that input. The query complexity of f is the minimum height of any decision tree computing f. A standard algorithm due to [GLR99, Aar03] inputs f's truth table and outputs f's query complexity in time $\widetilde{O}(3^n)$, where $\widetilde{O}$ hides a $\mathrm{poly}(n)$ factor. In terms of the input size $N = 2^n$, this is polynomial time $\widetilde{O}(N^{\log_2(3)})$. A simple extension of the algorithm outputs an optimal decision tree, not just its height. There is also a considerable amount of literature on properly learning decision trees given access to input-output pairs of f (e.g., see the recent papers [BHZ19, BLQT21] and the references within). The "given a truth table" problem corresponds to the extreme case where *all* input-output pairs of f are available.

We consider fault tolerant decision trees, in which a query to an input bit may or may not get the bit's actual value. This is motivated by scenarios where the input bits are the results of unreliable computations. Several models of this type have been introduced [KY90, FRPU94, RS91, SC04]. We focus on the original model from [KY90], in which a decision tree is only required to output

the correct bit when the total number of faults, over all queries to all variables, is at most some parameter E. Letting $\widetilde{O}$ hide a $\mathrm{poly}(n, E)$ factor, we prove:

Theorem 1. *Given as input the truth table of a function $f\colon \{0,1\}^n \to \{0,1\}$ and an integer $E \geq 0$, the E-fault tolerant query complexity of f can be computed in time $\widetilde{O}\big(\binom{n+E}{E} \cdot (2E+3)^n\big)$.*

When E is a constant, this running time is $\widetilde{O}\big(N^{\log_2(2E+3)}\big)$, which is polynomial in the size $N = 2^n$ of f's truth table.

Our algorithm generalizes the algorithm from [GLR99, Aar03] for the non-fault tolerant setting. The analysis of our algorithm is much more involved, largely because our set of dynamic programming subproblems has a complicated combinatorial structure. It is not straightforward to characterize or count the subproblems. This stems from the fact that—unlike in the non-fault tolerant setting—we cannot treat the input variables as independent of each other.

2 Decision trees

A *decision tree* on boolean variables $x_1, x_2, \ldots, x_n$ is a full binary tree where each internal node is labeled with a variable index $i \in [n] = \{1, 2, \ldots, n\}$ and each leaf is labeled with an output bit. On an input $x \in \{0,1\}^n$, a decision tree follows a root-to-leaf path: Upon reaching an internal node labeled i, the decision tree *queries* x_i and goes to the left child if $x_i = 0$ or to the right child if $x_i = 1$. The output is the label of the leaf reached. A decision tree computes $f\colon \{0,1\}^n \to \{0,1\}$ iff it outputs $f(x)$ for every input x. A decision tree's *height* is the number of edges on a longest root-to-leaf path, i.e., the maximum over all $x \in \{0,1\}^n$ of the number of queries made on input x. The (deterministic) *query complexity* of f, denoted $D(f)$, is the minimum height of any decision tree that computes f. We have $D(f) \leq n$ for all $f\colon \{0,1\}^n \to \{0,1\}$ since a decision tree can make n queries to determine the entire input x and then output $f(x)$. There is no reason for a decision tree computing f to query the same variable more than once along a root-to-leaf path.

In the fault tolerant setting, a query to x_i may get either x_i's actual value or the opposite bit—a *fault* or *mistake*. We allow a decision tree to query the same variable more than once along a root-to-leaf path, in which case some queries may get the actual value while others are faults. We consider the model introduced by [KY90] where $E \geq 0$ is a specified upper bound on the total number of faults over all variables (not per variable). A decision tree computes $f\colon \{0,1\}^n \to \{0,1\}$ while tolerating E faults iff it outputs $f(x)$ for every input x and every choice of at most E queries to be faults. More formally, for a node v, input x, and variable index i, define $\mathrm{faults}(v, x, i)$ as the number of nodes along the root-to-v path—excluding v itself—that are labeled i but the path goes to the wrong child, i.e., left child but $x_i = 1$, or right child but $x_i = 0$. Define $\mathrm{faults}(v, x) = \sum_{i=1}^{n} \mathrm{faults}(v, x, i)$. A decision tree is *$E$-fault tolerant* for f iff for every input x and every leaf v with $\mathrm{faults}(v, x) \leq E$, the label of v is $f(x)$. In other words, if we define $\mathrm{consistent}_E(v) = \{x : \mathrm{faults}(v, x) \leq E\}$ then for every leaf v, $f(x)$ must be the same for all $x \in \mathrm{consistent}_E(v)$. The *$E$-fault tolerant query complexity* of f, denoted $D_E(f)$, is the minimum height of any E-fault tolerant decision tree for f. We have $D(f) = D_0(f) \leq D_1(f) \leq D_2(f) \leq \cdots$. The following lemma does not seem to appear in the literature.

Lemma 1. *$D_E(f) \leq D(f) \cdot (E+1) + E$ for all f and E.*

Proof. Let T be a decision tree of height $D(f)$ computing f. We design T', which is an E-fault tolerant decision tree of height at most $D(f) \cdot (E+1) + E$ for f. The idea is that on any input, T' tracks T's root-to-leaf path but for each variable T queries along the way, T' queries the variable repeatedly until ascertaining its actual value:

- Suppose (for convenience) T first queries x_1. We have T' query x_1 until some bit has appeared $E+1$ times. This bit is x_1's actual value since otherwise x_1 would have $E+1$ faults. Let e_1 be the number of faults on x_1. The number of queries to x_1 is $E+1+e_1$.
- Suppose (for convenience) T next queries x_2. We have T' query x_2 until some bit has appeared $E+1-e_1$ times. This bit is x_2's actual value since otherwise x_1, x_2 would have $E+1$ faults. Let e_2 be the number of faults on x_2. The number of queries to x_2 is $E+1-e_1+e_2$.

