Lower Bounds for Coin-Weighing Problems

ERIC PURDY
University of Chicago

Among a set of n coins of two weights (good and bad), and using a balance, we wish to determine the number of bad coins using as few measurements as possible. There is a known adaptive decision tree that answers this question in $O((\log(n))^2)$ measurements, and a slight modification of this decision tree determines the parity of the number of bad coins in $O(\log n)$. In this paper, we prove an $\Omega(\sqrt{n})$ lower bound on the depth of any oblivious decision tree which solves either the counting or the parity problem. Our lower bound can also be applied to any function of high average sensitivity, which includes most random functions and most random symmetric functions. With a slight generalization of this result, we derive lower bounds for the size of threshold circuits for a wide class of Boolean functions.

We demonstrate an exponential gap between the non-adaptive and adaptive coin-weighing complexities of the counting and parity problems. We prove a tight $\Theta(\log n)$ bound on the adaptive coin-weighing complexity of the parity problem.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Complexity of Proof Procedures*; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Relations among complexity measures*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Coin-weighing problems

1. THE BAD COIN COUNTING PROBLEM

We have n coins, which can be good or bad; at least one of them is bad. All the good coins weigh the same, and all the bad coins weigh the same. The bad coins are lighter than the good coins. We wish to determine the *number* of bad coins using as few comparisons on a balance as possible. (We need to specify that at least one coin is bad, since we cannot possibly distinguish a set of all good coins from a set of all bad coins.) A related question, which we will refer to as the Bad Coin Parity problem, asks us to determine whether this number is even or odd.

This process can be represented as a ternary decision tree, since the balance can tip left, tip right, or balance on any given measurement. A decision tree can be *adaptive* or *oblivious*. An oblivious decision tree must use the same measurement at all nodes of a given depth, while an adaptive decision tree is not subject to this requirement, and may act differently on different branches of the tree.

Author's address: Eric Purdy, Department of Computer Science, 1100 E. 58th St., Chicago, IL 60637

An algorithm is known which solves the Bad Coin Counting Problem in $O((\log n)^2)$ measurements, and a simple modification solves the Bad Coin Parity Problem in $O(\log n)$ measurements. Both of these algorithms are adaptive.

1.1 Results

In this paper, we give a lower bound for the oblivious coin-weighing complexity of both the Bad Coin Counting Problem and the Bad Coin Parity Problem.

THEOREM 1.1. *Any oblivious decision tree which solves the Bad Coin Parity problem must use $\Omega(\sqrt{n})$ comparisons on the balance. Consequently, the same lower bound holds for the Bad Coin Counting Problem.*

Theorem 1.1 follows from the following more general lower bound on oblivious coin-weighing algorithms, which we prove in Section 4.

THEOREM 1.2. *Let $f : \{0,1\}^n \to \mathcal{R}$ be a function of n Boolean variables. We can identify any sets of coins C with a 0/1 vector x by numbering the coins and letting $x_i = 1$ iff the i-th coin is good. We can then try to determine $f(x)$ by applying measurements to C. Any oblivious coin-weighing algorithm for determining f in this way must use $\Omega(\alpha(f)/\sqrt{n})$ measurements, where $\alpha(f) = E_x[\#\{f(x \oplus e_i) \neq f(x)\}]$ is the average sensitivity of f.*

The corresponding function f for the Bad Coin Counting Problem and the Bad Coin Parity Problem are the Hamming weight function and the parity function, respectively. Both of these function have average sensitivity n, which gives us Theorem 1.1. Both random functions and random symmetric functions have average sensitivity $\Omega(n)$ with high probability, so we also have

THEOREM 1.3. *Let $f : \{0,1\}^n \to \{0,1\}$ be a random function, with each $f(x)$ chosen independently and uniformly at random. With probability at least 99%, the oblivious coin-weighing complexity of f is at least $\Omega(\sqrt{n})$. The same holds if "random function" is replaced by "random symmetric function", and we choose $f(x)$ independently for each value of $\|x\|_1$.*

Additionally, we prove the following corollary to Theorem 1.1 via a simple conversion argument:

COROLLARY 1.4. *Any adaptive coin-weighing algorithm that solves the Bad Coin Parity problem must use at least $\frac{1}{2}\log_3 n - O(1)$ comparisons on the balance.*

Since Algorithm 2 gives an upper bound of $2\log_2 n + O(1)$ for the adaptive coin-weighing complexity of the Bad Coin Parity Problem, we have established the following bound, tight up to a constant factor:

COROLLARY 1.5. *The adaptive coin-weighing complexity of the Bad Coin Parity problem is $\Theta(\log n)$.*

We have also demonstrated an exponential gap between the oblivious and adaptive coin-weighing complexities of the Bad Coin Parity Problem. We believe this to be the first example of such an exponential gap for a coin-weighing problem.

