Lower Bound of Coin Counting Problem

ARITRA BHADURI

M.Tech CS

STATISTICAL INDIAN INSTITUTE

भिन्नेष्वैक्यस्य दर्शनम्

UNITY IN DIVERSITY

Supervisor: Sourav Chakraborty

A thesis submitted in fulfilment of
the requirements for the degree of
M.Tech

ACMU
Indian Statistical Institute
Kolkata

10 July 2020

Abstract

We have n coins of two weights. We also have a balance scale to measure the weights of the coins. The objective is to find the number of heavy coins with as few measurement as possible. This problem is known as "coin-counting problem". A sub-problem of this problem is, optimally find if the number of the heavy coins is even or odd. This problem is known as "coin-parity problem". It was first proposed by "Laszlo Babai"of "University of Chicago". There is a known adaptive algorithm which solves the coin-counting problem in $O(log^2 n)$ time. By modifying that algorithm we can also solve the parity problem in $O(logn)$ time.
The oblivious lower bound of coin-counting problem is $O(\sqrt{n})$. This result was proved by "Eric Purdy" on the paper "Lower Bound of coin-counting problem" (1). In the first section of this thesis we have discussed about oblivious lower bound of the counting problem and showed a tight adaptive $\theta(logn)$ bound on coin parity problem. All these result are based on the eric purdy's "Lower Bound of coin-counting problems" paper.
There is a trivial adaptive lower bound of the coin-counting problem which is $logn$. As we can see adaptive coin-counting problem does not have a tight bound. The objective of this thesis is to give an improvement on the lower bound of adaptive coin counting problem. In chapter 4 we have given a proof of the adaptive lower bound of coin-counting problem is $log2n + loglogn$. We interpreted each one of the coin configuration into a boolean-vector. The main idea is to check which of these Boolean-vectors can go to the same leaf of the decision tree . This creates a partition of Boolean-vectors. By counting the partitions give us the total number of leaf nodes in the decision tree. Now taking log of this number gives us the height of the decision tree and that's our required lower bound.

Acknowledgements

I would like to express my special thanks of gratitude to my guide Prof. Sourav Chakraborty who gave me an opportunity to do this wonderful project , It also gave me glimpse of how to do Research and I came to know about so many new things in this area.

Contents

List of Figures

CHAPTER 1

Introduction

We have n coins and a balance scale. All the coins look alike, only the weights differ. Among the n coins, some are heavy and others are light. All the heavy coin weights the same so as the light coins. Let the weight of heavy coins be w_1 and the light coins be w_2. The main objective of the problem is to find the number of heavy coins using the balance scale as few times as possible. This problem is called **coin counting problem**.

Since the balance scale can give three outcomes, right or left inclined and balanced, we represent this problem as a decision tree problem. Decision tree can be of two types **Adaptive** and **Oblivious** decision tree. In an oblivious decision tree, each measurement does not depend on the previous measurements. Hence in each depth, all the measurements are equal. Unlike oblivious in adaptive decision tree each measurement depends upon its previous measurements so in each depth of the decision tree there can be different measurements. There are known adaptive and oblivious algorithms that solve the coin counting problem in $O(log^2n)$ and n time respectively.

Like the coin counting problem, there is a sub-problem of this problem known as **coin parity problem**. It states how optimally(as few measurements as possible) we can check if the number of heavy coins is even or odd. We also have adaptive and oblivious algorithm to solve the parity problem in $logn$ and n time respectively.

The main issue of this problem is that none of the adaptive and oblivious bounds for coin counting problem is tight. The adaptive lower bound is $logn$ and oblivious lower bound is $\Omega(\sqrt{n})$ (1). For parity problem only the adaptive bound is tight, $logn$. Oblivious lower bound for the parity problem is $\Omega(\sqrt{n})$.

problem	Adaptive	Oblivious
Counting	$log2n + loglogn$	$\Omega(\sqrt{n})$ (1)
Parity	$\Omega(logn)$	$\Omega(\sqrt{n})$ (1)

TABLE 1.1: Lower Bound of Adaptive and oblivious Approach

problem	Adaptive	Oblivious
Counting	O$(logn)^2$	n
Parity	O$(logn)$	n

TABLE 1.2: Upper Bound of Adaptive and Oblivious Algorithm

As we can see there is an exponential gap between the upper and lower bound for the coin counting problem. The main objective of this thesis is to give a better adaptive lower bound to reduce the gap.

