Algorithmica (2002) 34: 480–501
DOI: 10.1007/s00453-002-0981-6

Algorithmica

The Quantum Black-Box Complexity of Majority

Thomas P. Hayes,[1] Samuel Kutin,[2] and Dieter van Melkebeek[3]

Abstract. We describe a quantum black-box network computing the majority of N bits with zero-sided error ε using only $\frac{2}{3}N + O(\sqrt{N}\log(\varepsilon^{-1}\log N))$ queries: the algorithm returns the correct answer with probability at least $1-\varepsilon$, and "I don't know" otherwise. Our algorithm is given as a randomized "XOR decision tree" for which the number of queries on any input is strongly concentrated around a value of at most $\frac{2}{3}N$. We provide a nearly matching lower bound of $\frac{2}{3}N - O(\sqrt{N})$ on the expected number of queries on a worst-case input in the randomized XOR decision tree model with zero-sided error $o(1)$. Any classical randomized decision tree computing the majority on N bits with zero-sided error $\frac{1}{2}$ has cost N.

Key Words. Majority function, Quantum computing, Query complexity, Las Vegas algorithms.

1. Introduction. How do you tell how a committee of three people will vote on an issue? The obvious approach is to ask each individual what vote he or she is planning to cast. If the first two committee members agree, you can skip the third one, but, if they disagree, you need to talk to all three members.

Suppose, however, that you can perform quantum transformations on the committee members. This allows you to ask, with one quantum question, whether the first two members agree or disagree. If they agree, you can disregard the third member and ask one of the first two for her vote. If the first two disagree, you know their votes will cancel, so it suffices to ask the third member for his vote. Either way, you will learn the answer in only two queries.

In this paper we discuss generalizations of this procedure to arbitrarily many voters. We allow our algorithms to ask whether two voters agree at the cost of one query. We consider both deterministic and randomized algorithms, allowing different kinds of error. Our algorithms can be simulated very efficiently on quantum machines, yielding new upper bounds for the quantum complexity of the MAJORITY function.

1.1. *Overview.* Suppose we wish to compute the value $f(X)$ of a function f on $\{0,1\}^N$ where the input X is given to us as a *black-box* $X\colon \{0,\ldots,N-1\} \to \{0,1\}$. The cost of the computation will be the number of queries we make to the oracle X. In the classical case, this model of computation is known as a *decision tree*, and has been well studied.

More recently, a *quantum mechanical* version of the model has been considered, which is inherently probabilistic. Several complexity measures are investigated: the number of

[1] Department of Mathematics, University of Chicago, 5734 S. University Avenue, Chicago, IL 60637, USA. hayest@math.uchicago.edu.
[2] Department of Computer Science, University of Chicago, 1100 E. 58th Street, Chicago, IL 60637, USA. kutin@cs.uchicago.edu.
[3] Computer Sciences Department, University of Wisconsin, 1210 W. Dayton Street, Madison, WI 53706, USA. dieter@cs.wisc.edu.

Received March 1, 2001. Communicated by M. Mosca and A. Tapp.
Online publication September 16, 2002.

queries needed to compute f exactly, with zero-sided error ε, or with bounded error ε. Beals et al. [3] show that for any function f these measures are all polynomially related to the classical decision tree complexity. Beals et al. also look more closely at some specific functions f. In particular, they consider the majority function, whose decision tree complexity equals N. They prove that in the quantum model the exact and zero-sided error cost functions are between $N/2$ and N (for any $\varepsilon < 1$); a result by Paturi [9] implies that the bounded error cost function is $\Omega(N)$ (for any constant $\varepsilon < \frac{1}{2}$).

In this paper we investigate these cost measures for MAJORITY more closely. We provide improved upper bounds, as well as matching lower bounds in related models.

Our first result is a quantum black-box network which exactly computes MAJORITY using $N + 1 - w(N)$ queries, where $w(N)$ equals the number of ones in the binary expansion of N. So, for N of the form $2^n - 1$, we can save $\lfloor \log N \rfloor$ queries.

Our algorithm exploits the fact, due to Cleve et al. [5], that the XOR of two input bits can be determined in a single quantum query. In fact, our algorithm can be viewed as an XOR decision tree, i.e., a classical decision tree with the additional power of computing the XOR of two input bits at the cost of a single query. The complexity of MAJORITY in this model has been studied before [11], [1], [2], independently of the connection with quantum computation. A tight bound of $N + 1 - w(N)$ was known [11], [1]. We give a simpler proof for the lower bound which generalizes to the case where computing the parity of arbitrarily many input bits is permitted in one query. The lower bound shows that our procedure cannot be improved without at least introducing a new quantum trick.

Our main result is a quantum black-box network that computes MAJORITY with zero-sided error ε using only $\frac{2}{3}N + O(\sqrt{N \log(\varepsilon^{-1} \log N)})$ queries. For any positive ε we construct such a network. The algorithm can be viewed as a randomized variant of an XOR decision tree given by Alonso et al. [2]. We construct an exact randomized XOR decision tree with an expected number of queries of at most $\frac{2}{3}N + 2 \log N$ on any input. We argue that the number of queries is sufficiently concentrated to yield our main result.

Alonso et al. [2] show that the *average* cost of their algorithm over all N-bit inputs is $\frac{2}{3}N - \Omega(\sqrt{N})$. They also show that the average-case complexity of MAJORITY in the XOR decision tree model is at least $\frac{2}{3}N - O(\sqrt{N})$. We instead are interested in the cost of randomized XOR decision trees on *worst-case* inputs. A standard argument shows that the Alonso et al. lower bound also holds for the expected number of queries on a worst-case input. We also prove that classical randomized decision trees need N queries to compute MAJORITY with zero-sided error $\frac{1}{2}$.

In the general bounded-error setting, van Dam [12] has shown how to compute any function f using $\frac{1}{2}N + \sqrt{N \log \varepsilon^{-1}}$ quantum queries. We point out that van Dam's technique does not provide a zero-sided error network for MAJORITY of cost less than N. We prove that any classical randomized decision tree for MAJORITY has to have cost N to achieve a bounded error of at most $\frac{1}{4}$.

1.2. *Organization*. Section 2 provides some preliminaries, including background on the XOR decision tree model, the quantum black-box model, and their relationship. Section 3 describes and analyzes our quantum network for computing MAJORITY exactly using $N + 1 - w(N)$ queries. In Section 4 we discuss our randomized XOR decision tree for MAJORITY that has small zero-sided error and cost about $\frac{2}{3}N$, and we relate this to the zero-error quantum query complexity. In Section 5 we show that the exact algorithm of Section 3 is optimal in a generalized version of the XOR decision tree model. In Section 6 we discuss lower bounds for the cost of randomized XOR decision trees and classical randomized decision trees for computing MAJORITY. Finally, in Section 7, we give a table summarizing the known results and propose several questions for further research.

2. Preliminaries. We first introduce some general notation. Then we discuss XOR decision trees, quantum black-box networks, and their relationship.

Let $X = X_0 X_1 \cdots X_{N-1}$ be a Boolean string of length N. We often think of X as a function $X\colon \{0, 1, \ldots, N-1\} \to \{0, 1\}$. We define MAJORITY$(X)$ to be zero if X contains more zeros than ones, and one otherwise. This is a weak definition, which we use to establish our lower bounds. Our algorithms always yield a stronger result in that they will answer "tie" when the number of zeros and ones are equal. The *discrepancy* of X is the size of the majority, i.e., the absolute value of the difference in the number of zeros and ones. XOR denotes the exclusive OR of two bits, and PARITY(X) denotes $\sum X_i \bmod 2$.

For a positive integer N, the *Hamming weight* of N, denoted $w(N)$, is the number of ones in the standard binary representation for N. We use the following properties.

LEMMA 1. *For any integer* $N > 0$, $\sum_{k=1}^{\infty} \lfloor N/2^k \rfloor = N - w(N)$.