Theorem 1.1 is notable for being a rare instance in which a lower bound for a coin-weighing problem can be proved that is much stronger than that given by the

Table I. Currently known bounds on coin-weighing complexity of various problems. The column labeled "Info" gives the information theory lower bound. The columns labeled "Adaptive" and "Oblivious" give the known bounds on the adaptive and oblivious complexities of the problems, respectively. Citations in brackets are given in the text. Unmarked statements are obvious.

Problem	Info	Adaptive		Oblivious	
		Lower	Upper	Lower	Upper
Counting	$\log_3 n$	$\Omega(\log n)$	$O((\log n)^2)$ [F, Alg 3]	$\Omega(\sqrt{n})$ [Thm 1.1]	n
Parity	1	$\Omega(\log n)$ [Cor 1.4]	$O(\log n)$ [F, Alg 2]	$\Omega(\sqrt{n})$ [Thm 1.1]	n
Location	$\log_3 n$	$\Theta(\log n)$ [A]		$\Theta(\log n)$ [B]	
Diagnosis	$\log_3 \binom{n}{c}$ $(\log_3 2)n$	$\Theta\left(\log_3 \binom{n}{c}\right)$ [C]		$\Theta(n)$	
All Equal	1	$\Theta\left(\frac{\log n}{\log\log n}\right)$ [D]		$\Omega\left(\frac{\log n}{\log\log n}\right)$ [E]	?
Finding (Sec. 7.1)	$\geq \log_3 n$	$\Omega(\log n)$	$O(\log n)$ [F, Alg 1]	$\Omega(\log n)$	?
Splitting (Sec. 7.2)	$\geq \log_3 n$	$\Omega(\log n)$	$O(\log n)$ [F, Alg 2]	$\Omega(\log n)$	?

standard information theory argument. (Any tree of depth d with branching factor b has at most b^d leaves. Therefore, if a decision problem has N possible answers, any valid decision tree for this problem must have depth $d \geq \lceil \log_b N \rceil$.) For the Bad Coin Counting Problem, there is an exponential gap between these lower bounds. For the Bad Coin Parity Problem, the information theory lower bound is trivial, while our lower bound is essentially the best possible.

Our proofs are based in Boolean function analysis, in particular, the notion of average sensitivity of Boolean functions. As a further application of these techniques, we prove Theorem 5.2, a lower bound on the number of gates in the first level of a threshold circuit, which we present in section 5.

1.2 Folklore Results

For the sake of completeness, and also for lack of a convenient reference, we describe and prove several coin-weighing results that appear to be folklore. In Section 3, we prove that coin-weighing algorithms can be limited to comparing equally-sized sets of coins without any loss of power. In Section 7, we describe a number of coin-weighing algorithms, including the aforementioned algorithms for the Bad Coin Parity Problem and the Bad Coin Counting Problem.

1.3 Prior Work

A more well-known coin-weighing problem, which we will refer to as the Bad Coin Location problem, asks the following: given a set of n coins, *exactly* one of which is counterfeit (and thus lighter or heavier), how many weighings on a balance does it take to identify the counterfeit coin? This problem was solved in the adaptive case by F.J. Dyson in 1946 [Dyson 1946], and in the oblivious case by A. Born et al. [Born et al. 2003].

We can also look at the problem of finding out exactly which coins are bad when more than one can be bad. We will call this the Bad Coin Diagnosis problem. The coin-weighing complexity of this problem has been more or less resolved. Suppose that c of the n coins are counterfeit. In [Pyber 1986], it is proved that this problem can be solved in at most $\log_3 \binom{n}{c} + 15k$ weighings, if we know that $c < k$ for some fixed k. In [Aigner and Li 1997], it is proved that this problem can be solved in at most $\log_3 \binom{n}{k} + 1.5k + 9$, in the same situation.

In another variant, we may not have any prior knowledge of c, the number of bad coins. In [Hu et al. 1994], it is proved that this problem can be solved in at most $a \log_3 \binom{n}{c}$ measurements for some constant $a \leq 2 \log_2 3$.

In [Alon et al. 1996; Alon and Kozlov 1997; Alon and Vu 1997; Kozlov and Vu 1997], N. Alon, D.N. Kozlov, and V. Vu consider the question of whether all the coins in a set have the same weight. In the case where the coins have at most two distinct weights, which corresponds to our situation of good coins and bad coins, they prove that k weighings suffice to answer the question for $k^{(1/2+o(1))k}$ coins. Thus, to answer the all-equal question for n coins, we need $(2+o(1))\frac{\log n}{\log \log n}$ weighings. This is a much smaller number of weighings than in the current paper, showing that the all-equal problem (which corresponds roughly to the AND function) is significantly less complex in this model than the parity problem (which corresponds to the parity function).