We interpreted each coin configurations into a Boolean-vector. The idea is to check which of these vectors can go to the same leaf node of the decision tree. This creates partitions in the boolean-vectors. Vectors corresponding to the same partition goes to the same leaf node. So there is one-one relation between number of partitions of vectors and the number of leaf node. By analysing certain properties of the vectors we have proved the number of leafs nodes in the decision tree is at-least $2nlogn$. Therefore the lower bound is $log(2nlogn)$.

THEOREM 1.0.1. *Adaptive lower bound of coin-counting problem is* $log2n + loglogn$.

Overview: In chapter 2 we have introduced the required terminologies. In chapter 3 we discussed about the oblivious lower bound of the coin counting problem (1). A consequence of this result is oblivious lower bound of the parity problem. We also have given an algorithm for the adaptive coin counting problem. Chapter 4 is based on our original work. Here we have discussed about the properties of the boolean-vectors and using that proved the lower bound of coin-counting problem is $log(2nlong)$.

CHAPTER 2

Notation

Let's represent the n-coin set as a $\{0,1\}^n$ Boolean Vectors. Where 1 denotes a heavy coin and 0 denotes a light coin. Let $\overrightarrow{x}$ be such a vector where $x_i = 1$ if the ith position in the coin configuration has a heavy coin and $x_i = 0$ otherwise. We can also represent each measurement as a form of a vector too. Let $\overrightarrow{m}$ be such a measurement,

$$m_i = \begin{cases} 1 & \text{if ith coin is in the right side of the balance scale} \\ -1 & \text{if ith coin is in the left side of the balance scale} \\ 0 & \text{if ith coin doesn't take part in the measurement} \end{cases}$$

$$\overrightarrow{m}.\overrightarrow{x} = \Sigma_{i=1}^{n} m_i.x_i = \begin{cases} > 0 & \text{implies balance scale is right inclined} \\ = 0 & \text{implies balance scale is balanced} \\ < 0 & \text{implies balance scale is left inclined} \end{cases}$$

So we can reinterpret this problem as a ternary decision tree problem, whose each branch is a measurement and leaf nodes are the number of heavy coins. Our objective is to minimize the height of this decision tree.

Definition : $f : \{0,1\}^n \longrightarrow R$ be a boolean function where $\overrightarrow{x} \in \{0,1\}^n$ we denote $\overrightarrow{x^i}$ as a vector where all but the ith entry is different from $\overrightarrow{x}$. We define the **sensitivity** of f at $\overrightarrow{x}$ is the number of i such that $f(\overrightarrow{x^i}) \neq f(\overrightarrow{x})$. We denote this as $\sigma_x(f)$. The average sensitivity of f at $\overrightarrow{x}$ is denoted as $\alpha(f) = E(\sigma_x(f))$.

Observe the average sensitivity for both the parity and counting function is n, where n is the length of the Boolean vectors. For any x and i, $1 \leq i \leq n$, $f(\overrightarrow{x^i}) \neq f(\overrightarrow{x})$ because each time only one co-ordinate of the vector is changed from 0 to 1 or vice versa so the number of 1 is increasing or decreasing by one. So for each $\overrightarrow{x}$ $\sigma_x(f) = n$. Hence $\alpha(f) = n.2^n/2^n = n.$

Now lets define what's called a **sensitivity graph** of f.

Definition : Let G_f be a graph $G_f = (V, E_f)$ where $V = \{0,1\}^n$ and $(x, y) \in E_f$ such that:

- $y = x^i$ for some i
- $f(x) \neq f(y)$

Note: Degree of each $\overrightarrow{x}$ where $\overrightarrow{x} \in V$ is $\sigma_x(f)$. So the sum of the degree of G_f is $\Sigma_x \sigma_x(f) = 2^n\alpha(f)$. Therefore $|E_f| = 2^{n-1}\alpha(f)$.

CHAPTER 3

Prior Work

3.1 Oblivious Lower Bound of Coin Counting Problem

In this section we have discussed about the oblivious lower bound of coin counting problem. A corollary of this problem is coin parity problem. The proof is based on a paper called **Lower Bound for Coin-Weighting Problem** by **Eric Purdy** (1).

