Rounds vs Queries Trade-off in Noisy Computation
[Extended Abstract]

Navin Goyal* Michael Saks†

Abstract

We show that a noisy parallel decision tree making $O(n)$ queries needs $\Omega(\log^* n)$ rounds to compute OR of n bits. This answers a question of Newman [21]. We prove more general trade-offs between the number of queries and rounds. We also completely settle a similar question for computing MAX in the noisy comparison tree model; these results bring out interesting differences among the noise models.

1 Introduction

Noise is ubiquitous and there is a vast literature devoted to its impact on computation and communication. Computation in a noisy environment has been studied in many models including decision trees [12, 25, 10, 8, 21], formulas and circuits [23, 14, 28, 17, 9] , various kinds of communication protocols [15, 26, 18, 11, 21], sorting networks [19], cellular automata [13], quantum computation [3, 5], and other situations, eg [1, 7, 22, 16, 20, 29]. In this paper we will be concerned with the noisy boolean decision tree model and the noisy comparison tree models.

1.1 Noisy Boolean Decision Trees

In the usual (noise-free) model, a boolean function $f(x_1, \ldots, x_n)$ is to be computed by querying variables in an adaptive manner. An algorithm in this model is represented by a decision tree. In the noisy version of the model, each reported answer may be incorrect. Several models have been proposed to formalize noise. In the simplest model, *the random noise model*, there is a noise parameter $\varepsilon \in [0, 1/2)$ and the outcome of each query is incorrect independently with probability exactly ε. To compute f in this model means to produce an output that agrees with f with probability at least $1 - \delta$ where $\delta > 0$ is fixed.

A noise-free boolean decision tree of depth d can be simulated by a noisy decision tree of depth $O(d \log d)$: Each query of a variable in the noise-free setting is simulated by taking the majority of $\Omega(\log d)$ noisy queries of the same variable. Since a noise-free decision tree can compute any n-variate function in at most n queries, this shows that any boolean function can be computed in the noisy model with at most $O(n \log n)$ queries. Feige et al [12] showed that to compute MAJORITY and PARITY in the noisy decision tree model $\Omega(n \log n)$ queries are indeed necessary. In contrast, they showed that OR, which requires n queries in the noise-free model, can be computed in $O(n)$ queries in the noisy model.

There is a natural notion of parallelism for decision trees [30]. The algorithm works in rounds, and in each round it makes a set of queries in parallel. The goal is to keep both the total number of queries and the number of rounds small. See section 2 for precise definitions. Noisy parallel decision tree algorithms have been used to design protocols for the noisy radio broadcast model [21]; the number of queries in the decision tree model corresponds to the number of broadcasts needed.

The $O(n)$ query algorithm for OR of Feige et al [12] runs in $O(\log n)$ rounds and this was reduced to $O(\log^* n)$ rounds by Newman [21]. Newman raised the question: Is there a noisy decision tree for OR using $O(n)$ queries and $O(1)$ rounds? In this paper we answer this question in the negative by showing that any $O(n)$ query algorithm requires $\Omega(\log^* n)$ rounds.

THEOREM 1.1. *There is an integer n_0 depending on ε such that for all $n \geq n_0$ and $r \leq (\log^*_{1/\varepsilon} n)/2$, a (possibly randomized) parallel noisy decision tree computing OR of n input bits with query complexity $\leq \frac{1}{10^3} n \log^{(r)}_{1/\varepsilon} n$ needs at least r rounds. In particular, if the query complexity is $\leq cn$ for any constant c then $\Omega(\log^*_{1/\varepsilon} n)$ rounds are necessary.*

This is the first non-trivial trade-off lower bound between the number of rounds and the total number of queries for noisy parallel decision trees. The results are tight up to constants in light of the $O(\log^*_{1/\varepsilon} n)$ algorithm of Newman [21], and similar algorithms for other

*Department of Computer Science, Rutgers University, Piscataway, NJ 08854, USA, email: ngoyal@cs.rutgers.edu. Supported in part by Louis Bevier Fellowship.

†Department of Mathematics, Rutgers University, Piscataway, NJ 08854, USA, email: saks@math.rutgers.edu.

points on the trade-off curve are easy to derive.

It has been argued that the assumptions of the random noise model are inappropriate for designing algorithms, because an algorithm (or circuit, protocol etc) may exploit this regularity in noise (eg, [11]). Other stronger models of noise have been proposed which eliminate this regularity, and allow the noise to be adversarial. There are at least two such models: (1) Static adversary model (eg, [11]); this is called the fault tolerance model by Newman [21]. (2) Clairvoyant adversary model [11]. (There is no standard terminology, and sometimes the above names may be used for different models.) In the static adversary model the adversary can preset the probability of error for each query, the only restrictions are that each of these probabilities is at most ε, and the errors are independent. In the clairvoyant adversary model, we first produce a set of noise random variables as in the random noise model: For each possible query that the algorithm can make, we have a bit which is 1 with probability ε and 0 with probability $1-\varepsilon$, independent of the other bits. If the bit corresponding to a query is 1 then the answer to that query is allowed to be corrupted by the adversary otherwise it is not corrupted. Now with the knowledge of the decision tree and these bits adversary can choose to corrupt any set of answers that it is allowed to corrupt.