Finally, [Guy and Nowakowski 1995] is a nice survey article on coin-weighing algorithms.

We summarize the current state of these problems in Table I, along with two auxiliary problems described in Section 7. Citations are as follows

A	[Dyson 1946]
B	[Born et al. 2003]
C	[Aigner and Li 1997; Hu et al. 1994; Pyber 1986]
D	[Alon and Kozlov 1997; Alon et al. 1996; Alon and Vu 1997; Kozlov and Vu 1997]
E	[Alon and Kozlov 1997; Alon et al. 1996; Alon and Vu 1997; Kozlov and Vu 1997]
F	Folklore results described in Section 7.

2. PRELIMINARIES

We have a set C of n coins, each of which has one of two weights w_0, w_1, with $w_0 < w_1$. By numbering the coins in a consistent fashion, we see that C is completely described by the weights w_0, w_1, and a bit vector $\vec{c} \in \{0,1\}^n$, where

$$\vec{c}_i = j \iff i\text{-th coin in } C \text{ has weight } w_j.$$

A coin-weighing problem requires us to discover some property of C that is independent of the weights w_0, w_1. This necessarily takes the form of a function f mapping[1] $\vec{c} \in \{0,1\}^n$ to some domain $\mathcal{R}$.

An algorithm computes $f(\vec{c})$ by applying measurements to C. A measurement m is specified by two disjoint sets $L, R \subset C$ to be placed on the left and right side

[1]There is a small technicality here, in that we cannot distinguish between the cases where $\vec{c}$ is the all-zeros vector and the all-ones vector. Therefore, we only allow f such that $f(\vec{0}) = f(\vec{1})$.

of the balance, respectively. We can also specify m by a vector $\vec{m} \in \{-1, 0, 1\}^n$. If c is the i-th coin, $\vec{m}_i = 1$ if $c \in R$, -1 if $c \in L$, and 0 otherwise. The result of a measurement is then determined by which pan is heavier, which we denote as

$$w_{\vec{m}}(\vec{c}) = \operatorname{sgn}(\vec{m} \cdot \vec{c}) = \begin{cases} +1 & \sum_{i \in L} w_{\vec{c}_i} < \sum_{i \in R} w_{\vec{c}_i} \\ 0 & \sum_{i \in L} w_{\vec{c}_i} = \sum_{i \in R} w_{\vec{c}_i} \\ -1 & \sum_{i \in L} w_{\vec{c}_i} > \sum_{i \in R} w_{\vec{c}_i} \end{cases}.$$

We wish to compute $f(\vec{c})$ using as few measurements as possible in the worst case. We can represent the computation as a decision tree in which each non-leaf node is labeled with a measurement $\vec{m}$. Each non-leaf node has three children corresponding to the cases $w_{\vec{m}}(\vec{c}) = 1, -1, 0$. We compute $f(\vec{c})$ by applying measurement $\vec{m}$ and moving to the appropriate child until we come to a leaf. Leaves are labeled with values of $f(\vec{c})$ to specify the output of the algorithm on this computation path. To simplify the discussion, we will focus on the existence of a correct labeling.

Definition 2.1. A decision tree T for a coin weighing problem f is *sound* if each leaf of T determines a unique value of $f(\vec{c})$.

An *oblivious* decision tree cannot choose its measurements based on the outcome of previous measurements, so all the interior nodes at a given depth will be labeled with the same measurement. Therefore, an oblivious decision tree can be represented as a set $T = \{m_1, m_2, \ldots, m_s\}$ of measurements. The (adaptive) coin-weighing complexity of f is then the minimum depth of a sound decision tree for f. The oblivious coin-weighing complexity is the minimum depth of a sound oblivious decision tree for f.

3. ELIMINATING UNBALANCED MEASUREMENTS

Definition 3.1. We say that a measurement L, R is *balanced* if $|L| = |R|$, i.e., there are an equal number of coins on each side of the balance. We will say that the measurement is *left-heavy* if $|L| > |R|$, and define right-heaviness analogously.

For any decision tree T, we can define another decision tree $bal(T)$ which is balanced. We compute $bal(T)$ by iterating the following process:

(1) While T has an unbalanced measurement m at vertex t
  (a) Let T_-, T_+, T_0 be the subtrees rooted at the children of t
  (b) If m is right-heavy, replace the vertex t with the subtree T_+. Delete the subtrees T_-, T_0.
  (c) If m is left-heavy, replace the vertex t with the subtree T_-. Delete the subtrees T_+, T_0.

