Electronic Colloquium on Computational Complexity, Report No. 34 (2024)

The hardness of decision tree complexity

Bruno Loff and Alexey Milovanov

LASIGE, Faculdade de Ciências, Universidade de Lisboa

November 2023

Abstract

Let f be a Boolean function given as either a truth table or a circuit. How difficult is it to find the decision tree complexity, also known as deterministic query complexity, of f in both cases? We prove that this problem is NC^1-hard and PSPACE-hard, respectively. The second bound is tight, and the first bound is close to being tight.

Contents

ISSN 1433-8092

1 Introduction

The decision tree is one of the most important computational models in theoretical computer science. Decision trees were invented in the 50s with the purpose of analyzing data. In this context, at each node in the tree we *query* some feature of the data, which partitions the data points depending on the value of this chosen feature. The resulting partition at the leaves should allow us to better understand the data. Starting in the 1980s, several learning algorithms were developed that would process data and produce a *classifier*. Meaning, we assume the existence of some function f of which we know some sampled pairs $(x, f(x))$ (the data), and we wish to produce a decision tree that would be able to predict $f(x)$, even on a previously-unseen input x (some famous algorithms are CHAID, CART, ID3, and C4.5, see ([Kas80, BFOS84, Qui86, Qui93])). The goal here is to produce the smallest possible decision tree, while making the fewest possible mistakes. This task, of learning a decision tree for f from a collection examples $(x, f(x))$, is used in real-life applications, and we could call it the *learning problem* for decision trees.

But one can also consider a more *algorithmic problem*. Here, the function f is completely known (i.e., we know $(x, f(x))$ for all x), and we wish to produce a decision tree which computes f (e.g., without any mistakes). As far as we are aware it was in the 1970s (e.g. [Yao75]) when people first studied, for specific functions f, and for specific ways of querying the input x, how small can be the depth of a decision tree that computes $f(x)$ when given x as input. In the meantime, decision trees have become a ubiquitous computational model, useful in the study of various kinds of computation. To give a few examples, decision trees are relevant to the study of data structures (the cell-probe model [Pă08, Lar13]), cryptographic reductions (in black-box reductions [RTV04]), quantum algorithms (in separations CITE), communication complexity and proof complexity (in lifting theorems [Göö15, GJPW18, GPW15]).

There is a meta-complexity problem underlying this algorithmic problem: how does one determine the decision-tree complexity of a given function f? This question has been studied before, usually for the learning problem [KST23], but also for the algorithmic problem [Aar03].

Let us restrict our attention to the simplest of all decision-tree models, where the known function $f : \{0,1\}^n \to \{0,1\}$ is a Boolean function, and the computational model is deterministic decision tree that must compute $f(x)$ by querying the bits of x. Here, we can consider two scenarios, with respect to how f is given:

(tt-DT) We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.

(circuit-DT) We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

The meta-complexity problem is: We are given f as input, either as a truth-table (tt-DT) or as a circuit (circuit-DT) and we wish to find the deterministic *query complexity* complexity of f, namely, the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. How hard is this problem? In this paper, we give a satisfactory answer for both scenarios, essentially via the same technique.

It is not difficult to see that circuit-DT belongs to PSPACE: see Proposition 2.7 below. One could immediately conjecture that the problem is PSPACE-hard, but one soon comes across various difficulties in proving such a statement. (After the required definitions are in place, in Section 3.2, we discuss precisely where the difficulty lies.) Our first main result, Theorem 3.1, is showing that this problem is indeed PSPACE-hard.

It is also well-known, and appears as Proposition 2.6 below, that tt-DT belongs to P. Meaning, denoting the input length for tt-DT as $N = 2^n$, which is the length of the truth-table of a Boolean function $f : \{0,1\}^n \to \{0,1\}$, Proposition 2.6 states that tt-DT can be solved in poly(N) time. In Proposition 2.9, we observe that tt-DT can also be computed in parallel, namely, by a Boolean circuit of depth $O(\log N \log\log N)$ and size poly(N). This result is simple enough that it should be considered folklore: easy to prove for anyone who would care to do so. However, despite being a natural and fundamental problem, no matching lower-bound was known. Our second main result, Theorem 3.2, shows that the above bound is close to tight: tt-DT is NC^1-hard (under uniform NC^0 reductions).

2 Definitions and upper bounds

Definition 2.1. *We let* $\{0,1\}^n$ *denote the set of all binary strings of length* n*, sometimes called the* Boolean cube*. We let* $\{0,1,*\}^n$ *denote the set of* (n-bit) partial assignments*. The elements* $\rho \in \{0,1,*\}^n$ *are in bijection with the* Boolean subcubes*:*

$$\{x \in \{0,1\}^n \mid \forall i \in [n] \quad \rho_i \neq * \implies x_i = \rho_i\}.$$

We will denote this set also by ρ*, by abuse of notation.*

We let $[i \leftarrow b] = *^{i-1} b *^{n-i-1}$ assign b to the i-th coordinate. Two partial assignment ρ and ρ' are called compatible if $\rho_i \neq *$ or $\rho'_i \neq *$ implies $\rho_i = \rho'_i$.

