# On Boolean Decision Trees with Faulty Nodes

Claire Kenyon* Valerie King†

March 19, 1998

**Abstract**

We consider the problem of computing with faulty components in the context of the Boolean decision tree model, in which cost is measured by the number of input bits queried and the responses to queries are faulty with a fixed probability. We show that if $f$ can be represented in $k$-DNF form and in $j$-CNF form, then $O(n \log(\min(k, j)/q))$ queries suffice to compute $f$ with error probability less than $q$, where $n$ is the number of input bits.

## 1 Introduction

In this paper, we describe a method for performing reliable computation despite the presence of faulty components. This problem has been well studied in various contexts. The model we consider here is the noisy Boolean decision tree.

In a Boolean decision tree, the value of a function on $n$ bits is computed as follows: Each step consists of a query of an input bit, where the choice of the query may depend on the outcome of the previous queries. The cost of the algorithm is the number of queries required for the worst case input. The algorithm may be represented as a tree in which each node is a query,

*Current address: LIP ENS-Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France. Part of this work was done while the author was visiting NECI, Princeton, and at LIENS in Paris.

†Current address: Computer Science Department, University of Victoria, BC, Canada V8W 3P2. This work was done while the author was at NECI, Princeton.

and the computation ends when a leaf labelled with an output is reached. The cost is the depth of the tree.

In the noisy version, each response to a query is correct with probability $p$, where $p$ is a fixed constant greater than $1/2$. This induces a probability distribution on the computation paths followed for an input. We specify a tolerance parameter $q$, $0 < q < 1/2$ and require that for every input, the probability that the correct answer is reached is at least $1 - q$. The cost of the algorithm is, as before, the depth of the tree.

It is not difficult to see that if the noiseless Boolean decision tree depth is $d$, one can construct a noisy decison tree by repeating each query $\mathrm{O}(\log(d/q))$ times and using the majority of the responses to determine the next step.

In 1990, Feige, Peleg, Raghavan and Upfal [?] showed that this logarithmic blowup was necessary for some problems while unnecessary for others. Thus, they introduced the question of how one might characterize the noisy Boolean decision complexity of a function. Their paper gives tight bounds on the complexity of computing symmetric functions. In 1991, Reischuk and Schmeltz [?] proved that almost all Boolean functions require $\Omega(n \log n)$ queries.

In this paper we use a very different approach to design an algorithm whose cost matches the bounds of [?] for symmetric functions and generalizes that result as follows:

**Theorem 1** *Let $f$ be a Boolean function on $n$ bits such that $f$ can be represented in CNF form by clauses of size no greater than $N_0$ and in DNF form by clauses of size no greater than $N_1$. Let $p$ be the error probability. Then there is a Boolean decision tree with tolerance $q$ of depth $O(n \log(min\{N_0, N_1\}/q)/(2p-1)^3)$.*

For example, our algorithm shows that determining whether a graph contains a triangle can be decided with $O(n \log(1/q))$ noisy queries where $n$ is the total number of possible edges and $p$ is a constant, since $N_1 = 3$. This resolves an open question suggested by Feige, and is optimal to within a constant factor, when $q$ is a constant, since $n$ queries are required in the noiseless case.

We observe that an upper bound of $O(n \log(\max\{N_0, N_1\}/q))$ follows immediately from the trivial $O(d \log(d/q))$ algorithm mentioned above and Blum's observation (see [?]) that $d \leq N_0 N_1$ for any function $f$, where $d$ is the noiseless Boolean decision tree depth of $f$ and $N_0$ and $N_1$ are any numbers that satisfy the conditions of our theorem.

Other variants of the Boolean decision tree model, with error, have been studied. Our approach was inspired by a 1989 work [?] by Kenyon and A. Yao on the complexity of computing on Boolean decison trees where no more than a fixed number of queries were answered incorrectly. Feige, Peleg, Raghavan, and Upfal [?] and Reischuk and Schmeltz [?] also look at lower bounds for probabilistic noisy Boolean decision trees, in which coin-tossing is permitted in the algorithms. They also consider a static model in which the sequence of queries may not depend on the answers given. Feige, Peleg, Raghavan and Upfal also consider faulty computation in comparison decision trees and in parallel models of computation [?], [?]. Related results by Pippenger and others concern the amount of redundancy required for circuits whose gates are "noisy", see [?], [?], [?].