This process corresponds to always assuming that the larger set of coins will be heavier, and pretending that this is the outcome of the measurement.

In this section, we prove, for the sake of completeness, the following well known fact:

LEMMA 3.2. *The decision tree T for a coin-weighing problem f is sound if and only if $bal(T)$ is also sound. Thus, we can require our algorithms to use balanced measurements.*

The result of a measurement may depend on the values w_0, w_1. For example, two bad coins will outweigh one good coin if and only if $2 \cdot w_0 > w_1$. This leads to the following definition:

Definition 3.3. We say that the weights in C are "close together" if

$$w_0 < w_1 < \left(1 + \frac{1}{|C|}\right) \cdot w_0$$

Fact 3.4. In the case where the weights in C are close together, any non-balanced weighing will always tell us that the side with more coins is heavier. Therefore, only balanced weighings provide additional information in these cases.

Fact 3.5. The result of a balanced measurement does not depend on the relative weights of the good and bad coins, only on the number of good and bad coins in each tray.

PROPOSITION 3.6. *A decision tree T correctly deals with all close-together instances of the problem if and only if $bal(T)$ also does so.*

PROOF. We constructed $bal(T)$ in Definition 3.1 so that it does the same computation as T as long as all non-balanced measurements have the result that the side with more coins is heavier.

According to Fact 3.4, this is the case for C with close-together weights. Therefore, T correctly computes $f(C)$ if and only if $bal(T)$ does. $\square$

PROPOSITION 3.7. *For any decision tree T, $bal(T)$ is sound on close-together instances of the problem if and only if $bal(T)$ is sound on all instances.*

PROOF. By Fact 3.5, the outcome of a balanced measurement depends only on $\vec{c}$, not on w_0 and w_1. Since $bal(T)$ consists solely of balanced measurements, its output is also a function of $\vec{c}$.

Therefore, if $bal(T)$ is sound on close-together C, it is sound on all C. Conversely, if $bal(T)$ is sound on all C, it is sound on close-together C. $\square$

Propositions 3.6 and 3.7 combine to give us Lemma 3.2. Since $bal(T)$ has smaller depth than T, it is thus sufficient to deal only with decision trees consisting of balanced measurements.

4. THE LOWER BOUND ON OBLIVIOUS DECISION TREES

In this section, we prove Theorem 1.1. By Lemma 3.2, we can limit ourselves to balanced measurements.

Definition 4.1. Let $f : \{0,1\}^n \to \mathcal{R}$ be a function of n Boolean variables, and let $x \in \{0,1\}^n$. For $1 \le i \le n$, let x^i be x with its i-th coordinate flipped. The *sensitivity of f at x*, $\sigma_x(f)$, is the number of i such that $f(x^i) \neq f(x)$. The *average sensitivity of f*, $\alpha(f)$ is $E_x[\sigma_x(f)]$, where $x \in \{0,1\}^n$ is chosen uniformly at random.

We also define the sensitivity graph of f to be $G_f = (V, E_f)$ with $V = \{0,1\}^n$, and $(x, y) \in E_f$ iff:

(1) $y = x^i$ for some i

(2) $f(y) \neq f(x)$.

Note that $|E_f| = 2^{n-1}\alpha(f)$.

LEMMA 4.2 DECOMPOSITION LEMMA. *If $g_1, \ldots, g_r$ are functions from $\{0,1\}^n$ to $\mathcal{X}$, and h is a function from $\mathcal{X}^r$ to $\mathcal{R}$, and $f(x) = h(g_1(x), ..., g_r(x))$, then $\alpha(f) \leq \alpha(g_1) + ... + \alpha(g_r)$.*

PROOF. Consider the sensitivity graph $G_f = (V, E_f)$, and the sensitivity graphs G_{g_1},...,G_{g_r}. Let $(x, y) \in E_f$. In order for $h(g_1(x), \ldots, g_r(x)) \neq h(g_1(y), \ldots, g_r(y))$, we must have $g_i(x) \neq g_i(y)$ for some i, and then $(x, y) \in E_{g_i}$. Thus $E_f \subseteq \cup_i E_{g_i}$, and $|E_f| \leq \sum_i |E_{g_i}|$.

Therefore, $\alpha(f) \leq \sum_i \alpha(g_i)$. □

LEMMA 4.3. *The average sensitivity of a measurement function is $O(\sqrt{n})$* [2].

