

Lower Bounds on Two Coin-Weighing Problems

Eric Purdy *

September 25, 2008

Abstract

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. This demonstrates an exponential gap between the nonadaptive and adaptive decision tree complexities of these problems. It also implies the tight order of magnitude $\Theta(\log n)$ for the adaptive decision tree complexity of the parity problem. With a slight generalization of this result, we derive some lower bounds for the size of threshold circuits for a wide class of Boolean functions.

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.

An adaptive decision tree is known for the Bad Coin Counting problem which uses only $O((\log n)^2)$ measurements. Moreover, we show that a slight

*Department of Computer Science, University of Chicago. Email: epurdy@cs.uchicago.edu

modification of this decision tree solves the Bad Coin Parity problem. We have described this decision tree in Section 7. Our main result establishes an exponential gap between the oblivious and adaptive complexities of both the Bad Coin Counting problem and the Bad Coin Parity problem. The main result is this:

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.*

By a simple conversion argument, this result also implies an essentially tight lower bound on the adaptive decision tree complexity:

Corollary 1.2. *Any adaptive decision tree which solves the Bad Coin Parity problem must use (asymptotically) $\frac{1}{2} \log_3 n$ comparisons on the balance.*

Together with the decision tree described in Section 7 (see Observation 7.12) this implies that:

Corollary 1.3. *The adaptive decision tree complexity of the Bad Coin Parity problem is $\Theta(\log n)$.*

1.1 Relation to 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 [6], and in the oblivious case by A. Born et al. [5].

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 Determination problem. The coin-weighing complexity of this problem has been more or less resolved. Suppose that c of the n coins are counterfeit. In [9], 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 [7], it is proved that this problem can be solved in at most $a \log_3 \binom{n}{c}$ measurements for some constant $a \leq 2 \log_2 3$. This result holds without any prior knowledge of c .

In [2, 1, 3, 8], 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 (asymptotically) $(2 + o(1)) \frac{\log n}{\log \log n}$ 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).

Table 1.1 below summarizes the current state of these problems.

Any tree of depth d with branching factor b has at most b^d leaves. Therefore, if a decision problem has N possible answers, each must be assigned to a distinct leaf of any valid decision tree, and thus any valid decision tree for this problem must have depth $d \geq \lceil \log_b N \rceil$. The column labeled “Info” (for informational complexity) below gives the lower bound obtained from this standard information theory argument. The columns labeled “Adaptive” and “Oblivious” give the known bounds on the adaptive and oblivious complexities of the various problems, respectively.

Problem	Info	Adaptive	Oblivious
Bad Coin Counting	$\log_3 n$	$\Omega(\log n), O((\log n)^2)^*$	$\Omega(\sqrt{n})\dagger, O(n)$
Bad Coin Parity	1	$\Omega(\log n)\dagger, O(\log n)^*$	$\Omega(\sqrt{n})\dagger, O(n)$
Bad Coin Location	$\log_3 n$	$\Theta(\log n)$ [5]	$\Theta(\log n)$ [6]
Bad Coin Determination	$\log_3 \binom{n}{c}$	$\Theta(\log_3 \binom{n}{c})$ [1, 2, 3, 8]	?
All Equal	1	$\Theta\left(\frac{\log n}{\log \log n}\right)$ [7]	?

Table 1.1. Informational, Adaptive, and Oblivious Complexities of Three Coin-Weighing Problems

*Folklore, described in section 7

† Proved in the current paper

Statements without asterisks or references are obvious.

1.2 Organization

The rest of this paper is organized as follows: In Section 2, we give some definitions to formalize the notion of coin-weighing problems. In Section 3, we prove a technical lemma, that we can limit ourselves to measurements which have an equal number of coins on the two pans of the balance. In Section 4, we prove our main result, the lower bound on the depth of oblivious decision trees for the Bad Coin Parity problem. In Section 5, we give an application of this lower bound to some statements about the size of threshold circuits. In Section 6, we use our main result to deduce a lower bound on the depth of adaptive decision trees for the Bad Coin Parity problem. Finally, in Section 7, we describe several algorithms (which seem to be folklore) that solve the Bad Coin Counting and Bad Coin Parity problems.

