Improved bounds for the randomized decision tree complexity of recursive majority [*]

Frédéric Magniez[1], Ashwin Nayak[†2], Miklos Santha[‡1,3],
Jonah Sherman[4], Gábor Tardos[§5], and David Xiao[1]

[1]CNRS, LIAFA, Univ Paris Diderot, Paris, France
[2]C&O and IQC, University of Waterloo, Waterloo, Canada
[3]Centre for Quantum Technologies, National U. of Singapore, Singapore
[4]CS Division, University of California, Berkeley, USA
[5]Rényi Institute, Budapest, Hungary

Abstract

We consider the randomized decision tree complexity of the recursive 3-majority function. We prove a lower bound of $(1/2 - \delta) \cdot 2.57143^h$ for the two-sided-error randomized decision tree complexity of evaluating height h formulae with error $\delta \in [0, 1/2)$. This improves the lower bound of $(1 - 2\delta)(7/3)^h$ given by Jayram, Kumar, and Sivakumar (STOC'03), and the one of $(1 - 2\delta) \cdot 2.55^h$ given by Leonardos (ICALP'13). Second, we improve the upper bound by giving a new zero-error randomized decision tree algorithm that has complexity at most $(1.007) \cdot 2.64944^h$. The previous best known algorithm achieved complexity $(1.004) \cdot 2.65622^h$. The new lower bound follows from a better analysis of the base case of the recursion of Jayram *et al.* The new algorithm uses a novel "interleaving" of two recursive algorithms.

1 Introduction

Decision trees form a simple model for computing boolean functions by successively reading the input bits until the value of the function can be determined. In this model, the only cost we consider the number of input bits queried. This allows us to study the complexity of computing a function in terms of its structural properties. Formally, a *deterministic decision tree algorithm* A on n variables is a binary tree in which each internal node is labeled with an input variable x_i,

and the leaves of the tree are labeled by either 0 or 1. Each internal node has two outgoing edges, one labeled with 0, the other with 1. Every input $x = x_1 \ldots x_n$ determines a unique path in the tree leading from the root to a leaf: if an internal node is labeled by x_i, we follow either the 0 or the 1 outgoing edge according to the value of x_i. The value of the algorithm A on input x, denoted by $A(x)$, is the label of the leaf on this unique path. Thus, the algorithm A *computes* a boolean function $A : \{0,1\}^n \to \{0,1\}$.

We define the *cost* $\mathrm{C}(A, x)$ of a deterministic decision tree algorithm A on input x as the number of input bits queried by A on x. Let $\mathcal{P}_f$ be the set of all deterministic decision tree algorithms which compute f. The *deterministic complexity* of f is $\mathrm{D}(f) = \min_{A \in \mathcal{P}_f} \max_{x \in \{0,1\}^n} \mathrm{C}(A, x)$. Since every function can be evaluated after reading all the input variables, $\mathrm{D}(f) \leq n$.

In an extension of the deterministic model, we can also permit randomization in the computation. A *randomized decision tree algorithm* A on n variables is a distribution over all deterministic decision tree algorithms on n variables. Given an input x, the algorithm first samples a deterministic tree $B \in_{\mathrm{R}} A$, then evaluates $B(x)$. The error probability of A in computing f is given by $\max_{x \in \{0,1\}^n} \Pr_{B \in_{\mathrm{R}} A}[B(x) \neq f(x)]$. The *cost* of a randomized algorithm A on input x, denoted also by $\mathrm{C}(A, x)$, is the expected number of input bits queried by A on x. Let $\mathcal{P}_f^\delta$ be the set of randomized decision tree algorithms computing f with error at most δ. The two-sided bounded error *randomized complexity* of f with error $\delta \in [0, 1/2)$ is $\mathrm{R}_\delta(f) = \min_{A \in \mathcal{P}_f^\delta} \max_{x \in \{0,1\}^n} \mathrm{C}(A, x)$.

We write $\mathrm{R}(f)$ for $\mathrm{R}_0(f)$. By definition, for all $0 \leq \delta < 1/2$, it holds that $\mathrm{R}_\delta(f) \leq \mathrm{R}(f) \leq \mathrm{D}(f)$, and it is also known [BI87, HH87, Tar90] that $\mathrm{D}(f) \leq \mathrm{R}(f)^2$, and that for all constant $\delta \in (0, 1/2)$, $\mathrm{D}(f) \in \mathrm{O}(\mathrm{R}_\delta(f)^3)$ [Nis89].

Considerable attention in the literature has been given to the randomized complexity of functions computable by read-once formulae, which are boolean formulae in which every input variable appears only once. For a large class of well balanced formulae with NAND gates the exact randomized complexity is known. In particular, let NAND_h denote the *complete* binary tree of height h with NAND gates, where the inputs are at the $n = 2^h$ leaves. Snir [Sni95] has shown that $\mathrm{R}(\mathsf{NAND}_h) \in \mathrm{O}(n^c)$ where $c = \log_2\left(\frac{1+\sqrt{33}}{4}\right) \approx 0.753$. A matching $\Omega(n^c)$ lower bound was obtained by Saks and Wigderson [SW86], and extended to Monte Carlo algorithms (i.e., with constant error $\delta < 1/2$) by Santha [San95]. Since $\mathrm{D}(\mathsf{NAND}_h) = 2^h = n$ this implies that $\mathrm{R}(\mathsf{NAND}_h) \in \Theta(\mathrm{D}(\mathsf{NAND}_h)^c)$. Saks and Wigderson conjectured that this is the largest gap between deterministic and randomized complexity: *for every* boolean function f and constant $\delta \in [0, 1/2)$, $\mathrm{R}_\delta(f) \in \Omega(\mathrm{D}(f)^c)$. For the zero-error (Las Vegas) randomized complexity of read-once threshold formula of depth d, Heiman, Newman, and Wigderson [HNW90] proved a lower bound of $\Omega(n/2^d)$. Heiman and Wigderson [HW91] proved that the zero-error randomized complexity of every read-once formula f is at least $\Omega(\mathrm{D}(f)^{0.51})$.

After such progress, one would have hoped that the simple model of decision tree algorithms might shed more light on the power of randomness. But surprisingly, we know the exact randomized complexity of very few boolean functions. In particular, the randomized complexity of the recursive 3-majority function ($\mathsf{3\text{-}MAJ}_h$) is still open. This function, proposed by Boppana, was one of the earliest examples where randomized algorithms were found to be more powerful than deterministic decision trees [SW86]. It is a read-once formula on 3^h variables given by the complete ternary tree of height h whose internal nodes are majority gates. It is easy to check that $\mathrm{D}(\mathsf{3\text{-}MAJ}_h) = 3^h$, but there is a naive randomized recursive algorithm for $\mathsf{3\text{-}MAJ}_h$ that performs better: pick two random children of the root and recursively evaluate them, then evaluate the third child if the value is not yet determined. This has zero-error randomized complexity $(8/3)^h$. However, it was already

observed by Saks and Wigderson [SW86] that one can do even better than this naive algorithm. As for lower bounds, that reading 2^h variables is necessary for zero-error algorithms is easy to show. In spite of some similarities with the NAND_h function, no progress was reported on the randomized complexity of $\mathsf{3\text{-}MAJ}_h$ for 17 years. In 2003, Jayram, Kumar, and Sivakumar [JKS03] proposed an explicit randomized algorithm that achieves complexity $(1.004) \cdot 2.65622^h$, and beats the naive recursion. (Note, however, that the recurrence they derive in [JKS03, Appendix B] is incorrect.) They also prove a $(1 - 2\delta)(7/3)^h$ lower bound for the δ-error randomized decision tree complexity of $\mathsf{3\text{-}MAJ}_h$. In doing so, they introduce a powerful combinatorial technique for proving decision tree lower bounds.

In this paper, we considerably improve the lower bound obtained in [JKS03], first by proving that $\mathrm{R}_\delta(\mathsf{3\text{-}MAJ}_h) \geq (1 - 2\delta)(5/2)^h$, then further improving the base $5/2$. We also improve the upper bound by giving a new zero-error randomized decision tree algorithm.

Theorem 1.1. *For all $\delta \in [0, 1/2]$, we have*

$$(1/2 - \delta) \cdot 2.57143^h \quad \leq \quad \mathrm{R}_\delta(\mathsf{3\text{-}MAJ}_h) \quad \leq \quad (1.007) \cdot 2.64944^h \ .$$

In contrast to the randomized case, the bounded-error *quantum* query complexity of $\mathsf{3\text{-}MAJ}_h$ is known more precisely; it is in $\Theta(2^h)$ [RS08].

New lower bound. For the lower bound, Jayram *et al.* consider a complexity measure related to the distributional complexity of $\mathsf{3\text{-}MAJ}_h$ with respect to a specific "hard" distribution (cf. Section 2.3). The focus of the proof is a relationship between the complexity of evaluating formulae of height h to that of evaluating formulae of height $h - 1$. They derive a sophisticated recurrence relation between these two quantities, that finally implies that $\mathrm{R}_\delta(\mathsf{3\text{-}MAJ}_h) \geq \alpha(2 + q)^h$, where αq^h is a lower bound on the probability $p_{\delta,h}$ that a randomized algorithm with error at most δ queries a special variable, called the "absolute minority", on inputs drawn from the hard distribution. They observe that any randomized decision tree with error at most δ queries at least one variable with probability $1 - 2\delta$. This variable has probability 3^{-h} of being the absolute minority, so $q = 1/3$ and $\alpha = 1 - 2\delta$ satisfies the conditions and their lower bound follows.

We obtain new lower bounds by improving the bound on $p_{h,\delta}$. We start by proving that $p_{h,\delta} \geq (1 - 2\delta)2^{-h}$, i.e., increasing q to $1/2$, which immediately implies a better lower bound for $\mathrm{R}_\delta(\mathsf{3\text{-}MAJ}_h)$. To obtain this bound, we examine the relationship between $p_{h,\delta}$ and $p_{h-1,\delta}$, by encoding a height $h - 1$ instance into a height h instance, and using an algorithm for the latter instance. Analyzing this encoding requires understanding the behavior of all decision trees on 3 variables, and this can be done by exhaustively considering all such trees.

We further improve this lower bound by encoding height $h-2$ instances into height h instances, and prove $p_{h,\delta} \geq \alpha q^h$ for $q = \sqrt{7/24} > 0.54006$. For technical reasons we set $\alpha = 1/2 - \delta$ (half the value considered by Jayram *et al.* in their bound). For encodings of height $h - 3$ and $h - 4$ instances into height h instances, we use a computer to get the better estimates, with $q = (2203/12231)^{1/3} > 0.56474$ and $q = (216164/2027349)^{1/4} > 0.57143$, respectively.

The lower bound of $(1 - 2\delta)(5/2)^h$ mentioned above was presented in a preliminary version of this article [MNSX11]. Independent of the further improvements we make, Leonardos [Leo13] gave a lower bound of $\mathrm{R}_\delta(\mathsf{3\text{-}MAJ}_h) \geq (1 - 2\delta) \cdot 2.55^h$. His approach is different from ours, and is based on the method of generalized costs proposed by Saks and Wigderson [SW86]. The final lower bound $(1/2 - \delta) \cdot 2.57143^h$ we obtain surpasses the bound due to Leonardos.

New algorithm. The naive algorithm and the algorithm of Jayram *et al.* are examples of *depth-k* recursive algorithms for $\mathsf{3\text{-}MAJ}_h$, for $k = 1, 2$, respectively. A depth-k recursive algorithm

is a collection of subroutines, where each subroutine evaluates a node (possibly using information about other previously evaluated nodes), satisfying the following constraint: when a subroutine evaluates a node v, it is only allowed to call other subroutines to evaluate children of v at depth at most k, but is not allowed to call subroutines or otherwise evaluate children that are deeper than k. (Our notion of depth-one is identical to the terminology "directional" that appears in the literature. In particular, the naive recursive algorithm is a directional algorithm.)