PROOF. We begin with the following observation:

Observation 4.4. Let $\vec{c}, \vec{d}$ be two coin-vectors differing only in the i-th coordinate, with $\vec{c_i} = 1, \vec{d_i} = 0$. Then for any measurement vector m,

$$\vec{m} \cdot \vec{c} - \vec{m} \cdot \vec{d} = \vec{m} \cdot (\vec{c} - \vec{d}) = \vec{m} \cdot e_i = \vec{m}_i \in \{1, -1, 0\}.$$

If $m \cdot \vec{c} > 0$, we know $m \cdot \vec{d} \geq 0$. Therefore, if $(\vec{c}, \vec{d}) \in E_{\vec{m}}$, either $w_{\vec{m}}(\vec{c}) = 0$ or $w_{\vec{m}}(\vec{d}) = 0$.

Returning to the proof of the lemma, let $\vec{m}$ be a balanced measurement, with k coins on each side of the balance. Let $Supp(\vec{m}) = \{i \mid \vec{m}_i \neq 0\}$. Then $|Supp(\vec{m})| = 2k$.

Let $K_m = \{\vec{c} \mid w_{\vec{m}}(\vec{c}) = 0\}$. For $\vec{c} \in K_m$, let $N_m(\vec{c}) = \{\vec{d} \mid (\vec{c}, \vec{d}) \in E_{\vec{m}}\}$. Clearly these $\vec{d}$ are in one-to-one correspondence with $Supp(\vec{m})$, so $|N_m(\vec{c})| = |Supp(\vec{m})| = 2k$.

By Observation 4.4, all edges in $E_{\vec{m}}$ are between some $\vec{c} \in K_m$ and some $\vec{d}$ in the corresponding $N_{\vec{m}}(\vec{c})$. Thus $|E_{\vec{m}}| = \sum_{\vec{c} \in K_m} |N_{\vec{m}}(\vec{c})| = |K_m| \cdot 2k$.

$|K_m|$ is the number of ways to choose j coins out of k coins on the left side of the balance to be good, and j coins out of k coins on the right side to be good, and assigning the coins arbitrarily on the other $n - 2k$ coins, for each j between 0 and k.

$$|K_m| = \left(\sum_{j=0}^{k} \binom{k}{j}^2 2^{n-2k} \right) = \binom{2k}{k} 2^{n-2k}$$

Thus

$$|E_{\vec{m}}| = \binom{2k}{k} 2^{n-2k}(2k) = 2^n \cdot 2k \frac{\binom{2k}{k}}{2^{2k}} = O(2^n \cdot 2k \frac{1}{\sqrt{k}}) \leq O(2^n \sqrt{n}).$$

Since $|E_{\vec{m}}| = 2^{n-1}\alpha(m)$, we conclude that $\alpha(\vec{m}) = O(\sqrt{n})$.

□

[2]This lemma is a close cousin of the well-known result that monotone functions have average sensitivity $O(\sqrt{n})$

Together, the Decomposition Lemma and Lemma 4.3 establish our main result, Theorem 1.2.

For the parity function and the counting function on n bits, it is clear that the sensitivity at any point is n, since flipping any bit always changes its value. Thus the average sensitivity for these functions is n, and Theorem 1.2 implies Theorem 1.1.

If f is a random function or a random symmetric function, we know that f will have average sensitivity $\Omega(n)$ with probability at least 99%, which establishes Theorem 1.3.

5. APPLICATIONS TO CIRCUIT COMPLEXITY

In this section[3], we consider a very general class of circuits which includes threshold circuits, and use Lemma 4.2 to prove lower bounds on the size of such circuits.

Definition 5.1. Let $MonoLayer(n, k)$ be the following class of circuits: we have n inputs, each in $\{0, 1\}$. We have k gates on the first level, and all are required to compute monotone functions. We also allow NOT gates between the inputs and the first level, although we require that each gate at the first level receives at most one of x or $\bar{x}$. We allow an arbitrary number of arbitrary gates at all other levels.

THEOREM 5.2. *Consider a function $f : \{0, 1\}^n \to \mathcal{X}$. Let C be a $MonoLayer(n, k)$ circuit computing f. Then k, the number of gates at C's first level, must be at least $\Omega(\alpha(f)/\sqrt{n})$.*

PROOF. Let $x_1, \ldots, x_n$ be our inputs. Let $g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)$ be the functions computed by our gates at the first level.

We cite the following well-known fact:

Fact 5.3. Monotone functions have average sensitivity $O(\sqrt{n})$.

Thus the g_i have average sensitivity at most $O(\sqrt{n})$ (note that the average sensitivity of a function is not affected by negating some of its inputs). We can then apply Lemma 4.2, since the output of f must be computable from $g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)$.

Thus $\alpha(f) \le \sum_i \alpha(g_i) \le O(k\sqrt{n})$, and $k = \Omega(\alpha(f)/\sqrt{n})$, as desired. $\square$

6. THE LOWER BOUND ON ADAPTIVE DECISION TREES

In this section, we use Theorem 1.2 to prove Corollary 1.4. This follows from a general relation between adaptive and oblivious decision trees for the same decision problem.