2 Preliminaries

An instance of a coin weighing problem \mathcal{P} is a weight assignment function $w : \{1, 2, \dots, n\} \rightarrow \{w_{bad}, w_{good}\}$, where $0 < w_{bad} < w_{good}$. $w(i)$ represents the weight of the i th coin in our set of n coins. Alternatively, we may think of an instance as a partition of $\{1, 2, \dots, n\}$ into disjoint sets G and B , where $w(i) = w_{good}$ for every $i \in G$, and $w(i) = w_{bad}$ for every $i \in B$.

A coin weighing problem \mathcal{P} is then just a “question” function on the set of such instances. That is, we wish to determine the value of $\mathcal{P}(w)$ using as few measurements on the balance as possible.

A measurement corresponds to two disjoint sets $L, R \subset \{1, 2, \dots, n\}$, where $i \in L$ if the i th coin is on the left pan of the balance, and $i \in R$ if the i th coin is on the right pan of the balance. Note that we do not require all coins to be used in every measurement. If we let W_L (W_R , resp.) be the combined weight of the coins on the left (right, resp.) side of the balance under a given weight assignment w , the result of the measurement m is

$$res(m, w) = \begin{cases} R & \text{if } W_L < W_R \\ E & \text{if } W_L = W_R \\ L & \text{if } W_L > W_R \end{cases}$$

A decision tree for a coin weighing problem is a tree in which each interior node is labeled with a measurement. Each interior node has three children, one for each of the three outcomes of the measurement. When faced with an instance of the problem, we traverse this tree, always moving to the node corresponding to the result of the previous measurement.

An oblivious decision tree cannot base these measurements on the outcome of a previous measurement, so all the interior nodes at a given depth will be labeled with the same measurement. Therefore, an oblivious decision tree corresponds to a list $L = \{m_1, m_2, \dots, m_s\}$ of measurements on the balance.

Definition 2.1. A decision tree for a particular problem is *sound* if each leaf of the tree determines a unique value of $\mathcal{P}(w)$.

An oblivious decision tree corresponding to a list of measurements L is sound if and only if, for any two weight assignments w_1, w_2 ,

$$((\forall m_i \in L) (res(m_i, w_1) = res(m_i, w_2))) \implies \mathcal{P}(w_1) = \mathcal{P}(w_2).$$

3 Eliminating unbalanced measurements

Definition 3.1. We say that a measurement is *balanced* if there are an equal number of coins on each side of the balance. Also, for every oblivious decision tree T , we define $bal(T)$ to be another oblivious decision tree which consists of all balanced measurements from T .

In this section, we show that we can limit ourselves to using only balanced measurements. In particular, we prove

Lemma 3.2. *The decision tree L for a coin-weighing problem \mathcal{P} is sound if and only if $bal(L)$ is also sound.*

This fact is well-known, but we include a proof for the sake of completeness.

Let the good (bad, resp.) coins all have weight w_{good} (w_{bad} , resp.). The statement of the problem assumes

$$w_{bad} < w_{good}.$$

The result of a weighing may depend on the relative weights of the good and bad coins. For example, two bad coins will outweigh one good coin if and only if $2 \cdot w_{bad} > w_{good}$. This leads to the following definitions:

Definition 3.3. We say that the weights are “close together” if

$$w_{bad} < w_{good} < \left(1 + \frac{1}{n}\right) \cdot w_{bad}.$$

Fact 3.4. In the case where the relative weights 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 L correctly deals with all close-together instances of the problem if and only if $bal(L)$ also does so.*

Proof. By Fact 3.4, in close-together instances, non-balanced measurements do not depend on the type of coins involved, so they yield no additional information. Therefore, if L correctly identifies the number of bad coins, so will $bal(L)$. Conversely, since $bal(L) \subset L$, L will certainly solve any problem that $bal(L)$ solves. \square

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

Proof. By Fact 3.5, the result of a balanced measurement does not depend on the relative values of w_{good} and w_{bad} . Since $bal(L)$ consists solely of balanced measurements, the results of its measurements depend only on the number and location of the bad coins. Therefore, if $bal(L)$ is sound on close-together instances, it is sound on all instances. Conversely, if $bal(L)$ is sound on all instances, it is sound on close-together instances. \square

Propositions 3.6 and 3.7 combine to give us Lemma 3.2.

Since $bal(L)$ has lesser or equal length than L , 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 a slight generalization of Theorem 1.1. By Lemma 3.2, we can limit ourselves to balanced measurements.