$\vdots$

- Suppose (for convenience) T queries x_i in the i^{th} step. We have T' query x_i until some bit has appeared $E+1-e_1-e_2-\cdots-e_{i-1}$ times. This bit is x_i's actual value since otherwise $x_1, \ldots, x_i$ would have $E+1$ faults. Let e_i be the number of faults on x_i. The number of queries to x_i is $E+1-e_1-e_2-\cdots-e_{i-1}+e_i$.

$\vdots$

Upon reaching a leaf of T, we have T' output the same bit. If at most E faults occur, then T' finds the leaf T would reach. Since T computes f, T' is E-fault tolerant for f. If T makes q queries on input x, then T' makes at most

$$\begin{aligned}\textstyle\sum_{i=1}^{q}\big(E+1-(\sum_{j=1}^{i-1}e_j)+e_i\big) &= q\cdot(E+1)-\textstyle\sum_{i=1}^{q}(q-1-i)\cdot e_i\\ &\le q\cdot(E+1)+e_q\\ &\le D(f)\cdot(E+1)+E\end{aligned}$$

queries on input x, regardless of when the faults occur. □

Lemma 1 is tight for the And function on n bits:

Observation 1. *$D_E(\mathrm{And}_n) = n\cdot(E+1)+E$ for all n and E.*

Proof. An adversary can respond 1 to all queries, until some variable x_i has been queried E times and each other variable has been queried at least $E+1$ times, and then the adversary can start responding 0 to subsequent queries to x_i (and keep responding 1 to queries to other variables). After at most $n\cdot(E+1)+E-1$ queries, the current node v would have $\mathrm{faults}(v,y)\le E$ where y is the all 1s input (so $\mathrm{And}_n(y)=1$) and $\mathrm{faults}(v,z)\le E$ where z is all 1s except $z_i=0$ (so $\mathrm{And}_n(z)=0$), where i is the index of a variable that has been queried as 1 at most E times and as 0 at most E times. Thus v cannot be a leaf since $y,z\in\mathrm{consistent}_E(v)$ but $f(y)\neq f(z)$. That is, an E-fault tolerant decision tree for And_n must make at least $n\cdot(E+1)+E$ queries in the worst case, otherwise it may output the wrong bit for either y or z. □

Lemma 1 is not tight for all functions: Let Tribes_n be the function that interprets x as a $\sqrt{n}\times\sqrt{n}$ array of bits and $\mathrm{Tribes}_n(x)=1$ iff there exists an all-1 row in x. Then $D(\mathrm{Tribes}_n)=n$ but [KY90] proved that $D_E(\mathrm{Tribes}_n)=O(n+\sqrt{n}\cdot E)$. More generally, [KY90] proved that $D_E(f)=O(n+C(f)\cdot E)$ for all f and E, where $C(f)=\max(\text{CNF width of } f, \text{DNF width of } f)$ is the certificate complexity of f.

3 Patterns and redundant queries

A *pattern* is a tuple $p=(p_1,p_2,\ldots,p_n)=\big((p_{1,0},p_{1,1}),(p_{2,0},p_{2,1}),\ldots,(p_{n,0},p_{n,1})\big)$ where for all $i\in[n]$ and $b\in\{0,1\}$, $p_{i,b}$ is a nonnegative integer representing the number of queries to variable

x_i that got value b. Each node v in a decision tree has an associated pattern p^v that records the results of the queries along the root-to-v path but does not indicate the order of those queries. If v is the root, then $p^v = \big((0,0), (0,0), \ldots, (0,0)\big)$ since no queries have happened.

For a non-fault tolerant decision tree, in which no variable is ever re-queried, a pattern is traditionally viewed as a *restriction* $r \in \{0,1,*\}^n$ where $r_i = 0$ means $x_i = 0$ was queried, and $r_i = 1$ means $x_i = 1$ was queried, and $r_i = *$ means x_i remains unqueried. Compared to our more general notion of patterns, $r_i = 0$ corresponds to $p_i = (1,0)$, and $r_i = 1$ corresponds to $p_i = (0,1)$, and $r_i = *$ corresponds to $p_i = (0,0)$.

The combinatorial structure is more subtle for fault tolerant decision trees. For example, suppose $n = 2$, $E = 1$, the current pattern is $\big((1,0),(0,1)\big)$, and we query x_1. If the query gets $x_1 = 0$, then the pattern becomes $\big((2,0),(0,1)\big)$ and we know x_1 is actually 0 (otherwise x_1 would have two faults) but we do not know x_2's actual value. If the query gets $x_1 = 1$, then the pattern becomes $\big((1,1),(0,1)\big)$ and we know x_2 is actually 1 (otherwise x_1, x_2 would each have one fault) but we do not know x_1's actual value. The point is that querying a variable might reveal more information about *other* variables.