Definition 6.1. Let T be an adaptive decision tree. We define $L = list(T)$ to be the oblivious decision tree corresponding to the list of all of the measurements used in any sub-branch of T.

PROPOSITION 6.2. *Consider a function $f : \{0, 1\}^n \to \mathcal{X}$. If T is a sound adaptive decision tree for f, then $L = list(T)$ is a sound oblivious decision tree f.*

[3]We wish to thank the anonymous referee for telling us how to strengthen and greatly simplify the results of this section.

PROOF. Let T be a sound adaptive decision tree for f. If $\vec{c}$ and $\vec{d}$ are coin-vectors for which $f(\vec{c}) \neq f(\vec{d})$, some measurement $\vec{m}$ in some branch of T must return a different result for $\vec{c}$ and $\vec{d}$. Then $\vec{m}$ is in L, and we have proved that whichever leaf of L contains $\vec{c}$ does not contain $\vec{d}$. This holds for any $\vec{c}, \vec{d}$ such that $f(\vec{c}) \neq f(\vec{d})$, so L is sound. It is clear that L is oblivious. □

PROPOSITION 6.3. *Consider a function $f : \{0,1\}^n \to \mathcal{X}$. Let b represent the branching factor of a decision tree. If the oblivious complexity of f is C_{ob}, and the adaptive complexity of f is C_{ad}, then $C_{ad} \geq \log_b C_{ob}$.*

PROOF. Let T be a sound adaptive decision tree of minimal depth C_{ad}. Let $L = \mathit{list}(T)$. Since the branching factor of the trees is b, T can use at most b^d different measurements at depth d. Therefore, T can use at most $\sum_{d=0}^{C_{ad}-1} b^d \leq b^{C_{ad}}$ different measurement, and $|L| \leq b^{C_{ad}}$. By Proposition 6.2, we know that L is a sound oblivious decision tree. Therefore $|L| \geq C_{ob}$. Thus $b^{C_{ad}} \geq C_{ob}$, which yields the desired result. □

Now we prove Corollary 1.4.

PROOF PROOF OF COROLLARY 1.4. Let f be the parity function. Let C_{ob}, C_{ad} be the oblivious and adaptive complexities of f, respectively. $C_{ob} = \Omega(\sqrt{n})$ by Theorem 1.1. For sufficiently large n and some constant M, $C_{ob} \geq M\sqrt{n}$. Using Proposition 6.3:

$$C_{ad} \geq \log_3(C_{ob}) \geq \log_3(M\alpha(f)/\sqrt{n}) = \log_3(\alpha(f)) - \frac{1}{2}\log_3(n) + O(1)$$

which concludes the proof. □

7. EFFICIENT ALGORITHMS FOR PARITY AND COUNTING PROBLEMS

In this section, we present a number of efficient coin-weighing algorithms, some of them provably optimal. The results in this section appear to be folklore [4], but we include them for lack of a convenient reference.

In Section 7.1, we give an efficient algorithm for locating a single bad coin. In Section 7.2, we give an efficient algorithm for splitting a set of coins into two sets with an equal number of good coins and an equal number of bad coins. This is useful as a subroutine, and also allows us to solve the Bad Coin Parity Problem in $2\log_2 n + O(1)$ measurements. This not only closely matches the $\Omega(\log n)$ lower bound given by Theorem 1.4, but also demonstrates, in the light of Theorem 1.1, an exponential gap between the oblivious and adaptive complexities of the Bad Coin Parity Problem.

In Section 7.3, we give an efficient algorithm for the Bad Coin Counting Problem. Our algorithm uses $(\log_2 n)^2 + O(\log n)$ measurements. Together with Theorem 1.1, this demonstrates an exponential gap between the oblivious and adaptive complexities of the Bad Coin Counting Problem.

[4]The Bad Coin Counting Problem has been assigned to and solved by generations of students in Professor Babai's Algorithms class [Babai], but Professor Babai does not know the original source.

7.1 Finding a Coin of Known Weight

We are given a set of coins which contains at least one bad coin, and we want to identify at least one specific bad coin.