We present an improved depth-two recursive algorithm. To evaluate the root of the majority formula, we recursively evaluate one grandchild from each of two distinct children of the root. The grandchildren "give an opinion" about the values of their parents. The opinion guides the remaining computation in a natural manner: if the opinion indicates that the children are likely to agree, we evaluate the two children in sequence to confirm the opinion, otherwise we evaluate the third child. If at any point the opinion of the nodes evaluated so far changes, we modify future computations accordingly. A key innovation is the use of an algorithm optimized to compute the value of a *partially evaluated* formula. In the analysis, we recognize when incorrect opinions are formed, and take advantage of the fact that this happens with smaller probability.

We do not believe that the algorithm we present here is optimal. Indeed, we conjecture that even better algorithms exist that follow the same high level intuition applied to depth-k recursion, for $k > 2$. However, it seems new insights are required to analyze the performance of deeper recursions, as the formulas describing their complexity become unmanageable for $k > 2$.

Organization. We prepare the background for the main results Section 2. In Section 3 we prove the new lower bounds for $\mathsf{3\text{-}MAJ}_h$. The new algorithm for the problem is described and analyzed in Section 4.

2 Preliminaries

We write $u \in_{\mathrm{R}} D$ to state that u is sampled from the distribution D. If X is a finite set, we identify X with the uniform distribution over X, and so, for instance, $u \in_{\mathrm{R}} X$ denotes a uniform element of X.

2.1 Distributional Complexity

A variant of the randomized complexity we use is *distributional* complexity. Let $\mathcal{D}_n$ be the set of distributions over $\{0, 1\}^n$. The *cost* $\mathrm{C}(A, D)$ of a randomized decision tree algorithm A on n variables with respect to a distribution $D \in \mathcal{D}_n$ is the expected number of bits queried by A, where the expectation is taken over inputs sampled from D and the random coins of A. The *distributional complexity* of a function f on n variables for δ two-sided error is $\Delta_\delta(f) = \max_{D \in \mathcal{D}_n} \min_{A \in \mathcal{P}_f^\delta} \mathrm{C}(A, D)$. The following observation is a well established route to proving lower bounds on worst case complexity.

Proposition 2.1. $\mathrm{R}_\delta(f) \geq \Delta_\delta(f)$.

2.2 The $\mathsf{3\text{-}MAJ}_h$ Function and the Hard Distribution

Let $\mathsf{MAJ}(x)$ denote the boolean majority function of its input bits. The ternary majority function $\mathsf{3\text{-}MAJ}_h$ is defined recursively on $n = 3^h$ variables, for every $h \geq 0$. We omit the height h when it is obvious from context. For $h = 0$ it is the identity function. For $h > 0$, let x be an input of length

n and let $x^{(1)}, x^{(2)}, x^{(3)}$ be the first, second, and third $n/3$ variables of x. Then

$$\text{3-MAJ}_h(x) \quad = \quad \text{MAJ}(\text{3-MAJ}_{h-1}(x^{(1)}), \ \text{3-MAJ}_{h-1}(x^{(2)}), \ \text{3-MAJ}_{h-1}(x^{(3)})) \ .$$

In other terms, 3-MAJ_h is defined by the read-once formula on the complete ternary tree T_h of height h in which every internal node is a majority gate. We identify the leaves of T_h from left to right with the integers $1, \ldots, 3^h$. For an input $x \in \{0,1\}^h$, the bit x_i defines the *value* of the leaf i, and then the values of the internal nodes are evaluated recursively. The value of the root is $\text{3-MAJ}_h(x)$. For every node v in T_h different from the root, let $P(v)$ denote the parent of v. We say that v and w are siblings if $P(v) = P(w)$. For any node v in T_h, let $Z(v)$ denote the set of variables associated with the leaves in the subtree rooted at v. We say that a node v is at depth d in T_h if the distance between v and the root is d. The root is therefore at depth 0, and the leaves are at depth h.

We now define recursively, for every $h \geq 0$, the set $\mathcal{H}_h$ of *hard inputs* of height h. In the base case $\mathcal{H}_0 = \{0,1\}$. For $h > 0$, let

$$\mathcal{H}_h \quad = \quad \{(x,y,z) \in \mathcal{H}_{h-1} \times \mathcal{H}_{h-1} \times \mathcal{H}_{h-1} : \text{3-MAJ}_{h-1}(x), \text{3-MAJ}_{h-1}(y), \text{ and}$$
$$\text{3-MAJ}_{h-1}(z) \text{ are not all identical} \} \ .$$

The hard inputs consist of instances for which at each node v in the ternary tree, one child of v has value different from the value of v. The *hard distribution* on inputs of height h is defined to be the uniform distribution over $\mathcal{H}_h$. We call a hard input x *0-hard* or *1-hard* depending on whether $\text{3-MAJ}_h(x) = 0$ or 1. We write $\mathcal{H}_h^0$ for the set of 0-hard inputs and $\mathcal{H}_h^1$ for the set of 1-hard inputs. A moment's notice shows that the cardinality of both $\mathcal{H}_h^0$ and $\mathcal{H}_h^1$ is $3 \cdot 3^3 \cdots 3^{3^h}$.

For an $x \in \mathcal{H}_h$, the *minority path* $M(x)$ is the path, starting at the root, obtained by following the child whose value disagrees with its parent. For $0 \leq d \leq h$, the node of $M(x)$ at depth d is called the depth d minority node, and is denoted by $M(x)_d$. We call the leaf $M(x)_h$ of the minority path the *absolute minority* of x, and denote it by $m(x)$.

2.3 The Jayram-Kumar-Sivakumar Lower Bound

For a deterministic decision tree algorithm B computing 3-MAJ_h, let $L_B(x)$ denote the set of variables queried by B on input x. Recall that $\mathcal{P}_{\text{3-MAJ}_h}^\delta$ is the set of all randomized decision tree algorithms that compute 3-MAJ_h with two-sided error at most δ. Jayram *et al.* define the function $I_\delta(h,d)$, for $d \leq h$:

$$I_\delta(h,d) \quad = \quad \min_{A \in \mathcal{P}_{\text{3-MAJ}_h}^\delta} \mathbb{E}_{x \in_{\mathrm{R}} \mathcal{H}_h, B \in_{\mathrm{R}} A}[|Z(M(x)_d) \cap L_B(x)|] \ .$$

In words, it is the minimum over algorithms computing 3-MAJ_h, of the expected number of queries below the dth level minority node, over inputs from the hard distribution. Note that $I_\delta(h,0) = \min_{A \in \mathcal{P}_{\text{3-MAJ}_h}^\delta} \mathrm{C}(A, \mathcal{H}_h)$, and therefore by Proposition 2.1, $\mathrm{R}_\delta(\text{3-MAJ}_h) \geq I_\delta(h,0)$.

We define $p_{h,\delta} = I_\delta(h,h)$, which is the minimal probability that a δ-error algorithm A queries the absolute minority of a random hard x of height h.

Jayram *et al.* prove a recursive lower bound for $I_\delta(h,d)$ using information theoretic arguments. A more elementary proof can be found in [LNPV06].

5

Theorem 2.2 (Jayram, Kumar, Sivakumar [JKS03]). *For all $0 \leq d < h$:*

$$I_\delta(h, d) \quad \geq \quad I_\delta(h, d+1) + 2I_\delta(h-1, d) \ .$$

We do not know whether the above recurrence is optimal. A simple computation gives then the following lower bound on $I_\delta(h, d)$, for all $0 \leq d \leq h$, expressed as a function of the $p_{i,\delta}$'s:

$$I_\delta(h, d) \quad \geq \quad \sum_{i=d}^{h} \binom{h-d}{i-d} 2^{h-i} p_{h,\delta} \ .$$

When $d = 0$, this gives $I_\delta(h, 0) \geq \sum_{i=0}^{h} \binom{h}{i} 2^{h-i} p_{i,\delta}$. Putting this together with the fact that $R_\delta(\text{3-MAJ}_h) \geq I_\delta(h, 0)$, we get the following corollary:

Corollary 2.3. *Let $q, a > 0$ such that $p_{i,\delta} \geq a \cdot q^i$ for all $i \in \{0, 1, 2, \ldots, h\}$. Then $R_\delta(\text{3-MAJ}_h) \geq a(2+q)^h$.*

As mentioned in Section 1, Jayram *et al.* obtain the lower bound of $(1 - 2\delta)(7/3)^h$ from this corollary by observing that $p_{h,\delta} \geq (1 - 2\delta)(1/3)^h$.

3 Improved Lower Bounds

3.1 First Improvement

In this section, we develop a method to enhance the Jayram-Kumar-Sivakumar technique for establishing a lower bound for 3-MAJ. The enhancement comes from an improved estimate for $p_{h,\delta}$, the minimum probability with which a decision tree queries the absolute minority of an input drawn from the hard distribution.

Theorem 3.1. *For every error $\delta > 0$ and height $h \geq 0$, we have $p_{h,\delta} \geq (1 - 2\delta)2^{-h}$.*

Proof. We prove this theorem by induction. Clearly, $p_{0,\delta} \geq 1 - 2\delta$. It then suffices to show that $2p_{h,\delta} \geq p_{h-1,\delta}$ for $h \geq 1$. We do so by reduction as follows: let A be a randomized algorithm that achieves the minimal probability $p_{h,\delta}$ for height h formulae. We construct a randomized algorithm A' for height $h - 1$ formulae such that the probability that A' errs is at most δ, and A' queries the absolute minority with probability at most $2p_{h,\delta}$. Since $p_{h-1,\delta}$ is the minimum probability of querying the absolute minority in the hard distribution, computed over all randomized algorithms on inputs of height $h - 1$ with error at most δ, this implies that $2p_{h,\delta} \geq p_{h-1,\delta}$.

We now specify the reduction. For the sake of simplicity, we omit the error δ in the notation. We use the following definition:

Definition 3.2 (One level encoding scheme). *A one level encoding scheme is a bijection $\psi : \mathcal{H}_{h-1} \times \{1, 2, 3\}^{3^{h-1}} \to \mathcal{H}_h$, such that for all (y, r) in the domain, $\text{3-MAJ}_{h-1}(y) = \text{3-MAJ}_h(\psi(y, r))$.*

Let $c : \{0, 1\} \times \{1, 2, 3\} \to \mathcal{H}_1$ be a bijection satisfying $b = \text{MAJ}(c(b, s))$ for all inputs (b, s). Define the one level encoding scheme ψ induced by c as follows: $\psi(y, r) = x \in \mathcal{H}_h$ such that for all $1 \leq i \leq 3^{h-1}$, $(x_{3i-2}, x_{3i-1}, x_{3i}) = c(y_i, r_i)$.

To define A', we use the one level encoding scheme ψ induced by the following function: $c(y, 1) = y01$, $c(y, 2) = 1y0$, and $c(y, 3) = 01y$.