For a pattern p, we define $\text{faults}(p,x,i) = p_{i,\overline{x_i}}$ and $\text{faults}(p,x) = \sum_{i=1}^n \text{faults}(p,x,i)$ and $\text{consistent}_E(p) = \{x : \text{faults}(p,x) \le E\}$. Thus for any node v in a decision tree, $\text{faults}(v,x,i) = \text{faults}(p^v,x,i)$ and $\text{faults}(v,x) = \text{faults}(p^v,x)$ and $\text{consistent}_E(v) = \text{consistent}_E(p^v)$.

For a pattern p, querying x_i would be *redundant* (with respect to E) iff all $x \in \text{consistent}_E(p)$ have the same value of x_i. An internal node v in a decision tree is *redundant* (with respect to E) iff x_i is a redundant query given p^v, where i is the label of v. A decision tree with no redundant nodes is *sensible* (with respect to E).

Lemma 2. *For every f and E, there exists a minimum-height E-fault tolerant decision tree for f that is sensible.*

Proof. Consider any minimum-height E-fault tolerant decision tree for f. We claim that if v is a redundant node labeled i, and b is the common value of x_i for $x \in \text{consistent}_E(p^v)$, then the decision tree will still be E-fault tolerant for f after we replace v with v's child corresponding to query $x_i = b$ and discard the other child's subtree. Repeating this process to eliminate all redundant nodes yields a sensible decision tree that is no taller than the original.

To prove the claim, consider any leaf u in v's b-subtree in the original decision tree, and let w be the modified decision tree's leaf corresponding to u. We show that $\text{consistent}_E(p^w) = \text{consistent}_E(p^u)$, which implies that the modified decision tree is still E-fault tolerant for f since $f(x)$ is the same for all $x \in \text{consistent}_E(p^w)$. Trivially, $\text{consistent}_E(p^w) \supseteq \text{consistent}_E(p^u)$ since p^w is the same as p^u except $p^w_{i,b} = p^u_{i,b} - 1$ and so $\text{faults}(p^w,x) \le \text{faults}(p^u,x)$ for all x. To see that $\text{consistent}_E(p^w) \subseteq \text{consistent}_E(p^u)$, first note that $p^w_{j,c} \ge p^v_{j,c}$ for all $(j,c) \in [n]\times\{0,1\}$, because $p^w_{i,b} = p^u_{i,b} - 1 \ge p^v_{i,b}$ and $p^w_{j,c} = p^u_{j,c} \ge p^v_{j,c}$ for all $(j,c) \ne (i,b)$. Thus $\text{consistent}_E(p^w) \subseteq \text{consistent}_E(p^v)$, so $x_i = b$ for all $x \in \text{consistent}_E(p^w)$. This implies that $\text{faults}(p^u,x) = \text{faults}(p^w,x) \le E$ for all $x \in \text{consistent}_E(p^w)$. □

Lemma 3. *x_i is a redundant query given pattern p iff $\max_b(p_{i,b}) + \sum_{j\ne i} \min_b(p_{j,b}) > E$.*

Proof. $\Leftarrow$: Assume $\max_b(p_{i,b}) + \sum_{j\ne i} \min_b(p_{j,b}) > E$. For all $x \in \text{consistent}_E(p)$, we must have $x_i = \arg\max_b(p_{i,b})$ since otherwise $\text{faults}(p,x,i) = \max_b(p_{i,b})$ and $\text{faults}(p,x,j) \ge \min_b(p_{j,b})$ for all $j \ne i$ and thus $\text{faults}(p,x) \ge \max_b(p_{i,b}) + \sum_{j\ne i} \min_b(p_{j,b}) > E$, which would mean $x \notin \text{consistent}_E(p)$.

$\Rightarrow$: Assume $\max_b(p_{i,b}) + \sum_{j\ne i} \min_b(p_{j,b}) \le E$. Define $y, z \in \{0,1\}^n$ as follows: $y_i = 0$ and $z_i = 1$ and $y_j = z_j = \arg\max_b(p_{j,b})$ for all $j \ne i$. Then $\text{faults}(p,y,i), \text{faults}(p,z,i) \le \max_b(p_{i,b})$

and $\text{faults}(p, y, j) = \text{faults}(p, z, j) = \min_b(p_{j,b})$ for all $j \neq i$, and thus $\text{faults}(p, y), \text{faults}(p, z) \leq \max_b(p_{i,b}) + \sum_{j \neq i} \min_b(p_{j,b}) \leq E$, which means $y, z \in \text{consistent}_E(p)$. Thus x_i is not a redundant query given p. □

For a pattern p, $i \in [n]$, and $b \in \{0, 1\}$, define the pattern $p^{i,b}$ to be the same as p except $p^{i,b}_{i,b} = p_{i,b} + 1$. If an internal node v has pattern p and label i, then v's children have patterns $p^{i,0}$ and $p^{i,1}$.

Observation 2. *For all p and i,* $\text{consistent}_E(p) = \text{consistent}_E(p^{i,0}) \cup \text{consistent}_E(p^{i,1})$.

Proof. $\supseteq$: For all $b \in \{0, 1\}$, if $x \in \text{consistent}_E(p^{i,b})$ then $\text{faults}(p, x) \leq \text{faults}(p^{i,b}, x) \leq E$ and so $x \in \text{consistent}_E(p)$.

$\subseteq$: If $x \in \text{consistent}_E(p)$ then $\text{faults}(p^{i,x_i}, x) = \text{faults}(p, x) \leq E$ and so $x \in \text{consistent}_E(p^{i,x_i})$. □

By the way, $\text{consistent}_E(p^{i,0})$ and $\text{consistent}_E(p^{i,1})$ might not be disjoint.