# 2 Definitions and the Algorithm

## 2.1 Definitions

Following the standard definitions, we say that a pair $M = (T, F)$, where $T$ and $F$ are disjoint subsets of $\{x_1, ..., x_n\}$, is called a *minterm* of $f$ (resp. a *maxterm* of $f$) if the following holds:
**1•** if $x$ is such that for all $x_j$ in $T$, $x_j = 1$, and for all $x_j$ in $F$, $x_j = 0$, then $f(x) = 1$ (resp. $f(x) = 0$) regardless of the values of the other coordinates of $x$.
**2•** no other pair $(T', F')$ with $T' \subseteq T$ and $F' \subseteq F$ satisfies condition 1.

If $f$ can be represented in CNF and DNF form with clauses of size no greater than $N_0$ and $N_1$, respectively, then there is a set of minimal clauses, each of size no greater than by $N_0$, which are maxterms and a set of minimal clauses, each of size no greater than $N_1$ which are minterms. Moreover, every input in $f^{-1}(0)$ is consistent with at least one of these maxterms, and every input in $f^{-1}(1)$ is consistent with at least one of these minterms.

A function may have large minterms for example, yet have a DNF representation that does not require large clauses (minterms). For example, the storage access function is defined to be 1 iff in a string of $n = m + \log m$ bits, there is a 1 in the $i^{th}$ position, where $i$ is the value of the last $\log m$ bits. This function can be represented by an $O(\log n)$-DNF form even though $f$ has a minterm of size $\Theta(n)$. We note that the minimal $K$ such that $f$ can be represented in DNF form *and* CNF form with clauses of size no greater than $K$ is called the *non-deterministic* or *certificate complexity* of $f$ (see [?]).

*Note:* ¿From this point on in the paper, we fix a representation of $f$ in $N_1$-DNF form and another in $N_0$-CNF form, and use the terms "minterm" and "maxterm" to refer to the minterms and maxterms given by those representations.

## 2.2 Notations

Let $f$ be a Boolean function on $\vec{x} = (x_1, x_2, \ldots, x_n)$. At each step, the algorithm must choose a variable to query. The decision is based on the current "label" of the variables, where the *label* $v(x_j)$ *of variable* $x_j$ is the difference between the number of queries on $x_j$ which were answered by a 1, and the number of queries on $x_j$ which were answered by a 0, from the beginning of the algorithm until the current step:

$$v(x_j) = \#\text{1-answers on } x_j - \#\text{0-answers on } x_j \ .$$

Thus, the larger $v(x_j)$ is, the more confident we feel that $x_j = 1$.

To satisfy a minterm or maxterm $M = (T, F)$, we want every $x_j$ in $T$ to be 1 and every $x_j$ in $F$ to be 0. Thus we define the label of $x_j$ within minterm or maxterm $M$:

$$v_M(x_j) = \begin{cases} v(x_j) & \text{if } x_j \in T \\ -v(x_j) & \text{if } x_j \in F \\ \text{is undefined} & \text{otherwise} \end{cases} \ .$$

Then the label of a minterm or maxterm $M$ is defined as follows:

$$v(M) = \min_{x_j \in T \cup F} v_M(x_j) \ .$$

Thus if $v(M)$ is large we can feel confident that $M$ is satisfied by the input vector.

We can now describe the algorithm.

### 2.3 Algorithm

The basic idea is to look, at each step, for a likely minterm or maxterm, and try to make it more likely, before $cn\log(\min\{N_0, N_1\}/q)$ queries are used up, where $c$ is a constant dependent on $p$.

At each step, we do not know the underlying input vector $\vec{x}$, but we know what it looks like: $x_j$ "looks like" a 1 if $v(x_j) > 0$ and like a 0 if $v(x_j) < 0$, and it could be either if $v(x_j) = 0$. It is easy to see that at each step, there is at least one minterm or maxterm $M$ with a non-negative label. Moreover, it is impossible to have simultaneously $v(M_1) > 0$ and $v(M_0) > 0$ for a minterm $M_1$ and a maxterm $M_0$.