We can represent a possible assignment of weights to the coins by a $(0, 1)$ vector of length n , where a 0 (1 resp.) in the i th coordinate indicates that the i th coin is bad (good resp.) under that assignment. We can represent a measurement by a $(-1, 0, 1)$ vector of length n , where a 0 (1, -1 resp.) in the i th coordinate indicates that the i th coin is not measured (on the right pan of the balance, on the left pan of the balance, resp.) in that measurement.

Proposition 4.1. *A coin-vector c and a measurement-vector m cause the balance to tip right if $m \cdot c > 0$, left if $m \cdot c < 0$, and to balance if $m \cdot c = 0$.*

Proof. By Proposition 3.5 the result of the measurement depends only on which side of the pan has more good coins on it. $m \cdot c$ is equal to the number of good coins on the right pan of the balance minus the number of good coins on the left pan of the balance. Therefore, the balance behaves as stated. \square

Using these coin-vectors, we can consider our coin-weighing problem \mathcal{P} as a Boolean function.

Next, we define a graph $G = (V, E)$ with $V = \{0, 1\}^n$, and $E = \{(c, c') : \|c - c'\| = 1, \mathcal{P}(c) \neq \mathcal{P}(c')\}$. This graph is a subgraph of the n -dimensional hypercube. We note that there is an edge between two vertices exactly when the associated vectors differ in only one coordinate (or when the two associated assignments of weight to the coins differ only in the weight of one coin) and the problem $\mathcal{P}()$ has a different answer for these two inputs.

Observation 4.2. If two coin-vectors c_1, c_2 differ in only the i th coordinate, then for any measurement vector m ,

$$m \cdot c_1 - m \cdot c_2 = m \cdot (c_1 - c_2) = m \cdot e_i = 1, -1, \text{ or } 0.$$

Therefore, if $m \cdot c_1 > 0$, we know $m \cdot c_2 \geq 0$.

Now, suppose that $L = \{m_1, m_2, \dots, m_s\}$ is the list of measurements used in a sound oblivious decision tree.

Proposition 4.3. *If the oblivious decision tree using the measurement list $L = \{m_1, m_2, \dots, m_s\}$ is sound, then*

$$(c_1, c_2) \in E \implies \exists i : 1 \leq s, \text{ sign}(m_i \cdot c_1) \neq \text{sign}(m_i \cdot c_2).$$

Proof. If two vectors in V share an edge in E , they give different answers for $\mathcal{P}()$, and the decision tree must return different results for the two vectors. If the decision tree is to do this correctly some measurement must return different results for the two assignments. \square

This leads to a reasonably good test of the list L , which no sound list of measurements will fail. For each measurement $m \in L$, we delete E_m , the set of edges in G between two coin-vectors which would return different results on the balance. By Proposition 4.3, we must have no edges of G left when we have processed all the measurement vectors. Now, we are in a position to prove Theorem 1.1.

Definition 4.4. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function of n variables, and let $x \in \{0, 1\}^n$. For $i \leq i \leq n$, let x^i be x with its i th coordinate flipped. The *sensitivity of f at x* , $s_x(f)$, is the number of i such that $f(x^i) \neq f(x)$. The *average sensitivity of f* , $s_{ave}(f)$ is defined to be the average value of $s_x(f)$.

For the parity 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 parity function has average sensitivity n . Theorem 1.1 is then a special case of the following theorem:

Theorem 4.5. *Let f be a Boolean function of average sensitivity α . Then any oblivious coin-weighing procedure for f must use at least $c\alpha/\sqrt{n}$ measurements, for an absolute positive constant c .*

Proof. The number of vertices in V is 2^n , and the average degree of a vertex is α , so the total number of edges in G is

$$|E| = 2^{n-1}\alpha .$$

Let K_m be the set of vectors that return 0 under a given measurement m with k coins on each side of the balance. Also, for a given coin-vector c_0 and a given measurement m with $m \cdot c_0$, let $N_{m,c_0} = \{c \in V | m \cdot c \neq 0\}$

Each vector in K_m is connected to at most n other vectors by an edge in G . At most $2k$ of these vectors differ in one of the $2k$ coordinates that are involved in the measurement, so $|N_{m,c}| \leq 2k$, where k is the number of coins on either side of the balance in the measurement m .

By Observation 4.2, edges are deleted between vectors returning 0 under the given measurement and vectors returning something nonzero. Therefore

$$|E_m| = \sum_{c \in K_m} |N_{m,c}| \leq |K_m| \cdot 2k .$$

$|K_m|$ 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 .