The random noise model is the weakest of the three and the clairvoyant adversary model is the strongest. For upper bounds, one seeks to design algorithms that tolerate noise in the strongest model possible. Since our lower bound is proved for the weakest model, they hold for the other models as well.

Our lower bound for OR is deduced from a lower bound on a related problem, the "Which half?" problem. The input to this problem consists of $2n$ bits, exactly one of which is known to be 1 and the problem is to determine whether the 1 is in the first or second half of the bits. We prove that if the location of the 1 is chosen at random then this problem can't be answered with a linear number of queries and few rounds. The proof is by round elimination: we prove a lemma that shows that after the first round of queries, we are left with a problem that is at least as hard as the original problem on a not much smaller set of inputs. By repeatedly applying the lemma we are reduced to a one round algorithm, for which a lower bound is easy to prove.

1.2 Noisy Comparison Decision Trees In the familiar comparison decision tree model, the variables take values from an abstract totally ordered set and a single query consists of comparing two variables. The parallel version of this model was considered by Valiant [30]. The random noise model (and the other two models of noise) can be adapted in the obvious way to comparison decision trees. We are interested in the trade-offs between the total number of queries and the number of rounds used in computing the MAX of n inputs by a parallel noisy comparison decision tree. Such tree were considered by Feige et al [12], but they did not study the trade-off question. Among other results they showed that MAX can be computed in $O(\log n)$ rounds using $O(n)$ queries in the random noise models as well as other stronger models. We briefly review the known results for MAX in the noise-free models as we use these results, and also because then one can then see the impact of noise by comparing these results with the results in the noisy models.

Valiant showed that for deterministic algorithms $\Omega(\log\log n)$ rounds are essential if in each round the algorithm can make at most $O(n)$ queries. He gave an algorithm running in $O(\log\log n)$ rounds and making $O(n\log\log n)$ queries. This was improved to $O(\log\log n)$ rounds and $O(n)$ queries by Shiloach and Vishkin [27]. Thus for deterministic parallel decision trees $\Theta(\log\log n)$ rounds are necessary and sufficient if the total number of queries is $O(n)$. Reischuk [24] gave a randomized algorithms with $O(1)$ rounds and $O(n)$ queries. Clearly this is the best possible. Now we describe our results.

THEOREM 1.2. *For computing MAX of n variables using noisy randomized comparison decision trees making $O(n)$ queries, $\Theta(\log^*_{1/\varepsilon} n)$ rounds are necessary and sufficient. This is true for all three models of noise.*

The lower bound is proved by a reduction from OR; the upper bound adapts Reischuk's [24] algorithm to the noisy case. In the random noise model if the number of queries is $O(n)$ then $O(\log^*_{1/\varepsilon} n)$ rounds are sufficient also for the deterministic algorithms. Deterministic algorithms in the random noise model have access to $O(n)$ biased independent random bits, obtained for example by querying the same variable many times. These bits can be used to simulate the algorithm in theorem 1.2 which uses $O(n^{1/3}\log n)$ random bits.

In the fault tolerance or clairvoyant adversary models this is not possible. Indeed, in these models a deterministic algorithm making $O(n)$ queries needs $\Omega(\log\log n)$ queries. This is because the adversary can choose to introduce no noise, and then the problem becomes noise-free and Valiant's argument applies. We have

THEOREM 1.3. *There is a noisy deterministic parallel comparison decision tree computing MAX of n variables in $O(\log_{1/\varepsilon}\log_{1/\varepsilon} n)$ rounds and $O(n)$ queries in the fault tolerance and clairvoyant adversary models.*

The upper bound is obtained by adapting the algorithm of [27] to the noisy case. The main idea is that the algorithm of [27] leaves enough room to adjust its parameters so that one can make additional queries which make the algorithm robust to the noise.

At high level, our algorithms are robust to adversarial noise because they make decisions based on majority voting. Eg, to estimate a variable we query it some number of times and take its value to be the majority of the received bits. It is easy to see that probability of the estimate obtained this way being correct cannot be reduced by correcting some bits or reducing the noise probability.

The rest of this paper is organized as follows. Section 2 contains a description of the model and some definitions. Section 3 explains the adversary's strategy and the round elimination lemma. Section 4 has the proof of theorem 1.1. In section 5 we sketch the results in the comparison tree model. Section 6 concludes with some open problems.

2 Preliminaries

The *boolean decision tree model* is an abstraction of algorithms evaluating functions of n boolean variables $x_1, \ldots, x_n$ using queries to individual variables. We denote by $x = (x_1, \ldots, x_n)$ the vector of these variables. A boolean decision tree over boolean variables $\{x_1, \ldots, x_n\}$ is a rooted binary tree where each internal node of the tree is labeled by a variable, and the leaves are labeled 0 or 1. In a noise-free execution, the computation starts with the algorithm querying the variable labeling the root. Then the algorithm queries the left child if the answer is 0, and queries the right child if the answer is 1, and so on, until it gets to a leaf when it announces the label of the leaf as the answer of the computation. Thus each assignment to the variables determines a unique root-leaf path.