4 Valid patterns

A pattern p is *valid* (with respect to E) iff $p = p^v$ for some node v in some sensible decision tree. A centerpiece of the analysis of our algorithm is a characterization of valid patterns. To state this, we define:

- $\text{block}(p, d) = \{i \in [n] : |p_{i,0} - p_{i,1}| = d\}$ for each integer $d \geq 0$.
- $\text{min-faults}(p, D) = \sum_{d \geq D} \sum_{i \in \text{block}(p,d)} \min_b(p_{i,b})$ for each integer $D \geq 0$.
- $\text{min-faults}(p) = \text{min-faults}(p, 0) = \sum_{i=1}^{n} \min_b(p_{i,b})$.

Lemma 4. *p is valid iff* $\text{min-faults}(p) \leq E$ *and* $\text{min-faults}(p, D) \leq E - D + 1$ *for each $D \geq 2$ such that* $\text{block}(p, D) \neq \emptyset$.

Proof. We define some notation with respect to p: For each variable x_i, define $b_i = \arg\max_b(p_{i,b})$ (breaking the tie arbitrarily if $p_{i,0} = p_{i,1}$) and $d_i = |p_{i,0} - p_{i,1}| = p_{i,b_i} - p_{i,\overline{b_i}}$, so $i \in \text{block}(p, d_i)$.

$\Leftarrow$: Assume $\text{min-faults}(p) \leq E$ and $\text{min-faults}(p, D) \leq E - D + 1$ for each $D \geq 2$ such that $\text{block}(p, D) \neq \emptyset$. The latter actually holds for $D \geq 1$, not just for $D \geq 2$, because $\text{min-faults}(p) \leq E$ implies that $\text{min-faults}(p, 1) \leq \text{min-faults}(p) \leq E = E - 1 + 1$.

To show that p is valid, we exhibit a sensible decision tree consisting of a root-to-v path where $p^v = p$ and all nodes hanging off of this path are leaves (whose outputs are irrelevant). For each variable x_i in decreasing order of d_i (breaking ties arbitrarily), the path has $p_{i,\overline{b_i}}$ many $x_i = \overline{b_i}$ queries followed by p_{i,b_i} many $x_i = b_i$ queries. By definition, this path ends at a node v with $p^v = p$.

We argue that no internal node is redundant. Consider an internal node u labeled i. First suppose $d_i = 0$. Then $\max_b(p^u_{i,b}) \leq \min_b(p_{i,b})$ and $\min_b(p^u_{j,b}) \leq \min_b(p_{j,b})$ for each $j \neq i$. Thus

$$\begin{aligned} \max_b(p^u_{i,b}) + \sum_{j \neq i} \min_b(p^u_{j,b}) &\leq \sum_{j=1}^{n} \min_b(p_{j,b}) \\ &= \text{min-faults}(p) \\ &\leq E \end{aligned}$$

so u is not redundant, by Lemma 3. Now suppose $d_i \geq 1$. Then $\text{block}(p, d_i) \neq \emptyset$ since $i \in \text{block}(p, d_i)$, so $\text{min-faults}(p, d_i) \leq E - d_i + 1$ by assumption. We have $\max_b(p^u_{i,b}) \leq \max_b(p_{i,b}) - 1 =$

$\min_b(p_{i,b}) + d_i - 1$ because $d_i \geq 1$ and the last $x_i = b_i$ query happens either at u or farther down the path (since the $x_i = b_i$ queries happen after the $x_i = \overline{b_i}$ queries). We have $\min_b(p^u_{j,b}) \leq \min_b(p_{j,b})$ for each $j \neq i$, and $\min_b(p^u_{j,b}) = 0$ if $d_j < d_i$ because variables are queried in decreasing order of d_j. Thus

$$\begin{aligned} \max_b(p^u_{i,b}) + \textstyle\sum_{j \neq i} \min_b(p^u_{j,b}) &\leq d_i - 1 + \textstyle\sum_{d \geq d_i} \sum_{j \in \text{block}(p,d)} \min_b(p_{j,b}) \\ &= d_i - 1 + \text{min-faults}(p, d_i) \\ &\leq E \end{aligned}$$

so u is not redundant, by Lemma 3.

$\Rightarrow$: Assume p is valid, and let v be a node with $p^v = p$ in some sensible decision tree.

To see that min-faults$(p) \leq E$, consider v's parent u. (Or if v is the root, then this holds trivially.) We have $\min_b(p_{i,b}) \leq \max_b(p^u_{i,b})$ where i is the label of u, and $\min_b(p_{j,b}) = \min_b(p^u_{j,b})$ for each $j \neq i$. Thus

$$\begin{aligned} \text{min-faults}(p) &= \textstyle\sum_{j=1}^{n} \min_b(p_{j,b}) \\ &\leq \max_b(p^u_{i,b}) + \textstyle\sum_{j \neq i} \min_b(p^u_{j,b}) \\ &\leq E \end{aligned}$$

by Lemma 3, since u is not redundant.