On input y, algorithm A' picks a uniformly random string $r \in \{1, 2, 3\}^{3^{h-1}}$, and runs A on $x = \psi(y, r)$ and computes the same output. Notice that each bit of x_i of x is either determined by r alone or else it is $y_{\lceil i/3 \rceil}$. When A asks for a bit of x that is determined by r, then this value is "hard wired" in A' and A' makes no query. When A asks for a bit of x that is not determined by r, then and A' queries the corresponding bit of y. Observe that A' has error at most δ as $\mathsf{3\text{-}MAJ}_{h-1}(y) = \mathsf{3\text{-}MAJ}_h(\psi(y, r))$ for all r, and A has error at most δ. We claim that

$$2 \Pr_{B \in_R A,\ x \in_R \mathcal{H}_h} [B(x) \text{ queries } x_{m(x)}] \quad \geq \quad \Pr_{B \in_R A,\ (y,r) \in_R \mathcal{H}_h'} [B'(y, r) \text{ queries } y_{m(y)}] \ , \tag{1}$$

where $\mathcal{H}_h'$ is the uniform distribution over $\mathcal{H}_{h-1} \times \{1, 2, 3\}^{3^{h-1}}$ and B' is the algorithm that computes $x = \psi(y, r)$ and then evaluates $B(x)$. We prove this inequality by taking an appropriate partition of the probabilistic space of hard inputs $\mathcal{H}_h$, and prove Eq. (1) separately, on each set in the partition. For $h = 1$, the two classes of the partition are $\mathcal{H}_1^0$ and $\mathcal{H}_1^1$. For $h > 1$, the partition consists of the equivalence classes of the relation $\sim$ defined by $x \sim x'$ if $x_i = x_i'$ for all i satisfying $P(i) \neq P(m(x))$ in the tree T.

Because ψ is a bijection, observe that this also induces a partition of (y, r), where $(y, r) \sim (y', r')$ if and only if $\psi(y, r) \sim \psi(y', r')$. Also observe that every equivalence class contains three elements. Then Eq. (1) follows from the following stronger statement: for every equivalence class S, and for all B in the support of A, it holds that

$$2 \Pr_{x \in_R \mathcal{H}_h} [B(x) \text{ queries } x_{m(x)} \mid x \in S] \quad \geq \quad \Pr_{(y,r) \in_R \mathcal{H}_h'} [B'(y, r) \text{ queries } y_{m(y)} \mid \psi(y, r) \in S] \ . \tag{2}$$

The same proof applies to all sets S, but to simplify the notation, we consider a set S that satisfies the following: for $x \in S$, we have $m(x) \in \{1, 2, 3\}$ and $x_{m(x)} = 1$. Observe that for each $j > 3$, the jth bits of all three elements in S coincide. Therefore, the restriction of B to the variables (x_1, x_2, x_3), when looking only at the three inputs in S, is a well-defined decision tree on three variables. We call this restriction C, and formally it is defined as follows: for each query x_j made by B for $j > 3$, C simply uses the value of x_j that is shared by all $x \in S$ and that we hard-wire into C; for each query x_j made by B where $j \in \{1, 2, 3\}$, C actually queries x_j. Note that the restriction C does not necessarily compute $\mathsf{MAJ}_1(x_1, x_2, x_3)$, for two reasons. Firstly, C is derived from B, which may err on particular inputs. But even if $B(x)$ correctly computes $\mathsf{3\text{-}MAJ}_h(x)$, it might happen that B never queries any of x_1, x_2, x_3, or it might query one and never query a second one, etc.

For any $x \in S$, recall that we write (y, r) the unique solution of $\psi(y, r) = x$. It holds for our choice of S that $m(y) = 1$ because we assumed $m(x) \in \{1, 2, 3\}$ and also $y_1 = y_{m(y)} = 0$ because we assumed $x_{m(x)} = 1$.

Observe that, for inputs $x \in S$, B queries $x_{m(x)}$ if and only if C queries the minority among x_1, x_2, x_3. Also, $B'(y, r)$ queries $y_{m(y)}$ if and only if $C(\psi(0, r_1))$ queries x_{r_1} (cf. definition of c). Furthermore, the distribution of $x_1 x_2 x_3$ when $x \in_R S$ is uniform over $\mathcal{H}_1^0$. Similarly, the distribution of r_1 over uniform (y, r) conditioned on $\psi(y, r) \in S$ is identical to that of $(0, r_1) = \psi^{-1}(x_1 x_2 x_3)$ for $x_1 x_2 x_3 \in_R \mathcal{H}_1^0$. Thus Eq. (2) is equivalent to:

$$\Pr_{x \in_R \mathcal{H}_1^0} [C(x) \text{ queries } x_{r_1} \text{ where } \psi(0, r_1) = x] \quad \leq \quad 2 \Pr_{x \in_R \mathcal{H}_1^0} [C(x) \text{ queries } x_{m(x)}] \ . \tag{3}$$

In principle, one can prove this inequality by considering all the (finitely many) decision trees C on three variables. We present here a somewhat more compact argument.

If C does not query any bit, both sides of Eq. (3) are zero, so the inequality holds. We therefore assume that C makes at least one query and, without loss of generality, we also assume that the first query is x_1. We distinguish two cases.

If C makes a second query when the first query is evaluated to 0 then the right hand side of Eq. (3) is at least $4/3 = 2 \cdot (1/3 + 1/3)$ because there is a $1/3$ chance that the first query is $m(x)$ and $1/3$ chance that the second is $m(x)$. But the left hand side is at most 1, and therefore the inequality holds.

If C does not make a second query when the first query is evaluated to 0 then the left hand side is at most $2/3$ since for $x = 010$, we have $r_1 = 3$, but x_3 is not queried. With probability $1/3$ we have $m(x) = 1$, so the right hand side is at least $2/3$. We conclude that Eq. (3) holds for every C.

We remark that the decision tree algorithm making no queries is not the only one that makes Eq. (3) hold with equality. Another such algorithm is the following: first query x_1, if $x_1 = 0$, stop, else if $x_1 = 1$, query x_2 and stop.

To handle a general S, we replace $\{1, 2, 3\}$ with $m(x)$ and its two siblings. For S such that $x \in S$ satisfies $x_{m(x)} = 0$, the optimal algorithm C' is the same as the one described above, except that each 0 is changed to 1 and vice versa.

Therefore Eq. (3) holds for every C, which implies the theorem. $\square$

Combining Corollary 2.3 and Theorem 3.1, we obtain the following lower bound.

Corollary 3.3. $R_\delta(\text{3-MAJ}_h) \geq (1 - 2\delta)(5/2)^h$.

3.2 Further Improvement

The proof of Theorem 3.1 proceeds by proving a recurrence, using a one level encoding scheme, for the minimal probability $p_{h,\delta}$ that an algorithm queries the absolute minority bit. It is natural to ask whether this is the best possible recurrence. In this section, we show that it is indeed possible to improve the recurrence by using higher level encoding schemes.

In the following, we sometimes omit the error parameter δ from the notation in the interest of readability. Let $\mathcal{R} = \{0, 1\} \times \{1, 2, 3\}$, $\mathcal{R}_h^{(1)} = \mathcal{R}^{3^{h-1}}$, and $\mathcal{R}_h^{(k)} = \mathcal{R}_h^{(k-1)} \times \mathcal{R}^{3^{h-k}}$.

Definition 3.4 (Uniform k-level encoding scheme). *Let $c : \{0, 1\} \times \mathcal{R} \to \mathcal{H}_1$ be the function given by $c(y, (b, 1)) = yb(1 - b)$, $c(y, (b, 2)) = (1 - b)yb$, and $c(y, (b, 3)) = b(1 - b)y$. The uniform k-level encoding scheme $\psi^{(k)}$, for an integer $k \geq 1$ is defined by the following recursion:*

1. *For $h \geq 1$, $y \in \mathcal{H}_{h-1}$ and $r \in \mathcal{R}_h^{(1)}$ we set $\psi^{(1)}(y, r) = x \in \mathcal{H}_h$ such that $(x_{3i-2}, x_{3i-1}, x_{3i}) = c(y_i, r_i)$, for all $1 \leq i \leq 3^{h-1}$;*

2. *for $h \geq k > 1$, $y \in \mathcal{H}_{h-k}$ and $(R, r) \in \mathcal{R}_h^{(k)}$ we set $\psi^{(k)}(y, (R, r)) = \psi^{(k-1)}(\psi^{(1)}(y, r), R)$.*

The uniform 2-level encoding scheme is illustrated in Figure 1. This encoding is no longer a bijection. However, one can make essentially the same argument as Theorem 3.1. The advantage of this scheme over the one used earlier is the higher symmetry: while in the previous encoding, the instances in $\mathcal{H}_h$ are related by cyclic permutations of triples of three siblings, now the entire
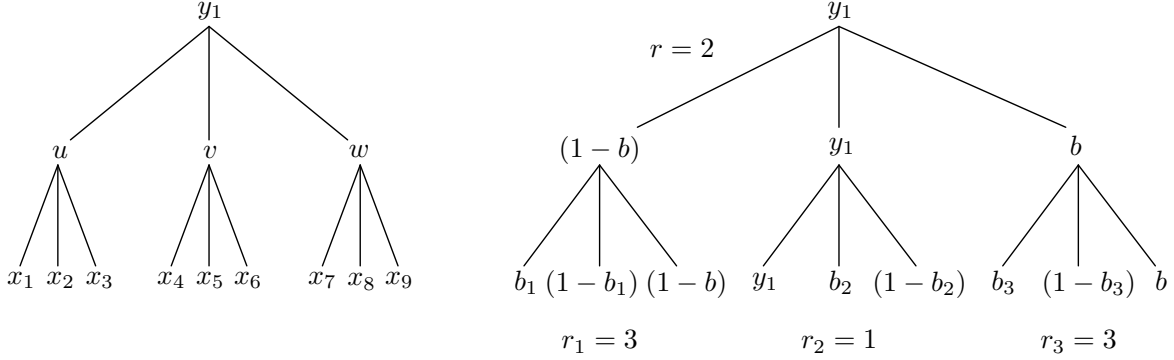
8

Figure 1: In a uniform 2-level encoding, a bit y_1 is encoded as the height-2 recursive majority of 9 bits $x_1 x_2 \cdots x_9$. On the right hand side, we give an example of the 9 bits with specific choices of r, r_1, r_2, r_3 for the two levels. In this example, when $y_1 = 0$, then b_3 encodes the absolute minority bit if $b = 1$ and $b_3 = 0$.

symmetric group acts on them. Because of this higher symmetry if one of three siblings has been queried, the remaining two still play symmetric roles.

We later use the following observations that hold for all $h \geq k \geq 1$:

1. For all $y \in \mathcal{H}_{h-k}$ and $r \in \mathcal{R}_h^{(k)}$ we have $\text{3-MAJ}_{h-k}(y) = \text{3-MAJ}_h(\psi^{(k)}(y, r))$.

2. For $(y, r) \in_{\text{R}} \mathcal{H}_{h-k} \times \mathcal{R}_h^{(k)}$ the value $\psi^{(k)}(y, r)$ is distributed uniformly in $\mathcal{H}_h$.

3. For each $r \in \mathcal{R}_h^{(k)}$ and index i in the range $1 \leq i \leq 3^{h-k}$ there is a unique index $q_i(r)$ in the range $(i-1)3^k + 1 \leq q_i(r) \leq i3^k$ such that for all $y \in \mathcal{H}_{h-k}$ we have $x_{q_i(r)} = y_i$, where $x = \psi_h^{(k)}(y, r)$. If $1 \leq j \leq 3^h$ but j is not equal to $q_i(r)$ for any i, then x_j is independent of the choice of y. We call these bits of x the *fixed bits*.

We use the uniform k-level encoding schemes to obtain better bounds on $p_{h,\delta}$. The argument is very similar to the argument in Theorem 3.1. We start with proving a lower bound on $p_{h,\delta}$ based on a parameter computable by considering all the (finitely many) decision trees acting on inputs from $\mathcal{H}_k^0$. Then we proceed to computing this parameter. The high symmetry helps in reducing the number of cases to be considered, but as k grows the length of the calculation increases rather rapidly. We explain the basic structure of the calculation and also include a short Python program implementing it in Appendix A. As an illustration, we do the calculation for $k = 2$ without the use of a computer. For $k = 3, 4$ we include the results of the program. A much more efficient algorithm would be needed to make the calculation for $k = 5$ feasible.