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

The number of edges deleted by a measurement m of k coins against k coins is thus bounded like so:

$$|E_m| \leq \binom{2k}{k} 2^{n-2k} (2k) = 2^n \cdot 2k \frac{\binom{2k}{k}}{2^{2k}}.$$

In the following calculations, we use \sim to refer to asymptotic equality, and we use $a_n \lesssim b_n$ to mean $a_n \sim \min\{a_n, b_n\}$.

Since

$$2k \cdot \frac{\binom{2k}{k}}{2^{2k}} \sim \frac{2k}{\sqrt{\pi k}} = \frac{2}{\sqrt{\pi}} \sqrt{k} \leq \frac{2}{\sqrt{\pi}} \sqrt{\frac{n}{2}} = \sqrt{\frac{2}{\pi}} \sqrt{n},$$

we obtain

$$|E_m| \lesssim 2^n \sqrt{\frac{2}{\pi}} \sqrt{n}.$$

If s is the number of weighing operations, and m is the measurement vector in L for which $|E_m|$ is largest,

$$|E_m| \cdot s \geq |E| = 2^{n-1} \alpha$$

Therefore,

$$s \geq 2^{n-1} \alpha / |E_m| \gtrsim \frac{2^{n-1} \alpha}{2^n \sqrt{\frac{2}{\pi}} \sqrt{n}} = \sqrt{\frac{\pi}{8}} \frac{\alpha}{\sqrt{n}}$$

and so we have $s = \Omega(\alpha/\sqrt{n})$. □

5 Applications to Circuit Complexity

In this section, we use Theorem 4.5 to prove some fairly general lower bounds on the size of threshold circuits.

Definition 5.1. Let $M(n)$ be some integer-valued function of n . Let $CW(M(n))$ be the class of Boolean functions f that can be correctly recognized by an oblivious coin-weighing algorithm using $M(n)$ balanced measurements, with access to an arbitrary number of coins of known weight.

Note that the proof of Theorem 4.5 did not really preclude the use of coins of known weight. The only step of the proof in which this would have made a difference would be in our calculation of the size of K_m , and we could have used Vandermonde's identity there to get $\binom{2k}{t} 2^{n-2k}$ for some t between 0 and $2k$. This is maximized at $t = k$, so our bound would only get stronger if coins of known weight were used.

Definition 5.2. Let $Thresh(M(n))$ be the class of Boolean functions which can be computed by circuits composed of threshold, AND, and OR gates, where the first layer of the circuit above the inputs contains at most $M(n)$ gates.

Proposition 5.3. $Thresh(M(n)) \subseteq CW(M(n))$

Proof. Firstly, by adding hard-wired inputs, we can simulate AND and OR gates by threshold gates. So we can assume that all the gates on the first layer are threshold gates.

Now, we wish to make all the threshold gates on the first layer balanced MAJORITY gates, in the sense that they receive an equal number of negated and non-negated literals. To do this, we augment the gates with negated and non-negated hardwired inputs of 0 and 1 in the proportion that we need them. This corresponds to adding coins of known weight to the balance pans in the coin-weighing model.

Next, we note that a balanced MAJORITY gate is true if and only if the corresponding measurement is true. (In the corresponding measurement, negated variables would correspond to coins on one side of the balance, and non-negated variables to coins on the other side.)

Therefore, if we can compute the function given the results of the measurements, we can compute it using the output of the MAJORITY gates. \square

This yields the following theorem:

Theorem 5.4. *For any function f of average sensitivity α , we have that $f \notin \text{Thresh}(\alpha/\sqrt{n})$.*

Thus functions of high average sensitivity require many gates on their first level.