LEMMA 1. *If* $g_1, ..., g_n : \{0,1\}^n \to \chi$ *and* h *be a function* $h : \chi \to R$*. Let* f *be real valued boolean function such that* $f(x) = h(g_1(x)....g_n(x))$ *then* $\alpha(f) \leq \Sigma_i \alpha(g_i)$*. Where* $\alpha(f)$ *denotes the average sensitivity of* f*. (1)*

PROOF. $f(x) \neq f(y)$ implies $h(g_1(x)....g_n(x)) \neq h(g_1(y)....g_n(y))$. So $\exists$ an i where $i \in (1, ..n)$ such that $g_i(x) \neq g_i(y)$. So if $(x, y) \in E_f$ then $(x, y) \in \cup_i E_{g_i}$ for some i $1 \leq i \leq n$. Therefore $|E_f| \leq \Sigma_i |E_{g_i}|$. But we know $|E_f| = 2^{n-1}\alpha(f)$. Hence $\alpha(f) \leq \Sigma_i \alpha(g_i)$. ■

Remark : Here E_f denotes the edge of the sensitivity graph of f.

Let f be a real boolean function $f : \{0,1\}^n \longrightarrow R$. $f(\overrightarrow{c})$ denotes the number of 1 in $\overrightarrow{c}$ where $\overrightarrow{c}$ denotes a Boolean vector. Let m be a measurement function such that $m : \{0,1\}^n \longrightarrow \chi$. In oblivious decision tree, measurements are independent of each other. So in oblivious scenario we can write them as a sequence of measurement function $m_1....m_r$ for any $x \in \{0,1\}^n$. Therefore our main objective is to find the least value for r such that $f(\overrightarrow{c}) = h(m_1(\overrightarrow{c})...m_r(\overrightarrow{c}))$ holds.

By applying the above lemma we can observe that $\alpha(f) \leq \Sigma_i \alpha(m_i) = r\alpha(m)$, so

$\alpha(f)/\alpha(m) = r$. We already know that when f is counting function $\alpha(f) = n$, consequently by computing $\alpha(m)$ we can directly get a lower-bound on r. We also know that when f is parity function then also $\alpha(f) = n$. Similarly by getting $\alpha(m)$ we can find the oblivious lower bound of the coin parity problem too.

3.1.1 Average Sensitivity of measurement function

LEMMA 2. *Average Sensitivity of measurement function is O($\sqrt{n}$). (1)*

PROOF. Let $\overrightarrow{c}$ and $\overrightarrow{d}$ be two boolean vectors such that $\overrightarrow{d}$ differs with $\overrightarrow{c}$ only at ith place therefore $\overrightarrow{c^i} = \overrightarrow{d}$ and $\overrightarrow{m}$ be a measurement vector. So if $0 < \overrightarrow{m}.\overrightarrow{c}$ then $0 \leq \overrightarrow{m}.\overrightarrow{d}$, because by only changing one coin from heavy to light or vice versa, scale can't go from left heavy to right heavy directly . So the scale gets balanced or stay inclined at the same side. This implies if $(\overrightarrow{x}, \overrightarrow{y}) \in E_m$ then either $\overrightarrow{m}.\overrightarrow{x} = 0$ or $\overrightarrow{m}.\overrightarrow{y} = 0$.
Let $C_m = \{\overrightarrow{c}|\overrightarrow{m}.\overrightarrow{c} = 0\}$ and $N_m(\overrightarrow{c}) = \{\overrightarrow{d}|(\overrightarrow{c}, \overrightarrow{d}) \in E_m\}$. Let there are k coins in each side of the weight scale. Therefore there are $2k$ co-ordinates of $\overrightarrow{c}$ such that by flipping these co-ordinate at a time results $\overrightarrow{m}.\overrightarrow{c^i} \neq \overrightarrow{m}.\overrightarrow{c}$. Hence $|N_m(\overrightarrow{c})| = 2k$. Observe all the edges of the sensitivity graph have edges between some $\overrightarrow{c} \in C_m$ and some $\overrightarrow{d} \in N_m(\overrightarrow{c})$. Thus $|E_m| = 2k.|C_m|$. We can reinterpret C_m as number of ways we can choose j heavy coins among k coins on each side of the scale such that the scale becomes balanced, $0 \leq j \leq k$. There are $n - 2k$ coins which are not on the scale so we can define them arbitrarily.

$$\begin{aligned} |C_m| &= \Sigma_{j=0}^{k} \binom{k}{j}^2 2^{n-2k} \\ &= \Sigma_{j=0}^{k} \binom{k}{j} \binom{k}{k-j} 2^{n-2k} \\ &= \binom{2k}{k} 2^{n-2k} \end{aligned} \tag{3.1}$$