To see that min-faults$(p, D) \leq E - D + 1$ for each $D \geq 2$ such that block$(p, D) \neq \emptyset$, consider the lowest node u on the root-to-v path (excluding v itself) such that u's label i satisfies $d_i \geq D$. Such u exists since $D \geq 1$ and block$(p, D) \neq \emptyset$. Let w be u's child on the root-to-v path (possibly $w = v$). Since u is the lowest such node, for each j with $d_j \geq D$ we have $p^w_{j,b} = p_{j,b}$ for both b and thus $\min_b(p^w_{j,b}) = \min_b(p_{j,b})$. We have $\min_b(p^w_{i,b}) = \max_b(p^w_{i,b}) - d_i \leq \max_b(p^u_{i,b}) - d_i + 1$ and $\min_b(p^w_{j,b}) = \min_b(p^u_{j,b})$ for each $j \neq i$. Thus

$$\begin{aligned} \text{min-faults}(p, D) &= \textstyle\sum_{d \geq D} \sum_{j \in \text{block}(p,d)} \min_b(p_{j,b}) \\ &= \textstyle\sum_{d \geq D} \sum_{j \in \text{block}(p,d)} \min_b(p^w_{j,b}) \\ &\leq \textstyle\sum_{j=1}^{n} \min_b(p^w_{j,b}) \\ &\leq \max_b(p^u_{i,b}) - d_i + 1 + \textstyle\sum_{j \neq i} \min_b(p^u_{j,b}) \\ &\leq E - d_i + 1 \\ &\leq E - D + 1 \end{aligned}$$

by Lemma 3, since u is not redundant. □

Corollary 1. *If p is valid then* consistent$_E(p) \neq \emptyset$.

Proof. $x \in \text{consistent}_E(p)$ where each $x_i = \arg\max_b(p_{i,b})$ since faults$(p, x) = \text{min-faults}(p) \leq E$. □

Lemma 5. *There are at most $\binom{n+E}{E} \cdot (2E+3)^n$ valid patterns.*

Proof. Choosing $p_{i,0}$ and $p_{i,1}$ is equivalent to choosing $\min_b(p_{i,b})$ and $p_{i,0} - p_{i,1}$ (no absolute value).

For valid p, there are $\binom{n+E}{E}$ possibilities for $\big(\min_b(p_{1,b}), \ldots, \min_b(p_{n,b})\big)$ since min-faults$(p) \leq E$ by Lemma 4.

For valid p, we have $\text{block}(p, D) = \emptyset$ for all $D \geq E + 2$ since otherwise we would have the contradiction $0 \leq \text{min-faults}(p, D) \leq E - D + 1 < 0$ by Lemma 4. (Intuitively, a sensible decision tree would never re-query x_i after $\max_b(p_{i,b}) = E+1$ since that would be redundant by Lemma 3.) Thus for each i, there are $2E+3$ possibilities for $p_{i,0} - p_{i,1}$, namely $E+1, E, \ldots, 0, \ldots, -E, -E-1$. Hence there are $(2E+3)^n$ possibilities for $\big(p_{1,0} - p_{1,1}, \ldots, p_{n,0} - p_{n,1}\big)$.

There are at most $\binom{n+E}{E} \cdot (2E+3)^n$ ways to form a valid p by choosing $\big(\min_b(p_{1,b}), \ldots, \min_b(p_{n,b})\big)$ and $\big(p_{1,0} - p_{1,1}, \ldots, p_{n,0} - p_{n,1}\big)$. □

As a sanity check, when $E = 0$, Lemma 4 implies that p is valid iff $\text{min-faults}(p) = 0$ and $\text{block}(p, D) = \emptyset$ for all $D \geq 2$, which happens iff $p_i \in \{(1,0), (0,1), (0,0)\}$ for each i. There are 3^n such patterns (corresponding to the restrictions in $\{0, 1, *\}^n$), which agrees with Lemma 5 when $E = 0$.

Lemma 5 generally overcounts the number of valid patterns because it ignores the constraints for $D \in \{2, \ldots, E+1\}$ in Lemma 4. The "minimum" and "difference" tuples are not independent. But Lemma 5 does not overcount by a lot: There are $\binom{n+E}{E}$ valid patterns with $p_{i,0} - p_{i,1} = 0$ for all i, and there are $(2E+3)^n$ valid patterns with $\min_b(p_{i,b}) = 0$ for all i. Thus there are at least

$$\max\big(\tbinom{n+E}{E}, (2E+3)^n\big) \;\geq\; \sqrt{\tbinom{n+E}{E} \cdot (2E+3)^n}$$

valid patterns, so Lemma 5 is at least quadratically tight.

When E is a constant, the number of valid patterns is $\Theta\big(\binom{n+E}{E} \cdot (2E+3)^n\big) = \Theta\big(n^E \cdot (2E+3)^n\big)$. This is because each of the $\binom{n+E}{E}$ possibilities for $\big(\min_b(p_{1,b}), \ldots, \min_b(p_{n,b})\big)$ has 0s in at least $n-E$ coordinates and thus gives rise to at least $(2E+3)^{n-E} = \Omega((2E+3)^n)$ valid patterns with the nonzero values of $p_{i,0} - p_{i,1}$ distributed only among the coordinates i with $\min_b(p_{i,b}) = 0$.