6 The Lower Bound on Adaptive Decision Trees

In this section, we use Theorem 1.1 to prove Corollary 1.2. This follows from a more 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 $L = \text{list}(T)$ 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. *If T is a sound adaptive decision tree, $L = \text{list}(T)$ is a sound oblivious decision tree.*

Proof. Suppose T is a sound adaptive decision tree. If c_1 and c_2 are coin-vectors which disagree on the number of bad coins, some measurement on some branch of T must return a different result for c_1 and c_2 . This measurement is in T , so there is a measurement in $L = \text{list}(T)$ which returns different results for c_1 and c_2 . Since this holds for every pair of coin-vectors which disagree on the number of bad coins, L is a sound oblivious decision tree. \square

Proposition 6.3. *Let b represent the branching factor of a decision tree. If the oblivious complexity of the decision problem is X , and the adaptive complexity of the same problem is Y , $Y \geq \log_b X$.*

Proof. Let T be an optimal adaptive decision tree, of depth Y . Let L be a list of all of the measurements used in any sub-branch of 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 $|L| = \sum_{d=0}^{Y-1} b^d \leq b^Y$. By Proposition 6.2, we know that L is sound as an oblivious decision tree if T is sound. Therefore $|L| \geq X$. Thus $b^Y \geq X$, which yields the desired result. \square

Now we prove Corollary 1.2.

Proof of Corollary 1.2. Let X be the oblivious complexity of a coin-weighing problem \mathcal{P} , of average sensitivity α . $X = \Omega(\alpha/\sqrt{n})$ by Theorem 1.1. For sufficiently large n and some constant C , $X \geq C\alpha/\sqrt{n}$. So (using \sim to denote asymptotic equality)

$$\log_3(X) \geq \log_3(C\alpha/\sqrt{n}) = \log_3(\alpha) - \frac{1}{2} \log_3(n) + O(1) \sim \log_3(\alpha) - \frac{1}{2} \log_3(n)$$

and thus by Proposition 6.3, the adaptive complexity of any coin weighing problem with average sensitivity α is at least (asymptotically) $\log_3(\alpha) - \frac{1}{2} \log_3(n)$. \square

7 A $O((\log n)^2)$ Adaptive Decision Process for Parity

The results in this section appear to be folklore ¹, but we include them for lack of a convenient reference. This result, together with Corollary 1.2 demonstrate the contrast between the adaptive and oblivious complexities of the Bad Coin Counting Problem.

Both the Bad Coin Counting Problem and the Bad Coin Parity problem have efficient adaptive decision trees, which are related to another problem, the Bad Coin Splitting problem. In this section, we describe a depth $O(\log n)$ adaptive decision tree for the Bad Coin Parity problem and a depth $O((\log n)^2)$ adaptive decision tree for the Bad Coin Counting problem. Compare these with the $\Omega(\log n)$ lower bound of Corollary 1.2 on the adaptive complexity of both problems.

Notation 7.1. We will use the notation $\text{insert}(a, S)$ to denote the operation of inserting element a in set S , the notation $\text{insert}(S, T)$ to denote the operation of inserting all the elements of the set S in the set T , and the notation $a := \text{remove}(S)$ to denote the operation of setting the variable a equal to an arbitrary element of S and removing that element from S .

¹This problem [4] has been assigned to and solved by generations of students in Professor Babai's Algorithms class, but Professor Babai does not know the original source.

7.1 Determining weight of all coins

A number of the procedures in the following sections make use of a rather trivial algorithm to determine the weight of all coins in a given set, which is based on the following observation. This procedure solves the Bad Coin Counting problem obliviously in n weighings.

Observation 7.2. If a set T contains k coins, and we weigh each coin c_i against the next coin c_{i+1} , then either all weighings will come out balanced (in which case we know that all the coins have the same weight) or at least one weighing will come out unbalanced. Suppose that we know that c_i weighs less than c_{i+1} . Then we know that the lighter coin is bad, and that the heavier coin is good. We also know the weight of every other coin, since there are only two possible weights, and we know whether any two adjacent coins have the same weight or different weights.

This observation yields the following procedure:

Pseudocode 7.3. *A procedure for determining the weight of all coins. Takes in a set S of coins and returns the set of bad coins. Note that we are assuming that at least one of the coins is bad.*

```
0  procedure weighall( $S$ )
1  Initialize:  $G := \emptyset, B := \emptyset, U := \emptyset$ 
2   $d := \text{remove}(S)$ 
3  for( $c$  a coin in  $S$ ) do
4      Weigh  $c$  against  $d$ 
5      if(weight( $c$ ) > weight( $d$ )) do
6          insert( $c, G$ )
7          if( $d \in U$ ) insert( $U, B$ )
8      end(if)
9      if(weight( $c$ ) < weight( $d$ )) do
10         insert( $c, B$ )
11         if( $d \in U$ ) insert( $U, G$ )
12     end(if)
13     if(weight( $c$ ) = weight( $d$ )) do
14         if( $d \in G$ ) insert( $c, G$ )
15         if( $d \in B$ ) insert( $c, B$ )
16         if( $d \in U$ ) insert( $c, U$ )
17     end(if)
18      $d := c$ 
19 end(for)
20 return  $B$ 
```

7.2 Finding a coin of known weight

In the following sections, it will occasionally be of value to find a coin of known weight.