Therefore

$$\begin{aligned} |E_m| &= \binom{2k}{k} 2^{n-2k}.2k \\ &= 2^n 2k \frac{\binom{2k}{k}}{2^{2k}} \end{aligned} \tag{3.2}$$

we know $|E_m| = 2^{n-1}\alpha(\overrightarrow{m}) = 2^n 2k\frac{\binom{2k}{k}}{2^{2k}}$. Therefore $\alpha(m) = 4k\frac{\binom{2k}{k}}{2^{2k}} = O(\sqrt{n})$. Observe by induction we can show $\frac{\binom{2k}{k}}{2^{2k}} \leq \frac{1}{\sqrt{k}}$. So $2k.\frac{\binom{2k}{k}}{2^{2k}} \leq 2k.\frac{1}{\sqrt{k}} \leq z.\sqrt{n}$, here z is an arbitrary constant.
Hence $\alpha(\overrightarrow{m}) = 4k\frac{\binom{2k}{k}}{2^{2k}} = O(2k.\frac{1}{\sqrt{k}}) = O(\sqrt{n})$.
We know $\alpha(f) = n$ where f is counting or parity function. By using $\alpha(f) = r\alpha(\overrightarrow{m})$ we get $r = \Omega(\sqrt{n})$. This shows that oblivious lower bound of coin counting problem is $\Omega(\sqrt{n})$. ■

COROLLARY 3.1.1. *Oblivious lower bound of coin parity problem is $\Omega(\sqrt{n})$.*

Note: $\alpha(f) = r\alpha(m)$ this is holds not only for parity and counting function but for any boolean function. So we can generalize the above result.

THEOREM 3.1.2. *Let $f : \{0,1\}^n \to R$ be a boolean function. Let C be a set of coins, $C = \{0,1\}^n$. Heavy coins are denoted by 1 and light coins are denoted by 0. We can determine $f(x)$ by applying measurements to C. Then any oblivious algorithm for determining $f(x)$ have to use $\Omega(\alpha(f)/\sqrt{n})$ measurements. (1)*

3.2 Adaptive Upper Bound of Coin Counting Problem

In this section, we will give an adaptive algorithm for the coin counting problem of order $O(log^2(n))$. The main idea is to divide the coin set into two equal parts such that we can get a balanced measurement on the scale. But we always can't find such a balance measure like when the heavy coins are odd in number. So instead of two parts lets divide the coin set into three parts. Let C be the total coin set. We partition C into C_1, C_2, C_3 such that $C_1 \& C_2$ contains same number of heavy coins and $|C_1| = |C_2|$, $|C_3| \leq 2$.

- When C contains even number of heavy coins and $|C|$ is even then $|C_3| = 0$
- When C contains even number of heavy coins and $|C|$ is odd then $|C_3| = 1$
- When C contains odd number of heavy coins and $|C|$ is odd then $|C_3| = 1$
- When C contains odd number of heavy coins and $|C|$ is even then $|C_3| = 2$

If we replace C with C_1 or C_2 and continue the process until only one coin left. Each time number of coins is reduced by half, so this process can go up to $O(logn)$ time.
If there exists a partition C_1 and C_2 such that measurement is balanced then C has even number of heavy coins, and if that's not the case then C has odd number of heavy coins. Therefore finding the partition C_1 , C_2 is same as solving the parity problem. If we can solve the parity problem in $O(logn)$ time. Then we are done.

PROPOSITION 3.2.1. *Given n coins we can find at-least a heavy coin in $(logn)$ time.*

Algorithm 1 *Finding at-least one heavy coin*

```
1: while $|C|$ > 1 do
2:     if $|C|$ is odd then
3:         Remove a arbitrary coin $C_x$ from $C$
4:     end if
5:     Divide $C$ into $C_1$ and $C_2$ so that $|C_1| = |C_2|$
6:     Compare $C_1$ and $C_2$ in the balance scale
7:     $C \longleftarrow$ {heaver of $C_1$ and $C_2$} $\cup C_x$
8: end while
9: return $C$
```