PROOF. Let $\ell = \lfloor \log N \rfloor$, and write $N = \sum_{j=0}^{\ell} b_j 2^j$, where $b_j \in \{0, 1\}$. We then have

$$\sum_{k=1}^{\infty} \left\lfloor \frac{N}{2^k} \right\rfloor = \sum_{k=1}^{\infty} \sum_{j=k}^{\ell} b_j 2^{j-k} = \sum_{j=1}^{\ell} b_j \sum_{k=1}^{j} 2^{j-k} = \sum_{j=1}^{\ell} b_j (2^j - 1) = \sum_{j=0}^{\ell} b_j 2^j - \sum_{j=0}^{\ell} b_j,$$

which is simply $N - w(N)$. □

COROLLARY 2. *For any integer* $N > 0$, $N!$ *is exactly divisible by* $2^{N-w(N)}$.

PROOF. For any positive integer k, there are exactly $\lfloor N/2^k \rfloor$ multiples of 2^k contributing to $N!$. So the exponent of the largest power of 2 dividing $N!$ is given by $\sum_{k=1}^{\infty} \lfloor N/2^k \rfloor$, which is equal to $N - w(N)$ by Lemma 1. □

2.1. XOR *Decision Trees*. An *XOR decision tree* is an algorithm for a given input length N which adaptively queries the input X and outputs a value. A query may be either

- X_i, where $0 \le i \le N - 1$, or
- $X_i \oplus X_j$, where $0 \le i, j \le N - 1$ and $\oplus$ denotes XOR.

The cost on a given input X is the number of queries made. The cost of an XOR decision tree is the maximum cost over all inputs of length N. An XOR decision tree can be viewed as a binary tree. The depth of this tree equals the cost of the XOR decision tree. We refer to Section 5.1 for a further generalization of XOR decision trees.

We define a *randomized XOR decision tree* $\mathcal{T}$ as an XOR decision tree in which we can toss a coin with arbitrary bias at any point in time, and proceed based on the outcome of the coin toss. Equivalently, we can view $\mathcal{T}$ as a probability distribution over (deterministic) XOR decision trees. The number of queries on a given input X is a random variable. We define the *cost* on input X as the maximum of this random variable, and the cost of $\mathcal{T}$ as the maximum cost over all inputs X.

The following definitions apply to a randomized decision tree $\mathcal{T}$ on N-bit inputs, and more generally to any probabilistic process $\mathcal{T}$ that takes a Boolean string of length N as input and outputs a value. Let f be a function on $\{0, 1\}^N$. If on any input X, $\mathcal{T}$ outputs $f(X)$ with probability at least $1-\varepsilon$, we say that $\mathcal{T}$ computes f with error ε. If $\mathcal{T}$ outputs $f(X)$ with probability at least $1-\varepsilon$ and says "I don't know" otherwise (i.e., $\mathcal{T}$ never produces an incorrect output) we say that $\mathcal{T}$ computes f with *zero-sided error* ε. In the case where $\varepsilon = 0$, we say that $\mathcal{T}$ *exactly* computes f.

A randomized decision tree that exactly computes f at cost C can trivially be transformed into a deterministic tree computing f at the same cost. It can often also be transformed into a randomized XOR decision tree for f with zero-sided error ε and cost $C' < C$, e.g., if on any input the number of queries is strongly concentrated around a value less than C'. More precisely, suppose that on any input X, with probability at least $1-\varepsilon$, $\mathcal{T}$ makes no more than C' queries. Then we can run $\mathcal{T}$, but as soon as we attempt to make more than C' queries, it stops the process and outputs "I don't know." The modified randomized decision tree has zero-sided error at most ε and cost at most C'.

2.2. *Quantum Black-Box Networks*. A quantum computer performs a sequence of unitary transformations $U_1, U_2, \ldots, U_T$ on a complex Hilbert space, called the *state space*. The state space has a canonical orthonormal basis which is indexed by the configurations s of some classical computer M. The basis state corresponding to s is denoted by $|s\rangle$.

The initial state φ_0 is a basis state. At any point in time t, $1 \le t \le T$, the state φ_t is obtained by applying U_t to φ_{t-1}, and can be written as

$$\varphi_t = \sum_s \alpha_{s,t}|s\rangle,$$

where $\sum_s |\alpha_{s,t}|^2 = 1$.

At time T, we *measure* the state φ_T. This is a probabilistic process that produces a basis state, where the probability of obtaining state $|s\rangle$ for any s equals $|\alpha_{s,T}|^2$. The output of the algorithm is the observed state $|s\rangle$ or some part of it.

We define the *quantum black-box model* following Deutsch and Jozsa [7]. In a quantum black-box network $\mathcal{A}$ for input length N, the initial state φ_0 is independent of the input $X = X_0 X_1 \cdots X_{N-1}$. We allow arbitrary unitary transformations independent of X. In addition, we allow $\mathcal{A}$ to make *quantum queries*. This is the transformation U taking the basis state $|i, b, z\rangle$ to $|i, b \oplus X_i, z\rangle$, where:

- i is a binary string of length $\log N$ denoting an index into the input X,
- b is the contents of the location where the result of the oracle query will be placed,
- z is a placeholder for the remainder of the state description,

and comma denotes concatenation.

We define the *cost* of $\mathcal{A}$ to be the number of times the query transformation U is performed; all other transformations are free.

The error notions introduced in Section 2.1 for arbitrary probabilistic processes also apply to quantum black-box networks.

2.3. *From* XOR *Decision Trees to Quantum Black-Box Networks*. Bernstein and Vazirani have shown [4] that a quantum computer can efficiently simulate classical deterministic and probabilistic computations. It is also known that we can efficiently compose quantum algorithms. In terms of quantum black-box networks these results imply that a classical randomized decision tree $\mathcal{T}$ that uses quantum black-box networks as subroutines can be efficiently simulated by a single quantum black-box network. The cost of the simulation will be the sum of the cost of T and the costs of the subroutines. Similarly, the error of the simulation will be bounded by the sum of the error of $\mathcal{T}$ and the errors of the subroutines. The simulation will have zero-sided error if all of the components do.

We describe our quantum black-box networks for MAJORITY as classical randomized decision trees that use the following exact quantum black-box network developed by Cleve et al. [5] for computing the XOR of two input bits.

LEMMA 3 [5]. *There exists a quantum black-box network of unit cost that on input two bits X_0 and X_1 exactly computes their XOR.*

The above argument shows that an XOR decision tree for a function f can be transformed into a quantum black-box network for f of the same cost. The transformation works in the exact setting, as well as for zero-sided or arbitrary error ε.

3. Computing MAJORITY Exactly.

In the Introduction we discussed how to use an XOR query to determine the MAJORITY of three input bits. In this section we generalize this idea to an input of arbitrary length. We first describe a general approach for constructing XOR decision trees or exact randomized XOR decision trees for MAJORITY. We call it the "homogeneous block approach." We use this approach to develop the "oblivious-pairing" algorithm, an XOR decision tree that computes MAJORITY exactly on N-bit inputs using at most $N + 1 - w(N)$ queries. In Section 5 we show that this is optimal.

The oblivious-pairing algorithm was first introduced and analyzed by Saks and Werman [11]. It forms a first step towards the zero-sided error randomized XOR decision tree for MAJORITY which we develop in Section 4.

3.1. *The Homogeneous Block Approach*. XOR queries allow us to compare bits of the input X. If the bits differ in value, we can discard them since the two of them together will not affect the majority value. If the bits have the same value, we can combine them into a homogeneous block of size 2, i.e., a subset of two input bits which we know have the same value but we do not know what that value is. More generally, we can apply the following operation "COMBINE" to two disjoint nonempty homogeneous blocks R and S. Suppose that $|R| \geq |S|$. We compare a bit from R with a bit from S. If the bits differ, we discard block S completely together with $|S|$ bits from block R. Otherwise, we combine blocks R and S into a single homogeneous block of size $|R| + |S|$.

In the homogeneous block approach, we keep track of a collection of disjoint nonempty homogeneous blocks with the property that the majority of the bits in the union of the blocks equals MAJORITY(X). We start out with the partition of the input into blocks of size 1, i.e., individual bits. Then we use some criterion to decide to which two blocks we apply the operation COMBINE. We keep doing so until we end up in a configuration consisting of an empty collection or one in which one of the blocks is larger than the union of all other blocks. In the former case, we have a tie. In the latter, the largest block determines the majority, and querying any of its bits gives us the value of the majority. One of these situations will eventually be reached since the number of blocks goes down by one or two in each step.