PROPOSITION 7.1. *Given n coins, at least one of which is bad, we can locate a bad coin in* $\log_2 n + O(1)$ *comparisons on the balance.*

Algorithm 1 Finding one bad coin

```
 1: procedure find_bad($S$):
 2:   while $|S| > 1$ do
 3:     $X \leftarrow \emptyset$
 4:     if $|S|$ is odd then
 5:       Remove an arbitrary coin $c$ from $S$
 6:       $X \leftarrow X \cup \{c\}$
 7:     end if
 8:     Split S into $S_1, S_2$, arbitrary sets of equal size
 9:     Weigh $S_1$ against $S_2$
10:     $S \leftarrow$ the lighter of $S_1$ and $S_2$ (breaking ties arbitrarily)
11:     $S \leftarrow S \cup X$
12:   end while
13:   return $d$, the sole coin in $S$
```

PROOF. Consider Algorithm 1. We know that S contains at least one bad coin when we enter the while loop in line 2, we we just need to prove that the while loop preserves this invariant, and we will return a bad coin in line 13.

Suppose $|S|$ is even. In line 10, we know that the lighter of S_1, S_2 has more bad coins, and thus at least one bad coin. Thus the invariant is preserved.

Suppose $|S|$ is odd. We remove a coin c from S in line 5. If c is bad, then the invariant is preserved, since we add c back in at line 11. Otherwise, we are back in the case where $|S|$ is even, and the invariant is still preserved.

The number of measurements used is $\lceil \log_2 n \rceil$, since we use one measurement in each iteration of the loop, and the loop runs $\lceil \log_2 n \rceil$ times before $|S| = 1$. □

7.2 Splitting the Coins Evenly and Solving the Bad Coin Parity Problem

Consider the following auxiliary problem:

Problem 7.2 *Bad Coin Splitting Problem.* Given a set S of n coins, we want to partition S into sets S_1, S_2, S_3 such that S_1 and S_2 contain the same number of bad coins and the same number of good coins, and $|S_3| \leq 2$. (We need S_3 in case there are an odd number of good coins or an odd number of bad coins.)

PROPOSITION 7.3. *The Bad Coin Splitting problem can be solved in* $\log_2 n + O(1)$ *measurements on the balance.*

Observation 7.4. Let A and B be disjoint sets of coins, with $|A| = |B|$, and let c_1, c_2 be two coins not in $A \cup B$. If $c_1 \cup A$ is heavier than $c_2 \cup B$, but $c_2 \cup A$ is lighter than $c_1 \cup B$, then:

Algorithm 2 Bad Coin Splitting Algorithm

```
1:  procedure split($S$):
2:      $n \leftarrow |S|, k \leftarrow 0, L \leftarrow 0, U \leftarrow n/2$
3:      $A_0 \leftarrow \{\frac{n}{2}+1, \dots, n\}$
4:      $B_0 \leftarrow \{1, \dots, \frac{n}{2}\}$
5:      $R_0 \leftarrow \text{compare}(A_0, B_0)$
6:      if $R_0 = 0$ then
7:          return $(A_0, B_0, \emptyset)$
8:      end if
9:      while $L < U - 1$ do
10:         $k \leftarrow \lfloor \frac{L+U}{2} \rfloor$
11:         $A_k \leftarrow \{1, \dots, k\} \cup \{\frac{n}{2}+k+1, \dots, n\}$
12:         $B_k \leftarrow \{k+1, \dots, \frac{n}{2}+k\}$
13:         $R_k \leftarrow \text{compare}(A_k, B_k)$
14:         if $R_k = 0$ then
15:             return $(A_k, B_k, \emptyset)$
16:         end if
17:         if $R_k = R_0$ then
18:             $U \leftarrow k$
19:         end if
20:         if $R_k = -R_0$ then
21:             $L \leftarrow k$
22:         end if
23:     end while
24:     return $(A_L, B_L, \{c_L, c_{L+1}\})$
```

(1) c_1 is a good coin,

(2) c_2 is a bad coin, and

(3) A and B have an equal number of bad coins.

Thus $A, B, \{c_1, c_2\}$ is a valid solution to Problem 7.2.

The splitting algorithm looks for sets satisfying the conditions of Observation 7.4, although it may instead find an exact split of the coins.

Observation 7.5. Suppose we have a set $S = \{c_1, \dots, c_n\}$ of n coins, and n is even. For $0 \leq k \leq n/2$, consider partitioning S into two sets of size $n/2$: $A_k = \{1, \dots, k\} \cup \{n/2 + k + 1, \dots, n\}$ and $B_k = \{k+1, \dots, n/2 + k\}$. Since A_k and B_k have the same size, if A_k weighs the same as B_k, then $A_k, B_k, \emptyset$ is a valid solution to Problem 7.2.

Let R_k be the outcome of measuring A_k against B_k, expressed as either $0, 1$, or -1, depending on whether A_k is the same weight as, heavier than, or lighter than B_k.