Let us fix $k \geq 1$ and let C be a deterministic decision tree algorithm on inputs of length 3^k that queries at least one variable. We define

$$
\alpha_C \quad = \quad \frac{\Pr_{x \in_{\text{R}} \mathcal{H}_k^0, (y,r) \in_{\text{R}} \psi^{-1}(x)}[C(x) \text{ queries } x_{q_1(r)}]}{\Pr_{x \in_{\text{R}} \mathcal{H}_k^0}[C(x) \text{ queries } x_{m(x)}]} \quad ,
$$

where $\psi = \psi_k^{(k)}$. Since C queries at least one bit, neither the numerator nor the denominator is zero. So α_C is well defined and positive. We emphasize that α_C does not depend on the output of

9

C, it depends only on which input bits C queries. We further define

$$\alpha_k \quad = \quad \max_C \alpha_C \ ,$$

where the maximum extends over all deterministic decision trees C on 3^k variables that query at least one variable.

Theorem 3.5. *For every $k \geq 1$, $h \geq 0$ integers and $\delta \geq 0$ real, we have*

$$p_{h,\delta} \quad \geq \quad (1 - 2\delta) \left(\frac{\alpha_k}{2^k} \right) \alpha_k^{-h/k} \ .$$

Therefore,

$$\mathrm{R}_\delta(\text{3-MAJ}_h) \quad \geq \quad (1 - 2\delta) \left(\frac{\alpha_k}{2^k} \right) \left(2 + \alpha_k^{-1/k} \right)^h \ .$$

Proof. We concentrate on the proof of the first statement; the second follows from Corollary 2.3.

The proof follows the same structure as that of Theorem 3.1 but instead uses a depth-k recursion: we show that

$$\alpha_k \, p_{h,\delta} \quad \geq \quad p_{h-k,\delta} \tag{4}$$

if $h \geq k$. To bound $p_{h,\delta}$ in the base cases $h < k$, we invoke Theorem 3.1, i.e., $p_{h,\delta} \geq (1 - 2\delta)/2^h$, and that $\alpha_k \leq 2^k$ (from the proof of the theorem).

It remains to prove Eq. (4). We proceed as in Theorem 3.1: we consider a randomized δ-error algorithm A for 3-MAJ$_h$ that achieves the minimum defining $p_{h,\delta}$ and construct a randomized algorithm A' for 3-MAJ$_{h-k}$ with the same error that queries the absolute minority of a uniformly random element of $\mathcal{H}_{h-k}$ with probability at most $\alpha_k \, p_{h,\delta}$

To define A', we use the uniform k-level encoding scheme $\psi = \psi^{(k)}$ (see Figure 1 for an illustration of the $k = 2$ case). On input $y \in \mathcal{H}_{h-k}$ the algorithm A' picks a uniformly random element r of $\mathcal{R}_h^{(k)}$ and simulates the decision tree A on input $x = \psi(y, r)$. Whenever A queries a fixed bit of x, the algorithm A' makes no query, and when A queries a bit $x_{q_i(r)}$, A' queries y_i. Define $\mathcal{H}'_h = \mathcal{H}_{h-k} \times \mathcal{R}_h^{(k)}$. Then $(y, r) \in \mathcal{H}'_h$ encodes $\psi(y, r) \in \mathcal{H}_h$.

We partition $\mathcal{H}_h$ into equivalence classes, this time into sets of size 3^l with $l = \sum_{i=0}^{k-1} 3^i$. For $h = k$, the two classes are $\mathcal{H}_k^0$ and $\mathcal{H}_k^1$. For $h > k$, an equivalence class consists of inputs that are identical everywhere except in the height-k subtree containing their absolute minority. More formally, recall that $P(i)$ denotes the parent of a node i in a tree, and let $P^{(k)}$ denote the k-fold composition of P with itself. In other words, $P^{(k)}(i)$ is the ancestor of the node i that is k levels above i. The partition of $\mathcal{H}_h$ for $h > k$ consists of the equivalence classes of the relation defined as $x \sim x'$ iff $x_i = x'_i$ for all i satisfying $P^{(k)}(i) \neq P^{(k)}(m(x))$ in the tree T_h.

Observe that the uniformity of the encoding implies that for every equivalence class S, and all decision trees B in the support of A:

$$\Pr_{x \in_\mathrm{R} \mathcal{H}_h} [B(x) \text{ queries } x_{m(x)} \mid x \in S] \quad = \quad \Pr_{(y,r) \in_\mathrm{R} \mathcal{H}'_h, x = \psi(y,r)} [B(x) \text{ queries } x_{m(x)} \mid x \in S] \ ,$$

and

$$\Pr_{(y,r) \in_\mathrm{R} \mathcal{H}'_h} [B'(y,r) \text{ queries } y_{m(y)} | \psi(y,r) \in S] \quad = \quad \Pr_{x \in_\mathrm{R} \mathcal{H}_h, (y,r) \in_\mathrm{R} \psi^{-1}(x)} [B'(y,r) \text{ queries } y_{m(y)} | x \in S] \ ,$$

10

where B' is the algorithm that first computes $x = \psi(y, r)$ and then evaluates $B(x)$. We then prove that for every equivalence class S, and all B in the support of A, it holds that:

$$\alpha_k \Pr_{x \in_R \mathcal{H}_h} [B(x) \text{ queries } x_{m(x)} \mid x \in S] \quad \geq \quad \Pr_{(y,r) \in_R \mathcal{H}'_h} [B'(y, r) \text{ queries } y_{m(y)} \mid \psi(y, r) \in S] . \quad (5)$$

Proving Eq. (5) for all B and S finishes the proof of the theorem.

Let us fix S and let z be the undetermined part of the input, i.e., the 3^k variables in the height-k subtree containing the absolute minority. Note that the set of possible values of z is either $\mathcal{H}_k^0$ or $\mathcal{H}_k^1$, depending on S. Now a deterministic decision tree B on inputs from S can be considered a deterministic decision tree C for z. Indeed, the queries B asks outside z have a deterministic answer in S that can be hard wired in C. In case C asks no queries at all, then Eq. 5 is satisfied with zero on both sides of the inequality. Otherwise, if the possible values of z come from $\mathcal{H}_k^0$, Eq. (5) follows from $\alpha_C \leq \alpha_k$ (which, in turn, comes from the definition of α_k as a maximum). Finally if the possible values of z are the 1-hard inputs, Eq. (5) is satisfied by symmetry. $\qquad \square$

To apply Theorem 3.5 we need to compute (or estimate) α_k. For any fixed k this is a finite computation, but it is infeasible to do this by enumerating over all possible decision trees C over 3^k variables, even for small values of k.

For a fixed integer $k \geq 1$ and real $\alpha \geq 0$, we introduce a function ρ_α on decision trees on 3^k variables:

$$\rho_\alpha(C) \quad = \quad \Pr_{x \in_R \mathcal{H}_k^0, (y,r) \in_R \psi^{-1}(x)} [C(x) \text{ queries } x_{q_1(r)}] - \alpha \Pr_{x \in_R \mathcal{H}_k^0} [C(x) \text{ queries } x_{m(x)}] . \quad (6)$$

For the decision tree C_0 not querying any variables we have $\rho_\alpha(C_0) = 0$, for other decision trees C we have $\rho_\alpha(C) > 0$ if and only if $\alpha_C > \alpha$. Thus, we have $\alpha_k > \alpha$ if and only if there exists C with $\rho_\alpha(C) > 0$. Finding the maximum, i.e., $\max_C \rho_\alpha(C)$ therefore answers the question whether $\alpha_k > \alpha$. We now focus on maximizing ρ_α for a given pair k, α. The advantage of this approach lies in the linearity of ρ_α, in a sense that we clarify below. This makes it easier to maximize $\rho_\alpha(C)$ than α_C itself.

Let us call a bit of the hard input $x \in \mathcal{H}_k$ *sensitive* if the flipping of this input bit flips the value of $\mathsf{3\text{-}MAJ}_k(x)$. Note that there are exactly 2^k such bits for each hard input, these are the ones where all nodes on the root to leaf path of the ternary tree evaluate to the same value.

Notice that for a fixed $x \in \mathcal{H}_k^0$ and $(y, r) \in_R \psi^{-1}(x)$ the position $q_1(r)$ (where the k-level encoding $\psi = \psi^{(k)}$ "hides" the input variable y) is uniformly distributed over the 2^k sensitive positions. Thus, we can simplify Eq. (6) defining ρ_α as follows:

$$\rho_\alpha(C) \quad = \quad 2^{-k} \pi_q(C) - \alpha \pi_m(C) , \quad (7)$$

where $\pi_q(C)$ is the expected number of sensitive bits queried by $C(x)$ for $x \in_R \mathcal{H}_k^0$ and π_m is the probability that the absolute minority bit is queried by C for $x \in_R \mathcal{H}_k^0$.

At any instant during the execution of a decision tree, we can partition the input variables into those that have been queried and those that have not. We call the set of pairs (x_i, a_i) of variables that have already been queried, along with their values, the *configuration* of the decision tree at that instant. The next action of the decision tree is either to stop (and produce an output that is not relevant to this analysis) or to choose a variable that has not yet been queried, and query it. In the latter case, the configuration after the query is determined by the value of the chosen variable.

A decision tree is determined by the actions it takes in the possible configurations. In a configuration γ, a decision tree that maximizes ρ_α takes an action that maximizes the linear combination in Eq. (7) *conditioned on reaching this configuration*. Namely it maximizes

$$\rho_\alpha(C, \gamma) \quad = \quad 2^{-k} P_q(C, \gamma) - \alpha P_m(C, \gamma) \ , \tag{8}$$

where $P_q(C, \gamma)$ is the expected number of sensitive bits queried by $C(x)$, when x is a uniformly random 0-hard x consistent with γ, while $P_m(C, \gamma)$ is the probability that $C(x)$ queries the absolute minority bit for a uniformly random 0-hard x consistent with γ. The optimal action in a configuration γ can therefore be found independently of the actions taken at configurations inconsistent with γ. (A similar statement for the maximization of α_C is false.)

Note that $\rho_\alpha(C, \gamma)$ is easy to compute if C stops at γ. If C queries a new variable at γ, then $\rho_\alpha(C, \gamma)$ is given by a convex combination of $\rho_\alpha(C, \gamma')$ and $\rho_\alpha(C, \gamma'')$, where γ' and γ'' are the two configurations resulting from the query. This leads to the following dynamic programming algorithm: consider all configurations in an order in which evaluating further variables yields configurations considered earlier. For each configuration γ we iterate through all actions to find an optimal one and store the value of $\rho_\alpha(C, \gamma)$ for the optimal C. We have $\rho_\alpha(C) = \rho_\alpha(C, \emptyset)$, where $\emptyset$ is the initial configuration (in which no variable has been queried).

The number of all possible configurations is 3^{3^k}. This makes the above algorithm infeasible even for $k = 3$. We reduce the number of configurations considered significantly by appealing to simple properties of 0-hard inputs:

1. We use the symmetries of the hard distribution. All the configurations γ in an orbit generated by the automorphisms of the ternary tree give rise to the same value for $\rho_\alpha(C, \gamma)$. We consider only one configuration in each such equivalence class.

2. We single out two types of configurations in which an optimal action is clear without the need for further computation. First, if the configuration uniquely determines the value of the root (i.e., is not consistent with a 1-input), an optimal strategy is to stop. Second, if an unqueried variable is known not to be the absolute minority variable, querying it does not decrease the objective function. We may thus assume that an optimal decision tree queries this variable.