PROOF. We are assuming there exists at least one heavy coin. C be the total coin set. If $|C|$ is even then divide into C_1 and C_2 such that $|C_1| = |C_2|$. If $|C|$ is odd then remove a coin arbitrarily C_x and do the above. Now replace C with $C_1 \cup C_x$ if the balance scale is inclined towards C_1 else replace with C with $C_2 \cup C_x$ and continue the process. Each time the total number of coins reducing by half. So this process will continue $logn$ times. Each time we are choosing the side which has equal or more number of heavy coins than the other side. So finally this process will return a heavy coin. Hence we can find a heavy coin in $(logn)$ time. ■

3.2.1 Coin Parity Problem

PROPOSITION 3.2.2. *Coin parity problem can be solved in $2logn$ time.*

Algorithm 2 *Coin Parity Problem*

```
1: $n \longleftarrow |C|$, $L \longleftarrow 1$, $U \longleftarrow n/2$
2: $C_1 \longleftarrow 1, ...., n/2$
3: $D_1 \longleftarrow n/2+1, ...., n$
4: $M = Compare(C_1, C_2)$
5: if $M = 0$ then
6:     Return $(C_1, C_2, \phi)$
7: end if
8: while $U - 1 > L$ do
9:     $k \longleftarrow \lfloor \frac{L+U}{2} \rfloor$
10:    $C_k \longleftarrow \{1, ...., k\} \cup \{n/2 + k + 1, ...., n\}$
11:    $D_k \longleftarrow \{k+1, ...., n/2 + k\}$
12:    $M_k = Compare(C_k, D_k)$
13:    if $M_k = 0$ then
14:        return $(C_k, D_k, \phi)$
15:    end if
16:    if $M_k = M$ then
17:        $U = k$
18:    end if
19:    if $M_k = -M$ then
20:        $L = k$
21:    end if
22: end while
23: return $(C_k, D_k, \{c_k, c_{k+1}\})$
```

PROOF. Let there are total $|C| = n$ coins. If n is even, divide it into two parts C_1 and C_2, each containing an equal number of coins. Else select a heavy coin using proposition 3.2.1 and put it aside. let's call that coin $c_k \in C_3$ and then partition it into two equal parts. So now C is partitioned into 3 parts $C = C_1 \cup C_2 \cup C_3$. Let M be the measurement function. If balance scale is right inclined, left inclined, or balanced then the value of M be 1, -1, 0 respectively. Now put C_1 and C_2 into the balance scale. If $M = 0$ we are done else shift half the number of coins from each side to the other side on the scale. If the sign of M does not change then shift half of the coins among the coins which didn't change side in the last measurement but changed its side last time when the sign of M was changed, to the other side of the scale. If the sign of M is changed then shift half of the coins among the coins which changed its side in the last measurement to the other side. We will continue the process until we get a balanced measurement or only two coins are left to change the side. Each time number of coins are which are changing side is reducing by half. So this process can go up to $(logn)$ time. If we get a balanced measurement then wlog C_1 = coins on the left side of the scale and C_2 = coins on the right side of the scale will be our required partition of C. When the number of coins is even and we can't get a balance measurement then last two coins which change its side in this process among them one is heavy and another one is light so by removing the last two coins, remaining coins from the above process will get our required C_1 and C_2. When the number of coins is odd by using algorithm 1 we remove a heavy coin c_k if now the number of heavy coin turns odd then we can't get a balanced measure. Among c_k and last two remaining coins, two of them have the same weight(pigeon hole principle). Add those two coins on each side of the scale then we will get a balanced measurement and coins on each side of the scale will from our required partitions of C. Hence this solves the parity problem in $(2logn)$ time. ∎

PROPOSITION 3.2.3. *Let $f : \{0,1\}^n \longrightarrow R$ be a function. d be the branching factor of the decision tree. C_{ab} and C_{ob} denotes the adaptive and oblivious lower bound of f. Then $d^{C_{ab}} \geq C_{ob}$.*

PROOF. Let D be a decision tree with branching factor of d. The total number of measurement D can have is at-most $\sum_{a=0}^{C_{ab}-1} d^a$. So $C_{ob} \leq \sum_{a=0}^{C_{ab}-1} d^a$. But $\sum_{a=0}^{C_{ab}-1} d^a \leq d^{C_{ab}}$. Therefore $d^{C_{ab}} \geq C_{ob}$. ∎

COROLLARY 3.2.4. *Time complexity of Adaptive coin parity problem is $\theta(logn)$.*

PROOF. Let f be the parity function. C_{ab} and C_{ob} denotes the adaptive and oblivious lower bound of f. By Corollary 3.1.1 we know $C_{ob} = \Omega(\sqrt{n})$. By Proposition 3.2.3 and Theorem 3.1.2

$$C_{ab} \geq log(C_{ob}) \geq log(M\alpha(f)/\sqrt{n}) = \frac{1}{2}logn + c \tag{3.3}$$

Here M and c are arbitrary constants. Hence the complexity of the parity problem is $\theta(logn)$.