If $\rho, \rho' \in \{0,1,*\}^n$ are compatible, then $\rho \cdot \rho'$ is the partial assignment such that $(\rho \cdot \rho')_i$ equals to ρ_i if $\rho_i \neq *$, equals ρ'_i if $\rho'_i \neq *$, and equals $*$ if $\rho_i = \rho'_i = *$.

If $|\rho^{-1}(*)| = \ell$ and $y \in \{0,1\}^\ell$, we let $\rho(y) \in \{0,1\}^n$ be the binary string which equals ρ where $\rho_i \neq *$, and equals y in the remaining coordinates (which are filled in order).

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and a partial assignment $\rho \in \{0,1,*\}^n$ with $\ell = |\rho^{-1}(*)|$ the number of $*$, we let $f|_\rho : \{0,1\}^\ell \to \{0,1\}$ be given by

$$f|_\rho(y) = f(\rho(y)).$$

I.e. it is the restriction of f to the Boolean subcube ρ.

Definition 2.2. *A* deterministic decision tree *over $\{0,1\}^n$ is a rooted, labelled, ordered binary tree T:*

- *Each non-leaf node v is labelled by an index $i_v \in [n]$.*
- *Each non-leaf node v has two children v_0 and v_1.*

Associated with each node v of T is a partial assignment ρ_v

- *The root is associated with $*^n$.*
- *For a non-leaf node v, and for both $b \in \{0,1\}$, $\rho_{v_b} = \rho_v \cdot [x_{i_v} = b]$.*

Definition 2.3. *The* computation *of T on input $x \in \{0,1\}^n$ is a path in T, which begins at the root, and proceeds at each node v by going to the child $v_{x_{i_v}}$, until it reaches a leaf.*

It is easy to see that ρ_v is the set of inputs whose computation goes through v. We have $(\rho_v)_i = *$ if and only if the coordinate x_i has not yet been queried in the computation between the root and node v, i.e., at or before v. If x_i has been queried at or before v, then $(\rho_v)_i = x_i$ for every x whose computation goes through v.

Definition 2.4. *We say that T* computes *a function $f : \{0,1\}^n \to \{0,1\}$ if, for every leaf v of T, f is constant on ρ_v. The* deterministic query complexity *of a function $f : \{0,1\}^n \to \{0,1\}$, which will be denoted $\mathsf{D}(f)$, is the smallest depth of any decision tree that computes f.*

Definition 2.5. *We let* tt-DT *be the computational problem where we are given the truth-table of a function $f : \{0,1\}^n \to \{0,1\}$, and wish to compute* $\mathsf{D}(f)$. *We let* circuit-DT *be the computational problem where we are given a Boolean circuit C computing a function $f : \{0,1\}^n \to \{0,1\}$, and we wish to compute* $\mathsf{D}(f)$.

The simplest observation that one can make is that tt-DT has a polynomial-time algorithm.

Proposition 2.6. *tt-DT belongs to P. More precisely, there is an algorithm that computes the DT-complexity of an n-ary Boolean function in time $O(3^n \cdot n) = O(N^{1.585\dots} \log N)$, where $N = 2^n$.*

This algorithm should be considered folklore, but we include it here because it is simple and insightful. Variants of it appear, e.g., in[GLR99, Aar03].

Proof. The main, crucial observation is that the best decision tree for f must first choose a coordinate i, query x_i, and then run the best decision tree for $f|_{[i \leftarrow x_i]}$. In general, for any partial assignment $\rho \in \{0,1,*\}^n$, $\mathsf{D}(f|_\rho) = 0$ if f is constant on ρ, and otherwise

$$\mathsf{D}(f|_\rho) = \min_{i \in \rho^{-1}(*)} 1 + \max_{b \in \{0,1\}} \mathsf{D}(f|_{\rho \cdot [i \leftarrow b]}). \qquad (*)$$

This gives us a dynamic programming algorithm: knowing $\mathsf{D}(f|_\rho)$ for all partial assignments $\rho \in \{0,1,*\}^n$ with $|\rho^{-1}(*)| = \ell$ free variables, we can use the above formula to compute $\mathsf{D}(f|_\rho)$ for all $\rho \in \{0,1,*\}^n$ with $|\rho^{-1}(*)| = \ell + 1$ free variables. Finally $f = f|_{*^n}$, so we learned $\mathsf{D}(f)$. There are 3^n partial assignments in total, and each computation $\mathsf{D}(f|_\rho)$ takes time $O(n)$ in a random-access machine. □

Some more insight will come from the following *game reformulation* of the statement "$\mathsf{D}(f) \leq k$". Consider the following game between two players, Alice and Bob. The game lasts for k steps. At every step, Alice chooses a variable x_i, and Bob sets a Boolean value to the corresponding variable, either $x_i = 0$ or $x_i = 1$. After k steps, Alice wins $f|_\rho$ is constant on the partial assignment ρ corresponding to Alice and Bob's moves; otherwise, Bob wins. It follows that Alice has a winning strategy in this game if and only if $\mathsf{D}(f) \leq k$. Indeed, if $\mathsf{D}(f) \leq k$, then Alice can make moves according to the corresponding tree, and if $\mathsf{D}(f) > k$, then Bob wins because this inequality means that for every i, the decision tree complexity of $f|_{[i\leftarrow 1]}$ or $f|_{[i\leftarrow 0]}$ is at least k. Bob's strategy is then to repeatedly choose the value $b \in \{0,1\}$ that maximizes $\mathsf{D}(f|_{[i\leftarrow b]})$.