The algorithm is now very simple. Formally, it goes as follows:

Initialization: *Choose a minterm $M = (T, F)$.*
Repeat:
- *Query $x_i \in T \cup F$ such that $v_M(x_i)$ is minimum*
- *if $v(M) < 0$, replace $M$ by another minterm or maxterm with a non-negative label.*

Until:
(1) $\exists M_1$ *minterm such that* $v(M_1) \geq c\log(N_0/q)$, or
(2) $\exists M_0$ *maxterm such that* $v(M_0) \geq c\log(N_1/q)$, or
(3) *the total number of queries has exceeded* $cn\log(\min\{N_0, N_1\}/q)$.
In Case (1), output "$f = 1$", else output "$f = 0$".
The constant $c$ satisfies $c = \Theta(1/(2p-1)^3)$. We assume $q > 16/n$.

# 3 Correctness

What is the probability that the algorithm is wrong? In this section, we show that the probability that the algorithm outputs the wrong answer when it terminates by condition (1) or (2) is less than $q/2$. In the next section, we show that the probability that the algorithm terminates by condition (1) or (2) is at least $1 - q/2$. Thus the probabililty of failure is less than $q$.

Our analysis uses facts about random walks in $\mathcal{Z}$.

### 3.1 Behaviour of random walks

For a given input $I$ to $f$, and a variable $x_i$ which has been queried exactly $k$ times,

let

$$h_{x_i}(k) = \begin{cases} v(x_i) & \text{if } x_i = 1 \text{ in } I \\ -v(x_i) & \text{if } x_i = 0 \text{ in } I \ . \end{cases}$$

Then for any $x_i$, $h_{x_i}(k) = 0$ initially, and $h_{x_i}(k+1) = h_{x_i}(k) + 1$ with probability $p$ and $h_{x_i}(k+1) = h_{x_i}(k) - 1$ with probability $(1-p)$. The function $h$ describes a weighted random walk on a line. (Since the behavior of $h_{x_i}$ is the same for all $x_i$, we will drop the $x_i$ from the notation where there is no ambiguity.)

We will need the following lemmas.

**Lemma 1** *Let $l \geq 0$. The probability that the random walk $(h(k))$ ever reaches level $-l$ is*

$$\left(\frac{1-p}{p}\right)^l \ .$$

**Proof:** Let $p_{-l}$ denote the probability that the random walk ever reaches level $-l$. Let $\alpha$ denote the probability that $h(k)$ is non-negative for all $k$. Then $p_{-1} = 1 - \alpha$, and it is easy to see that $p_{-l-1} = p_{-l}p_{-1} = (1-\alpha)^l$. A random walk which starts from 0 and always stays non-negative must begin by a step up from 0 to 1 and later drop by at most 1 overall. Thus $\alpha = p(1 - p_{-2}) = p(1-(1-\alpha)^2)$, from which we obtain $\alpha = (2p-1)/p$ and lemma 1 follows.•

**Corollary 1** *The probability that the random walk $h(k)$, starting at level 0 ever returns to level 0 is $2-2p$.*

**Proof:** If $h(1) = 1$, then the probability that $h$ falls below 1 is $1-\alpha$, where $\alpha$ is defined as in the proof of Lemma 1. If $h(1) = -1$, then $h = 0$ with probability 1. Hence the probability is $p(1-\alpha) + 1 - p$. •

**Lemma 2** *Let $l \geq 0$. Let $X_l$ denote the last time $k$ such that $h(k) = l$. Then:*

$$E(X) = \frac{1}{2p-1}l + \frac{4p(1-p)}{(2p-1)^2}$$

$$\sigma^2(X) = \frac{4p(1-p)}{(2p-1)^3}l + \frac{8p(1-p)}{(2p-1)^4}.$$

**Proof:** $\Pr\{X_l = k\}$ is the probability that $h(k) = l$ and $h$ always stays above $l$ after time $k$. From lemma 1, we get

$$\Pr\{X_l = k\} = \Pr\{h(k) = l\} p(1 - \frac{1-p}{p}) = (2p-1)\Pr\{h(k) = l\}.$$