Proposition 7.4. *Given n coins, at least one of which is bad, we can locate a bad coin in (asymptotically) $2 \log_2 n$ comparisons on the balance.*

Throughout this paper, we have been assuming that at least one of the coins is bad. Therefore, we can mount a search for a bad coin using this assumption and the following observation.

Observation 7.5. Suppose that we know that a set S contains at least one bad coin. If S contains an even number of coins, we can divide S into two sets of equal size and weigh them against each other. If one is lighter, it must contain a bad coin; otherwise, both must contain at least one bad coin.

This observation lead to the following decision process for finding one bad coin:

Pseudocode 7.6. *Finding one bad coin*

```
0  procedure badcoin( $S$ )
1  Initialize:  $S :=$  set of coins,  $T := \emptyset$ 
2  while  $|S| > 1$  do
3      If  $|S|$  is odd, take one coin out of  $S$  and put it in  $T$ .
4      Split  $S$  into  $S_1 \cup S_2$  arbitrarily
5      Weigh  $S_1$  against  $S_2$ 
6      Let  $S :=$  the lighter of  $S_1$  and  $S_2$ 
7  end(while)
8  Let  $d$  be the sole coin in  $S$ .
9  return remove(weighall( $\{d\} \cup S$ )) (See Pseudocode 7.3)
```

Proof of Proposition 7.4. Every time the **while** loop in lines 2-7 runs, S is halved in size. Therefore, since $|S|$ is n initially, this loop runs at most $\lceil \log_2 n \rceil$ times, with one measurement in each iteration. T can grow by at most one coin in each iteration of the **while** loop in lines 2-7, so T can have at most $\lceil \log_2 n \rceil$ elements. The call to procedure weighall in line 9 requires $|T|$ measurements, so this is an additional $\lceil \log_2 n \rceil$ measurements, for a total of $2 \cdot \lceil \log_2 n \rceil$ measurements.

By Observation 7.5, after every iteration of the loop, there is a bad coin in either S or T . Thus, by Observation 7.2, after we perform the weighings in the procedure weighall, we either know the location of a bad coin, or we know that all the coins in S and T have the same weight. In this case, since at least one is bad, they are all bad. Therefore, after performing the weighings specified by Pseudocode 7.6, we know the location of at least one bad coin. \square

Remark 7.7. We can improve this procedure to use only $\log_2 n + O(\log \log n)$ measurements. Instead of calling the procedure $\text{weighall}(\{d\} \cup S)$ in line 9, we can call $\text{badcoin}(\{d\} \cup S)$, which will require only $O(\log \log n)$ measurements, by the above analysis.

7.3 The Bad Coin Splitting Problem and the Bad Coin Parity Problem

The adaptive decision tree is constructed by way of the following auxiliary problem:

Problem 7.8 (Bad Coin Splitting Problem). Given n coins, each of weight either w_{good} or w_{bad} , split the coins into two piles of as equal weight as possible. That is, if n is even and b , the number of bad coins, is also even, then each pile should have $n/2$ coins, including $b/2$ bad ones. If n is even but b is odd, divide the coins into two piles of $n/2$ coins each, so that one pile contains $(b+1)/2$ bad coins, and the other pile contains $(b-1)/2$ bad coins. If n is odd, remove one coin and split the rest as above.

Proposition 7.9. *The Bad Coin Splitting problem can be solved in $O(\log n)$ measurements on the balance.*

The decision tree is based on the following observation. Note that, because the setup of the problem, we can assume we are trying to split an even number of coins.

Observation 7.10. Suppose that all the coins except for two (c_1 and c_2) are divided into 2 subsets A and B of equal size. Furthermore, suppose that the set $\{c_1\} \cup A$ is heavier than the set $\{c_2\} \cup B$, but that the set $\{c_2\} \cup A$ is lighter than the set $\{c_1\} \cup B$. Then the coin c_1 is good, the coin c_2 is bad, and the sets A and B have equal weight. Thus, $A \cup \{c_1\}$ and $B \cup \{c_2\}$, and $A \cup \{c_2\}$ and $B \cup \{c_1\}$, are both valid splittings of the coins.