∎

PROPOSITION 3.2.5. *Coin counting problem can solved adaptively in $O(log^2 n)$ time.*

Algorithm 3 *Coin Counting Problem*

```
 1: Function($C$)
 2: if $|C| = 1$ then
 3:     Return 1
 4: end if
 5: if $|C|$ is odd then
 6:     Using algorithm 1 remove $c_0$ from $C$
 7:     $C \longleftarrow C - \{c_0\}$
 8:     $C_3 \longleftarrow c_0$
 9: end if
10: Partition $C$ into $C_1, C_2, C_3$ using algorithm 2
11: $T \longleftarrow$ Count total number of heavy coin in $C_3$
12: $C \longleftarrow C_1$
13: return $2.$(Function($C$) $+T$)
```

PROOF. Finding Heavy Coin using Algorithm1 takes $logn$ time. we can Partition C into C_1,C_2,C_3 in $logn$ time too. Because the number of heavy coins is the same in C_1 and C_2 so it's sufficient to count any one of them and multiply by 2. This process will continue up to $logn$ time. Hence the time complexity of the adaptive coin-counting problem is $O(log^2 n)$. ■

CHAPTER 4

Adaptive Lower Bound of Coin Counting problem

In this section, we will present our findings on the lower bound of the adaptive coin-counting problem.

Remark : Trivial lower bound of adaptive coin counting problem is $logn$.

In a decision tree setting each leaf node is assigned to a particular value of k, where k denotes as the total number of heavy coins in the coin set. In n number of coins, there are n different choices for k. So the number of leaves in the decision tree is at-least n. Therefore the height of the tree is at-least $logn$. Hence the lower bound is $logn$.

There are a total 2^n numbers of possible Boolean-vectors when the total number of coins is n. If all the coin vectors go to a different leaf then the complexity of lower bound would become $log(2^n) = n$ but by Proposition 3.2.5, we know an algorithm that solves the adaptive coin-counting problem in $O(log^2 n)$ time. So the main idea is to find among the 2^n vectors which of these vectors can go to the same leaf in the decision tree.

4.1 Properties of leaves on the Decision tree

To go to the same leaf these Boolean-vectors have to satisfy certain properties. So let's first discuss about the properties of the Boolean-vectors. We assume heavy coins as 1 and light coins as 0 in the boolean-vectors.

(1) Coin vectors with different number of heavy coins will go to different leafs.

(2) Let $\overrightarrow{c_1}, ..., \overrightarrow{c_k} \in \{0,1\}^n$ all have g number of 1 in them. Let's call this support of a vector. So support(c_i) = g. Let for any $j \in \{0,k\}$ $\overrightarrow{c_j}$ has 1 in these $\{k_1, ..., k_g\}$ positions. Such that $\forall i \neq j$ $\overrightarrow{c_i}$ will not have 1's in those positions. Then all these boolean-vector will go to different leaves in the decision tree.

(3) Let k be the number of $1's$ in the Boolean-vector. Let $\overrightarrow{c_1}, ..., \overrightarrow{c_n} \in \{0,1\}^n$ and c be some non-zero arbitrary scalar. If $c(\overrightarrow{c_1} + ... + \overrightarrow{c_n}) \in \{0,1\}^n$ then $\overrightarrow{c_1}, ..., \overrightarrow{c_n}$ can't go to the same leaf.

(4) Let the number of heavy coin be k and n be the total number of coins. Let $\overrightarrow{c_1}, ..., \overrightarrow{c_n}$ goes to same leaf. Then compliment of these Boolean-vectors will also go to the same leaf and when the number of heavy coin are k and $n-k$ the corresponding number of leafs are equal.

(5) Let the number of heavy coin be k and n be the total number of coins. Among them set of vectors whose ith co-ordinate is 1 can't go to the same leaf.