In the *noisy boolean decision tree* model [12], on each fixed input the output of the algorithm is a random variable determined as follows: Each query is answered incorrectly (independently of the other queries) with probability ε, where $\varepsilon \in [0, 1/2)$ is a constant. We say that a decision tree computes a function $f : \{0,1\}^n \to \{0,1\}$ if for all inputs x, it outputs $f(x)$ with probability $\geq 2/3$, where the probability is taken over the random noise. Of course, $2/3$ is an arbitrary constant, and we could choose it to be any constant in $(1/2, 1)$ using standard amplification.

In a *parallel* noisy decision tree (explicitly defined by Newman [21]), leaves are labeled 0 or 1 as before. Each internal node is labeled by a multiset of variables which corresponds to the queries asked at that node, and there is a branch from the node for each of the possible outcomes of the queries. The output of a noisy parallel decision tree on a given input is a random variable whose distribution is given as in the sequential case.

The query complexity of a decision tree can be defined in several ways. One is the number of queries made on the worst case input for the worst choice of noise. This is just the depth of the decision tree. We can consider for every input the average number of queries made on it (where the average is over the random noise) and take the maximum of this over the inputs. Finally, we can average the number of queries on both the inputs and the random noise. This last measure depends on the distribution on the input. (The fourth possibility of averaging over the input and taking the worst case over the noise does not seem very interesting.)

For parallel decision trees, we get two complexity measures; each of them can be considered in the various average and worst case scenarios as above. One measure comes from counting the total number of queries made in a computation; this corresponds to the usual decision tree query complexity. The other measure comes from the depth of the leaf reached. This gives the number of rounds used by the tree.

In this paper for the boolean decision tree case we are concerned with (1) the query complexity averaged over the noise and worst case over the input; (2) number of rounds taken worst case over both noise and input. (Actually, we can take the query complexity to be average both over the input and the noise for a certain distribution over the input but we do not state our results in that form for simplicity.) One can derive a trade-off similar to but slightly weaker than the one in theorem 1.1 if one uses the number of rounds averaged over the noise and worst case over the input.

All these measures carry over easily to the comparison tree case. Here we will use the worst case measures.

For reals $x \geq a > 1$, and integer $r \geq 0$, $\log_a^{(r)} x$ denotes $\log(\log(\ldots(\log x)\ldots))$ with r logarithms. In particular, $\log_a^0 x = x$. $\log_a^* x$ denotes minimum integer r such that $\log_a^{(r)} x \leq 1$. As usual, $[a, b]$ denotes the set $\{a, \ldots, b\}$, and $[a]$ denotes the set $[1, a]$. For a bit vector u, denote the number of 1s in it by $|u|$. Unit vectors are denoted by $e_1, \ldots, e_n$, so e_i denotes the vector with all except the ith bit 0. The all 0's vector is denoted by just 0. The input is denoted $x = (x_1, \ldots, x_n)$. When the base is not specified the base of logarithm is 2. Notation $O(.)$, $\Omega(.)$ may hide factors depending on ε.

Throughout, we ignore the integrality issues when dealing with fractions, logarithms etc. This has negligible effect; and if desired, it is easy, though tedious, to take integrality explicitly into account.

Instead of working with OR directly, it is more

convenient to work with the following problem which we call the *Which Half? Problem* (WHP).

DEFINITION 2.1. *(WHP_n) Input: A bit vector x of length $2n$, generated by sampling uniformly from the set $\{e_1, \ldots, e_{2n}\}$. Output: 0, if the bit with value 1 appears among the first n bits; 1 otherwise.*

LEMMA 2.1. *If there is a qn-query r-round algorithm for OR_n then there is a qn-query r-round algorithm for WHP_n.*

Proof. Let A be a qn-query r-round algorithm for OR_n. Now if we feed A with the last n bits of the input to WHP_n then it computes the answer to WHP_n.

3 The round elimination lemma

As stated in the introduction, the round elimination lemma says that if there is an r-round algorithm for WHP_n using at most qn queries then there is an $r-1$-round algorithm for $WHP_{n'}$ with at most qn queries whose error probability is not much larger. The precise statement of this lemma requires two additional parameters θ and λ, which should be thought of as large real numbers.

LEMMA 3.1. *(Round elimination) Let $\theta \geq 10, \lambda \geq 10, n, q$ be positive integers such that $\sqrt{n} > \lambda\theta\,(1/\varepsilon)^{\theta q}$. If there is an r-round deterministic algorithm for WHP_n with qn queries and error probability δ, then there is an $(r-1)$-round deterministic algorithm for $WHP_{n'}$ with qn queries and error probability $\leq \delta/(1 - 4/\theta - e^{-2(\lambda-1)^2})$, where $n' = (1-1/\theta)\varepsilon^{\theta q} n/2$.*

Proof. We introduce an *oracle* who at the end of the first round of queries, provides some additional information to the algorithm, which includes revealing the true values of some of the variables. Clearly this can not increase the complexity of the algorithm (since the algorithm can ignore this information.) The behavior of the oracle depends on the integer parameters θ and λ mentioned in the lemma. We will show that (1) with probability close to 1, the location of the 1 is among the unrevealed variables, (2) the number of unrevealed variables in each half is the same and both are at least n', (3) conditioned on the results of the first round of queries and the information provided by the oracle, and conditioned on the event that the location of the 1 has not been revealed, the location of the 1 is uniformly distributed among the unrevealed locations.