The decision process consists of looking for such sets. We can best describe this precisely in terms of some variables which we will now define. Let n be even.

Let $A_k = \{1, \dots, k\} \cup \{\frac{n}{2} + k + 1, \dots, n\}$, and let $B_k = \{k + 1, \dots, \frac{n}{2} + k\}$. A_k and B_k are disjoint, each has size $n/2$, and $A_k \cup B_k = \{1, \dots, n\}$. Let

$$s_k = \sum_{i \in A_k} w_i - \sum_{i \in B_k} w_i = \sum_{i=1}^k w_i - \sum_{i=k+1}^{\frac{n}{2}+k} w_i + \sum_{i=\frac{n}{2}+k+1}^n w_i,$$

where w_i is the weight of the i th coin. Then A_k is heavier than (lighter than, of the same weight as, resp.) B_k if s_k is positive (negative, zero, resp.).

Let m_k be the measurement of A_k against B_k . Since $|A_k| = |B_k|$, m_k is a balanced measurement. The result of m_k depends only on the sign of s_k . Since $A_{n/2} = B_0$ and $A_0 = B_{n/2}$, $s_{n/2} = -s_0$.

If $s_{n/2} = 0$, the measurement $m_{n/2}$ will balance, giving a splitting of the coins. Otherwise, $s_{n/2} \neq 0$, so we have $s_0 < 0 < s_{n/2}$ or $s_{n/2} < 0 < s_0$. This means that there is some ℓ where s_ℓ and $s_{\ell-1}$ do not have the same sign. If s_ℓ or $s_{\ell-1}$ is zero, then the corresponding measurement will balance, giving a splitting of the coins. Otherwise, two measurements which differ only in the placement of one coin give opposite results. By Observation 7.10, we now have a valid splitting of the coins.

Therefore, we can solve the splitting problem in $\log_2(n)$ measurements using binary search to find such an l . At each step, we pick a k in the middle of the interval $[L, U]$ in which we are searching for l . If m_k balances, we are done. If m_k agrees with m_0 , then we know there is such an l between $k + 1$ and U . If m_k agrees with $m_{n/2}$, then there is such an l between L and $k - 1$. Repeating this process, we will find such an l , which will give us a splitting of the coins. This process is implemented by the following pseudocode:

Pseudocode 7.11. *Solving the Bad Coin Splitting problem. This procedure return a 5-tuple: (A, B, f, c_1, c_2) , where A and B are sets of nearly equal weight as requested in the problem, f is a flag which tells whether they are of equal weight, and c_1 (c_2 resp.) are a good coin (bad coin resp.) from the heavier (lighter resp.) of A and B , or null if A and B are of equal weight.*

```

0  procedure split( $S$ )
1  Initialize:  $k := 0, L = 0, U = n/2$ 
2  Let  $A_0 = \{\frac{n}{2} + 1, \dots, n\}$ 
3  Let  $B_0 = \{1, \dots, \frac{n}{2}\}$ 
4  Measure the coins with labels in  $A$  against the coins with labels in  $B$ .
   Let  $R$  be the result of this weighing
5  If ( $R = \text{"equal"}$ ) return  $(A_0, B_0, \text{equal}, \text{null}, \text{null})$ 
6  while ( $L < (U - 1)$ ) do
7     Let  $k = \lfloor \frac{L+U}{2} \rfloor$ 
8     Let  $A_k = \{1, \dots, k\} \cup \{\frac{n}{2} + k + 1, \dots, n\}$ 
9     Let  $B_k = \{k + 1, \dots, \frac{n}{2} + k\}$ 
10    Measure the coins with labels in  $A_k$  against the coins with labels in  $B_k$ .
     Let  $R'$  be the result of this weighing.
11    If  $R = R'$ , let  $U = k$ 
12    If  $R = \text{opposite of } R'$ , let  $L = k$ 
13    If  $R = \text{"equal"}$  return  $(A_k, B_k, \text{equal}, \text{null}, \text{null})$ 
14 end(while)
15 return  $(A_L, B_L, \text{notequal}, c_L, c_{L+1})$ 

```

Proposition 7.12. *The Bad Coin Parity problem can be solved using $2 \log_2 n$ measurements on the balance.*

Proof. If we have an even number of coins, we can split the coins exactly evenly if and only if there are an even number of bad coins. Therefore, if we have an

even number of coins, a solution to the splitting problem is also a solution to the Bad Coin Parity problem. If we have an odd number of coins, then we can use the algorithm from Pseudocode 7.6 to find one bad coin. After removing this coin, we can deal with the rest using the splitting problem. Therefore, we can solve the Bad Coin Parity problem using asymptotically $\log_2 n$ measurements for even values of n , and $2 \cdot \log_2 n$ measurements for odd values of n . \square

7.4 Using the Splitting Problem

In this section, we discuss how to use the solution of Problem 7.8 to solve the Bad Coin Counting problem.