Let $p_k = \Pr\{h(k) = l\}$. Assume that $l$ is even: $l = 2l'$. Then:

$$p_k = \begin{cases} 0 & \text{if } k \text{ is odd} \\ \binom{2k}{\frac{k-l}{2}} p^{\frac{k+l}{2}} (1-p)^{\frac{k-l}{2}} & \text{if } k \text{ is even} \end{cases}$$

Let $f(z) = \sum_k \binom{2k}{k-l'} z^{k-l'}$. From [?] I, 1.2.6, equation (17) and exercice 26, we obtain $f(z) = p^{-l}/(2p-1)$, where $z = p(p-1)$. Thus $\partial p / \partial z = 1/(2p-1)$ and $f'(z) = p^{-l-1}(-l(2p-1) - 2p)/(2p-1)^3$. We can use that to compute the expectation:

$$\begin{aligned} E(X) &= \sum_k k \Pr\{X = k\} \\ &= l + (2p-1) \sum_k 2(k - l') p_{2k} \\ &= l + 2(2p-1) p^l z f'(z) \\ &= \frac{l}{2p-1} + \frac{4p(1-p)}{(2p-1)^2}. \end{aligned}$$

The variance of $X$ is $\sigma^2(X) = \sum_k k^2 \Pr\{X = k\} - E(X)^2$, and after a few manipulations, we get

$$\sigma^2(X) = 4(2p-1) p^l z^2 f''(z) + 2(E(X) - l) - (E(X) - l)^2.$$

Differentiating $f'$ with respect to $z$, we compute

$$f''(z) = \frac{p^{-l-2} l^2}{(2p-1)^3} + \frac{p^{-l-2}(8p-1)l}{(2p-1)^4} + \frac{12p^{-l}}{(2p-1)^5}.$$

We finally obtain:

$$\sigma^2(X) = \frac{4p(1-p)l}{(2p-1)^3} + \frac{8p(1-p)}{(2p-1)^4}. \bullet$$

**Lemma 3** *(Chebyshev) Given $n$ random, independent variables $X_1, ..., X_n$ with variances ${\sigma_i}^2$ and expectations $E_i$, the probability that the sum of the variables exceeds $nt + (E_1 + ... + E_n)$ is less than $({\sigma_1}^2 + ... + {\sigma_n}^2)/n^2t^2$.*

At some point in the random walk, since $p$ is a constant greater than $1/2$, $h$ will not fall below 0 again and will instead approach infinity. We say that the value of the variable $x_j$ is *decided* after $h_{x_j}$ has crossed 0 for the last time.

## 3.2 Termination Conditions (1) and (2)

Let us suppose that the input $I$ is such that $f(I) = 1$ (the other case is similar). Then there is at least one minterm $M_1$ satisfied by $I$. What is the probability that the algorithm terminates by condition (2) and outputs "$f(I) = 0$"? This occurs only if it finds a maxterm $M_0$ which seems to be satisfied, i.e., which has a high label.

Since $M_0$ and $M_1$ are incompatible, there is some variable $x_i$ in $T_0 \cap F_1$ or $T_1 \cap F_0$ whose label is misleading, i.e., at the termination point,

$$v_{M_0}(x_i) \geq c \log(N_1/q) \enspace ,$$

and

$$h_{x_i} \leq -c \log(N_1/q) \enspace .$$

The probability of a wrong output given input $I$ is the probability that some variable in $M_1$ has $h \leq -c\log(N_1/q)$:

$$\begin{aligned} &\Pr\{\text{wrong output}\} \leq \\ &\quad \leq \ \Pr\{\exists x_i \in M_1, h_{x_i} \leq -c\log(N_1/q)\} \\ &\quad \leq \ |T_1 \cup F_1| \Pr\{h \leq -c\log(N_1/q)\}. \end{aligned}$$

By Lemma **??** the probability that $h$ ever reaches level $-c\log N_1/q$ is

$$\left(\frac{1-p}{p}\right)^{c\log(N_1/q)} < \frac{q}{2N_1}$$

provided that $c \geq 2/\log(p/1-p)$. (Note that for $p = 1/2 + \epsilon$, $\log(p/1-p) \sim 4\epsilon$. Therefore $2/\log(p/1-p) \leq c = \Theta(1/(2p-1)^3)$ for an appropriate choice of the constant factor.)