One can algorithmically find the winner in this game by a simple recursive algorithm. It is easy to see that, if a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is given as a Boolean circuit C, an algorithm can decide which of the two players has a winning strategy, using $\mathrm{poly}(n, |C|)$ memory. So, we get the following:

Proposition 2.7. *circuit-DT belongs to PSPACE.*

One may now ask whether Proposition 2.6 can be at all improved. Indeed, the algorithm can be parallelized. First, a definition:

Definition 2.8. *For $i \in \mathbb{N}$, we let $\mathrm{NC^i}$ denote the class of functions $f : \{0,1\}^n \to \{0,1\}^m$ computable by Boolean circuits with binary AND and OR gates, and unary NOT gates, in depth $O((\log n)^i)$ and size $\mathrm{poly}(n,m)$. We let $\widetilde{\mathrm{NC}}^1$ denote the class of functions $f : \{0,1\}^n \to \{0,1\}^m$ computable by such circuits in depth $O(\log n \cdot \log\log n)$ and size $\mathrm{poly}(n,m)$.*

We now claim the following.

Proposition 2.9. *tt-DT can be computed by a Boolean circuit of size $O(3^n \cdot \mathrm{poly}(n)) = O(N^{1.583\ldots}\,\mathrm{polylog}\,N)$ and depth $O(n \log n) = O(\log N \log\log N)$. Hence, tt-DT is in $\widetilde{NC}^1$.*

Proof. There's two different ways of seeing this. Using the fundamental equation $(*)$, and using the game definition above.

The equation $(*)$ directly gives us a circuit that uses $O(3^n)$ min, max, increment, and all-equal gates: for each partial assignment ρ we check if $f|_\rho$ is constant using an all-equal gate, otherwise we compute the formula given by $(*)$. This circuit has depth $O(n)$ using such gates. Implementing such gates using binary fan-in Boolean gates will result in a circuit of depth $O(n \log n)$.

Or instead, we see "$\mathsf{D}(f) \le k$" as the existence of a winning strategy for Alice in the above game. The tree of this game can be transformed into a Boolean circuit. The possible moves of Alice correspond to an n-ary disjunction, and Bob's moves correspond to a 2-ary conjunction. After k steps in the game, the winner is given by whether $f|_\rho$ is constant, which is computed by an all-equals gate. This gives us a formula of depth $O(k \log n)$ and size $O(n^k)$. By being careful to merge nodes where the previous moves of Alice and Bob give the same partial assignment, we get a circuit whose size will never exceed $O(3^n \mathrm{poly}(n))$. $\square$

3 Lower bounds

Our two main theorems are the following.

Theorem 3.1. *circuit-DT is PSPACE-hard under polynomial-time reductions.*

Theorem 3.2. *tt-DT is* NC^1*-hard under* NC^0*-reduction.*

The first theorem is well understood, but the second requires some clarification regarding reductions and uniformity. Recall the definition of DLOGTIME-uniform NC^1-cicuits from [BIS90].

Definition 3.3. *Let* $\mathcal{C} = \{C_n \mid n \in \mathbb{N}\}$ *be a family of Boolean circuits, so that* C_n *computes a function* $f : \{0,1\}^n \to \{0,1\}^{m(n)}$.

- *The "direct connection language" of* $\mathcal{C}$ *is the set of tuples* $\langle 0^n, t, a, b\rangle$, *where* $a \in [|C_n|]$ *and* $b \in \{0\} \cup [|C_n|]$ *are gate numbers in* C_n, t *is the type of gate* a *(AND, OR, NOT, or input gate* x_i*), and gate* b *is a child of gate* a *(and equals* 0 *if* a *is a leaf).*
- *The circuit family* $\mathcal{C}$ *is DLOGTIME-uniform if its direct connection language can be recognized in* $O(\log(n))$ *time by a deterministic multi-tape Turing machine (with an index tape for random access to the input).*

We say that language $A \subseteq \{0,1\}^*$ *is* NC^0*-reduced to language* $B \subseteq \{0,1\}^*$, *which we write* $A \leq_{\mathrm{NC}^0} B$, *if there is a DLOGTIME-uniform family of* NC^0*-circuits* $\mathcal{C} = \{C_n\}$ *such that, for every* $x \in \{0,1\}^n$, $x \in A$ *iff* $C_n(x) \in B$.