Let k be the (random) index such that $x_k = 1$. The first round of queries is specified by a vector $m = (m_1, \ldots, m_n)$ where m_i is the number of times x_i is queried. By hypothesis $m_1 + \ldots + m_n \leq qn$.

Let B_1 (resp. B_2) consist of the $\lfloor n/\theta \rfloor$ indices among the first half (resp. second half) of the variables that are queried most often (breaking ties arbitrarily). (Henceforth we omit $\lfloor \cdot \rfloor$ brackets.) The oracle reveals all of the values x_i for $i \in B = B_1 \cup B_2$. If $k \in B$ this ends the algorithm. Otherwise, let $A_1 = [n] - B_1$ and $A_2 = [n+1, 2n] - B_2$. It follows from the definition of B_1 and B_2 that $|A_1| = |A_2| = (1-\theta)n$ and for $i \in A$, $m_i \leq \theta q$. For each $i \in A = A_1 \cup A_2$, the oracle performs $\theta q - m_i$ additional queries to x_i whose answers are given to the algorithm for "free". Thus the algorithm gets exactly θq noisy values of x_i for each $i \in A$. Let m_i be the number of reported values of x_i that are 1. For $j \in \{1, 2\}$, let S_j be the set of indices $i \in A_j$ for which $m_i = m_k$. The oracle reveals the values of the variables outside of $S = S_1 \cup S_2$.

Let $s_j = |S_j|$. If $\min\{s_1, s_2\} < n'$ then the oracle reveals x_k, ending the computation. Otherwise, the oracle chooses $|s_1 - s_2|$ indices at random from the larger of S_1, S_2 and reveals those values. Let S_1', S_2' be the subsets of S_1, S_2 that remain unrevealed. It is immediate from the definition of the oracle that if the oracle does not reveal x_k, then $|S_1'| = |S_2'| \geq n'$ and conditioned on the query answers and the information provided by the oracle, the value of k is uniformly distributed on $|S_1' \cup S_2'|$.

We next give an upper bound on the probability that the oracle reveals x_k. There are three ways this can happen: (E_1) $x_k \in B$; (E_2) $\min(s_1, s_2) \leq n'$, (E_3) x_k is among the $|s_1 - s_2|$ variables of $S_1 \cup S_2$ revealed. We bound each of these probabilities.

For $E1$ Since there are a total of at most qn queries, at most n/θ variables are queried more than θq times, so $\Pr[x_k \in B] \leq 1/\theta$.

For $E2$ and $E3$, assume that E_1 does not happen. For $i \in A$, let Y_i be the indicator of the event that $m_i = m_k$. $Y_k = 1$ and conditioned on m_k and the event $i \neq k$, the probability p that $Y_i = 1$ is at least $\varepsilon^{m_i}(1-\varepsilon)^{\theta q - m_i} \geq \varepsilon^{q\theta}$.

Let Z denote the random variable which is the sum of $n(1-1/\theta)$ independent Bernoulli random variables with probability p. Then s_1 (resp. s_2) has a distribution of the form $Z+W$ where W is a positive random variable that is at most 1 which accounts for the contribution of Y_k. Let $\mu = (1-1/\theta)np$, which is the expectation of Z. Let F_j be the event that $|s_j - \mu| \leq \lambda\sqrt{n(1-1/theta)} + 1$.

By a version of the Chernoff bound for biased random variables (see, e.g., [4], corr. A.1.7, page 266),

$$\begin{aligned} & \Pr[\bar{F_j}] \leq \Pr|Z - \mu| \leq \lambda\sqrt{n(1-1/\theta)} \\ \leq \;& 2e^{-2(\lambda-1)^2}. \end{aligned}$$

Thus, conditioned on $E1$ not happening, F_1 and F_2

both hold with probability at least $1-4e^{-2(\lambda-1)^2}$. For the given value of λ, F_1 and F_2 imply that $s_1, s_2 \geq n'$ and so $E2$ holds.

Finally conditioned on $E1, F_1, F_2$, the probability that the event $E3$ occurs if x_k is among the $|s_1 - s_2|$ variables of $S_1 \cup S_2$ that the oracle reveals, and this happens with probability $|s_1 - s_2|/|s_1 + s_2|$. Given F_1 and F_2 this is at most $(\lambda\sqrt{n(1-1/\theta)}+1)/n'$ which is bounded below by $3/\theta$ from our assumption $\sqrt{n} > \lambda\theta\,(1/\varepsilon)^{\theta q}$.

So the probability of x_k getting revealed in the first round is $\leq \frac{1}{\theta} + 4e^{-2(\lambda-1)^2} + \frac{3}{\theta} = \frac{4}{\theta} + 4e^{-2(\lambda-1)^2} = t$. Therefore, with probability $1-t$, the algorithm gets an instance of WHP_m for m such that $p(v)(1-1/\theta)n - \lambda\sqrt{(1-1/\theta)n} \leq m \leq (p(v)n - \lambda\sqrt{n})$.