Thus $\Pr\{$ wrong output $\} < N_1 q/2N_1 = q/2$.

# 4 Termination Condition (3)–Limits on the Number of Queries

We show that the probability that condition (3) is reached before condition (1) or (2) is satisfied is less than $q/2$.

Let us assume that $N_1 \leq N_0$ (the other case is similar).

We note that we may assume that $N_1 < n/\log n$ and $q > 16/n$ for otherwise, the trivial algorithm of querying each variable $\log(n/q)$ times and taking the majority does as well as our algorithm.

We partition the queries into the following classes:

**Class 1:** Queries made on any variable before its value is decided, that is, before $h$ remains positive.

*Claim: The probability that $\alpha n$ queries do not suffice is less than $1/n$, where $\alpha = O(1/(2p-1)^2)$.*

*Proof:* Let $Y_i$ = the number of queries made on variable $x_i$ before $h_{x_i} = 0$ for the last time. Then let $Y = \sum_{j=1}^{n} Y_i$. ¿From Lemma **??** we have that for each $i$, $E[Y_i] = 4p(1-p)/(2p-1)^2$ and $\sigma^2(Y_i) = 8p(1-p)/(2p-1)^4$. Lemma **??** then implies the claim. •

**Class 2:** Queries made during a test of a maxterm which were made on a variable after its value is decided.

*Claim: The probability that $O(\beta n)$ queries do not suffice is less than $1/n$, where $\beta = O(\log(N_1/q)/(2p-1)^2)$.*

*Proof:* Let $Y_i$ be the number of queries made on variable $x_i$ after $h = 0$ for the last time and until $h$ reaches $\log(N_1/q)$ or $x_i$ is no longer queried. By Lemma **??**, we have that $E[Y_i] = c\log(N_1/q)/(2p-1) + 4p(1-p)/(2p-1)^2$ and $\sigma^2(Y_i) = 4p(1-p)c\log(N_1/q)/(2p-1)^3 + 8p(1-p)/(2p-1)^4$. Lemma **??** concludes the proof of the claim. •

**Class 3:** Queries done on variables in the test of the final minterm $M$ (if the algorithm ends up deciding that $f$ must be 1), after its value is decided.

*Claim: Assume that $N_1 < n/\log n$ and $q > 16/n$. The probability that $\gamma n$ queries do not suffice is less than $1/n$, where $\gamma = O(1/(2p-1)^2)$.*

*Proof:* Let $Y_i$ be the number of queries made on variable $x_i$ after $h = 0$ for the last time and until $h$ reaches $\log(N_0/q)$. Let $Y = \sum_{i|x_i \in M} Y_i$. By Lemma **??**, we have that $E[Y_i] = c\log(N_0/q)/(2p-1) + 4p(1-p)/(2p-1)^2$ and $\sigma^2(Y_i) = 4p(1-p)c\log(N_0/q)/(2p-1)^3 + 8p(1-p)/(2p-1)^4$. By Lemma **??**,

$$\Pr\{Y > \frac{cN_1\log(N_0/q)}{(2p-1)} + \frac{4p(1-p)N_1}{(2p-1)^2} + N_1 t\}$$
$$\leq \left[\frac{4p(1-p)cN_1\log(N_0/q)}{(2p-1)^3} + \frac{8p(1-p)N_1}{(2p-1)^4}\right] / ({N_1}^2 t^2).$$

Since $N_1 < n/\log n$ and $q > 16/n$, we have: $N_1 \log(N_0/q) < 2n$. Using those facts and choosing $t = O[(n/N_1)(1/2p-1)^2]$, we get the statement of the claim. •

**Class 4:** Queries done on variables after their values are decided and during the test of a non-final minterm.

This is the only somewhat complicated case. The ideas is to show that for each variable $x_j$, there is a bound $b_j$ such that $x_j$ is only queried during the test of a non-final minterm when $h_{x_j} < b_j$, and these bounds $b_j$ are such that with probability $1 - q/8$, $\sum_{j=1}^{n} b_j = O(n\log N_1/(2p-1)^2)$. It follows from Lemmas **??** and **??** that with probability at least $1-q/4$, $O(n\log N_1/(2p-1)^3)$ queries suffice.