   We implement the second type of action as follows. The nodes in the path to the absolute minority evaluate alternately to 0 and 1. If a node at odd depth evaluates to 0 or at even depth evaluates to 1, then it is not on the absolute minority path. In this case, all variables in the subtree rooted at the node are queried by an optimal decision tree. If a node at odd depth evaluates to 1, or at even depth evaluates to 0, then its *siblings* are not on the absolute minority path. In this case, the variables in the subtrees rooted at the siblings are queried by an optimal decision tree.

We call a configuration γ *unstable* if there is an action that an optimal decision tree may take in γ as described in point 2. We call the configuration *stable* otherwise. Note that the value of the root is not uniquely determined by a stable configuration. It suffices to store $\rho_\alpha(C, \gamma)$ for stable γ. If $\rho_\alpha(C, \gamma)$ is needed for some unstable configuration γ we apply the above rules (possibly multiple times) until the value of the root is determined, or we arrive at a stable configuration. We then compute $\rho_\alpha(C, \gamma)$ using the appropriate stored values.

The lone stable configuration for height 0 is $\emptyset$, the one in which the input variable has not been queried. Consider a stable configuration for height-k formulae, for $k \geq 1$, and the restrictions of

the configuration to the subtrees rooted at the three children of the root. Call the configuration obtained by negating the values of the variables in a configuration its *dual*. It is straightforward to verify that either a restriction uniquely determines the value of the corresponding child, or it is a dual of a stable configuration for height $(k-1)$. No child of the root can be known to have value 1 in a stable configuration and at most one of them can be known to have value 0. So an equivalence class of stable configurations for height k is determined by a multiset of (equivalence classes of) stable configurations for height $(k-1)$ of size 2 (when one child is known to have value 0) or of size 3 (when the values of the children are all undetermined). This characterization gives us the following recurrence relation for N_k, the number of equivalence classes of stable configurations for height k:

$$N_k = \binom{N_{k-1}+1}{2} + \binom{N_{k-1}+2}{3} ,$$

with the initial condition $N_0 = 1$. The recurrence gives us

$$\begin{aligned}
N_1 &= 2 \\
N_2 &= 7 \\
N_3 &= 112 \\
N_4 &= 246,792 \qquad \text{and} \\
N_5 &= 2,505,258,478,767,772 .
\end{aligned}$$

This makes the dynamic programming approach for optimizing $\rho_\alpha(C)$ feasible for $k \leq 4$. This approach is implemented by the Python program presented in Appendix A. In order to avoid dealing with dual configurations, the Python program considers NOT-3-MAJ, the recursive negated majority-of-three function. This is the function computed by the a negated majority gate in every internal node of a complete ternary tree.

Recall that using an algorithm to maximize $\rho_\alpha(C)$ we can check whether $\alpha_k > \alpha$. Instead of a binary search we find the exact value of α_k as follows. With little modification, the algorithm we present for maximizing $\rho_\alpha(C)$ also produces the value α_{C^*} for the optimal decision tree C^*. We then start with an arbitrary $\alpha \leq \alpha_k$, and repeatedly optimize ρ_α updating the estimate α to the last value α_{C^*}, until we find that $\max_C \rho_\alpha(C) = 0$. This heuristic finds the maximum α_k in a finite number of iterations. Instead of bounding the number of iterations in general we mention that starting from the initial value $\alpha = 0$ the heuristic gives us α_k in at most four iterations when $k = 2, 3, 4$. The computations show:

$$\begin{aligned}
\alpha_1 &= 2 , \\
\alpha_2 &= \frac{24}{7} , \\
\alpha_3 &= \frac{12231}{2203} , \quad \text{and} \\
\alpha_4 &= \frac{2027349}{216164} .
\end{aligned}$$

Using the value of α_4, Theorem 3.5 yields the following bound.

Corollary 3.6.

$$\mathrm{R}_\delta(\text{3-MAJ}_h) \geq (1/2 - \delta)\left(2 + \left(\frac{216164}{2027349}\right)^{1/4}\right)^h > (1/2 - \delta)2.57143^h .$$

3.3 Analysis of the 2-level encoding

As an illustration of the use of higher level encodings we explicitly derive a second order recurrence for $p_{h,\delta}$ using 2-level encodings. We fix $k = 2$ and consider deterministic decision trees C on 9 variables. We run these decision trees on inputs from $\mathcal{H}_2^0$.

From the proof of Theorem 3.1, we have $\alpha_C \leq 4$ for all C. We may verify that $\alpha_{C_0} = 3$ for the decision tree C_0 with the following strategy: first query x_1, if $x_1 = 1$, stop, else if $x_1 = 0$, query x_2 and x_3; then if $\mathsf{MAJ}(x_1, x_2, x_3) = 0$, stop, else query all remaining bits and stop. These bounds show that $3 \leq \alpha_2 \leq 4$, so it suffices to consider ρ_α for the values of α in the range $[3, 4]$. We prove below that for these values of α a single decision tree C' maximizes $\rho_\alpha(C)$ among the deterministic decision trees that query at least one variable. This decision tree C' is given in Figure 2. We state the optimality of C' in the following lemma.

Lemma 3.7. *Let C be any deterministic decision tree on 9-bit inputs that makes at least one query and let C' be the decision tree depicted in Figure 2. Then for all $\alpha \in [3, 4]$, $\rho_\alpha(C) \leq \rho_\alpha(C')$.*

Proof. Recall that the action in a configuration γ of the decision tree C that maximizes $\rho_\alpha(C)$ is the one that maximizes $\rho_\alpha(C, \gamma) = 2^{-2}P_q(C, \gamma) - \alpha P_m(C, \gamma)$. To simplify notation, we write ρ, P_q and P_m for $\rho_\alpha(C, \gamma), P_q(C, \gamma)$, and $P_m(C, \gamma)$, respectively, if the decision tree C and configuration γ considered are clear from the context.

We call any set of 3 sibling nodes a *clause*, so that $\{1, 2, 3\}$, $\{4, 5, 6\}$ and $\{7, 8, 9\}$ are clauses. We say a clause is *evaluated* if its majority is uniquely determined by the configuration under consideration.

We argue that an algorithm that maximizes $\rho_\alpha(C)$ takes certain actions, without loss of generality. We begin with three rules that are special cases of the general rules from Section 3.2 that we used to reduce the number of configurations considered.

1. If a bit is evaluated to 0, then evaluate all remaining bits in its clause.

2. If two bits in a clause are evaluated to 1 (this is the minority clause), evaluate all remaining bits in the other clauses.

3. if two clauses have been evaluated to 0, then stop.

In what follows we systematically consider all stable configurations for height 2 inputs, i.e., the ones in which the above three rules do not apply. For each such configuration, we determine what action(s) an optimal decision may take next, without loss of generality, in order to maximize ρ_α.

4. A single majority (0) clause is evaluated and either no variables are evaluated in either of the other clauses or a single 1 is evaluated in both the other clauses. In this case, stopping is the best strategy. Indeed, $m(x)$ has not been queried yet. Therefore if we stop, then $P_m = 0$ and $P_q = 2$. This gives $\rho = 1/2$. But if C continues by querying at least one more bit, then $P_m \geq 1/6$ or $P_m \geq 1/4$ (since there are either 6 or 4 remaining unqueried variables, respectively, and they are symmetric) and $P_q \leq 4$. Therefore, $\rho = 2^{-2}P_q - \alpha P_m \leq 1 - \alpha/6 \leq 1/2$ since $\alpha \geq 3$.

5. A single majority clause is evaluated and one more bit is evaluated to 1, but nothing more. There are 9 inputs consistent with this configuration. We argue that stopping is best, as in the previous case. The argument is more involved because there is no global symmetry

between the unqueried variables. We separately compare stopping with querying a variable inside or outside the untouched clause.

If we stop, then $P_q = 2$ and $P_m = 0$, so we have $\rho = 1/2$.

If we query a variable in the clause containing the evaluated bit 1, then there are 3 consistent inputs in which the next queried bit is $m(x)$. So we have $P_m \geq 1/3$ and $P_q \leq 4$. Since $\alpha \geq 3$, $\rho \leq 0$.

If we query a variable in the untouched clause, then there is 1 out of the 9 consistent inputs for which this next queried variable is $m(x)$, making $P_m \geq 1/9$. There are 4 more consistent inputs for which this variable evaluates to 1. In this case we arrive at the configuration covered by item 4 above. Using that rule, the algorithm stops, leaving 2 out the 4 sensitive bits unqueried. Thus, we have $P_q \leq 4 - 4/9 \cdot 2 = 28/9$ and $\rho \leq 4/9$, which is lesser than the $1/2$ obtained if we stop.

6. A single 1 has been evaluated in each of the three clauses and no other bit has been queried. In this case reading another bit is the best strategy (the choice of which bit is unimportant because of symmetry). Observe that no bit 0 has been evaluated yet. Therefore if C stops, we have $\rho = P_q = P_m = 0$. If C continues to query another bit, which bit it queries does not matter by symmetry. Then the rest of the algorithm is determined by the earlier rules yielding $P_q = 8/3$, $P_m = 1/6$, and $\rho = 2/3 - \alpha/6 \geq 0$.

7. A single bit 1 has been evaluated in each of two different clauses and the third clause is untouched. Then it is best to evaluate a bit of the third clause. If C stops we have $\rho = P_m = P_q = 0$.

   If C reads another bit in one of the clauses containing a single 1 bit, then the rest of the decision tree algorithm is determined by the earlier rules and we get $P_q = 14/5$ and $P_m = 1/5$ with $\rho = 7/10 - \alpha/5$. Note that whether this option is better than stopping depends on the value of α.

   If C reads a bit in the untouched clause, then by using the rules already presented in the previous cases, we calculate $P_q = 12/5$ and $P_m = 2/15$, yielding $\rho = 3/5 - 2\alpha/15$. This is more than both 0 and $7/10 - \alpha/5$ in the entire range of α we are considering.

8. A single bit 1 has been evaluated in one clause and no other clauses are touched. Then it is best to evaluate a bit of another clause. If C stops, then we have $\rho = P_m = P_q = 0$.

   If C evaluates another bit in the clause containing 1, then the rest of C is determined by earlier rules and we have $P_q = 3$, $P_m = 1/4$ and $\rho = 3/4 - \alpha/4 \leq 0$.

   But if C queries a bit in an untouched clause, then similar calculations yield $P_q = 7/3$, $P_m = 5/36$ and $\rho = 7/12 - 5\alpha/36 > 0$, making this the best choice.

Following all the above rules and always choosing the smallest index when symmetry allows us to choose, we arrive at a well defined decision tree, namely C'. This finishes the proof of the lemma. $\qquad \square$

The above lemma immediately gives us the value of α_2.
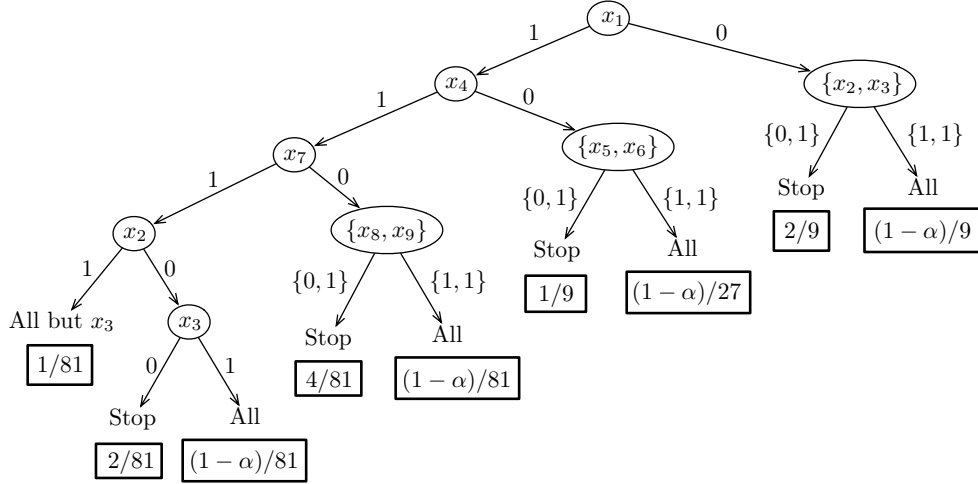
Theorem 3.8. $\alpha_2 = 24/7$.