The algorithm has $r-1$ rounds to work on this instance and the error probability δ' that it can afford is such that $\delta'(1-t) \leq \delta$, hence $\delta' \leq \delta/(1-t)$. Since $p(v) \geq \varepsilon^{q\theta}$, this gives us an $(r-1)$-round, qn-query, δ'-error probability algorithm for $WHP_{n'}$, where n' is as in the statement of the theorem.

4 Proof of the main theorem

In this section we prove the main theorem using lemma 3.1. For $i \geq 0$, define,

$$\theta_i = (100)2^i,$$

$$d_i = 10\theta_i,$$

$$\lambda_i = 10 + i.$$

LEMMA 4.1. *Let $i \in [r]$, and let n_i be such that $\log_{1/\varepsilon}^{(i-1)} n_i \geq \frac{10}{\varepsilon^2}$. If there is an i-round algorithm for WHP_{n_i} with error probability $\delta \leq 1/3$, and average number of queries $\leq \frac{\log_{1/\varepsilon}^{(i)} n_i}{d_{r-i}}$, then there is an $(i-1)$-round algorithm for $WHP_{n_{i-1}}$ with $n_{i-1} = \varepsilon^{\frac{\log_{1/\varepsilon}^{(i)} n}{10}} n_i/4$ and error probability $\delta_{i-1} \leq \delta_i/(1 - 4/\theta_{r-i} - e^{-2(\lambda_{r-i}-1)^2})$ and average number of queries $\leq \frac{\log_{1/\varepsilon}^{(i-1)} n_{i-1}}{d_{r-i+1}}$.*

Proof. Let $r \leq (\log_{1/\varepsilon}^{*} n)/2$. Note that for all sufficiently large n, the assumption $\sqrt{n} > \lambda\theta\,(1/\varepsilon)^{\theta q}$ in lemma 3.1 is satisfied for $(\lambda, \theta) = (\lambda_i, \theta_i)$, and $q_i = \frac{\log_{1/\varepsilon}^{(i)} n_i}{d_{r-i}}$, where $i \in [r]$. Below we assume that our n's will be large enough to satisfy this assumption. By lemma 3.1, with the adversary choosing $\theta = \theta_{r-i}$ and $\lambda = \lambda_{r-i}$ there is an $(i-1)$-round algorithm for $WHP_{n_{i-1}}$ with total $\frac{\log_{1/\varepsilon}^{(i)} n_i}{d_{r-i}} n$ queries and error probability $\delta_{i-1} = \delta_i/(1 - 4/\theta_{r-i} - e^{-2(\lambda_{r-i}-1)^2})$, where

$$n_{i-1} = (1 - 1/\theta_{r-i})\varepsilon^{\theta_{r-i}\frac{\log_{1/\varepsilon}^{(i)} n}{d_{r-i}}} n/2 > \varepsilon^{\frac{\log_{1/\varepsilon}^{(i)} n}{10}} n/4.$$

The average number of queries used by this algorithm is

$$\begin{aligned} & \frac{\log_{1/\varepsilon}^{(i)} n_i}{d_{r-i}} \frac{n_i}{n_{i-1}} \leq \frac{4\log_{1/\varepsilon}^{(i)} n_i}{d_{r-i}} \left(\frac{1}{\varepsilon}\right)^{\frac{\log_{1/\varepsilon}^{(i)} n_i}{10}} \\ \leq & \frac{4\log_{1/\varepsilon}^{(i)} n_i}{d_{r-i}} (\log_{1/\varepsilon}^{(i-1)} n_i)^{1/10} \\ \leq & \frac{\log_{1/\varepsilon}^{(i-1)} n_{i-1}}{d_{r-i+1}}. \end{aligned}$$

The last inequality follows from our assumption that $\log_{1/\varepsilon}^{(i-1)} n_i \geq \frac{10}{\varepsilon^2}$. This completes the proof of the lemma.

Let $n = n_r$ satisfy $\log_{1/\varepsilon}^{(r-1)} n_r \geq \frac{10}{\varepsilon^2}$. Suppose that we have an r-round algorithm for WHP_{n_r} with error probability $\delta_r \leq 1/3$, and average number of queries $\leq \frac{\log_{1/\varepsilon}^{(r)} n_r}{d_0}$, then applying lemma 4.1 repeateadly r times we obtain a 0-round algorithm (that is an algorithm which makes no queries) for WHP_{n_0}, where

$$\begin{aligned} n_0 &= \frac{n}{2^{2(r-1)}} \varepsilon^{\frac{1}{10}(\log_{1/\varepsilon}^{(r)} n_r + \ldots + \log_{1/\varepsilon}^{(1)} n_1)} \\ &\geq \frac{n}{2^{2(r-1)}} \varepsilon^{\frac{1}{10}(\log_{1/\varepsilon}^{(r)} n + \ldots + \log_{1/\varepsilon}^{(1)} n)} \\ &\geq n^{1/4}. \end{aligned}$$

This algorithm has error probability

$$\begin{aligned} \delta_0 &\leq \delta_r/(\prod_{j=1}^{r}(1 - 4/\theta_j - 4e^{-2(\lambda_j-1)^2})) \\ &\leq \delta_r/(1 - \sum_{j=1}^{r}(4/\theta_j + 4e^{-2(\lambda_j-1)^2})) \\ &\leq \delta_r/(1 - 4/200 - 1/100) \\ &\leq \delta(1 + 1/10) \leq \frac{1}{3}\frac{11}{10} = \frac{11}{30}. \end{aligned}$$