Building a homogeneous block of size k requires only $k - 1$ comparisons between the bits in the block. In general, the number of comparisons performed upon reaching a configuration consisting of ℓ homogeneous blocks equals $N - \ell - c$, where c denotes the number of times two blocks cancelled each other out completely. It follows that, compared with the trivial procedure of querying every input bit, the homogeneous block approach saves one query for every block in the final configuration except the dominating block, and one for every cancellation of equal-sized blocks.

3.2. *The Oblivious-Pairing Algorithm.* In the oblivious-pairing algorithm we first build homogeneous blocks of size 2 by pairing up the initial blocks of size 1, leaving the last block of size 1 untouched when N is odd. Then we build blocks of size 4 out of the blocks of size 2, possibly leaving the last block of size 2 untouched, etc. In general, during the kth phase of the algorithm, we pairwise COMBINE the homogeneous blocks of size 2^{k-1} to either cancel or form homogeneous blocks of size 2^k. There will be at most one block of size 2^{k-1} left after the end of the kth phase.

There can be at most $\lfloor \log N \rfloor$ phases. Afterwards, either there are no blocks left, in which case we have a "tie," or else all remaining blocks have sizes that are different powers of 2. The largest block then dominates all the others combined and dictates the majority.

We provide the pseudo-code for the oblivious-pairing algorithm in Figure 1. We keep track of the collection of disjoint nonempty homogeneous blocks as a list $\mathcal{S} \doteq (S_j)_{j=1}^{\ell}$ of subsets of $\{0, 1, \ldots, N-1\}$ of nonincreasing size. We always compare two consecutive blocks in the list, say S_i and S_{i+1}, a procedure captured by the subroutine COMBINE. We also use the following notation: if X is homogeneous on a subset S of $\{0, 1, \ldots, N-1\}$, we write X_S for the value of any bit X_i, $i \in S$.

For any positive integer k, the blocks of size 2^{k-1} are pairwise disjoint. We pair them up during the kth phase of the algorithm. It follows that the number of COMBINE operations during the kth phase is bounded from above by $\lfloor N/2^k \rfloor$. Each application of COMBINE involves one XOR. Therefore, Lemma 1 gives us an upper bound of $N - w(N)$ on the total number of XORs. There can be at most one more query, for a total of $N + 1 - w(N)$. This total is reached, e.g., for homogeneous inputs (all zeros or all ones). There are no cancellations on homogeneous inputs, and $w(N)$ is the smallest number of power-of-2 blocks that add up to N. We conclude:

THEOREM 4 [11]. *The oblivious-pairing algorithm for* MAJORITY *on N-bit inputs has* XOR *decision tree cost $N + 1 - w(N)$.*

Input: $X \doteq (X_i)_{i=0}^{N-1} \in \{0, 1\}^N$
Output: MAJORITY(X)
Notation: $\ell \doteq |\mathcal{S}|$
$S_j \doteq j$th element of $\mathcal{S}$, $1 \leq j \leq \ell$
$X_{S_j} \doteq X_i$ for any $i \in S_j$, $1 \leq j \leq \ell$
Subroutine: COMBINE$(\mathcal{S}, i, X)$
if XOR$(X_{S_i}, X_{S_{i+1}}) = 0$
then replace S_i, S_{i+1} in $\mathcal{S}$ by $S_i \cup S_{i+1}$
else remove S_i, S_{i+1} from $\mathcal{S}$
Algorithm:
$\mathcal{S} \leftarrow (\{i\})_{i=0}^{N-1}$
for $k = 1, 2, \ldots, \lfloor \log N \rfloor$
while $I \doteq \{j \mid 1 \leq j < \ell \text{ and } |S_j| = |S_{j+1}| = 2^{k-1}\} \neq \emptyset$
$i \leftarrow \min I$
COMBINE$(\mathcal{S}, i, X)$
if $\ell = 0$
then return "tie"
else return X_{S_1}

Fig. 1. The oblivious-pairing algorithm.

COROLLARY 5. *We can compute* MAJORITY *exactly on N-bit inputs using at most $N + 1 - w(N)$ quantum black-box queries.*

4. Computing MAJORITY **with Zero-Sided Error.** In Section 3 we considered the oblivious-pairing XOR decision tree. We showed that it has a cost of $N - w(N) + 1$. We now consider exact randomized XOR decision trees for MAJORITY. Our main result is the randomized greedy-pairing algorithm, for which the number of queries on any input is highly concentrated around a value of about $\frac{2}{3}N$ on a worst-case input. Using the techniques discussed in Sections 2.1 and 2.3, this gives us a randomized XOR decision tree and a quantum black-box network with small zero-sided error of cost about $\frac{2}{3}N$. In Section 6 we give a nearly matching lower bound on the expected number of queries on a worst-case input for randomized XOR decision trees with small zero-sided error.

In Section 4.1 we discuss a simple randomized version of the oblivious-pairing algorithm. We carefully analyze the number of queries it makes, as we will need that result later. In Section 4.2 we describe a deterministic algorithm of Alonso et al. [2], the greedy-pairing algorithm, for which the average number of queries over all N-bit inputs is roughly $\frac{2}{3}N$. In Section 4.3 we analyze a randomized version of the greedy-pairing algorithm. We prove that the number of queries it makes is with high probability not much larger than $\frac{2}{3}N$.

4.1. *The Randomized Oblivious-Pairing Algorithm.* The oblivious-pairing algorithm is efficient when we can get pairs of blocks to cancel. Recall that the number of XORs made in any homogeneous block algorithm for MAJORITY equals $N - \ell - c$, where ℓ denotes the number of blocks at the end, and c is the number of cancellations of equal-sized blocks that occurred. In the oblivious-pairing algorithm, ℓ can be at most $\log N$, so not much savings can be expected from that term. The number of cancellations can be much larger. On the input 010101 . . ., all $N/2$ pairs of individual bits cancel, and we can declare a tie with only $N/2$ queries. However, even if we know the input is perfectly balanced, there is no guarantee that any cancellations occur until the very end.

One natural approach is to permute the input bits randomly before we begin the algorithm: Choose some permutation π of $\{0, 1, \ldots, N-1\}$ uniformly at random, let $X'_i = X_{\pi(i)}$, and run the oblivious-pairing algorithm on the input X'. The distribution of the number of queries on a given input now only depends on the number of ones and the number of zeros it contains.

Consider the randomized oblivious-pairing algorithm running on a perfectly balanced input of length N. We perform $N/2$ queries comparing individual bits; we expect roughly half of those to cancel, and half to yield homogeneous blocks of size 2. We next pair up the $N/4$ blocks of size 2, which takes $N/8$ queries. Again, we expect roughly half of those queries to cancel, and half to yield blocks of size 4. The overall number of queries should then be about

$$\frac{N}{2} + \frac{N}{8} + \frac{N}{32} + \cdots = \frac{2}{3}N.$$

We prove below that the number of queries the oblivious-pairing algorithm makes on a balanced input is indeed highly concentrated around $\frac{2}{3}N$.

However, consider a homogeneous input. Permuting the input bits has no effect; the input remains homogeneous, blocks will never cancel, and the randomized oblivious-pairing algorithm still takes $N - w(N) + 1$ queries. We need to do something else to reduce the computation cost on such inputs. We return to this question in Section 4.2.

Before doing so, we prove the following theorem about the number of comparisons the oblivious-pairing algorithm makes on input X. We use the theorem in our analysis of our main result in Section 4.3.

THEOREM 6. *There exists a constant d such that the following holds. Let $C_{\mathrm{OP}}(X)$ denote the number of comparisons the oblivious-pairing algorithm makes on input X. Let $N > 0$, and let $A + B = N$, $A, B \geq 0$. Let X be chosen uniformly at random from all strings of A ones and B zeros. Then for any $r \geq 1$,*

$$\Pr_X[C_{\mathrm{OP}}(X) \geq N - \tfrac{2}{3}\min(A, B) + d\sqrt{rN}] \leq 2^{-r}\log N.$$

The proof of Theorem 6 uses the following tail law.