Figure 2: A pictorial representation of the decision tree C'. "Stop" indicates that the algorithm stops and produces an output (that is not relevant to our analysis). "All" indicates that the algorithm queries all remaining variables. In the leftmost branch, all variables but x_3 are queried. The contribution to $\rho_\alpha(C')$ by each branch of C' is written under it in a box.

Proof. As observed in the paragraph before Lemma 3.7 we have $3 \le \alpha_2 \le 4$. By the lemma we know that $\rho_\alpha(C)$ in this range is maximized by either C' or the decision tree that does not query any variable. The latter gives $\rho_\alpha = 0$. We have $\alpha \ge \alpha_2$ if and only if this maximum is 0, so we are done if we calculate $\rho_\alpha(C')$. The contribution of each branch of the algorithm is given in Figure 2. Together these sum to $(48 - 14\alpha)/81$. This is positive for $\alpha < 24/7$, so we have $\alpha_2 = \alpha_{C'} = 24/7$. □

As a consequence of the value α_2, we get a slightly weaker lower bound than the one in Corollary 3.6. However, this bound has the advantage that we have an explicit proof for it.

Corollary 3.9. $\mathrm{R}_\delta(\text{3-MAJ}_h) \quad \ge \quad (1/2 - \delta)\left(2 + \sqrt{7/24}\right)^h \quad > \quad (1/2 - \delta)\,2.54006^h.$

4   Improved Depth-Two Algorithm

In this section, we present a new zero-error algorithm for computing 3-MAJ_h. For the key ideas behind it, we refer the reader to Section 1.

As before, we identify the formula 3-MAJ_h with a complete ternary tree of height h. We are given an assignment to the 3^h variables (equivalently, the leaves) of 3-MAJ_h, which may be accessed by querying the variables. In the description of the algorithm we adopt the following convention. Once the algorithm has determined the value b of the subformula rooted at a node v of the formula 3-MAJ_h, we also use v to denote this bit value b.

The algorithm is a combination of two depth-2 recursive algorithms. The first one, EVALUATE (see Algorithm 1), takes a node v of height $h(v)$, and evaluates the subformula rooted at v. The interesting case, when $h(v) > 1$, is depicted in Figure 3. The first step, permuting the input, means
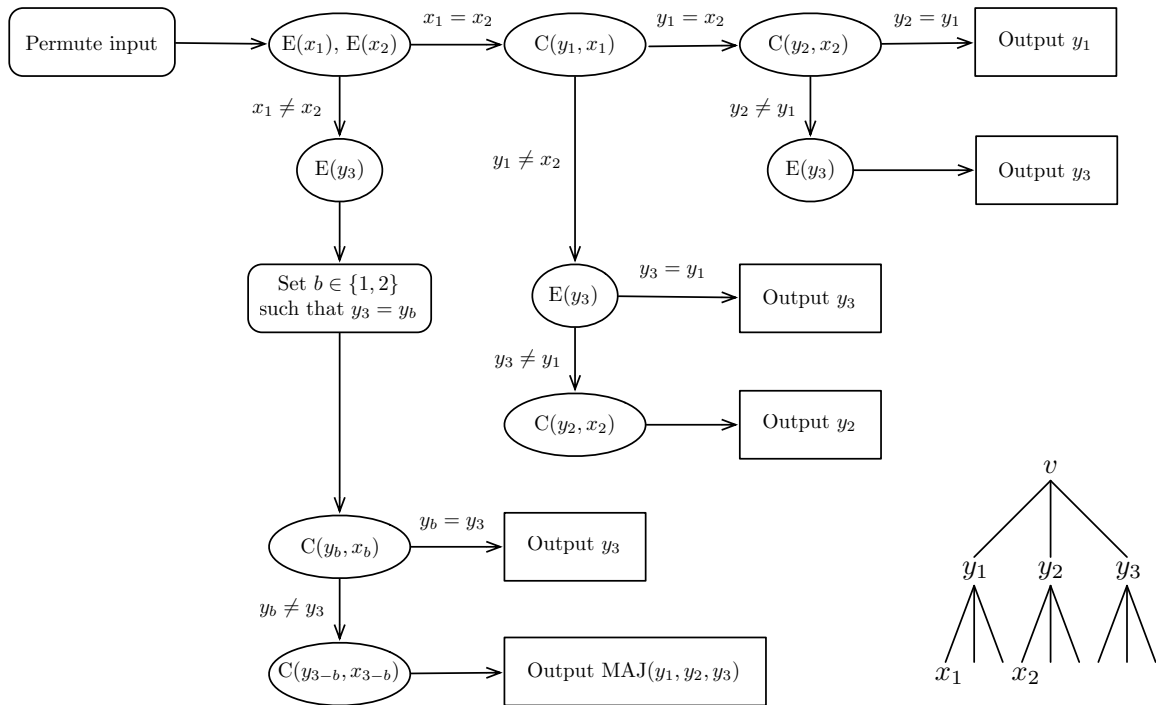
Figure 3: Pictorial representation of algorithm EVALUATE on a subformula of height $h(v) \geq 2$ rooted at v. It is abbreviated by the letter 'E' when called recursively on descendants of v. The letter 'C' abbreviates the second algorithm COMPLETE depicted in Figure 4.

applying a random permutation to the children y_1, y_2, y_3 of v and independent random permutations to each of the three sets of grandchildren.

The second algorithm, COMPLETE (see Algorithm 2), is depicted in Figure 4. It takes two arguments v, y_1, and completes the evaluation of the subformula 3-MAJ_h rooted at node v, where $h(v) \geq 1$, and y_1 is a child of v whose value has already been evaluated. The first step, permuting the input, means applying a random permutation to the children y_2, y_3 of v and independent random permutations to each of the two sets of grandchildren of y_2, y_3. Note that this is similar in form to the depth 2 algorithm due to [JKS03].

To evaluate an input of height h, we invoke EVALUATE(r), where r is the root. The correctness of the two algorithms follows by inspection—they determine the values of as many children of the node v as is required to compute the value of v.

For the complexity analysis, we study the expected number of queries they make for a worst-case input of fixed height h. (*A priori*, we do not know if such an input is a hard input as defined in Section 2.2.) Let $T(h)$ be the worst-case complexity of EVALUATE(v) for v of height h. For COMPLETE(v, y_1), we distinguish between two cases. Let y_1 be the child of node v that has already been evaluated. The complexity given that y_1 is the minority child of v is denoted by S^{m}, and the complexity given that it is a majority child is denoted by S^{M}.

The heart of the analysis is the following set of recurrences that relate T, S^{M} and S^{m} to each other.

17

Algorithm 1 EVALUATE(v): evaluate a node v.

Input: Node v with subtree of height $h(v)$.
Output: the bit value 3-MAJ$_h(Z(v))$ of the subformula rooted at v

Let $h = h(v)$

if $h = 0$ **then**   ▷ First base case: $h = 0$ (v is a leaf)
    Query $Z(v)$ to get its value a; **return** a
end if

Let y_1, y_2, y_3 be a uniformly random permutation of the children of v ▷ v has height $h \geq 1$

if $h = 1$ **then**   ▷ Second base case: $h = 1$
    EVALUATE(y_1) and EVALUATE(y_2)
    if $y_1 = y_2$ **then return** y_1
    else return EVALUATE(y_3)
    end if
end if

                                    ▷ Recursive case: v has height $h \geq 2$; use the attached figure as a guide
Let x_1 and x_2 be chosen uniformly at random from the children of y_1 and y_2, respectively

EVALUATE(x_1) and EVALUATE(x_2)

if $x_1 \neq x_2$ **then**
    EVALUATE(y_3)
    Let $b \in \{1, 2\}$ be such that $x_b = y_3$
    COMPLETE(y_b, x_b)
    if $y_b = y_3$ **then return** y_b
    else return COMPLETE(y_{3-b}, x_{3-b})
    end if
else [$x_1 = x_2$]
    COMPLETE(y_1, x_1)
    if $y_1 = x_1$ **then**
        COMPLETE(y_2, x_2)
        if $y_2 = y_1$ **then return** y_1
        else [$y_2 \neq y_1$] **return** EVALUATE(y_3)
        end if
    else [$y_1 \neq x_1$]
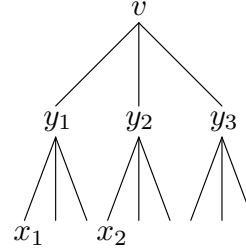   EVALUATE(y_3)
        if $y_3 = y_1$ **then return** y_1
        else return COMPLETE(y_2, x_2)
        end if
    end if
end if

Algorithm 2 COMPLETE(v, y_1): finish the evaluation of the subformula rooted at node v
---
Input: Node v of height $h(v)$; child y_1 of v which has already been evaluated
Output: the bit value 3-MAJ$_h(Z(v))$

Let $h = h(v)$

Let y_2, y_3 be a uniformly random permutation of the two children of v other than y_1

if $h = 1$ **then**   ▷ Base case
    EVALUATE(y_2)
    if $y_2 = y_1$ **then return** y_1
    else return EVALUATE(y_3)
    end if
end if

Let x_2 be chosen uniformly at random from the children of y_2 ▷ Recursive case
                                                      ▷ use the attached figure as a guide

EVALUATE(x_2)

if $y_1 \neq x_2$ **then**
    EVALUATE(y_3)
    if $y_1 = y_3$ **then return** y_1
    else return COMPLETE(y_2, x_2)
    end if
else $[y_1 = x_2]$
    COMPLETE(y_2, x_2)
    if $y_1 = y_2$ **then return** y_1
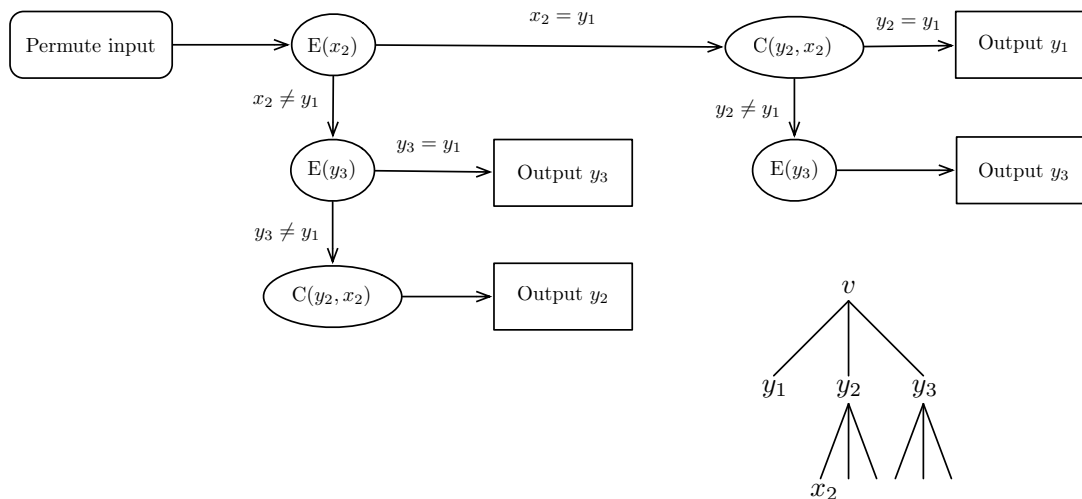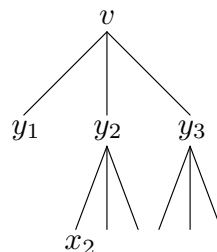   **else return** EVALUATE(y_3)
    end if
end if

Figure 4: Pictorial representation of algorithm COMPLETE on a subformula of height $h \geq 1$ rooted at v one child y_1 of which has already been evaluated. It is abbreviated by the letter 'C' when called recursively on descendants of v. Calls to EVALUATE are denoted 'E'.