It is not difficult to see that this type of reduction satisfies the natural properties:

- (*transitivity*) If $A \leq_{\mathrm{NC}^0} B$ and $B \leq_{\mathrm{NC}^0} C$ then $A \leq_{\mathrm{NC}^0} C$.
- (*closure*) If $A \leq_{\mathrm{NC}^0} B$ and $B \in \mathrm{NC}^1$ then $A \in \mathrm{NC}^1$ (this holds for both uniform and non-uniform variants of NC^1).

3.1 TQBF

The proof of theorems 3.1 and 3.2 are similar: we prove that TQBF reduces to DT. Recall that TQBF (True Quantified Boolean Formula) is the problem of determining the truth of a formula

$$\exists y_1 \forall x_1 \ldots \forall x_n h(y_1, x_1, \ldots y_n, x_n), \tag{†}$$

where h is some Boolean function. As we did for decision-tree complexity, let us consider two variants of the TQBF problem:

tt-TQBF We are given as input a Boolean function $h : \{0,1\}^{2n} \to \{0,1\}$ as a truth table, and wish to know whether (†) holds.

circuit-TQBF We are given h as a circuit, and wish to know whether (†) holds.

It is well-known that circuit-TQBF is PSPACE-complete. It turns out that tt-TQBF is NC^1-complete:

Theorem 3.4. *tt-TQBF is* NC^1*-complete under* $\leq_{\mathrm{NC}^0}$ *reductions.*

To prove Theorem 3.4 we use the following result of Barrington.

Theorem 3.5 ([Bar86, BIS90])**.** *The* S_5 identity problem*,* $S_5\mathsf{IP}$*, is the problem of deciding if the product of given permutations from* S_5 *is equal to the identity. Then* $S_5\mathsf{IP}$ *is* NC^1*-complete under* $\leq_{\mathrm{NC}^0}$ *reductions.*

This theorem was proved in [Bar86], and in [BIS90] the authors verified that the reasoning proves the desired statement with DLOGTIME-uniform NC^0 reductions.

Proof of Theorem 3.4. It is easy to see that tt-TQBF is in NC^1: the formula (†) is a Boolean formula of depth n whose leaves are entries in the truth-table of h. To prove that tt-TQBF is NC^1-hard, the idea is to consider $S_5\mathsf{IP}$ as a game that can be interpreted as a TQBF formula.

Consider the input of $S_5\mathsf{IP}$: permutations $\pi_1, \ldots, \pi_N$, with $N = 2^n$. Imagine two players Alice and Bob; Alice wants to prove that the product is equal to the identity permutation and Bob does not trust her.

They play in the following game. Bob asks: what is the product of the first half of the permutations, i.e. $\pi_1 \cdot \ldots \cdot \pi_{\lfloor \frac{N}{2} \rfloor}$. Alice states that this product is some permutation σ_1. Additionally, since Alice is trying to prove to Bob that all the product of all permutations is equal to the identity, she is also implicitly stating that the product $\pi_{\lfloor \frac{N}{2} \rfloor + 1} \cdot \ldots \cdot \pi_N$ is equal to σ_1^{-1}. Bob does not trust Alice, and so he chooses $b_1 \in \{0,1\}$ to mean that he believes one of Alice's statements to be false. I.e. he chooses $b_1 = 0$ if he believes that the first part is actually not σ_1, and he sets $b_1 = 1$ if he believes that the second part is not σ_1^{-1}. After this, a similar procedure repeats: Alice states that the value of the product of the first half of Bob's chosen part is σ_2 (this is one quarter of all permutations), which implies that the second half of Bob's chosen part is $\sigma_2^{-1}\sigma_1^{(-1)^{b_1}}$. Then Bob chooses one of these two quarters $b_2 \in \{0,1\}$ where he believes Alice's statement is false, and so on.

This game produces a sequence $\sigma_1, b_1, \ldots, \sigma_n, b_n$. At the end of the game, the winner is identified by whether $\alpha = \pi_i$ or not, where α and i are inferred from the sequence. This game can be interpreted as a TQBF: each statement σ_i of Alice can be encoded as 7 bits (since $5! < 2^7$), and each choice b_i of Bob can be encoded as 1 bit. Every value of the truth table of h for tt-TQBF can be constructed from Alice and Bob's moves and the corresponding value of some π_i.

Indeed, $h(\sigma_1, b_1, \ldots, \sigma_n, b_n) = 1$ if and only if $\alpha = \pi_i$ for the appropriate $\alpha \in S_5$ and $i \in [n]$. Hence, this reduction is in NC^0, since every value in the truth-table of h (i.e. the winner in the corresponding game) depends only on one permutation π_i of the input, which is encoded using a constant number of bits.

We further claim that the corresponding NC^0 reduction can be made DLOGTIME uniform. By logarithmic time we mean $O(\log N) = O(n)$. We first describe an algorithm which, when given $\sigma_1, b_1, \ldots \sigma_n, b_n$ (i.e. the moves in Alice-Bob game), outputs the index i and the permutation α required to compute the bit $h(\sigma_1, b_1, \ldots \sigma_n, b_n) = [\alpha = \pi_i?]$ of h's truth table.