LEMMA 7. *There exists a constant d' such that the following holds. Let $c(X)$ denote the number of cancellations during the first phase of the oblivious-pairing algorithm on* *input X. Let $N > 0$, and let $A + B = N$, $A, B \geq 0$. Let X be chosen uniformly at random from all strings of A ones and B zeros. Then for every $r \geq 1$,*

$$\Pr_X\left[\left|c(X) - \frac{AB}{N}\right| \geq d'\sqrt{rN}\right] \leq 2^{-r}.$$

The combinatorial problem underlying Lemma 7 is a special case of "Levene's matching problem" [6], and has been well studied. We suspect that the tail law given in Lemma 7 is known but have not been able to find a reference. We include a proof sketch in the Appendix.

PROOF OF THEOREM 6. The proof goes by induction on N. We first do the induction step.

Assume without loss of generality that $A \geq B$. Look at the sequence of homogeneous blocks of size 2 after the first phase of oblivious-pairing on input X. Let X' denote the input obtained by replacing each block in this sequence by a single bit of the same value. We have that $C_{\mathrm{OP}}(X) = \lfloor N/2 \rfloor + C_{\mathrm{OP}}(X')$.

Let A' denote the number of ones in X', B' the number of zeros, and $N' = A' + B'$. Note that $N' = \lfloor N/2 \rfloor - c(X)$, $B' = \lfloor (B - c(X))/2 \rfloor$, and $A' \geq B'$.

Conditioned on A' and B', the distribution of X' is uniform. Therefore, by our induction hypothesis, we have that with probability at least $1 - 2^{-r} \log N'$,

$$\begin{aligned}
C_{\mathrm{OP}}(X') &\leq N' + \tfrac{2}{3}B' + d\sqrt{rN'} \\
&= \left\lfloor \frac{N}{2} \right\rfloor - c(X) - \frac{2}{3}\left\lfloor \frac{B - c(X)}{2} \right\rfloor + d\sqrt{rN'} \\
&\leq \frac{N}{2} - \frac{B}{3} - \frac{2}{3}c(X) + d\sqrt{rN'} + \tfrac{1}{3}.
\end{aligned}$$

By Lemma 7, with probability at least $1 - 2^{-r}$,

$$c(X) \geq \frac{AB}{N} - d'\sqrt{rN} \geq \frac{B}{2} - d'\sqrt{rN}.$$

Taking everything together, and using that fact that $N' \leq N/2$, we have that with probability at least $1 - 2^{-r}\log N' - 2^{-r} \geq 1 - 2^{-r}\log N$,

$$\begin{aligned}
C_{\mathrm{OP}}(X) &\leq N - \tfrac{2}{3}B + \left(\frac{2d'}{3} + \frac{d}{\sqrt{2}}\right)\sqrt{rN} + \tfrac{1}{3} \\
&\leq N - \tfrac{2}{3}B + d\sqrt{rN},
\end{aligned}$$

provided d is large enough that $(2d' + 1)/3 \leq (1 - 1/\sqrt{2})d$. This proves the induction step.

By picking d larger as needed, we can take care of the base cases. □

Theorem 6 can be strengthened to show that the random variable $C_{\mathrm{OP}}(X)$ is strongly concentrated around a value slightly smaller than $N - \frac{2}{3}\min(A, B)$. We omit the precise expression for the concentration point, as it is rather cumbersome and is not needed for what follows. A proof similar to the above (but simpler and not relying on Lemma 7) shows that the expected value of $C_{\text{OP}}(X)$ in Theorem 6 is bounded above by $N - \frac{2}{3}\min(A, B)$.

4.2. *The Greedy-Pairing Algorithm.* As we mentioned in Section 4.1, the oblivious-pairing algorithm requires $N - w(N) + 1$ queries on the all ones input, whether or not we randomize. In contrast, the trivial algorithm for MAJORITY, which simply queries bits until the observed discrepancy is larger than the number of bits remaining, takes $\lfloor N/2 \rfloor + 1$ queries on the all ones input. Therefore, we should be able to improve the oblivious-pairing algorithm.

The oblivious-pairing algorithm always COMBINEs two smallest blocks of equal size. A first idea is that we may always decide to COMBINE two *largest* blocks of equal size instead, and stop as soon as the largest block (if any) is larger than the union of the other blocks. This leads to an improvement on some inputs, e.g., on homogeneous inputs of length $N = 2^k - 1$: we build up a block of size 2^{k-1} using $\lfloor N/2 \rfloor$ XORs and query one bit in that block, for a total cost of $\lfloor N/2 \rfloor + 1$. However, on homogeneous inputs of length $N = 2^k + 1$, we still make $N - 1$ queries: we construct a block S_1 of size 2^{k-1}, and then perform another $2^{k-1} - 1$ queries to form another large block, even though one additional query combining S_1 with another bit would guarantee a majority.

In order to do better, we should allow COMBINE operations on blocks of unequal size. As cancellations of blocks of equal size are beneficial, we still prefer to COMBINE such blocks, but we should only do so if we reasonably expect the answer to be useful. Alonso et al. [2] introduce a homogeneous block algorithm for MAJORITY which does just this: they COMBINE two blocks only if they are sure they will need to know the answer. We call this the "greedy-pairing" algorithm.

More precisely, the greedy-pairing algorithm works as follows. Suppose that in some step we find a pair S_i, S_{i+1} of large blocks of equal size. Instead of automatically combining these two blocks, however, we now ask a question: Are we sure this is necessary? In other words, if we assumed all blocks up to i all agreed, would that still not be enough to determine a majority? If the answer is yes, we COMBINE the two blocks. If the answer is no, then we try to build up the largest block by running COMBINE on S_1 and S_2.

When we compare two blocks of the same size, we are trying to gain by cancelling and reducing ℓ by two in a single step. When we compare two blocks of different sizes, we are trying to gain by greedily constructing a large enough block to guarantee a majority.

Since the only COMBINE operations between blocks of unequal size involve S_1, all blocks except possibly S_1 will have sizes that are powers of 2. Say $|S_j| = 2^{s_j}$, $2 \leq j \leq \ell \doteq |\mathcal{S}|$, where the s_j's are integers. The size of S_1 can be written as $|S_1| = (2m+1)2^{s_1}$ for some integers m and s_1. Note that $s_1 \geq s_2 \geq \cdots \geq s_\ell$. We think of S_1 as being composed of several power-of-2 blocks. The smallest such subblock has size 2^{s_1}.

The precise criterion we use to determine which blocks S_i and S_{i+1} to compare is given in the pseudo-code of Figure 2. Note that the smallest j such that $s_j = s_{j+1}$ exists during each execution of the while loop. If there were no such j, the block S_1 would dominate all the other blocks combined and we would have exited the loop.

Input: $X \doteq (X_i)_{i=0}^{N-1} \in \{0,1\}^N$
Output: MAJORITY(X)
Notation: $\ell \doteq |\mathcal{S}|$
$S_j \doteq j$th element of $\mathcal{S}$, $1 \le j \le \ell$
$X_{S_j} \doteq X_i$ for any $i \in S_j$, $1 \le j \le \ell$
$s_1 \doteq$ largest integer t such that 2^t divides $|S_1|$
$s_j \doteq |S_j|$, $2 \le j \le \ell$
Subroutine: COMBINE$(\mathcal{S}, i, X)$
if XOR$(X_{S_i}, X_{S_{i+1}}) = 0$
then replace S_i, S_{i+1} in $\mathcal{S}$ by $S_i \cup S_{i+1}$
else if $|S_i| > |S_{i+1}|$
then remove $|S_{i+1}|$ elements from S_i
remove S_{i+1} from $\mathcal{S}$
else remove S_i, S_{i+1} from $\mathcal{S}$
Algorithm:
$\mathcal{S} \leftarrow (\{i\})_{i=0}^{N-1}$
while $\ell > 0$ and $|S_1| \le \sum_{j=2}^{\ell} |S_j|$
$i \leftarrow$ smallest integer j such that $s_j = s_{j+1}$
if $\sum_{j=1}^{i} |S_j| > \sum_{j=i+1}^{\ell} |S_j|$
then $i \leftarrow 1$
COMBINE$(\mathcal{S}, i, X)$
if $\ell = 0$
then return "tie"
else return X_{S_1}

Fig. 2. The greedy-pairing algorithm.