The algorithm queries one clause for some time, then another clause for some time, and so on. We call *minterm test* a sequence of consecutive queries performed by the algorithm on the same minterm.

For each test of a minterm $M$, and for every variable $x \in M$, we have $v_M(x) \geq 0$ at the beginning of the minterm test; if $M$ is non-final, there is a *spoiling* variable $x_s$ such that $v_M(x_s) = -1$ at the end of the minterm test. In between, $v_M(x_s)$ wanders in the non-negative range. (Note that $M$ may be queried again during another minterm test).

*Claim: Let $l$ be the maximum level reached by $v_M(x_s)$ during the minterm test. Every $x_j \in M$ which is queried during that minterm test has $v_M(x_j) \leq l+1$ at any time during the minterm test.*

*Proof:* At each step, the algorithm queries a variable $x_j$ only if $v_M(x_j)$ is minimal over all labels of variables in the same minterm. Thus $v_M(x_j) \leq$

$v_M(x_s) \le l$ just before the query, and $v_M(x_j) \le l+1$ after the query. •

It follows from the claim that the maximum value that $h_{x_j}$ can attain during a test of a non-final minterm is $l'+1$ where $l'$ is the maximum, over all non-final minterm tests involving $x_j$, of the highest label attained by the spoiling variable during that minterm test.

To analyze the heights of the spoiling variables, we will use the fact that $h(x_s)$ crosses the axis during the minterm test which it spoils. Let us call the maximum value which each $|h_{x_i}|$ attains between each two consecutive times it equals 0 a *peak*.

In the worst case, the largest $n/N_1$ peaks are heights of labels of spoiling variables. Also in the worst case, the largest peak "affects" $N_1 - 1$ variables, the second largest affects a disjoint set of $N_1 - 1$ variables, etc. Let $L$ be a multiset containing the $n/N_1$ largest peaks. We have

$$\sum_{j=1}^{n} \max\{h_{x_j} \text{ attained during tests of a non-final minterm}\}$$

$$\le \sum_{l \in L} (l+1) N_1 \ \ .$$

It remains to prove the following:

*Claim: The probability is at least* $1 - 1/8q$ *that:*

$$\sum_{l \in L} (l+1) N_1 = O\left(\frac{n \log N_1}{(2p-1)^2}\right) \ \ .$$

*Proof:* We first show that with high probability the number of peaks is small and that a peak is unlikely to be large. Using those two facts, proved with the lemmas below, we show a bound on $\sum_{l \in L}(l)$.

**Lemma 4** *Let* $X_i$ *denote the number of passes through* 0 *by variable* $x_i$. *Then:*

$$E(X_i) = \frac{2(1-p)}{2p-1}$$

$$\sigma^2(X_i) = \frac{2(1-p)}{(2p-1)^2}.$$

*Proof of lemma:* Let $p_k = \Pr\{X_i \geq k\}$. We have: $p_0 = 1$, and for $k \geq 1$, $p_k = p_1 p_{k-1}$. From Corollary **??**, $p_1 = 2(1-p)$ hence the lemma. $\bullet$

By Lemma 3, the probability that $X_1 + ... + X_n > \delta n$ is less than $1/n$ for $\delta = O(1/(2p-1))$. In other words, with probability at least $1 - 1/n$, there are fewer than $\delta n$ peaks.

**Lemma 5** *Let $X$ be a random variable which represents the value of a peak. Then* $\Pr\{X \geq i\} \leq ((1-p)/p)^{i-1}$.