We can say something a little stronger than the previous paragraph. Suppose $1 \leq E \leq n$. There are $\binom{n}{E}$ possibilities of $\big(\min_b(p_{1,b}), \ldots, \min_b(p_{n,b})\big)$ where $\min_b(p_{i,b}) = 1$ for E coordinates i and $\min_b(p_{i,b}) = 0$ for $n - E$ coordinates i. For each of those possibilities: There are $(2E+3)^{n-E}$ possibilities for the tuple of $p_{i,0} - p_{i,1}$ for all i with $\min_b(p_{i,b}) = 0$. By Stirling's formula (where e is the base of the natural log) there are $E! \geq (E/\mathrm{e})^E$ permutations of $[E]$, which we view as possibilities for the tuple of $|p_{i,0} - p_{i,1}|$ for all i with $\min_b(p_{i,b}) = 1$. There are 2^E possibilities for the tuple of $\text{sign}(p_{i,0} - p_{i,1})$ for all i with $\min_b(p_{i,b}) = 1$. In total, we just described at least

$$\tbinom{n}{E} \cdot (2E+3)^{n-E} \cdot (E/\mathrm{e})^E \cdot 2^E \qquad (\dagger)$$

patterns p that are each valid, since $\text{min-faults}(p) = E$, and $\text{min-faults}(p, D) = E - D + 1$ for each $2 \leq D \leq E$, and $\text{block}(p, D) = \emptyset$ for each $D > E$. Let us compare this to the upper bound of Lemma 5. Assuming $1 \leq E \leq n/2$, we have

$$\begin{aligned}
\tbinom{n+E}{E} / \tbinom{n}{E} \;&=\; \frac{(n+E) \cdot (n+E-1) \cdots (n+1)}{n \cdot (n-1) \cdots (n-E+1)} \\
&=\; \big(1 + E/n\big) \cdot \big(1 + E/(n-1)\big) \cdots \big(1 + E/(n-E+1)\big) \\
&\leq\; 2^E
\end{aligned}$$

and

$$\frac{(2E+3)^n}{(2E+3)^{n-E} \cdot (E/\mathrm{e})^E \cdot 2^E} \;=\; \Big(\frac{(2E+3) \cdot \mathrm{e}}{2E}\Big)^E \;=\; \big((1 + 3/(2E)) \cdot \mathrm{e}\big)^E \;\leq\; 7^E.$$

So, the upper bound $\binom{n+E}{E} \cdot (2E+3)^n$, divided by the lower bound $(\dagger)$, is at most 14^E. Thus when $E = O(\log n)$, Lemma 5 only overcounts the number of valid patterns by a fairly insignificant $\text{poly}(n)$ factor. Computational experiments suggest that Lemma 5 is fairly tight for all E, but it remains open to prove this.

```
complexity(f, E):
    initialize empty dictionary opt
    (h, ℓ) ← solve(((0,0), ..., (0,0)))
    return h
```

```
solve(p):
    if opt contains key p: return opt[p]
    h ← ∞
    for i ∈ [n]:
        if min-faults(p) + |p_{i,0} − p_{i,1}| ≤ E:
            (h_0, ℓ_0) ← solve(p^{i,0})
            (h_1, ℓ_1) ← solve(p^{i,1})
            if h = ∞ and h_0 = h_1 = 0 and ℓ_0 = ℓ_1: (h, ℓ) ← (0, ℓ_0) and break out of loop
            if h > 1 + max(h_0, h_1): (h, ℓ) ← (1 + max(h_0, h_1), i)
    if h = ∞: (h, ℓ) ← (0, f(x)) where x_i = arg max_b(p_{i,b}) for each i ∈ [n]
    opt[p] ← (h, ℓ)
    return (h, ℓ)
```

Figure 1: Algorithm to compute $D_E(f)$

5 The algorithm: Proof of Theorem 1

We design a dynamic programming algorithm for the following computational problem: Given as input the truth table of a function $f\colon \{0,1\}^n \to \{0,1\}$ and an integer $E \geq 0$, output $D_E(f)$.

For each valid pattern p, we have a subproblem whose solution is a pair (h, ℓ) where h is the minimum height of any sensible decision tree that is E-fault tolerant for f assuming the root is defined to have pattern p, and ℓ is the root's label in one such decision tree. If $h = 0$ then the root is a leaf that outputs $\ell \in \{0,1\}$. If $h > 0$ then the root is an internal node that queries x_ℓ where $\ell \in [n]$. We store the solution (h, ℓ) to subproblem p in a dictionary data structure *opt* where p is the key and (h, ℓ) is the value. We use memoized recursion, rather than traditional dynamic programming, because enumerating the valid patterns is not straightforward.

Figure 1 shows the algorithm. If p is valid then solve(p) returns the solution (h, ℓ), after computing it and storing it in *opt*[p] if it was not already there. We assume f, n, E, and *opt* are globally accessible in calls to the solve subroutine. The loop iterates through variables that are not redundant queries given p, by Lemma 3. By induction, we may assume solve($p^{i,0}$) and solve($p^{i,1}$) return correct solutions, since $p^{i,0}$ and $p^{i,1}$ are valid if x_i is not a redundant query given p.