Lemma 4.1. *We have $S^m(1) = 2$, $S^M(1) = \frac{3}{2}$, $T(0) = 1$, and $T(1) = \frac{8}{3}$.*
For all $h \geq 1$, it holds that

$$S^M(h) \leq S^m(h) \qquad and \qquad S^M(h) \leq T(h) \ . \tag{9}$$

Finally, for all $h \geq 2$, it holds that

$$S^m(h) \ = \ T(h-2) + T(h-1) + \frac{2}{3} S^M(h-1) + \frac{1}{3} S^m(h-1) \ , \tag{10}$$

$$S^M(h) \ = \ T(h-2) + \frac{2}{3} T(h-1) + \frac{1}{3} S^M(h-1) + \frac{1}{3} S^m(h-1) \ , \quad and \tag{11}$$

$$T(h) \ = \ 2\,T(h-2) + \frac{23}{27} T(h-1) + \frac{26}{27} S^M(h-1) + \frac{18}{27} S^m(h-1) \ . \tag{12}$$

Proof. We prove these relations by induction. The bounds for $h \in \{0, 1\}$ follow immediately by inspection of the algorithms. To prove the statement for $h \geq 2$, we assume the recurrences hold for all $l < h$. Observe that it suffices to prove Equations (10), (11), (12) for height h, since the values of the coefficients immediately imply that Inequalities (9) holds for h as well.

Equation (10). Since COMPLETE(v, y_1) always starts by computing the value of a grandchild x_2 of v, we get the first term $T(h-2)$ in Eq. (10). It remains to show that the worst-case complexity of the remaining queries is $T(h-1) + (2/3)S^M(h-1) + (1/3)S^m(h-1)$.

Since y_1 is the minority child of v, we have that $y_1 \neq y_2 = y_3$. The complexity of the remaining steps is summarized in the next table in the case that the three children of node y_2 are not all equal. In each line of the table, the worst case complexity is computed given the event in the first cell of the line. The second cell in the line is the probability of the event in the first cell over the random permutation of the children of y_2. This gives a contribution of $T(h-1) + (2/3)S^M(h-1) + (1/3)S^m(h-1)$.

$S^m(h)$ (we have $y_1 \neq y_2 = y_3$)		
event	probability	complexity
$y_2 = x_2$	2/3	$T(h-1) + S^M(h-1)$
$y_2 \neq x_2$	1/3	$T(h-1) + S^m(h-1)$

This table corresponds to the worst case, as the only other case is when all children of y_2 are equal, in which the cost is $T(h-1) + S^M(h-1)$. Applying Inequality (9) for $h-1$, this is a smaller contribution than the case where the children are not all equal.

Therefore the worst case complexity for S^m is given by Eq. (10). We follow the same convention and appeal to this kind of argument also while deriving the other two recurrence relations.

Equation (11). Since COMPLETE(v, y_1) always starts by computing the value of a grandchild x_2 of v, we get the first term $T(h-2)$ in Eq. (11). There are then two possible patterns, depending on whether the three children y_1, y_2, y_3 of v are all equal. If $y_1 = y_2 = y_3$, we have in the case that all children of y_2 are not equal that:

$S^M(h)$ if $y_1 = y_2 = y_3$		
event	probability	complexity
$y_2 = x_2$	2/3	$S^M(h-1)$
$y_2 \neq x_2$	1/3	$T(h-1)$

20

As in the above analysis of Eq. (10), applying Inequalities (9) for height $h - 1$ we get that the complexity in the case when all children of y_2 are equal is bounded above by the complexity when the children are not all equal. Therefore the above table describes the worst-case complexity for the case when $y_1 = y_2 = y_3$.

If y_1, y_2, y_3 are not all equal, we have two events $y_1 = y_2 \neq y_3$ or $y_1 = y_3 \neq y_2$ of equal probability as y_1 is a majority child of v. This leads to the following tables for the case where the children of y_2 are not all equal

$S^{\mathrm{M}}(h)$ given $y_1 = y_2 \neq y_3$		
event	prob.	complexity
$y_2 = x_2$	2/3	$S^{\mathrm{M}}(h-1)$
$y_2 \neq x_2$	1/3	$T(h-1) + S^{\mathrm{m}}(h-1)$

$S^{\mathrm{M}}(h)$ given $y_1 = y_3 \neq y_2$		
event	prob.	complexity
$y_2 = x_2$	2/3	$T(h-1)$
$y_2 \neq x_2$	1/3	$T(h-1) + S^{\mathrm{m}}(h-1)$

As before, we apply Inequalities (9) for height $h - 1$ to see that the worst case occurs when the children of y_2 are not all equal.

From the above tables, we deduce that the worst-case complexity occurs on inputs where y_1, y_2, y_3 are not all equal. This is because Inequalities (9) for height $h-1$ imply that, line by line, the complexities in the table for the case $y_1 = y_2 = y_3$ are upper bounded by the corresponding entries in each of the latter two tables. To conclude Eq. (11), recall that the two events $y_1 = y_2 \neq y_3$ and $y_1 = y_3 \neq y_2$ occur with probability $1/2$ each:

$$
\begin{aligned}
S^{\mathrm{M}}(h) &= T(h-2) + \frac{1}{2}\left[\frac{2}{3}S^{\mathrm{M}}(h-1) + \frac{1}{3}\left(T(h-1) + S^{\mathrm{m}}(h-1)\right)\right] \\
&\quad + \frac{1}{2}\left[\frac{2}{3}T(h-1) + \frac{1}{3}\left(T(h-1) + S^{\mathrm{m}}(h-1)\right)\right] .
\end{aligned}
$$

Equation (12). Since EVALUATE(v) starts with two calls to itself to compute x_1, x_2, we get the first term $2\,T(h-2)$ on the right hand side. For the remaining terms, we consider two possible cases, depending on whether the three children y_1, y_2, y_3 of v are equal. If $y_1 = y_2 = y_3$, assuming that the children of y_1 are not all equal, and the same for the children of y_2, we have

$T(h)$ given $y_1 = y_2 = y_3$		
event	probability	complexity
$y_1 = x_1, y_2 = x_2$	4/9	$2S^{\mathrm{M}}(h-1)$
$y_1 = x_1, y_2 \neq x_2$	2/9	$T(h-1) + S^{\mathrm{M}}(h-1)$
$y_1 \neq x_1, y_2 = x_2$	2/9	$T(h-1) + S^{\mathrm{M}}(h-1)$
$y_1 \neq x_1, y_2 \neq x_2$	1/9	$T(h-1) + S^{\mathrm{m}}(h-1)$

As before, the complexities are in non-decreasing order, and we observe that Inequalities (9) for height $h-1$ imply that in a worst case input the children of y_1 are not all equal, and that the same holds for the children of y_2.

If y_1, y_2, y_3 are not all equal, we have three events $y_1 = y_2 \neq y_3$, $y_1 \neq y_2 = y_3$ and $y_3 = y_1 \neq y_2$

each of which occurs with probability $1/3$. This leads to the following analyses

$T(h)$ given $y_1 = y_2 \neq y_3$		
event	probability	complexity
$y_1 = x_1, y_2 = x_2$	4/9	$2S^{\mathrm{M}}(h-1)$
$y_1 = x_1, y_2 \neq x_2$	2/9	$T(h-1) + S^{\mathrm{M}}(h-1) + S^{\mathrm{m}}(h-1)$
$y_1 \neq x_1, y_2 = x_2$	2/9	$T(h-1) + S^{\mathrm{M}}(h-1) + S^{\mathrm{m}}(h-1)$
$y_1 \neq x_1, y_2 \neq x_2$	1/9	$T(h-1) + 2S^{\mathrm{m}}(h-1)$

$T(h)$ given $y_1 \neq y_2 = y_3$		
event	probability	complexity
$y_1 = x_1, y_2 = x_2$	4/9	$T(h-1) + S^{\mathrm{M}}(h-1)$
$y_1 = x_1, y_2 \neq x_2$	2/9	$T(h-1) + S^{\mathrm{M}}(h-1) + S^{\mathrm{m}}(h-1)$
$y_1 \neq x_1, y_2 = x_2$	2/9	$T(h-1) + S^{\mathrm{M}}(h-1) + S^{\mathrm{m}}(h-1)$
$y_1 \neq x_1, y_2 \neq x_2$	1/9	$T(h-1) + 2S^{\mathrm{m}}(h-1)$

$T(h)$ given $y_3 = y_1 \neq y_2$		
event	probability	complexity
$y_1 = x_1, y_2 = x_2$	4/9	$T(h-1) + S^{\mathrm{M}}(h-1)$
$y_1 = x_1, y_2 \neq x_2$	2/9	$T(h-1) + S^{\mathrm{M}}(h-1) + S^{\mathrm{m}}(h-1)$
$y_1 \neq x_1, y_2 = x_2$	2/9	$T(h-1) + S^{\mathrm{m}}(h-1)$
$y_1 \neq x_1, y_2 \neq x_2$	1/9	$T(h-1) + 2S^{\mathrm{m}}(h-1)$

In all three events, we observe that Inequalities (9) for height $h-1$ imply that in a worst case input, the children of y_1 are not all equal, and the same holds for the children of y_2.

Applying Inequalities (9) for height $h-1$, it follows that line by line the complexities in the last three tables are at least the complexities in the table for the case $y_1 = y_2 = y_3$. Therefore the worst case also corresponds to an input in which y_1, y_2, y_3 are not all equal. We conclude Eq. (12) as before, by taking the expectation of the complexities in the last three tables.

Theorem 4.2. $T(h), S^{\mathrm{M}}(h)$, and $S^{\mathrm{m}}(h)$ are all in $\mathrm{O}(\alpha^h)$, where $\alpha \leq 2.64944$.

Proof. We make an ansatz $T(h) \leq a\,\alpha^h$, $S^{\mathrm{M}}(h) \leq b\,\alpha^h$, and $S^{\mathrm{m}}(h) \leq c\,\alpha^h$, and find constants a, b, c, α for which we may prove these inequalities by induction.

The base cases tell us that $2 \leq c\alpha$, $\frac{3}{2} \leq b\alpha$, $1 \leq a$, and $\frac{8}{3} \leq a\alpha$.

Assuming we have constants that satisfy these conditions, and that the inequalities hold for all appropriate $l < h$, for some $h \geq 2$, we derive sufficient conditions for the inductive step to go through.