*Proof of lemma:* Let $t$ be a time at which $h = 0$. Then the probability that $X \geq i$ is the probability that $|h(k)_{k \geq t}|$ attains at least the value $i$ before returning to 0, given that a peak occurred. First, we compute the probability that $|h(k)_{k \geq t}|$ attained at least $i$ before returning to 0, and then divide by the probability that a peak occurred, i.e. that $h$ returned to 0. Let $Y$ be the random variable which is equal to $X$ if a peak occurs and which is 0 is $h$ never returns to 0 after time $t$.

With probability $1-p$, $h$ starts by going down, i.e. $h(t+1) = -1$. Let $h'(k) = h(k-1) + 1$. Then

$$\Pr\{Y \geq i \mid h(t+1) = -1\} \leq \Pr\{h' \text{ ever reaches level } -(i-1)\} \leq (\frac{1-p}{p})^{i-1}$$

by Lemma 1.

With probability $p$, $h$ starts by going up. Let $w$ be the first time when $h = i$. Let $h'(k) = h(k-w) - i$, $k \geq w$. Then:

$$\Pr\{Y \geq i \mid h(t+1) = 1\} \leq \Pr\{h' \text{ ever reaches level } -i)\} \leq (\frac{1-p}{p})^{i}.$$

by Lemma 1.

Combining these results gives an upper bound of:

$$\Pr\{Y \geq i\} \leq (1-p)(\frac{1-p}{p})^{i-1} + p(\frac{1-p}{p})^{i} = (2-2p)(\frac{1-p}{p})^{i-1}.$$

Dividing this expression by the probability that a random walk beginning at 0 returns to 0 (see Corollary **??**) yields the lemma. $\bullet$

We are now ready to prove the claim.

Given a set of peaks, the probability that each peak $i$ attains a specific value $l_i$ is $[(1-p)/p]^{\sum_i (l_i - 1)}$.

Recall that $L$ contains the $n/N_1$ highest peaks, and that there are fewer than $\delta n$ peaks with high probability. The probability that the values in $L$ sum to $B$ is no greater than the number of ways of partitioning B among the $n/N_1$ elements in $L$ times the probability that each peak in $L$ reaches the value assigned to it by the partition.

To give an upper bound on the probability that *any* multiset of size $n/N_1$ of peaks sums to $B'$, we multiply by the number of ways of choosing $n/N_1$ peaks from the total of $\delta n$ peaks.

Finally, we give an upper bound on the probability that any subset sums to more than B, by summing these probabilities for all $B' > B$.

Hence, the probability that the $n/N_1$ largest peaks sum to more than $B$ is bounded above by

$$\sum_{B'>B}^{\infty} \binom{\delta n}{n/N_1} \binom{B' + n/N_1 - 1}{n/N_1 - 1} (\frac{1-p}{p})^{B'-n/N_1} < q/8,$$

using the well-known bound

$$\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k,$$

and taking

$$\begin{aligned} B &= O(\frac{n}{N_1}\frac{\log \delta N_1}{\log(\frac{p}{1-p})} + \frac{\log q}{\log(\frac{p}{1-p})}). \\ &= O(\frac{n}{N_1}\frac{\log \delta N_1}{(2p-1)^2} \end{aligned}$$

Thus, with probability at least $1 - q/8$,

$$\sum_{j=1}^{n} \max\{h_{x_j} |\text{ attained during tests of a nonfinal minterm}\}$$

$$< N_1 B$$

$$= O(n \log N_1/(2p-1)^2).\bullet$$

We have now shown that with probability at least $1 - q/8$, each $h_{x_j}$ is queried only while its value is less than $b_j$ where $\sum_j b_j == O(n \log N_1/(2p - 1)^2)$. Lemmas 2 and 3 imply that with probability at least $1/n < q/8$, each

$h_{x_j}$ will exceed $b_j$ after a total of $O(n \log N_1/(2p-1)^3$ queries. Hence, with probability at least $1-q/4$, $O(n \log N_1/(2p-1)^3$ queries suffice for Class 4.

Finally we conclude that the probability that $O(n \log(N_1/q)/(2p-1)^3)$ queries do not suffice is no greater than the sum of the probabilities that $O(n \log(N_1/q)/(2p-1)^3)$ queries do not suffice in of Cases (1)-(4), which is no greater than $q/2$, for $q > 16/n$.