The algorithm looks at every pair of moves $(\sigma_1, b_1), \ldots (\sigma_n, b_n)$ one time and maintains in memory a permutation α, an index $k \in [n]$, and two indexes $1 \le s \le t \le N$. These values in memory have the following meaning: after round k, Alice has stated that $\pi_s \ldots \pi_t = \alpha$. Initially α is the identity, $k = 0$, $i = 1$, and $j = N$.

The move of Alice in the k-th round is some permutation $\sigma_k \in S_5$, and Alice states that $\pi_s \ldots \pi_m = \sigma_k$, where $m = \frac{s+t}{2}$. The move of Bob (some bit b_k) is a choice "left" or "right". If Bob's answer is "left", then we set $s := s, t := m$, $\alpha := \sigma$ and if Bob's choice is "right" then we set $s := m+1$, $t := t$, and $\alpha := \sigma_k^{-1} \cdot \alpha$. Each such calculation can be done in constant time, so the entire computation can be done in linear time.

The above circuit is very simple, and the above algorithm shows how to compute the connections between the various gates. Since this algorithms runs in $O(\log N)$ time, this circuit is DLOGTIME-uniform. $\square$

3.2 TQBF vs DT

We have now laid out enough definitions that we can discuss the crucial difficulty in proving our main result (Theorems 3.1 and 3.2). We would like to prove that circuit-DT is PSPACE-hard, and we know that circuit-TQBF is PSPACE-hard. We would like to prove that tt-DT is NC^1-hard, and we know that tt-TQBF is NC^1-hard. The fundamental difference between the two problems can be understood by looking them as a game.

In TQBF, we have two disjoint sets of variables: Alice sets the y_i variables and Bob sets the x_i variables. In DT, we have a single set of variables: Alice chooses a variable, and Bob sets the variable. Now, what we would like to do, is to simulate the TQBF game using the DT game. The difficulty is that it is not at all obvious how such a simulation should proceed.

To simulate a TQBF game over variables $y_1, x_1, \dots, y_n, x_n$, we use a DT game over variables $y_1, y_1', x_1, x_1', \dots, y_n, y_n', x_n, x_n'$, and some other auxiliary variables. The hope is that the DT instance works in such a way that Alice must play by choosing first y_i and then y_i', or first y_i' and then y_i. Whichever order she chooses, Bob must play by setting the first chosen y_i or y_i' variable to 1, and by setting the second chosen to 0. Then Alice must play by choosing one of the two x_i or x_i' variables, and Bob can set it anyway he wants. This way we simulate the TQBF game using the DT game. The difficulty is now in ensuring that Alice and Bob are indeed forced to play by the above "*standard*" strategies. For this, some additional gadgets will be put in place, so that any deviation from the above standard strategies will cause the deviating player to loose the DT game. So let us begin.

3.3 First auxiliary function

In the reduction we will need an example of a Boolean function W such that for some variable it holds that $\mathsf{D}(W) = \mathsf{D}(W|_{p=1}) \gg \mathsf{D}(W|_{p=0})$. This function will be some kind of product of the following function $w : \{0,1\}^4 \to \{0,1\}$:

$$w(p, a_0, a_1, r) = \begin{cases} 0 & \text{if } p = 1, \text{ and } a_0 = a_1, \\ a_r & \text{if } p = 0, \text{ or } a_0 \neq a_1. \end{cases}$$

Lemma 3.6 ([Zha])**.** *The function w has the following properties:*

1. $\mathsf{D}(w) = 3$*;*
2. $\mathsf{D}(w|_{p=1}) = 3$*;*
3. $\mathsf{D}(w|_{p=0}) = 2$*.*

Proof. (1) To determine w one can ask the values of a_0 and a_1. If $a_0 \neq a_1$ then it is enough to know r to determine the value of the function. If $a_0 = a_1$ then it is enough to know p (if $p = 0$ then $w = a_0 = a_1$, if $p = 1$ then $w = 0$). The proof of the lower bound follows from the second item.

(2) The upper-bound follows from (1). The lower-bound is proven by brute force. We have

$$w(1, a_0, a_1, r) = \begin{cases} 0 & \text{if } a_0 = a_1, \\ a_r & \text{if } a_0 \neq a_1. \end{cases}$$

If we choose to query a_0 and a_1 first, and they differ, we still need to query r. If we choose to query a_i and r first, and $r = 1 - i$, we still need to query a_{1-i}.
(3) To determine $w(0, a_0, a_1, r)$ we may ask r then a_r. It is easy to see that one question is not enough. □

Denote by $W_k : \{0,1\}^{1+3k}$ the Boolean function given by:

$$W_k(p, a_0^1, a_1^1, r^1, \ldots, a_0^k, a_1^k, r^k) = w(p, a_0^1, a_1^1, r^1) \oplus \ldots \oplus w(p, a_0^k, a_1^k, r^k).$$

Lemma 3.7. *The function W_k has the following properties:*

1. $\mathsf{D}(W_k) = 3k$*; Moreover, there is a strategy for the second player (who sets the values) such that if the first player (Alice) chooses variable p then she looses.*