The key to the good performance of the greedy-pairing algorithm is the following observation. Let M denote the index of the $(\lfloor N/2 \rfloor + 1)$st input bit agreeing with the majority. If X is balanced, let $M \doteq N$. Let Y denote the substring consisting of the first M bits of X, and Z the remainder of X. Then the greedy-pairing algorithm never performs any comparisons involving bits of Z. This is because Y forces the majority in all of X, and the greedy-pairing algorithm only involves a new bit b in a comparison if the bits before b cannot force the majority of X.

This is the way the greedy-pairing algorithm saves queries compared with the oblivious-pairing algorithm: by not making the comparisons the oblivious-pairing algorithm makes involving bits of Z. On Y, the greedy-pairing algorithm makes some of the comparisons the oblivious-pairing algorithm makes, but possibly also makes some others. We need to show that there are not too many other queries, or at least that we can account for most of them by queries the oblivious-pairing algorithm makes on Y but the greedy-pairing algorithm does not. We prove next that there are at most $O(\log^2 N)$ queries that we cannot account for in that way.

THEOREM 8. *Let $C_{\mathrm{GP}}(X)$ denote the number of comparisons the greedy-pairing algorithm makes on input X, and let C_{OP} be defined as in Theorem* 6. *There exists a constant d such that on any binary input X of length N,*

$$C_{\mathrm{GP}}(X) \leq C_{\mathrm{OP}}(Y) + d\log^2 N,$$

where Y denotes the first M bits of X and M the position of the $(\lfloor N/2 \rfloor + 1)$st *bit in X agreeing with the majority. When X is balanced, $M \doteq N$ and $Y \doteq X$.*

In fact, a refinement of the argument below shows that

$$C_{\mathrm{OP}}(Y) \leq C_{\mathrm{GP}}(X) \leq C_{\mathrm{OP}}(Y) + \max(2\lfloor \log N \rfloor - 3, 0),$$

which is tight. However, the relationship as stated in Theorem 8 is strong enough for our purposes.

In order to prove Theorem 8, we need the following properties of the greedy-pairing algorithm. They deal with the technical concept of an "unusual comparison," which is a comparison between S_1 and S_2 with $s_1 \neq s_2$. These are precisely the comparisons between blocks of different sizes, provided we view a comparison with S_1 as one with the last subblock of S_1 of size 2^{s_1}.

LEMMA 9. *Consider running the greedy-pairing algorithm on an input X and call a comparison unusual if it is between S_1 and S_2, and $s_1 \neq s_2$. Let s be an integer. Let T be the first point in time there is an unusual comparison with $s_2 \leq s$. (If there is no such comparison, we let T denote the end of the algorithm.) Then the following hold*:

1. *All comparisons the greedy-pairing algorithm makes before T with $|S_{i+1}| \leq 2^s$ are also made by the oblivious-pairing algorithm on input X.*
2. *After T, the greedy-pairing algorithm makes no comparisons with $|S_{i+1}| \geq 2^s$ and $i > 1$, and none with $|S_{i+1}| > 2^s$ and $i = 1$.*
3. *Let B_j denote the jth block S_2 of size 2^s which the greedy-pairing algorithm compares with S_1 at and after T. Then the sequence $B_1, B_2, \ldots, B_r$ is one of successive blocks of size 2^s produced by the oblivious-pairing algorithm on input X.*
4. *The outcome of each of the greedy-pairing comparisons referred to in* 3 *is "unequal" for $S_2 = B_j$, $1 \leq j < r$.*

PROOF OF LEMMA 9. We prove claim 1 by contradiction. Suppose that, at some time before T, the greedy algorithm makes a comparison with $|S_{i+1}| = 2^u$, where $u \leq s$, which is not made by the oblivious-pairing algorithm. Consider the first such time U. Since $U < T$, the comparison at time U is not unusual. Since no unusual comparisons with $|S_{i+1}| \leq 2^u$ have occurred, we must have $|S_i| = |S_{i+1}|$; and, by our choice of U, both S_i and S_{i+1} are also formed by the oblivious-pairing algorithm.

By our choice of U, any earlier blocks of size 2^u must have been compared as they are in the oblivious-pairing algorithm. In particular, there must be an even number of them. So, blocks S_i and S_{i+1} must be the jth and $(j+1)$st blocks of size 2^u formed by the oblivious-pairing algorithm for some odd j. Hence, this comparison is also made by the oblivious-pairing algorithm, contradicting our choice of U.

We now consider claim 2. Clearly, at time T, only $|S_1|$ can have size larger than 2^s, and, if $s_2 > 0$, $|S_i| < |S_2|$ for $i > 2$. (Proof: if $|S_2| = |S_3|$, then S_3 must have been formed at some time $U < T$; since the algorithm did not do an unusual comparison at time U, it would have chosen to compare S_2 and S_3 at time $U + 1$.)

At time T, let j be the smallest index such that $|S_j| = |S_{j+1}|$. Then we must have $\sum_{k=1}^{j} |S_k| > \frac{1}{2} \sum_{k=1}^{\ell} |S_k|$. Since all blocks up to S_j have different sizes, and $|S_2| \leq 2^s$, we conclude that, at time T,

$$|S_1| + 2^{s+1} - 1 > \frac{1}{2} \sum_{k=1}^{\ell} |S_k|. \tag{1}$$

Once inequality (1) holds, it remains true for the remainder of the algorithm. (No comparison can increase the right-hand side. The left side is decreased only by an "unequal" comparison between S_1 and S_2, in which case both sides decrease by $|S_2|$.)

So, suppose that, at some later time, there is a block of size 2^s, and, if a comparison were done between some S_i and S_{i+1}, it would form a second such block. By (1), the greedy-pairing algorithm would choose to do a comparison between S_1 and S_2 instead. Hence, from time T onward, there can be at most one block S_i of size 2^s for $i > 1$, which proves claim 2.

To prove claim 3, we let U be the first time (if any) that there is an unusual comparison with $s_2 < s$. (If $s_2 < s$ at time T, then $U = T$ and $r = 0$.) By claim 2, all blocks B_j are formed before time U. So, by claim 1 applied to $s - 1$, the comparisons which form those blocks are all performed by the oblivious-pairing algorithm.

Finally, by the above reasoning, at the time that S_1 is compared with B_j, there is no other block of size 2^s. If the comparison were "equal," then the next comparison would be unusual as well, with $|S_2| < 2^s$, and no additional blocks B_j would form. We conclude that, for each $j < r$, the comparison between S_1 and B_j is "unequal." □

Using Lemma 9, we can prove Theorem 8 as follows.

PROOF OF THEOREM 8. Fix a nonnegative integer s and look at the comparisons the greedy-pairing algorithm makes on input X with $|S_{i+1}| = 2^s$. Let T be as defined in Lemma 9.

By claim 1 of Lemma 9, all such comparisons before T are also made by the oblivious-pairing algorithm on input X at some point in time. As the greedy-pairing algorithm only involves bits of Y in comparisons, these comparisons are actually made by the oblivious-pairing algorithm on input Y.

By claims 2 and 3 of Lemma 9, there are r more comparisons the greedy-pairing algorithm makes at and after time T with $|S_{i+1}| = 2^s$. With these comparisons, we can associate the comparisons the oblivious-pairing algorithm makes involving the blocks $B_1, B_2, \ldots, B_r$ and their superblocks. By claim 3, $B_1, B_2, \ldots, B_{r-1}$ are subsequent blocks the oblivious-pairing algorithm produces during phase s. By claim 4, all of them have the same value. The oblivious-pairing algorithm will spend at least $r-1-\lceil\log(r-1)\rceil$ comparisons on combining the blocks $B_1, \ldots, B_{r-1}$. So, the greedy-pairing algorithm makes at most $r-(r-1-\lceil\log(r-1)\rceil) \leq 2+\log N$ queries with $|S_{i+1}| = 2^s$ which we cannot account for by queries the oblivious-pairing algorithm makes on Y. Adding this surplus over all values of s we get that

$$C_{\mathrm{GP}}(X) \leq C_{\mathrm{OP}}(Y) + (2+\log N)\log N,$$

which establishes the upper bound. □

We point out that Alonso et al. [2] showed that the average case complexity of the greedy-pairing algorithm is optimal up to an $O(\log N)$ term. In particular, they established the following upper bound.