By lemma 3.1 this gives a 0-round algorithm for WHP_{n_1} with error probability $\frac{11}{30} < 1/2$. But this is impossible, as clearly any 0-round algorithm (that is, an algorithm which does not look at the input) for WHP_n, with $n \geq 1$, has error probability $\geq 1/2$. Hence our initial assumption about the algorithm for WHP_{n_r} was invalid. Hence any deterministic algorithm for WHP_n working in r rounds must use $(n\log_{1/\varepsilon}^{(r)} n)/d_0 = (n\log_{1/\varepsilon}^{(r)} n)/10^3$ queries.

Since we have given a single distribution that is hard for all deterministic algorithms, it follows (by an appropriate variant of Yao's Lemma [31]) that the lower bound extends to randomized decision trees.

For the "in particular" part of the theorem, let $r = (\log^*_{1/\varepsilon} n)/2$, then for any constant c for all sufficiently large n we have $\frac{1}{10^3}\log^{(r)}_{1/\varepsilon} n > cn$. Hence by the first part of the theorem at least $r = \Omega(\log^*_{1/\varepsilon} n)$ rounds are necessary if the number of queries is cn.

REMARK 4.1. *Theorem 1.1 and its proof easily generalize to the following setting where the algorithm has a* helper *who is allowed to set the error probability of each query to any number in* $[\varepsilon, 1/2]$. *This generalization will be useful in the proof of lower bound for* MAX.

5 Comparison Trees

Proof. (sketch for theorem 1.2)

Lower bound. As mentioned before, we prove the trade-off lower bound for MAX by reducing OR to it. We are given the input $x = (x_1, \ldots, x_n)$ for OR, we define input $y_i := x_i + i/2n$ for the MAX problem. Adding $i/2n$ ensures that all the inputs are distinct. To compute $OR(x)$, it is sufficient to compute $MAX(y)$ because, if say y_i is the answer returned by the algorithm for MAX then we can just query x_i $\log n$ times and get its value, which is the value of OR, reliably. To find $MAX(y)$ we simulate the comparison queries of the algorithm for MAX using the boolean queries: To compare y_i and y_j we query x_i and x_j, call the returned values x'_i and x'_j, and return y_i if $x'_i + i/2n > x'_j + j/2n$, else return y_j.

In the above simulation, the probability of error for the comparison query is not a fixed constant and may depend on the values of x_i and x_j. Eg, if $(x_1, x_2) = (0, 0)$, then $(y_1, y_2) = (1/2, 1)$, and so $y_1 > y_2$. The probability that y_1 is the answer of comparison between x_1 and x_2 is $\varepsilon(1-\varepsilon)$ corresponding to the event $(x'_1, x'_2) = (1, 0)$. On the other hand, if $(x_1, x_2) = (0, 1)$, then $(y_1, y_2) = (1/2, 3/2)$, and so $y_1 > y_2$. But now the probability that y_1 is the answer of comparison between x_1 and x_2 is ε^2, again corresponding to the event $(x'_1, x'_2) = (1, 0)$. This issue can be resolved by remark 4.1: Helper manipulates the error probability for the boolean queries in the range $[\varepsilon, 1/2]$, so that the error probabilities for the comparison queries are all the same. Thus we get a lower bound for MAX in the random noise model.

Upper bound. We borrow the sampling idea used by Reischuk [24] to give a noise-free algorithm working in $O(1)$ rounds and $O(n)$ queries. We use a subroutine called $APXM(S, x)$ (for *approximate maximum*), where $S \subseteq U$, and $x \in U$. $APXM(S, x)$ returns a set $T \subseteq S$ such that with high probability, say $> 99/100$, $|T| \leq 2(n - rank(S, x))$ and $\max(S) \in T$ and $x \in T$. $rank(S, x)$ is the number of elements in S smaller than x. It runs in $O(\log^*_{1/\varepsilon} n)$ rounds and $O(n)$ queries. $APXM$ is closely related to the OR problem, and uses ideas from the $\log^* n$-round algorithm of Newman [21] for OR.

We assume that $\varepsilon < 1/4$. $APXM(S, x)$ works as follows. Let S be the set of input variables. $APXM(S, x)$ produces sets $S = S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots$. Set $n_i = |S_i|$, $q_0 = 100$, and $q_i = 100(n/n_i)2^{-(i+1)}$. In round i, it produces S_i by doing q_i comparisons for each variable in S_{i-1} with x, choosing the variables which come out greater in the majority of comparisons. Computation continues until $n_{i+1} > 10^4 e^{-q_i/8} n_i$. If S_r is the last set produced then $APXM(S, x)$ returns $S_r \cup \{x\}$.

We briefly indicate the analysis. By the Chernoff bound, the probability that in the ith round for a variable in S_i the majority vote of comparisons yields wrong answer is $\leq e^{-q_i/8}$. Hence, by another application of the Chernoff bound, $n_{i+1} \approx n_i e^{-q_i/8}$ with high probability, provided that $(n - rank(S, x)) \ll n_{i+1}$. Hence, $q_{i+1} \approx q_i e^{q_i/8}$. This shows that q_i grows as a tower of height i. Hence there are $O(\log^*_{1/\varepsilon} n)$ rounds. The probability that $\max(S)$ gets eliminated $\approx \sum_{i \geq 0} e^{-q_i/8} \ll 1$.