2. $\mathsf{D}(W_k|_{p=1}) = 3k$*;*

3. $\mathsf{D}(W_k|_{p=0}) = 2k$*.*

Proof of Lemma 3.7. It is easy to see that if functions f and g have disjoint variables then $\mathsf{D}(f \oplus g) = \mathsf{D}(f) + \mathsf{D}(g)$. By these reasons the second and the third items are direct corollaries of the same items in Lemma 3.6. To prove the first item just consider the following obvious inequalities:

$$3k = \mathsf{D}(W_k|_{p=1}) \leq \mathsf{D}(W_k) \leq k \cdot \mathsf{D}(w) = 3k.$$ □

3.4 Second auxiliary function

Another tool in the reduction is the following function:

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n, \text{ where:}$$

- every f_i is defined as $f_i(y_i, y_i') = y_i \wedge y_i'$;

- every g_i is defined as

$$g_i(y_1, y_1', x_1, x_1', \ldots y_i, y_i', x_i, x_i') = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

We claim that $\mathsf{D}(F_n) = 3n$. Moreover, we want to claim some properties of the corresponded game between Alice (who chooses variables) and Bob (who sets values). We would like to say that Alice and Bob must follow certain *standard* strategies, or else they will loose the DT game.

- **Standard strategies for Alice.** In a standard strategy for Alice, she spends $2n$ questions to ask about the variables y_i and y'_i for all i. She can ask these questions in an arbitrary order.

  She also spends n questions to ask about exactly one of the two variables x_i and x'_i, for each i. Here the order is crucial. The correct variable to choose depends on the value of $f_1 \oplus g_1 \oplus \cdots \oplus g_{i-1} \oplus f_i$: she chooses x_i if this is 1, and x'_i if this is 0, as in the definition of g_i above. So, before asking about x_i or x'_i, Alice must first ask about $y_1, y'_1, \ldots, y_{i-1}, y'_{i-1}$ and about the appropriate variable in every couple (x_j, x'_j) for all $j < i$. This defines f_j, for all $j \leq i$, and g_j, for all $j < i$.

- **Standard strategies for Bob.** A standard strategy for Bob is any strategy such that, the first time Alice asks about one of the two variables y_i or y'_i, Bob answers 1.

Lemma 3.8. *It holds that $\mathsf{D}(F_n) = 3n$ and, furthermore,*

1. *If Alice plays according to a standard strategy, she wins the DT game within $3n$ rounds.*

2. *If Alice does not play according to a standard strategy then Bob can play in such a way that Alice does not win within $3n$ rounds.*

3. *If, while Alice is playing according to a standard strategy, Bob does not answer according to one of his standard strategies, then Alice can win the DT game in strictly fewer than $3n$ rounds.*

Proof. The first observation follows from items 1 and 2. Item 1 follows because, in a standard strategy, Alice has asked about all variables in the functions f_i, and because she asked about the relevant variable among x_i and x'_i, she also learned all the g_i.

Now we prove item 2. Assume that Alice does not play according to a standard strategy. It means that either (a) Alice does not ask about some y_i or y'_i, or (b) she did not ask about one of the x_i or x'_i, or (c.i) she asks about x_i or x'_i before seeing all the variables of $f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i$, or

(d) for some i, she saw all the variables of $f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i$, but asked about the wrong x_i or x'_i.

In case (a), Alice did not ask about some y_i or y'_i, so that the value of f_i is not defined and independent of the remaining values, and hence the value of F_n is also not defined, so if Bob plays according to any standard strategy, they end in a non-monochromatic subcube, and Alice looses. Likewise, in cases (b) and (d), g_i is undefined and independent of the remaining values, and hence F_n is also undefined, and Alice also looses.

Now suppose we are in case (c.i), but not (a), (b), or (d), or (c.j), for any $j < i$. Then, Alice has asked about one of the variables x_i or x'_i, but she did so before asking every variable of $f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i$. We then claim that Bob can answer Alice's questions in such a way that Alice will need to ask at least $3n+1$ questions in total to know the value of F. Indeed, Alice has asked about x_i or x'_i before $f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i$ became defined. So after choosing some value for the variable x_i or x'_i, Bob can still answer Alice's questions by a standard strategy such that g_i equals the variable x'_i or x_i which Alice did not choose. The remaining questions Bob answers according to an arbitrary standard strategy. Since Alice wasted (at least) one question by asking an irrelevant variable, it follows that she needs $3n$ other questions to fix the value of F_n.

Finally, to prove item 3, suppose that the first time Alice asks one of the two variables y_i or y'_i, Bob answers 0. Then the function $f_i = y_i \wedge y'_i$ becomes fixed and equal to 0, so Alice can fix the value of F without ever asking about the other variable. And so Alice wins within $3n-1$ moves. □

3.5 The reduction

Proof of theorems 3.1 and 3.2. We reduce the TQBF instance

$$\exists y_1 \forall x_1 \ldots \forall x_n h(y_1, x_1, \ldots y_n, x_n) \tag{1}$$

to computing the query complexity $\mathsf{D}(D)$ of the function:

$$D = \begin{cases} q, & \text{if } G_n \vee p = 0 \\ W_{10n} \oplus F_n, & \text{otherwise.} \end{cases}$$

where

$$D = D(x_1, x'_1, y_1, y'_1, \ldots, x_n, x'_n, y_n, y'_n, p, q, a_0^1, a_1^1, r^1, \ldots, a_0^{10n}, a_1^{10n}, r^{10n})$$

is a Boolean function on $2n + 2 + 30n$ variables.