THEOREM 10 [2]. *The average number of comparisons made by the greedy-pairing algorithm over all N-bit inputs equals*

$$\frac{2N}{3} - \sqrt{\frac{8N}{9\pi}} + O(\log N).$$

The term $\sqrt{8N/9\pi}$ comes from the average discrepancy over all N-bit inputs, which is $\sqrt{2N/\pi} + O(1)$.

However, the analysis by Alonso et al. is not sufficient for our purposes. We need an algorithm which performs well on the worst-case input. It is with this goal in mind that we now study a randomized version of the greedy-pairing algorithm.

4.3. *The Randomized Greedy-Pairing Algorithm*. In Section 4.1 we randomized the oblivious-pairing algorithm by first applying a random permutation π to the input bits. We can use the same technique to randomize the greedy-pairing algorithm. This is the algorithm which leads to our main result.

The following analysis is essential.

THEOREM 11. *There exists a constant d such that the following holds. Let $C_{\mathrm{GP}}(X)$ denote the number of comparisons the greedy-pairing algorithm makes on input X. Let $N > 0$, and let $A + B = N$, $A, B \geq 0$. Let X be chosen uniformly at random from all strings of A ones and B zeros. Then for every $r \geq 1$,*

$$\Pr_X[C_{\mathrm{GP}}(X) \geq \tfrac{1}{2}N + \tfrac{1}{3}\min(A,B) + d\sqrt{rN}] \leq 2^{-r}\log N.$$

Theorem 11 shows that the worst-case inputs for the randomized greedy-pairing algorithm are the balanced ones. We conclude:

COROLLARY 12. *There exists a constant d such that for any positive ε and any binary string X of length N, with probability at least $1-\varepsilon$ the randomized greedy-pairing algorithm makes no more than $\frac{2}{3}N + d\sqrt{N\log(\varepsilon^{-1}\log N)}$ comparisons on input X.*

As with Theorem 6, we make use of a concentration result in the proof of Theorem 11. Here, as there, we believe this result is already known, but have not found a reference. A proof sketch is in the Appendix.

LEMMA 13. *There exists a constant d such that the following holds. Let $N = A+B > 0$, where $A \geq B \geq 0$. Let X be chosen uniformly at random from all strings of A ones and B zeros. Let M denote the index of the* $(\lfloor N/2 \rfloor + 1)$st *one of X. If $A = B$, let $M \doteq N$. Then for every $r \geq 1$,*

$$\Pr_X \left[\left| M - \frac{N^2}{2A} \right| > d\sqrt{rN} \right] \leq 2^{-r}.$$

PROOF OF THEOREM 11. Let Y be the string consisting of the first M bits of X, where, as before, M is the position of the $(\lfloor N/2 \rfloor + 1)$st bit in X agreeing with the majority. When X is balanced, $M \doteq N$ and $Y \doteq X$. By Theorem 8,

$$C_{\text{GP}}(X) \leq C_{\text{OP}}(Y) + O(\log^2 N).$$

If X is exactly balanced, then $Y = X$ is uniformly distributed among strings having $N/2$ ones and $N/2$ zeros. In this case, we have reduced to Theorem 6.

Suppose X is not exactly balanced. Without loss of generality, let $A > B$. Y has exactly $(\lfloor N/2 \rfloor + 1)$ ones and $M - (\lfloor N/2 \rfloor + 1)$ zeros, the Mth bit being a one. Let Y' be the string of length $M - 1$ obtained by dropping the last one from Y.

Conditioned on M being fixed, Y' is uniformly distributed among strings having $\lfloor N/2 \rfloor$ ones and $M - 1 - \lfloor N/2 \rfloor$ zeros. Hence Theorem 6 applied to Y' yields

$$C_{\text{OP}}(Y') \leq M - \frac{2}{3}\left(M - \left\lfloor \frac{N}{2} \right\rfloor\right) + d\sqrt{rM} \leq \frac{M+N}{3} + d\sqrt{rN}$$

with probability at least $2^{-r} \log N$.

Since Y differs from Y' only in the rightmost bit, oblivious pairing does all the same comparisons on Y as on Y', plus at most one additional comparison per phase. Hence

$$C_{\text{OP}}(Y) \leq C_{\text{OP}}(Y') + \log N.$$

Putting this together,

$$C_{\text{GP}}(X) \leq \frac{M+N}{3} + d\sqrt{rN} + O(\log^2 N)$$

with probability at least $1 - 2^{-r} \log N$. Since $M \leq N$, this is already enough to establish Corollary 12.

By Lemma 13, $M \leq N^2/2A + d'\sqrt{rN}$ with probability at least $1 - 2^{-r}$. Hence, with probability at least $1 - 2^{-r}(1 + \log N)$,

$$\begin{aligned} C_{\text{GP}}(X) &\leq \frac{N^2 + 2AN}{6A} + (d + d')\sqrt{rN} + O(\log^2 N) \\ &= \frac{3AN + BN}{6A} + d''\sqrt{rN} \end{aligned}$$

$$\leq \frac{3AN + 2AB}{6A} + d''\sqrt{rN}$$
$$= \frac{N}{2} + \frac{B}{3} + d''\sqrt{rN}. \qquad \square$$

Theorem 11 can be strengthened to show that the random variable $C_{\mathrm{GP}}(X)$ is strongly concentrated around a value slightly smaller than $N/2 - \min(A, B)/3$. We omit the precise expression for the concentration point, as it is rather cumbersome and not needed for what follows.

A simplified version of the proof of Theorem 11 shows that the expected number of comparisons the randomized greedy pairing algorithm makes on an N-bit input with A ones and B zeros is bounded above by

$$\tfrac{1}{2}N + \tfrac{1}{3}\min(A, B) + 2\log N = \tfrac{2}{3}N - \tfrac{1}{6}D + 2\log N,$$

where D denotes the discrepancy of the input. This gives us a bound of the form $\frac{2}{3}N - O(\sqrt{N})$ on the average-case cost of the (randomized) greedy-pairing algorithm. However, the constant hidden in the $O(\sqrt{N})$ term is not as good as that achieved by Alonso et al. [2] in Theorem 10.

Using the techniques from Section 2.3, Corollary 12 yields our main result.

THEOREM 14 (Main Result). *There exists a constant d such that, for any positive integer N and any $\varepsilon > 0$, there exists a quantum black-box network of cost*

$$\tfrac{2}{3}N + d\sqrt{N\log(\varepsilon^{-1}\log N)}$$

that computes the majority of N bits with zero-sided error ε.

5. Lower Bounds for Computing MAJORITY **Exactly.** Beals et al. [3] establish a lower bound of $N/2$ quantum queries for computing MAJORITY exactly. In this section we show that any XOR decision tree computing MAJORITY must use at least $N + 1 - w(N)$. Hence, the oblivious-pairing algorithm of Section 3 is optimal.

We first define a more general model of computation, a decision tree relative to a set of functions. We then show that, relative to the collection of all parity functions, the oblivious-pairing algorithm is the best possible.

Recall that the classical decision tree complexity of MAJORITY equals N.

5.1. *Relative Decision Tree Complexity.* A decision tree relative to a class of functions $\mathcal{G}$ is one which is permitted to apply any function from $\mathcal{G}$ to a subset of the input bits (taken in any order) at unit cost.

DEFINITION 15 ($\mathcal{G}$-Decision Tree). Let $\mathcal{G} = \{g_1, g_2, \ldots\}$ be a collection of functions where g_k is a function on M_k bits. A *$\mathcal{G}$-decision tree* is a deterministic algorithm for a given input length N which can query its input bits $X_0, \ldots, X_{N-1}$, and which can also perform queries of the form $g_k(X_{\sigma(0)}, \ldots, X_{\sigma(M_k-1)})$, where σ is a one-to-one function from $\{0, \ldots, M_k - 1\}$ to $\{0, \ldots, N - 1\}$. The cost of a $\mathcal{G}$-decision tree is the maximum over all N-bit inputs of the total number of queries performed on that input, including individual input bits as well as functions g_k.