In the algorithm we also need to compute the maximum of sets of size $O(n^{1/3})$. This can be done in a straightforward manner: Compare $c \log n$ times all $O(n^{2/3})$ pairs, for each pair choose that variable as the greater of the two which comes greater in the majority of comparisons. For large constant c this gives the correct total order on the variables with probability $1 - 1/n^{\Omega(1)}$. This procedure uses $O(n^{2/3} \log n)$ comparison.

Now we can present our algorithm for MAX. Let T denote the set of inputs.

1 Uniformly randomly choose $T_1 \subset T$ of size $n^{1/3}$.
$m_1 = \max(T_1)$.
$T_2 = APXM(T, m_1)$.
If $|T_2| > 10n^{2/3}$, HALT with ERROR.

2 Uniformly randomly choose $T_3 \subset T_2$ of size $n^{1/3}$.
$m_2 = \max(T_3)$.
$T_4 = APXM(T_3, m_2)$.
If $|T_4| > 10n^{1/3}$, HALT with ERROR.

3 Output $\max(T_4)$.

The above algorithm uses randomness for choosing T_1 and T_3. For each of these it needs $O(\log \binom{n}{n^{1/3}}) = O(n^{1/3} \log n)$ random bits. It can be simulated by a $O(n)$-query, $\log^*_{1/\varepsilon} n$-round deterministic algorithm which has access to $O(n)$ biased random bits. Simulation requires some care, and can be done for example by approximating the distributions for T_1 and T_2.

Proof. (sketch for theorem 1.3) We describe a deterministic algorithm computing MAX in $O(n)$ queries and $O(\log\log n)$ rounds in the fault-tolerance and clairvoyant adversary models. The algorithm is obtained by modifying the algorithm of [27] for the same problem in the noise-free model.

The algorithm works in two phases. The first phase has $2\log\log n$ rounds, and is similar to the NBA tournament type algorithm of Feige et al [12]. Divide the input variables into groups of size 2. For each group ask 3 times which variable is greater. For each group choose the variable which is greater in the majority of comparisons. These variables participate in the second round, where we again divide these variables into groups of size 2, and for each groups ask 5 queries, and so on. In general, in the ith round, each comparison is made $2i+1$ times. So after $2\log\log n$ rounds we have $n' = n/(\log n)^2$ variables.

For the second phase consider a leveled rooted tree T (not to be confused with a decision tree) with n' leaves. An internal node with ℓ leaves as descendents has $\sqrt{\ell}$ children. It is easy to see that such a tree has height $\log\log n'$. In the second phase the algorithm assigns the n' variables from the first phase to the n' leaves of T. Now the algorithm proceeds level by level starting from the leaves. Leaves have height 0. We inductively describe round i of the second phase. From the previous rounds the algorithm has assigned a variable to each node in T at height $i-1$. For each node v at level i we have a group of variables which correspond to the children of v. Now the algorithm will find the maximum of each group and assign it to the node at level i associated with the group. It remains to specify how this maximum is found. This is done in the most simple way: For each pair make $c_1\log n$ comparisons for some constant c_1, and choose the variable which comes out greater in the majority of comparisons as greater. If it does not get a total order on the elements in a group then it halts with error. Else, if it gets a total order on the elements of the group then choose the maximum variable in this total order. Choose the maximum variable in this total order. The variable labeling the root is the answer.

Clearly the number of rounds taken by this algorithm is $2\log\log n + \log\log n' < 3\log\log n$. The number of queries in the first phase is $O(n)$, because in round i the number of queries is $(2i+1)n/2^i$. For each round in the second phase, the number of queries is $n'\log n$. The number of rounds in the second phase is $< \log\log n$. So the total number of queries in the second phase is $< n'\log n\log\log n = \frac{n}{(\log n)^2}\log n\log\log n < n$.

By the Chernoff bound the probability that the maximum gets eliminated in the ith round of the first phase is $\approx c_2^i$ for some small constant c_2 which can be chosen to be small by choosing ε to be small. Hence the probability of the maximum getting eliminated in the first round is $\approx \sum_{i\geq 1} c_2^i < 2c_2 \ll 1$. In the second phase by choosing c_1 large enough, again by the Chernoff bound we get that the probability of error for any pair is $\leq 1/n^5$. Since the total number of pairs being compared is $O(n)$, with probability $1-1/n^4$ answer for every pair is correct, and for each group the right total order is found, and the maximum is computed correctly. The algorithm is robust to adversarial noise because it makes decisions based on majority voting.

6 Further Work

There are several results about the query complexity (without restriction on rounds) of some specific boolean functions, eg, MAJORITY and PARITY. This paper gives trade-off results between rounds and queries for OR function. It would be nice to obtain general characterizations for query complexity and trade-off for any given boolean function.

For the noisy comparison tree model, one can ask similar questions, as we studied above for MAX, for SELECTION. Note that answers to these questions are known for the noise-free comparison tree model by Ajtai et al [2] and Reischuk [24] for deterministic and randomized algorithms respectively.