$W_{10n} = W_{10n}(p, a_0^1, a_1^1, r^1, \ldots, a_0^{10n}, a_1^{10n}, r^{10n})$

was defined in Section 3.3.

$$F_n = F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n')$$

was defined in Section 3.4. And

$$G_n = G_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n')$$

is a (previously undefined) Boolean function on $2n$ variables, given by:

$$G_n = h(y_1, x_1, \ldots, y_n, x_n) \vee \bigvee_{i=1}^{n} (y_i \wedge y_i') \vee \bigvee_{i=1}^{n} x_i \oplus x_i'$$

First we note that this reduction can be computed in polynomial time if h is given as a circuit and by a NC^0-circuit if h is given a truth-table: every value of D depends only on one value of h. To see DLOGTIME-uniformity one can check that, given values for the input variables of D, one can calculate the ouput, as a function of h, in time $O(n)$. We simply calculate every value in the above expression for D in time $O(n)$, and as a result, the output will be either 1, 0, $h_n(\ldots)$ or $\neg h_n(\ldots)$. So the truth table of D can be computed from the truth table of h by a DLOGTIME-uniform NC^0-reduction.

We now claim that (1) is true if and only if $\mathsf{D}(D) \leq 33n$.

When (1) holds

First we prove if (1) is true, i.e., Alice has a winning strategy in the TQBF-game, then $\mathsf{D}(D) \leq 33n$, i.e. Alice has a winning strategy in the DT-game, which allows her to win within $33n$ steps.

In the TQBF-game Alice first chooses y_1, then y_2 as a function of y_1 and (Bob's choice for) x_1 then y_3 as a function from y_1, x_1, y_2, x_2, and so on. We wish to translate such a strategy for the TQBF-game, into a strategy for the DT-game. The translation is the following.

Alice begins by fixing the value of F_n to a constant. She will do so, by using the following standard DT-strategy (*standard* as in the proof of Lemma 3.8). If the TQBF-strategy of Alice sets $y_1 = 1$, then the standard DT-strategy of Alice first asks about the variable y_1, and then asks about the variable y_1'. If the TQBF-strategy of Alice sets $y_1 = 0$, the standard DT-strategy of Alice swaps the order: first asking about y_1' and then about y_1. The standard DT-strategy of Alice then asks about the appropriate relevant variable x_1 or x'_1 (according to her standard strategy). She considers the value of the chosen variable (x_1 or x'_1) to be the value Bob has chosen for x_1 in TQBF-game. Then, the DT-strategy of Alice asks about y_2 and y'_2 in some order, that again depends on the corresponding value for y_2 in the TQBF-strategy, and so on.

Now, either Bob follows a standard strategy (as in Lemma 3.8), or not. If he does, then (because the TQBF-strategy is a winning strategy for Alice) the value of h is equal to 1 and hence G_n is also equal to 1 and therefore D is equal to $F_n \oplus W_{10n}$. Note, that the value of F_n is already defined. By Lemma 3.7 Alice can define the value of W_{10n} in remaining $30n$ moves, so she wins.

If Bob does not use a standard strategy, Alice can define the value of F_n in fewer than $3n$ moves. She can do it by Lemma 3.8. Then Alice asks about p. Then, either:

- *Bob answers* $p = 1$. In this case, D is equal to $F_n \oplus W_{10n}$, F_n is already defined and Alice can ask at least $30n$ questions, which is enough to determine the value of W_{10n}.

- *Bob answers* $p = 0$. In this case by Lemma 3.7 Alice can define the value of $W_{10n}|_{p=0}$ in $20n$ questions, so she has at least $10n$ questions left. She uses these questions to ask about all the variables $q, x_1, x'_1, y_1, y'_1, \ldots, x_n, x'_n, y_n$ and y'_n. Therefore, now Alice knows the values of W_{10n}, q, p, G_n, and F_n, so Alice knows the value of D.

When (1) is false

Now we prove that if (1) is false then Alice cannot win in $33n$ moves. First we prove the following

Lemma 3.9. *Assume that* (1) *is false. Then, Bob can answer the questions of Alice about the variables of* F_n *in such a way that*

- F_n *will not be defined as long as Alice has asked fewer than* $3n$ *questions.*

- *If Alice has asked exactly* $3n$ *questions, then either (i) the value of* F_n *is not defined, or (ii) the value of* F_n *is defined, but there exists an assignment of variables not asked by Alice such that* $G_n = 0$.

Proof of Lemma 3.9. From Lemma 3.8 we know that, if Alice does not play according to a standard strategy, she will need more than $3n$ questions to define F_n. So we can assume that Alice plays according to a standard strategy. Bob will also play according to a standard strategy, with the following additional constraint: for every pair $\{y_i, y_i'\}$ Bob will answer 0 to the second requested variable. (Recall that, a standard strategy for Bob is one where he answers 1 for the first requested variable in the pair.)