DEFINITION 16 ($\mathcal{G}$-Decision Tree Complexity). Let f be a function on $\{0, 1\}^N$. The *$\mathcal{G}$-decision tree complexity of f*, denoted $D^{\mathcal{G}}(f)$, is the minimum cost of a $\mathcal{G}$-decision tree computing f. When $\mathcal{G} = \{g\}$, we write this simply as $D^g(f)$.

We consider two instances, namely $\mathcal{G} = \{\text{XOR}\}$ and $\mathcal{G} = \mathcal{PARITY}$, where $\mathcal{PARITY}$ denotes the collections of all PARITY functions (on any number of bits).

We trivially have that $D^{\mathcal{PARITY}}(f) \leq D^{\text{XOR}}(f)$ for any function f. The discussion in Section 2.3 shows that there exists a quantum black-box network that computes f exactly with cost at most $D^{\text{XOR}}(f)$.

The following lemma establishes a limit on how much we can expect $\mathcal{PARITY}$ to help simplify the computation of a function f. It is an extension of a result of Rivest and Vuillemin [10] for standard decision trees.

LEMMA 17. *Let f be a Boolean function on $\{0, 1\}^N$. If $D^{\mathcal{PARITY}}(f) \leq d$, then 2^{N-d} divides $|f^{-1}(1)|$.*

PROOF. Each leaf of the decision tree corresponds to a set of inputs: those inputs for which the computation terminates at that leaf. These sets partition $\{0, 1\}^N$; in particular, the accepting leaves partition $f^{-1}(1)$. So it suffices to prove that the size of the set corresponding to any leaf is divisible by 2^{N-d}.

View $\{0, 1\}^N$ as a vector space of dimension N over GF(2) (with coordinate-wise addition). Each parity query or input bit query is of the form: "Is the input in a subspace of codimension 1?" (A subspace has *codimension c* if it has dimension $N - c$.) If every response is "yes," then the set corresponding to the leaf is also a subspace; since at most d questions were asked, this space is of codimension at most d. If some response is "no," then the set is an affine subspace. This is either empty or nonempty of codimension at most d. In every case, the size of the set is a multiple of 2^{N-d}. □

5.2. *Lower Bound for* MAJORITY. As we have noted, the oblivious-pairing algorithm in Section 3 is an XOR decision tree. Theorem 4 therefore implies that $D^{\text{XOR}}(\text{MAJORITY}) \leq N+1-w(N)$. We now show that equality actually holds. Hence, the oblivious-pairing algorithm is optimal.

THEOREM 18. $D^{\mathcal{PARITY}}(\text{MAJORITY}) = D^{\text{XOR}}(\text{MAJORITY}) = N + 1 - w(N)$.

PROOF. As noted above, we already know that

$$D^{\text{XOR}}(\text{MAJORITY}) \leq N + 1 - w(N)$$

by Theorem 4. Since $D^{\mathcal{PARITY}}(\text{MAJORITY}) \leq D^{\text{XOR}}(\text{MAJORITY})$, it suffices to show that

$$D^{\mathcal{PARITY}}(\text{MAJORITY}) \geq N + 1 - w(N).$$

We use Lemma 17 to do so; the first step is to compute what power of 2 divides $|\text{MAJORITY}^{-1}(1)|$.

We first consider the case where N is even, say $N = 2m$. The 2^{2m} possible inputs can be divided into three types: those with more ones than zeros, those with more zeros than ones, and the $\binom{2m}{m}$ perfectly balanced inputs. The number of inputs with a majority of ones is therefore $2^{2m-1} - \frac{1}{2}\binom{2m}{m}$. Since $\binom{2m}{m} = (2m)!/(m!)^2$, Corollary 2 states that $\binom{2m}{m}$ is exactly divisible by 2^k for $k = (2m - w(2m)) - 2(m - w(m)) = w(2m) = w(N)$. Therefore, since $w(N) < N$, $|\text{MAJORITY}^{-1}(1)|$ is exactly divisible by $2^{w(N)-1}$.

If we had $D^{\mathcal{PARITY}}(\text{MAJORITY}) \leq N - w(N)$, then, by Lemma 17, we would have $2^{w(N)}$ dividing $|\text{MAJORITY}^{-1}(1)|$. Since this is false, we must have $D^{\mathcal{PARITY}}(\text{MAJORITY}) \geq N + 1 - w(N)$ for even N.

When N is odd, we note that we can use an algorithm for MAJORITY on N variables to solve the problem on $N - 1$ variables: pad the $N - 1$ input bits with one zero. Since the above argument for the even case only relies on the number of inputs mapped to one, we thus conclude that $D^{\mathcal{PARITY}}(\text{MAJORITY})$ for N odd is at least $(N-1)+1-w(N-1) = N + 1 - w(N)$, which proves the desired result. □

6. Lower Bounds for Computing MAJORITY with Zero-Sided Error.

Beals et al. [3] prove a lower bound of $\frac{1}{2}N$ on the number of queries a quantum black-box network needs to compute MAJORITY on N-bit strings with zero-sided error $\varepsilon < 1$. We will show that the cost of a randomized XOR decision tree computing MAJORITY with zero-sided error $\varepsilon = o(1)$ cannot be reduced below $\frac{2}{3}N - o(N)$. We will also prove that any classical randomized decision tree with zero-sided error $\varepsilon = \frac{1}{2}$ has to have cost at least N. In fact, we will show the stronger result that any classical randomized decision tree with arbitrary error bounded by $\frac{1}{4}$ has cost at least N.

This result about randomized XOR decision trees follows directly from the average case lower bound of Alonso et al. [2] using a standard argument.

THEOREM 19 [2]. *There exists a constant d such that the following holds for any input length N. For any* XOR *decision tree computing* MAJORITY, *the average cost over all inputs of length N is at least*

$$\tfrac{2}{3}N - \sqrt{\frac{8N}{9\pi}} - d.$$

COROLLARY 20. *There exists a constant d such that the following holds for any input length N. For any randomized* XOR *decision tree computing* MAJORITY *exactly, there exists an input of length N such that the expected number of queries is at least $\frac{2}{3}N - \sqrt{8N/9\pi} - d$ on that input.*

PROOF. Let $g(N)$ denote $\frac{2}{3}N - \sqrt{8N/9\pi} - d$ from Theorem 19.

Look at the randomized XOR decision tree $\mathcal{T}$ as a distribution over deterministic XOR decision trees $\{\mathcal{T}_i\}$. Each deterministic tree $\mathcal{T}_i$ in the support of $\mathcal{T}$ computes MAJORITY exactly. By Theorem 19, the average cost of each $\mathcal{T}_i$ is at least $g(N)$. Consequently, the expected average cost of $\mathcal{T}$ is at least $g(N)$. Therefore, there exists an input on which the expected number of queries is at least $g(N)$. □

A randomized XOR decision tree $\mathcal{T}$ with zero-sided error ε and cost C, can be transformed into an exact randomized XOR decision tree $\mathcal{T}'$ for the same function with an expected number of queries of at most $C + \varepsilon(N - C) \leq C + \varepsilon N$ on any input. We just run $\mathcal{T}$ and whenever it is about to answer "I don't know," we query individual bits until we know the entire input. Using Corollary 20, we obtain:

THEOREM 21. *Any randomized* XOR *decision tree computing* MAJORITY *on N-bit inputs with zero-sided error ε has cost at least $\frac{2}{3}N - \varepsilon N - O(\sqrt{N})$.*

In contrast, a classical randomized decision tree needs N queries to compute MAJORITY with error ε for any sufficiently small constant ε.

THEOREM 22. *Any randomized decision tree that computes the* MAJORITY *of N bits with bounded error $\varepsilon \leq \frac{1}{4}$ has cost at least N.*

PROOF. Let t denote $\lceil N/2 \rceil$. Suppose there exists a randomized decision tree T that computes MAJORITY on N-bit inputs with bounded error $\varepsilon \leq \frac{1}{4}$ and cost at most $N-1$. Without loss of generality, we can assume that T always queries exactly $N - 1$ of the N input bits.

Consider a deterministic tree of cost $N - 1$ and suppose we pick an N-bit input uniformly at random among those with exactly A ones. Then the probability that the unique bit not queried is a one equals A/N. Also, for any final state s of T, the probability that we end up there only depends on the number of ones seen when we reach s.