By the induction hypothesis, Lemma 4.1, and the ansatz, we have

$$S^{\mathrm{m}}(h) \leq a\,\alpha^{h-2} + a\,\alpha^{h-1} + \frac{2b}{3}\,\alpha^{h-1} + \frac{c}{3}\,\alpha^{h-1} \ ,$$

$$S^{\mathrm{M}}(h) \leq a\,\alpha^{h-2} + \frac{2a}{3}\,\alpha^{h-1} + \frac{b}{3}\,\alpha^{h-1} + \frac{c}{3}\,\alpha^{h-1} \ , \quad \text{and}$$

$$T(h) \leq 2a\,\alpha^{h-2} + \frac{23a}{27}\,\alpha^{h-1} + \frac{26}{27}\,\alpha^{h-1} + \frac{18}{27}\,\alpha^{h-1} \ .$$

These would imply the required bounds on $S^{\mathrm{m}}(h), S^{\mathrm{M}}(h), T(h)$ if

$$a + \tfrac{3a+2b+c}{3}\alpha \leq c\,\alpha^2 \ , \qquad a + \tfrac{2a+b+c}{3}\alpha \leq b\,\alpha^2 \ , \quad \text{and} \qquad 2a + \tfrac{23a+26b+18c}{27}\alpha \leq a\,\alpha^2 \ . \tag{13}$$

The choice $\alpha = 2.64944$, $a = 1.02$, $b = 0.559576 \times a$, and $c = 0.755791 \times a$ satisfies the base case as well as all the Inequalities (13), so the theorem holds by induction. □

5   Concluding remarks

In this article, we revisited a technique due to Jayram, Kumar, and Sivakumar for proving a lower bound on the decision tree complexity of 3-MAJ$_h$, the recursive majority-of-three function of height h. We showed that it could be enhanced by obtaining better estimates on the probability $p_{h,\delta}$ with which the absolute minority variable is queried under the hard distribution. The new estimates are obtained by considering highly symmetric encodings of height-$(h-k)$ inputs into height-h inputs. The analysis of the encodings quickly becomes intractable with growing k. However, by appealing to the underlying symmetry in the function, the analysis can be executed explicitly for $k = 1, 2$, and with the aid of a computer for $k \leq 4$. This leaves us with several immediate questions about the technique:

1. Is there is a more efficient algorithm for the analysis?

2. Is there a succinct, explicit analysis for larger values of k?

3. What is the best lower bound we can obtain using this technique?

4. Does this technique give us any intuition into more efficient algorithms?

We also present a new (more efficient) algorithm for 3-MAJ$_h$ based on the idea that a partial evaluation of a formula helps us form an opinion about the value of its subformulae. We use the opinions at a certain stage of the algorithm to choose the next variable to query. Additionally, we use a depth-2 recursive algorithm that is optimized for computing the value of a partially evaluated formula. It is likely that algorithms with depth k recursion, with $k > 2$ give us further improvements in efficiency. However, their analysis seems to be beyond the scope of the techniques used in this work.

References

[BI87]   M. Blum and R. Impagliazzo. General oracle and oracle classes. In *Proceedings of 28th IEEE Symposium on Foundations of Computer Science*, pages 118–126, 1987.

[HH87]   J. Hartmanis and L. Hemachandra. One-way functions, robustness, and non-isomorphism of NP-complete sets. In *Proceedings of 2nd Structure in Complexity Theory Conference*, pages 160–173, 1987.

[HNW90] R. Heiman, I. Newman, and A. Wigderson. On read-once threshold formulae and their randomized decision tree complexity. In *Proceedings of 5th Structure in Complexity Theory*, pages 78–87, 1990.

[HW91]   R. Heiman and A. Wigderson. Randomized versus deterministic decision tree complexity for read-once boolean functions. In *Proceedings of 6th Structure in Complexity Theory Conference*, pages 172–179, 1991.

[JKS03]   T. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complex-
            ity. In *Proceedings of 35th ACM Symposium on Theory of Computing*, pages 673–682,
            2003.

[Leo13]   Nikos Leonardos. An improved lower bound for the randomized decision tree complexity
            of recursive majority. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska,
            and David Peleg, editors, *Proceedings of 40th International Colloquium on Automata,
            Languages and Programming*, volume 7965 of *Lecture Notes in Computer Science*, pages
            696–708. Springer, 2013.

[LNPV06]   I. Landau, A. Nachmias, Y. Peres, and S. Vanniasegaram. The lower bound for
            evaluating a recursive ternary majority function: an entropy-free proof. Techni-
            cal report, Department of Statistics, University of California, Berkeley, CA, USA,
            `http://www.stat.berkeley.edu/110`, 2006. Undergraduate Research Report.

[MNSX11]   F. Magniez, A. Nayak, M. Santha, and D. Xiao. Improved bounds for the randomized
            decision tree complexity of recursive majority. In *Proceedings of 38th International
            Colloquium on Automata, Languages and Programming*, pages 317–329, 2011.

[Nis89]   N. Nisan. CREW PRAMs and decision trees. In *Proceedings of 21st Annual ACM
            Symposium on Theory of Computing*, pages 327–335, 1989.

[RS08]   B. Reichardt and Spalek. Span-program-based quantum algorithm for evaluating formu-
            las. In *Proceedings of 40th ACM Symposium on Theory of Computing*, pages 103–112,
            2008.

[San95]   M. Santha. On the Monte Carlo boolean decision tree complexity of read-once formulae.
            Random Structures and Algorithms, 6(1):75–87, 1995.

[Sni95]   M. Snir. Lower bounds for probabilistic linear decision trees. *Theoretical Computer
            Science*, 38:69–82, 1995.

[SW86]   M. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of
            evaluating game trees. In *Proceedings of 27th Annual Symposium on Foundations of
            Computer Science*, pages 29–38, 1986.

[Tar90]   G. Tardos. Query complexity or why is it difficult to separate $\mathbf{NP}^A \cap \mathbf{coNP}^A$ from $\mathbf{P}^A$
            by a random oracle. *Combinatorica*, 9:385–392, 1990.

A   Python program

The following Python program is also available at:

   https://www.dropbox.com/s/wcrdoib5h918p2e/commented-majority.py

```
def prep(dep):
  global depth,g,d
  # depth = depth of recursive NOT-3-MAJ circuits considered
  # g[i] = list of depth i stable configurations for 0 <= i <= depth
  # for a record r representing a configuration of depth i
  # r[0] = # of evaluations giving 0
```

```
  # r[1] = # of evaluations giving 1
  # r[2] = # of queried sensitive bits in all evaluations with correct
      output
  # r[3:6] = indices of the three children, sorted (where applicable)
  # d[i] = dictionary that tells the index of a stable configuration of
      depth i from the indices of its 3 depth i-1 children
  # prep sets the values of these global variables (will not be changed)
  depth=dep
  g=[[[0,1,depth%2],[1,1,0]]]
  # two stable configurations in g[0]: a variable set to 1 and a not
      queried variable
  d=[{}]
  # empty dictionary in d[0]
  for i in range(1,depth+1):
    # building g[i] and d[i] under the names gg and dd
    gl=g[-1]
    gg=[[0,1,((depth-i)%2)*2**i]]
    # the first record represents a fully queried subtree of depth i
        evaluating to 1
    dd={}
    for a in range(len(gl)):
      for b in range(max(a,1),len(gl)):
        for c in range(b,len(gl)):
          # enforcing a <= b <= c and no two fully evaluated siblings in a
              stable configuration
          dd[(a,b,c)]=len(gg)
          A,B,C=gl[a],gl[b],gl[c]
          gg.append([A[0]*B[1]*C[1]+A[1]*B[0]*C[1]+A[1]*B[1]*C[0],A[1]*B
              [0]*C[0]+A[0]*B[1]*C[0]+A[0]*B[0]*C[1],A[0]*B[1]*C[2]+A[0]*B
              [2]*C[1]+A[1]*B[0]*C[2]+A[1]*B[2]*C[0]+A[2]*B[0]*C[1]+A[2]*B
              [1]*C[0],a,b,c])
          # inserting the current record in gg and dd - formula is long
              but simple
    g.append(gg)
    d.append(dd)
    # inserting the final gg and dd as g[i] and d[i]

def check(a0,a1):
  global alpha,ot,opt
  # alpha is considered as the rational a0/a1 but the integers are stored
  # ot is a table dynamically built for the strategy C maximizing
      rho_alpha(C)
  # ot[a][0] = # of inputs in which the optimal strategy C applied after
      configuration g[depth][a] queries abs. minority
  # ot[a][1] = # total # of sensitive bits revealed for C applied after g[
      depth][a]
  alpha=[a0,a1]
  ot=[[0,0]]
  # g[depth][0] is impossible
  for a in range(1,len(g[-1])):
    opt=[0,g[-1][a][2]]
```

```python
    # opt is the best current guess for ot[a]
    # here it is set to value corresponding to stopping at the
        configuration g[depth][a]
    # opt[0] = 0 as the absolute minority is not queried in any stable
        configuration
    adj(depth,a,[],0)
    # here adj is applied to the root of the current configuration g[depth
        ][a]
    # it crawls through the entire tree and updates opt if querying a
        variable is better than the current optimum
    ot.append(opt)
    # opt is now the correct optimum - it is inserted in the list
  return ot[-1]
  # here max_C rho_alpha(C) = ot[-1][1]/(2**depth*N)-alpha*ot[-1]/N, where
      N is the total # of 0-hard inputs
  # also: alpha(C) = ot[-1][1]/(ot[-1][0]*2**depth) (unless ot[-1]=[0,0]
    and C queries no input bit)


def adj(i,a,s,t):
  global ot,opt
  # here we consider what happens if a vertex W in a stable configuration
      is evaluated to 0 or 1
  # g[i][a] = the configuration below W
  # s is a list of depth-i pairs of siblings to add to g[i][a] to arrive
      to the stable depth d configuration considered
  # first goal compute tt as follows
  # tt[0] # of inputs in which the optimal strategy C applied after W is
      set to 0 (instable) queries abs. minority
  # tt[1] total # of sensitive bits revealed in same situation
  # tt[2] # of consistent inputs with W on the root to absolute minority
      path
  # t = (same as tt but for the parent of W)
  if s==[]:
    tt=[0,2**i,1,0,0]
    # no input considered gives 0 at the root
  else:
    s1,s2=g[i][s[0][0]],g[i][s[0][1]]
    # the siblings of W
    nn,ne=s1[1]*s2[1],s1[0]*s2[1]+s2[0]*s1[1]
    tt=[nn*t[0]+ne*t[3],nn*t[1]+ne*t[4],nn*t[2]]
    # based on the rule to evaluate the siblings of W if W evaluates to 0
    if s[0][0]==0:
      tt.extend([s2[0]*t[0],s2[0]*t[1]])
      # based on the fact that if W evaluates to 1 and a so does a sibling
        , then the parent evaluates to 0
    else:
      # we have a stable configurationa and use d to find its index in g[
          depth]
      w,j=0,i
      for si in s:
        j=j+1
```

```
          if w<=si[0]:
            triple=(w,si[0],si[1])
          elif w<=si[1]:
            triple=(si[0],w,si[1])
          else:
            triple=(si[0],si[1],w)
          # here we sorted the three siblings w, si[0] and si[1]
          w=d[j][triple]
        # by now w is index of the full stable configuration and ot[w]
          contains the numbers we seek
        tt.extend(ot[w])
  if i>0:
    # if W is not a variable we recursively call adj on its non-evaluated
      children
    A=g[i][a]
    if A[3]>0:
      adj(i-1,A[3],[A[4:6]]+s,tt)
    adj(i-1,A[4],[[A[3],A[5]]]+s,tt)
    adj(i-1,A[5],[A[3:5]]+s,tt)
  else:
    # if W is a variable we compute the effects of querying it, compare to
      opt and update opt if needed
    pair=[tt[0]+tt[2]+tt[3],tt[1]+tt[4]]
    if (pair[0]-opt[0])*alpha[0]*2**depth<(pair[1]-opt[1])*alpha[1]:
      opt=pair

import fractions
def alpha(dep):
  # finds minimal alpha with rho_alpha(C) <= 0 for all C = max alpha(C)
    for non-empty C
  # by recursively applying check(alpha) that either gives a higher alpha
    or confirms that alpha is optimal
  prep(dep)
  p=0
  q=1
  x=check(p,q)
  while (x != [0,0]):
    f=fractions.Fraction(x[1],x[0]*(2**dep))
    p=f.numerator
    q=f.denominator
    x=check(p,q)
  print("For depth ",dep,", optimal alpha is ",f)
  print("leading to a lower bound of (",2+(p/q)**(-1/dep),")^h")
```