References

[1] M. Ajtai. The invasiveness of off-line memory checking. *STOC* 2002, 504–513.

[2] M. Ajtai, J. Komlos, W .L Steiger, E. Szemeredi. Optimal parallel selection has complexity $O(\log\log n)$. *J. Comput. Syst. Sci.* 38(1): 125–133, 1989.

[3] D. Aharonov, M. Ben-Or. Fault Tolerant Quantum Computation with Constant Error. *STOC* 1997, 176–188.

[4] N. Alon, J. Spencer. The Probabilistic Method. Second Edition. *John Wiley & Sons*, 2000.

[5] H. Buhrman, I. Newman, H. Röhrig, R. de Wolf. Robust quantum algorithms and polynomials. *Quantum Physics abstracts*, number quant-ph/0309220.

[6] A. El Gamal. Open problems presented at the 1984 workshop on Specific Problems in Communication and Computation sponsored by Bell Communication Research.

[7] W. Evans, C. Kenyon, Y. Peres, L. J. Schulman. Broadcasting on trees and the Ising model. Annals of Applied Probability, 10(2), 2000, pp. 410–433.

[8] W. S. Evans, N. Pippenger. Average-Case Lower Bounds for Noisy Boolean Decision Trees. *SIAM J. Comput.*, 28(2): 433–446 (1998).

[9] W. S. Evans., L. J. Schulman. Signal propagation

and noisy circuits. IEEE Transactions on Information Theory, 44(3), May 1998, 1299–1305.

[10] U. Feige. On the Complexity of Finite Random Functions. *Inf. Process. Lett.*, 44(6): 295–296 (1992).

[11] U. Feige, J. Kilian. Finding OR in a noisy broadcast network. *Inf. Process. Lett.* 73(1-2): 69–75 (2000).

[12] U. Feige, P. Raghavan, D. Peleg, E. Upfal. Computing with Noisy Information. *SIAM J. Comput.*, 23(5): 1001–1018 (1994).

[13] P. Gács. Reliable Cellular Automata with Self-Organization. *FOCS* 1997, 90–99.

[14] P. Gács, A. Gál. Lower bounds for the complexity of reliable Boolean circuits with noisy gates. *IEEE Transactions on Information Theory*, Vol. 40, No. 2, 1994, pp. 579–583.

[15] R. G. Gallager. Finding parity in simple broadcast networks. *IEEE Transactions on Information Theory*, Vol. 34, 1988, 176–180.

[16] A. Kalai, R. Servedio. Boosting in the Presence of Noise. *35th Annual Symposium on Theory of Computing (STOC)*, 2003, pp. 196–205.

[17] D. J. Kleitman, F. T. Leighton, Y. Ma. On the Design of Reliable Boolean Circuits That Contain Partially Unreliable Gates. *J. Comput. Syst. Sci.* 55(3): 385–401 (1997)

[18] E. Kushilevitz, Y. Mansour. Computation in Noisy Radio Networks. *SODA* 1998: 236–243.

[19] F. T. Leighton, Y. Ma. Tight Bounds on the Size of Fault-Tolerant Merging and Sorting Networks with Destructive Faults. *SIAM J. Comput.*, 29(1): 258–273 (1999).

[20] E. Mossel. Survey: Information flow on trees. *In Graphs, Morphisms and Statistical Physics. DIMACS series in discrete mathematics and theoretical computer science J. Nestril and P. Winkler editors.* (2004).

[21] I. Newman. Computing in fault tolerance broadcast networks. 19th IEEE Annual Conference on Computational Complexity, 2004, 113–122.

[22] A. Pelc. Searching games with errors—fifty years of coping with liars. *Theoret. Comput. Sci.*, 270 (2002), no. 1-2, 71–109.

[23] N. Pippenger. On the Networks of Noisy Gates. *FOCS* 1985, 30–36.

[24] R. Reischuk. Probabilistic Parallel Algorithms for Sorting and Selection. *SIAM J. Computing*, 14(2): 396–409, 1985.

[25] R. Reischuk, B. Schmeltz. Reliable Computation with Noisy Circuits and Decision Trees—A General $n \log n$ Lower Bound. FOCS 1991: 602–611.

[26] L. Schulman. Coding for Interactive Communication. *IEEE Trans. Information Theory*, 42(6) Part I, 1745–1756, Nov.1996.

[27] Y. Shiloach, U. Vishkin. Finding the maximum, merging and sorting in a parallel computation model. *Journal of Algorithms*, 2(1): 88–102, 1981.

[28] D. A. Spielman. Highly Fault-Tolerant Parallel Computation. *FOCS* 1996, 154–163.

[29] M. Szegedy, X. Chen. Computing Boolean Functions from Multiple Faulty Copies of Input Bits. *LATIN* 2002, 539–553.

[30] L. G. Valiant. Parallelism in comparison problems. *SIAM J. Comp*, 4(3): 348–355, 1975.

[31] A. Yao, Probabilistic computations: Towards a unified measure of complexity, *In Proceedings of the Seventeenth IEEE Conference on Foundations of Computer Science*, 1977, pages 222–227.