Base cases are when the optimal height is 0, i.e., the root should be a leaf because $f(x)$ is the same for all $x \in \text{consistent}_E(p)$. One way to detect this would be to loop through all $x \in \{0,1\}^n$, compute faults(p, x), and compare $f(x)$ for all x with faults$(p, x) \leq E$, but that would incur $\widetilde{O}(2^n)$ time overhead. Our algorithm detects base cases more efficiently. There are two kinds of base cases:

- $|\text{consistent}_E(p)| = 1$: By definition, every x_i is a redundant query given p, so we still have $h = \infty$ after the loop. Thus solve(p) correctly computes the solution as $(0, f(x))$ where x is the unique element of $\text{consistent}_E(p)$. (There is no tie for $\arg\max_b(p_{i,b})$, since min-faults$(p) \leq E$ and min-faults$(p) + |p_{i,0} - p_{i,1}| > E$ implies $p_{i,0} \neq p_{i,1}$.)
- $|\text{consistent}_E(p)| > 1$ but $f(x)$ is the same for all $x \in \text{consistent}_E(p)$: By definition, at least

```
construct-tree(p):
    (h, ℓ) ← opt[p]
    create a node v labeled ℓ
    if h > 0:
        v's left child ← construct-tree(p^{ℓ,0})
        v's right child ← construct-tree(p^{ℓ,1})
    return v
```

Figure 2: Constructing an optimal decision tree

one x_i is not a redundant query given p. When the loop reaches the first such index i, we have $h = \infty$ and $h_0 = h_1 = 0$ and $\ell_0 = \ell_1$ since $f(x)$ is the same for all $x \in \text{consistent}_E(p^{i,0}) \cup \text{consistent}_E(p^{i,1})$ by Observation 2 ($\supseteq$). Thus solve(p) breaks out of the loop and correctly computes the solution as $(0, \ell_0)$ (skipping over "if $h = \infty$" after the loop since $h = 0$ now).

Non-base cases are when the optimal height is positive, i.e., the root should be an internal node because $f(x)$ is not the same for all $x \in \text{consistent}_E(p)$. Since we only consider sensible decision trees, the loop just tries every possible non-redundant query x_i the root could make, computes the height $1 + \max(h_0, h_1)$ of an optimal decision tree having i as the root label, lets h be the minimum of these heights, and lets ℓ be the index of the variable that achieves the minimum.[1] The condition "if $h = \infty$ and $h_0 = h_1 = 0$ and $\ell_0 = \ell_1$" is false since otherwise $f(x)$ would be the same for all $x \in \text{consistent}_E(p^{i,0}) \cup \text{consistent}_E(p^{i,1})$ and so p would be a base case by Observation 2 ($\subseteq$). Also, "if $h = \infty$" is false after the loop since at least one x_i is non-redundant given p, and h becomes finite during such an iteration. Thus for a non-base case, (h, ℓ) is correct at the end of solve(p).

Finally, complexity(f, E) returns the minimum height of any sensible E-fault tolerant decision tree for f, which equals $D_E(f)$ by Lemma 2.

Now we analyze the running time. A call to solve(p) is *fresh* iff *opt* does not yet contain the key p. We charge the cost of a nonfresh call to the parent call. There are at most $\binom{n+E}{E} \cdot (2E+3)^n$ fresh calls by Lemma 5, and each fresh call takes time $\text{poly}(n, E) = \widetilde{O}(1)$ (excluding any fresh recursive calls it makes), assuming the dictionary is implemented with a self-balancing search tree. Thus the running time of complexity(f, E) is $\widetilde{O}\big(\binom{n+E}{E} \cdot (2E+3)^n\big)$.

After running solve$\big((0,0), \ldots, (0,0)\big)\big)$, we can construct a minimum-height sensible E-fault tolerant decision tree for f by straightforwardly retracing the optimal choices for the relevant subproblems (Figure 2). In the non-fault tolerant setting ($E = 0$), constructing the decision tree is faster than solving all the subproblems: $\widetilde{O}(2^n)$ compared to $\widetilde{O}(3^n)$. In contrast, in the fault tolerant setting, constructing the decision tree can be much slower than solving all the subproblems—potentially as slow as $\widetilde{O}(2^{n \cdot (E+1)+E})$ by Lemma 1—since a fault tolerant decision tree can be a sprawling behemoth with many duplicated subtrees.

Another minor extension handles partial functions $f\colon \{0,1\}^n \to \{0,1,?\}$ where $f(x) = ?$ means that $f(x)$ is undefined and we do not care what a decision tree outputs on input x. A decision tree is E-fault tolerant for a partial function f iff for every leaf v, its label equals $f(x)$ for all $x \in \text{consistent}_E(v)$ such that $f(x) \neq ?$. To compute $D_E(f)$ for a partial function f, we can modify solve(p) so that $opt[p] = (0, ?)$ means that $f(x) = ?$ for all $x \in \text{consistent}_E(p)$, and $opt[p] = (0, \ell)$ for $\ell \in \{0,1\}$ means that there exists an $x \in \text{consistent}_E(p)$ such that $f(x) \neq ?$ and that $f(x) = \ell$

[1] If we wanted to minimize the number of leaves rather than the height, we could just change $1 + \max(h_0, h_1)$ to $h_0 + h_1$ and let $h = 1$ for base cases.

for all such x. If $h = \infty$ and $h_0 = h_1 = 0$ then: If $\ell_0 = \ell_1$ or $\ell_1 = ?$, we assign $(h, \ell) \leftarrow (0, \ell_0)$. Else if $\ell_0 = ?$, we assign $(h, \ell) \leftarrow (0, \ell_1)$. Else we do not assign (h, ℓ) and do not break out of the loop here.

6 Future directions

It is open to prove asymptotically tight bounds on the number of valid patterns for all n and E.

When E is super-constant, our algorithm's running time is super-polynomial in the input size. Can we prove a complexity lower bound on the problem for large E? Even in the non-fault tolerant setting, can fine-grained complexity techniques show that the $\widetilde{O}(N^{\log_2(3)})$ running time is optimal under standard assumptions?