Look at T as a probability distribution over deterministic trees of cost $N - 1$. Among all final states that have seen $t - 1$ ones, let α be the weighted fraction that outputs zero. Consider the input distribution that is a convex combination of β times the uniform distribution over inputs with exactly $t - 1$ ones, and $1 - \beta$ times the uniform distribution over inputs with exactly t ones. By the above observations, the probability of error is at least

$$\beta\left(1 - \frac{t-1}{N}\right)(1 - \alpha) + (1 - \beta)\frac{t}{N}\alpha.$$

Picking $\beta \doteq t/(N + 1)$ makes the factors of $(1 - \alpha)$ and α equal, so we get that the probability of error is at least $t(N - t + 1)/N(N + 1)$, which exceeds $\frac{1}{4}$. This contradicts the assumption that $\varepsilon \leq \frac{1}{4}$. □

We note that the bound of $\frac{1}{4}$ in Theorem 22 is essentially tight. Using the notation from the above proof, the following algorithm does the job: Query $N - 1$ bits in random order and output 0 if fewer than $t - 1$ of them are one, 1 if more, and the outcome of a (biased) coin toss otherwise.

COROLLARY 23. *Any randomized decision tree that computes the* MAJORITY *of* N *bits with zero-sided error* $\varepsilon \leq \frac{1}{2}$ *has cost at least* N.

PROOF. Transform the randomized decision tree $\mathcal{A}$ with zero-sided error ε into the randomized decision tree $\mathcal{A}'$ as follows: Whenever $\mathcal{A}$ says "I don't know," output the outcome of a fair coin toss; otherwise answer the same as $\mathcal{A}$. $\mathcal{A}'$ has two-sided error $\varepsilon/2$. Then apply Theorem 22 to $\mathcal{A}'$. □

Again, the bound of $\frac{1}{2}$ is essentially tight.

7. Open Questions. We summarize the known results in Table 1. We fix the error ε in the table to N^{-2}. This leads to several natural open questions.

- Our results for exact and zero-sided error in the XOR decision tree model are quite tight. The corresponding results in the quantum black-box model are not. Can we narrow the gap? The quantum black-box model is more powerful than the XOR decision tree model, so we may be able to improve the quantum upper bound by applying some other technique to MAJORITY.
- On the other hand, we may be able to improve the quantum lower bounds, in particular in the two-sided error case. The best lower bound we currently know is $\Omega(N)$ for any constant error ratio less than $\frac{1}{2}$. This follows from Paturi's [9] result that the approximating degree of the majority function (see, for example, [8] for a definition) is $\Omega(N)$, and the observation by Beals et al. that half the approximating degree is a lower bound for the quantum black-box complexity in the bounded error setting. The constant hidden in the $\Omega(N)$ of Paturi's result is much smaller than one. A constant of one would show that van Dam's approach is essentially optimal for MAJORITY.
- In this paper we focused on the exact and zero-sided error settings. The results in the table for two-sided error XOR decision trees trivially follow from the classical lower bound (Theorem 22) and the upper bound in the zero-sided error setting (Corollary 12). Can we exploit the two-sided error relaxation? How about the one-sided error setting?
- The $O(\sqrt{N})$ term in the lower bound for the cost of a zero-sided error randomized XOR decision tree comes from the average size of the discrepancy of a random input. It seems likely that, if we restrict to balanced inputs, we can improve this lower bound to $\frac{2}{3}N - O(\log N)$. Can we do so?

Table 1. Known results.

Cost of MAJORITY	Quantum black-box model		XOR decision tree model	
	Lower bound	Upper bound	Lower bound	Upper bound
Exact	$N/2$ [3]	$N - w(N) + 1$	$N - w(N) + 1$ [11]	$N - w(N) + 1$ [11]
Zero-sided error	$\frac{1}{2}N$ [3]	$\frac{2}{3}N + O(\sqrt{N \log N})$	$\frac{2}{3}N - O(\sqrt{N})$ [2]	$\frac{2}{3}N + O(\sqrt{N \log N})$
Two-sided error	$\Omega(N)$ [3]	$\frac{1}{2}N + O(\sqrt{N \log N})$ [12]	$\frac{1}{2}N$	$\frac{2}{3}N + O(\sqrt{N \log N})$

Acknowledgments. The authors thank László Babai, Eric Bach, Harry Buhrman, Jin-Yi Cai, Ian Dinwoodie, Murali K. Ganapathy, Pradyut Shah, Janos Simon, Daniel Štefankovič, Ronald de Wolf, and the anonymous referees for stimulating discussions, pointers to the literature, and helpful comments on earlier versions of the paper.

Appendix. Tail Laws. In this section we present proof sketches of two of our more technical results, Lemmas 7 and 13. Both of these tail laws can be established using Azuma's inequality. We instead provide proof sketches from first principles.

PROOF SKETCH FOR LEMMA 7. The number of inputs X yielding exactly C cancellations in round 1 can be expressed using a trinomial coefficient:

$$F(C) = \binom{\lfloor N/2 \rfloor}{C,\ \lfloor (A-C)/2 \rfloor,\ \lfloor (B-C)/2 \rfloor} 2^C,$$

assuming, if N is even, that not all of A, B, C are odd (in which case there are no such inputs).

We treat the cases of C even and C odd separately, letting

$$f(t) = \binom{\lfloor N/2 \rfloor}{2t,\ \lfloor A/2 \rfloor - t,\ \lfloor B/2 \rfloor - t} 4^t$$

and

$$g(t) = \binom{\lfloor N/2 \rfloor}{2t+1,\ \lfloor (A-1)/2 \rfloor - t,\ \lfloor (B-1)/2 \rfloor - t} 2^{2t+1}.$$

It is a matter of basic algebra to locate the maxima of f and g, although somewhat tedious because the answers are affected by the parities of A and B. In all cases, the maxima for f and for g occur at integers within 2 units distance from $AB/2N$.

The functions f and g both fall off very sharply away from their maxima, and in fact satisfy much stronger tail inequalities than what we need, which can be proved by standard techniques.

From this, it follows that F has its maximum within 4 units of AB/N, and that F also falls away very sharply away from this maximum. The concentration result for $c(X)$ follows. □

PROOF SKETCH FOR LEMMA 13. Fix an integer $0 \leq k \leq N$. The probability that exactly i ones occur in the first k bits of X is

$$\binom{k}{i}\binom{N-k}{A-i} \Big/ \binom{N}{A}$$

The distribution of i is concentrated at kA/N, which can be shown using standard techniques.

Choosing $k_1 = N^2/2A - d\sqrt{rN}$ and $k_2 = N^2/2A + d\sqrt{rN}$, the concentration results for i imply that, with probability at least $1 - 2^{-r}$, at most $\lfloor N/2 \rfloor$ ones occur in the first k_1 bits of X and at least $\lfloor N/2 \rfloor$ ones occur in the first k_2 bits of X. Hence $k_1 \leq M \leq k_2$. □

References

[1] Laurent Alonso, Edward M. Reingold, and René Schott. Determining the majority. *Information Processing Letters*, 47(5):253–255, 1993.

[2] Laurent Alonso, Edward M. Reingold, and René Schott. The average-case complexity of determining the majority. *SIAM Journal on Computing*, 26(1):1–14, 1997.

[3] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. In *Proceedings of the* 39*th IEEE FOCS*, pages 352–361, 1998.

[4] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, October 1997.

[5] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society*, *London*, *Series A*, 454:339–354, 1998. quant-ph/9708016.

[6] F. N. David and P. E. Barton. *Combinatorial Chance*. Griffin, 1962, Chapter 12.

[7] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society*, *London*, *Series A*, 439:553–558, 1992.

[8] Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994.

[9] Ramamohan Paturi. On the degree of polynomials that approximate symmetric Boolean functions. In *Proceedings of the* 24*th ACM STOC*, pages 468–474, 1992.

[10] Ronald L. Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3:371–384, 1976.

[11] Michael E. Saks and Michael Werman. On computing majority by comparisons. *Combinatorica*, 11(4):383–387, 1991.

[12] Wim van Dam. Quantum oracle interrogation: getting all information for almost half the price. In *Proceedings of the* 39*th IEEE FOCS*, pages 362–367, 1998.