Proposition 7.13. *The Bad Coin Counting problem can be solved adaptively in $O((\log n)^2)$ measurements.*

Pseudocode 7.14. *We can determine the number of bad coins using the following recursive algorithm, which makes use of the solution to the Splitting Problem.*

```

0 Initialize:  $k := 0$ ,  $result := 0$ ,  $S :=$  set of coins,  $T := \emptyset$ 
1  $c_0 = \text{badcoin}(S)$  (See Pseudocode 7.6)
2  $S := \text{remove}(c_0, S)$ 
2 procedure count( $S$ )
3   if  $|S| > 1$  do
4     if ( $|S|$  is odd) do
5        $c = \text{remove}(S)$ 
5       insert( $c, T$ )
5     end(if)
5     ( $S_1, S_2, f, c_1, c_2$ ) = split( $S$ ) (See Pseudocode 7.11)
6     if (flag = notequal) do
7       insert( $c_1, c_2, T$ )
8     Let  $k := \text{weighall}(T \cup \{c_0\}) - 1$ . (See Pseudocode 7.3)
9      $T := \emptyset$ 
7     end(if)
10    return  $2 * \text{count}(S_1) + k$ 
11  end(if)
12  elsedo
13    return  $-\text{weighall}(S \cup \{c_0\}) - 1$ 
15 end(procedure) count)
16 result := count( $S - \{c_0\}$ ) + 1

```

Proof of Proposition 7.13. Finding the bad coin in line 1 requires at most $\lceil \log_2 n \rceil$ measurements, as described in the analysis of Pseudocode 7.6. Every time that the procedure “count” is run, we use at most $\lceil \log_2 n \rceil$ measurements in line 5, and at most 3 measurements in line 7. Since S is halved in size every time the

procedure is run, it can be run at most $\lceil \log_2 n \rceil$ times. Therefore, the total number of measurements used by this algorithm is asymptotically $(\log_2 n) \cdot (\log_2 n)$, or $O((\log n)^2)$. \square

References

- [1] N. Alon and D. N. Kozlov, “Coins with arbitrary weights,” *J. Algorithms*, vol. 25, no. 1, pp. 162–176, 1997.
- [2] N. Alon, D. N. Kozlov, and V. H. Vu, “The geometry of coin-weighing problems,” in *FOCS*, 1996, pp. 524–532.
- [3] N. Alon and V. H. Vu, “Anti-hadamard matrices, coin weighing, threshold gates, and indecomposable hypergraphs,” *J. Comb. Theory, Ser. A*, vol. 79, no. 1, pp. 133–160, 1997.
- [4] L. Babai, “Algorithms,” homework problem, University of Chicago, <http://www.classes.cs.uchicago.edu/archive/2005/winter/27200-1/assignments/03.pdf>.
- [5] A. Born, C. A. J. Hurkens, and G. J. Woeginger, “How to detect a counterfeit coin: Adaptive versus non-adaptive solutions,” *Inf. Process. Lett.*, vol. 86, no. 3, pp. 137–141, 2003.
- [6] F. J. Dyson, “The problem of the pennies,” *The Mathematical Gazette*, vol. 30, no. 29, pp. 231–234, 1946.
- [7] X.-D. Hu, P. D. Chen, and F. K. Hwang, “A new competitive algorithm for the counterfeit coin problem,” *Inf. Process. Lett.*, vol. 51, no. 4, pp. 213–218, 1994.
- [8] D. N. Kozlov and V. H. Vu, “Coins and cones,” *J. Comb. Theory, Ser. A*, vol. 78, no. 1, pp. 1–14, 1997.
- [9] L. Pyber, “How to find many counterfeit coins?” *Graphs and Combinatorics*, vol. 2, no. 1, pp. 173–177, 1986.