We claim that if (1) is false, and Alice uses a standard strategy, then Bob can make $x_i = x_i'$ for every i, and $h = 0$, thus forcing $G_n = 0$. Indeed, in a standard strategy Alice asks variables in the right order, so that Bob can set x_i and x_i' to the same value, according to his winning strategy in the TQBF-game. This makes the value of h equal to 0. □

Now we are ready to describe the strategy for Bob. To recall: we are assuming that (1) is false, and we will devise a strategy for Bob that will leave D undefined unless Alice asks more than $33n$ queries.

Bob's strategy.

- For the variables that define F_n, Bob uses the strategy from Lemma 3.9.
- If Alice asks about p then Bob answers 1.
- For the variables that define W_{10n}, Bob uses some best strategy for $W_{10n}|_{p=1}$.
- For q the answer is arbitrary.

We argue that if Bob uses this strategy then Alice cannot define the value of D in $33n$ questions. Indeed, assume that Alice asks about p. Then $p = 1$ and the value of D is equal to the value of $F_n \oplus W_{10n}$. Now either Alice asked $< 3n$ questions about F_n or $< 30n$ questions about $W_{10n}|_{p=1}$. Either way, one of these functions is not defined and hence D is also not defined.

Now assume that Alice does not ask about p. Then by setting $p = 1$, we can always force the value of D to be equal to $F_n \oplus W_{10n}$, so this value must be defined. But to define F_n and W_{10n}, Alice must spend all her $33n$ questions, and so she cannot ask about q. Moreover, to learn the value of F_n, she must spend exactly $3n$ questions about F_n, but she cannot spend any more. From Lemma 3.9, there must then exist some setting of the variables Alice didn't ask, such that $G_n = 0$. Then, by way of this assignment together with $p = 0$, D becomes equal to q, which Alice did not ask and hence is not defined. □

4 Open questions

1. What is the exact time-complexity of tt-DT? Is it possible to improve $O(3^n n)$-algorithm of Proposition 2.6? Is it possible to prove any non-trivial bounds (for example, under the Exponential Time Hypothesis)?

2. Is it possible to improve the $O(\log N \log\log N)$-depth bound of Proposition 2.9?

3. What is the exact time, space, and circuit complexity of the problem of finding the minimum *size* of a decision tree that computes a given Boolean function? It is known that this problem belongs to P[Aar03], but the best depth upper-bound we know is $O((\log N)^2)$. It seemed to us that our reduction cannot be adapted to this case without a significantly new idea.

4. What can we say about the problem of *approximating* decision-tree complexity? One can consider, in the reduction above, the function $D_n^1 \oplus \cdot D_n^k$ instead of D_n for some k, where all D_n^i are the same functions as D_n with fresh variables. It allows to prove that the problem of the approximation of DT with constant term has the same complexity as exact calculation of DT. Is it possible to improve on this result?

Acknowledgments

The authors would like to thank Wei Zhan for his answer at StackOverflow. This work was funded by the European Union (ERC, HOFGA, 101041696). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. It was also supported by FCT through the LASIGE Research Unit, ref. UIDB/00408/2020 and ref. UIDP/00408/2020, and by CMAFcIO, FCT Project UIDB/04561/2020, `https://doi.org/10.54499/UIDB/04561/2020`.

References

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

[Bar86] David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 1–5, 1986.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Chapman and Hall/CRC, 1984.

[BIS90] David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[GJPW18] Mika Göös, T. S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Trans. Comput. Theory*, 10(1):4:1–4:20, 2018.

[GLR99] David Guijarro, Vıctor Lavın, and Vijay Raghavan. Exact learning when irrelevant variables abound. *Information Processing Letters*, 70(5):233–239, 1999.

[Göö15] Mika Göös. Lower bounds for clique vs. independent set. *Electron. Colloquium Comput. Complex.*, TR15-012, 2015.

[GPW15] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1077–1088. IEEE Computer Society, 2015.

[Kas80] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127, 1980.

[KST23] C. Koch, C. Strassle, and L. Tan. Properly learning decision trees with queries is np-hard. pages 2383–2407, nov 2023.

[Lar13] Kasper Green Larsen. *Models and techniques for proving data structure lower bounds.* PhD thesis, Aarhus University, 2013.

[Pă08] Mihai Pătraşcu. *Lower bound techniques for data structures.* PhD thesis, Massachusetts Institute of Technology, 2008.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[RTV04] Omer Reingold, Luca Trevisan, and Salil Vadhan. Notions of reducibility between cryptographic primitives. In *Theory of Cryptography Conference*, pages 1–20. Springer, 2004.

[Yao75] Andrew Chi-Chih Yao. On the complexity of comparison problems using linear functions. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 85–89. IEEE Computer Society, 1975.

[Zha] Wei Zhan. `https://cstheory.stackexchange.com/questions/52546/a-question-about-decision-tree-complexity`.