Suppose that $R_0 \neq 0$. Then $R_{n/2} = -R_0$, since $A_{n/2} = B_0$ and $B_{n/2} = A_0$. In this case, there must be at least one k such that $R_k \neq R_{k+1}$. Given such a k, either one of R_k, R_{k+1} is zero, giving a split of S, or $R_k = -R_{k+1}$, and we are in the situation of Observation 7.4. In this case, $A_k \setminus \{n/2+k+1\}, B_k \setminus \{k+1\}, \{k+$

$1, n/2 + k + 1\}$ is a valid split of S.

Using Observation 7.5 allows us to efficiently split the coins in a binary search:

Proposition 7.6. *The Bad Coin Parity problem can be solved using* $2 \log_2 n + O(1)$ *measurements on the balance.*

Proof. Let S_1, S_2, S_3 be a valid split of S, which we can find in $\log_2 n + O(1)$ measurements using the splitting algorithm. Let d be a bad coin, found using Algorithm 1, again in $\log_2 n + O(1)$ measurements.

Since S_1 and S_2 contain the same number of bad coins, and thus an even number in total, we need only determine the number of coins in S_3, which we can do in at most two measurements by weighing each coin against d. □

7.3 Solving the Bad Coin Counting Problem

Proposition 7.7. *The Bad Coin Counting problem can be solved adaptively in* $(\log_2 n)^2 + O(\log n)$ *measurements.*

Algorithm 3 Counting the Bad Coins

```
 1: $c_0 \leftarrow$ badcoin($S$) (Algorithm 1)
 2: $S \leftarrow S \setminus \{c_0\}$
 3: procedure count($S$)
 4:   if $|S| = 1$ then
 5:     Weigh the sole coin $c$ in $S$ against $c_0$
 6:     return 1 if $c$ is bad, 0 otherwise
 7:   end if
 8:   $T \leftarrow \emptyset$
 9:   if $|S|$ is odd then
10:     Let $c$ be an arbitrary coin in $S$
11:     $S \leftarrow S \setminus \{c\}$
12:     $T \leftarrow T \cup \{c\}$
13:   end if
14:   $(S_1, S_2, S_3) \leftarrow$ split($S$) (Algorithm 2)
15:   $T \leftarrow T \cup S_3$
16:   Weigh each coin in $T$ against $c_0$, whose weight is known.
17:   Let $t$ be the number of bad coins in $T$.
18:   return $2 * \text{count}(S_1) + t$
19:
20: return count($S$) + 1
```

Proof Proof of Proposition 7.7. Correctness is straightforward: since S_1 and S_2 contain the same number of bad coins in line 14, it is sufficient to count the bad coins in S_1 and double that number.

Finding the bad coin in line 1 requires at most $\lceil \log_2 n \rceil$ measurements by Proposition 7.1. Every time that the procedure count is run, we use at most $\lceil \log_2 n \rceil$ measurements in line 14, and at most 3 measurements in line 16. Since S is halved in size every time count is called, it can be called at most $\lceil \log_2 n \rceil$ times. Therefore, the total number of measurements used by this algorithm is at most $(\log_2 n)^2 + O(\log n)$. $\square$

REFERENCES

Aigner, M. and Li, A. 1997. Searching for counterfeit coins. *Graphs and Combinatorics 13,* 1 (March), 9–20.

Alon, N. and Kozlov, D. N. 1997. Coins with arbitrary weights. *J. Algorithms 25,* 1, 162–176.

Alon, N., Kozlov, D. N., and Vu, V. H. 1996. The geometry of coin-weighing problems. In *FOCS*. 524–532.

Alon, N. and Vu, V. H. 1997. Anti-hadamard matrices, coin weighing, threshold gates, and indecomposable hypergraphs. *J. Comb. Theory, Ser. A 79,* 1, 133–160.

Babai, L. Algorithms. Homework problem, University of Chicago, `http://www.classes.cs.uchicago.edu/archive/2005/winter/27200-1/assignments/03.pdf`.

Born, A., Hurkens, C. A. J., and Woeginger, G. J. 2003. How to detect a counterfeit coin: Adaptive versus non-adaptive solutions. *Inf. Process. Lett. 86,* 3, 137–141.

Dyson, F. J. 1946. The problem of the pennies. *The Mathematical Gazette 30,* 29, 231–234.

Guy, R. K. and Nowakowski, R. J. 1995. Coin-weighing problems. *The American Mathematical Monthly 102,* 2, 164–167.

Hu, X.-D., Chen, P. D., and Hwang, F. K. 1994. A new competitive algorithm for the counterfeit coin problem. *Inf. Process. Lett. 51,* 4, 213–218.

Kozlov, D. N. and Vu, V. H. 1997. Coins and cones. *J. Comb. Theory, Ser. A 78,* 1, 1–14.

Pyber, L. 1986. How to find many counterfeit coins? *Graphs and Combinatorics 2,* 1, 173–177.