It remains open to study the complexity of computing fault tolerant *randomized* query complexity. The algorithm from [Aar03] for computing non-fault tolerant randomized query complexity does not seem to generalize to the fault tolerant setting.

It is also open to study the analogous question for other models of fault tolerant decision trees, such as those introduced in [FRPU94]. To the best of our knowledge, "complexity of complexity" has not been studied for models of fault tolerant circuits, formulas, branching programs, or communication protocols.

Acknowledgments

This work was supported by NSF grant CCF-1657377.

References

[Aar03] Scott Aaronson. Algorithms for boolean function query properties. *SIAM Journal on Computing*, 32(5):1140–1157, 2003. doi:10.1137/S0097539700379644.

[AD17] Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. *Information and Computation*, 256:2–8, 2017. doi:10.1016/j.ic.2017.04.004.

[AGvM+18] Eric Allender, Joshua Grochow, Dieter van Melkebeek, Cristopher Moore, and Andrew Morgan. Minimum circuit size, graph isomorphism, and related problems. *SIAM Journal on Computing*, 47(4):1339–1372, 2018. doi:10.1137/17M1157970.

[AHM+06] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. Minimizing DNF formulas and $\mathrm{AC}^0{}_{\mathrm{d}}$ circuits given a truth table. In *Proceedings of the 21st Conference on Computational Complexity (CCC)*, pages 237–251. IEEE, 2006. doi:10.1109/CCC.2006.27.

[AKRR03] Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. Derandomization and distinguishing complexity. In *Proceedings of the 18th Conference on Computational Complexity (CCC)*, pages 209–220. IEEE, 2003. doi:10.1109/CCC.2003.1214421.

[All20] Eric Allender. The new complexity landscape around circuit minimization. In *Proceedings of the 14th Conference on Language and Automata Theory and Applications (LATA)*, pages 3–16. Springer, 2020. doi:10.1007/978-3-030-40608-0_1.

[BHZ19] Nader Bshouty and Catherine Haddad-Zaknoon. Adaptive exact learning of decision trees from membership queries. In *Proceedings of the 30th International Conference on Algorithmic Learning Theory (ALT)*, pages 207–234. PMLR, 2019.

[BLQT21] Guy Blanc, Jane Lange, Mingda Qiao, and Li-Yang Tan. Properly learning decision trees in almost polynomial time. In *Proceedings of the 62nd Symposium on Foundations of Computer Science (FOCS)*, pages 920–929. IEEE, 2021. doi:10.1109/FOCS52979.2021.00093.

[FRPU94] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

[FS90] Steven Friedman and Kenneth Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, 39(5):710–713, 1990. doi:10.1109/12.53586.

[GLR99] David Guijarro, Víctor Lavín, and Vijay Raghavan. Exact learning when irrelevant variables abound. *Information Processing Letters*, 70(5):233–239, 1999. doi:10.1016/S0020-0190(99)00063-0.

[HIL21] Shuichi Hirahara, Rahul Ilango, and Bruno Loff. Hardness of constant-round communication complexity. In *Proceedings of the 36th Computational Complexity Conference (CCC)*, pages 31:1–31:30. Schloss Dagstuhl, 2021. doi:10.4230/LIPIcs.CCC.2021.31.

[Ila20] Rahul Ilango. Constant depth formula and partial function versions of MCSP are hard. In *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*, pages 424–433. IEEE, 2020. doi:10.1109/FOCS46700.2020.00047.

[Ila21] Rahul Ilango. The minimum formula size problem is (ETH) hard. In *Proceedings of the 62nd Symposium on Foundations of Computer Science (FOCS)*, pages 427–432. IEEE, 2021. doi:10.1109/FOCS52979.2021.00050.

[ILO20] Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. NP-hardness of circuit minimization for multi-output functions. In *Proceedings of the 35th Computational Complexity Conference (CCC)*, pages 22:1–22:36. Schloss Dagstuhl, 2020. doi:10.4230/LIPIcs.CCC.2020.22.

[KC00] Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *Proceedings of the 32nd Symposium on Theory of Computing (STOC)*, pages 73–79, 2000. doi:10.1145/335305.335314.

[KW09] Eyal Kushilevitz and Enav Weinreb. On the complexity of communication complexity. In *Proceedings of the 41st Symposium on Theory of Computing (STOC)*, pages 465–474. ACM, 2009. doi:10.1145/1536414.1536479.

[KY90] Claire Kenyon and Andrew Yao. On evaluating boolean functions with unreliable tests. *International Journal of Foundations of Computer Science*, 1(1):1–10, 1990. doi:10.1142/S0129054190000023.

[Rav13] Netanel Raviv. Truth table minimization of computational models. Technical Report 1306.3766, arXiv, 2013.

[RS91] Rüdiger Reischuk and Bernd Schmeltz. Reliable computation with noisy circuits and decision trees—a general $n \log n$ lower bound. In *Proceedings of the 32nd Symposium on Foundations of Computer Science (FOCS)*, pages 602–611. IEEE, 1991. doi:10.1109/SFCS.1991.185425.

[SC04] Mario Szegedy and Xiaomin Chen. Computing boolean functions from multiple faulty copies of input bits. *Theoretical Computer Science*, 321(1):149–170, 2004. doi:10.1016/j.tcs.2003.07.001.