# 5 Further Directions

The bounds given in Theorem 1 were shown to be tight for symmetric functions (see [**?**] ). However, they are not tight for all functions. Consider the following function on $x_1, x_2, ..., x_n$ bits, defined when $\sqrt{n}$ is an integer. Let

$$f = 1 \text{ iff } \exists i \in \{0, 1, ..., \sqrt{n}-1\} \text{ s. t. } x_{i\sqrt{n}+1} = x_{i\sqrt{n}+2} = ... = x_{(i+1)\sqrt{n}} = 1.$$

The minterms and maxterms of $f$ are all of size $\sqrt{n}$, hence Theorem 1 gives an $O(n \log(n/q(n)))$ upper bound on the cost of determining $f$ when the queries are answered with probability of correctness at least a constant $p$. But we can construct a faster algorithm for this.

By Theorem 1, there is an algorithm which takes $O(\sqrt{n}/\log(1/q))$ queries to compute the AND of $\sqrt{n}$ bits with probability no more than $q$ probability of error, where the queries are correct with probability at least $p$. Call this DO_AND $(x_1, x_2, ..., x_{\sqrt{n}}, p, q)$. Similarly, there is an algorithm which takes $O(\sqrt{n}/\log(1/q)$ queries to compute the OR of $\sqrt{n}$ bits with probability no more than $q$ of error, where the queries are correct with probability at least $p$. Call this DO_OR $(x_1, x_2, ..., x_{\sqrt{n}}, p, q)$.

To compute $f$ with probability of error no greater than $q'(n)$, we modify DO_OR $(x_1, x_2, ..., x_{\sqrt{n}}, p, q'(n))$. by replacing each occurrence of a query $i$ with a call to DO_AND$(x_{i\sqrt{n}+1}, ...x_{(i+1)\sqrt{n}}, p, 1-p)$.

Since each computation of DO_AND( $x_{i\sqrt{n}+1}, ...x_{(i+1)\sqrt{n}}, p, 1-p)$ takes $O(\sqrt{n})$ time, the overall cost is $O(n/\log(1/q'(n)))$.

This example suggests that savings are possible, and additional criteria need to be considered when determining the noisy Boolean decision tree complexity of a function.

# References

[1] Uriel Feige, David Peleg, Prabhakar Raghavan and Eli Upfal. *Computing with Noisy Information*, $22^{nd}$ Annual ACM Symposium on Theory of Computing, 1990, 128-137.

[2] Claire Kenyon and Andrew C. Yao. *On evaluating boolean functions with unreliable tests*, International Journal of Foundations of Computer Science, 1, 1(1990), 1-10.

[3] William Feller. *An Introduction to Probability Theory and its Applications*, Volume 1, Wiley and Sons 1957.

[4] Anna Gal *Lower Bounds for the Complexity for Reliable Boolean Circuits with Noisy Gates*, FOCS '91, 594-601.

[5] N. Nisan *CREW PRAMs and Decision Trees* STOC '89, 327-335.

[6] N. Pippenger *On Networks of Noisy Gates* FOCS '85, 30-38.

[7] R. Reischuk and B. Schmeltz. *Reliable Computation with Noisy Circuits and Decision Trees – A General* $n \log n$ *Lower Bound*, FOCS '91, 602-611.

[8] E. F. More and C. Shannon. *Reliable circuits using less reliable relays i-ii*, J. Franklin Institute, 1956, 262:191-208.

[9] J. Von Neumann. *Probabilistic logics and the synthesis of reliable organisms from unreliable components*, in Automata Studies, 43-98, Princeton University Press, Princeton, NJ, 1956.

[10] A. Pelc. *Searching with known error probability.* Theoretical Computer Science, 63: 185-202, 1989.

[11] R. L. Rivest, A. R. Meyer, D. J. Kleitman, K. Winklmann, and J. Spencer. *Coping with errors in binary search procedures*, J. of Computer and System Sciences, 20: 396-404, 1980.

[12] D. E. Knuth, *The art of computer programming*, Vol I: Fundamental Algorithms, Addison-Wesley, 1968.