PROPOSITION 4.1.1. *: Let $\overrightarrow{c_1}, ..., \overrightarrow{c_k} \in \{0,1\}^n$ all have g number of 1 in them. Let for any $j \in \{0,k\}$ $\overrightarrow{c_j}$ has 1 in these $\{k_1, ..., k_g\}$ positions. Such that $\forall i \neq j$ $\overrightarrow{c_i}$ will not have 1's in those positions. Then all these coin vector will go to different leafs in the decision tree.*

PROOF. We will prove the result using contradiction. Let for any $i, j \in \{1, k\}$ coins $\overrightarrow{c_i}, \overrightarrow{c_j}$ goes to the same leaf. This implies these two follow the same path in the decision tree. Hence they have the same measurement applied to them. Let $\overrightarrow{m_1}, .., \overrightarrow{m_r}$ are the measurements applied on $\overrightarrow{c_i}$ and $\overrightarrow{c_j}$. Then $\forall i, j \in \{1, ..., k\}$ and $i \neq j$

$$\begin{gathered}
sign(\overrightarrow{m_1}.\overrightarrow{c_i}) = sign(\overrightarrow{m_1}.\overrightarrow{c_j}) = sign(\overrightarrow{m_1}.\overrightarrow{(c_i + c_j)}) \\
sign(\overrightarrow{m_2}.\overrightarrow{c_i}) = sign(\overrightarrow{m_2}.\overrightarrow{c_j}) = sign(\overrightarrow{m_2}.\overrightarrow{(c_i + c_j)}) \\
. \\
. \\
. \\
sign(\overrightarrow{m_r}.\overrightarrow{c_i}) = sign(\overrightarrow{m_r}.\overrightarrow{c_j}) = sign(\overrightarrow{m_r}.\overrightarrow{(c_i + c_j)})
\end{gathered}$$

So $\overrightarrow{(c_i + c_j)}$ will also go to the same leaf as $\{\overrightarrow{c_i}, \overrightarrow{c_j}\}$ but the number of $1's$ in $\overrightarrow{(c_i + c_j)}$ is $2.g \neq g$. Then $\overrightarrow{(c_i + c_j)}$ can't go to the same leaf as $\{\overrightarrow{c_i}, \overrightarrow{c_j}\}$. Hence contradiction $\forall i, j \in \{1, ..., k\}$ where $i \neq j$, $\{\overrightarrow{c_i}, \overrightarrow{c_j}\}$ will go to the different leaves. ∎

Example: Let $n = 6$ and $k = 2$, then $\{110000, 001100, 000011\}$ will go to three different leaves in the tree.

PROPOSITION 4.1.2. *Let k be the number of $1's$ in the Boolean-vector. Let $\{\overrightarrow{c_1}, ..., \overrightarrow{c_n}\} \in \{0,1\}^n$ and c be some non-zero arbitrary scalar. If $c\overrightarrow{(c_1 + ... + c_n)} \in \{0,1\}^n$ then $\{\overrightarrow{c_1}, ..., \overrightarrow{c_n}\}$ can't go to the same leaf.*

PROOF. We will prove this result using contradiction. Idea is similar to the above proposition. Let $\{\overrightarrow{c_1}, ..., \overrightarrow{c_n}\}$ goes to same leaf and $\{\overrightarrow{m_1}, .., \overrightarrow{m_k}\}$ be the measurements applied on $\overrightarrow{c_i}'s$. Then $\forall i \in (1, .., n)$

$$\begin{aligned}
sign(\overrightarrow{m_1}.\overrightarrow{c_1}) &= ... = sign(\overrightarrow{m_1}.\overrightarrow{c_n}) = sign(\overrightarrow{m_1}.\overrightarrow{(c_1 + ... + c_n)})\\
sign(\overrightarrow{m_2}.\overrightarrow{c_i}) &= ... = sign(\overrightarrow{m_2}.\overrightarrow{c_n}) = sign(\overrightarrow{m_2}.\overrightarrow{(c_1 + ... + c_n)})\\
&.\\
&.\\
&.\\
sign(\overrightarrow{m_k}.\overrightarrow{c_1}) &= ... = sign(\overrightarrow{m_k}.\overrightarrow{c_n}) = sign(\overrightarrow{m_k}.\overrightarrow{(c_1 + ... + c_n)})
\end{aligned}$$

As we know for any two arbitrary vectors $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$ $sign(\overrightarrow{v_1}.\overrightarrow{v_2}) = sign(\overrightarrow{v_1}.(u\overrightarrow{v_2}))$ where u be an arbitrary scalar. Hence $sign(\overrightarrow{m_1}\overrightarrow{(c_1 + ... + c_n)}) = sign(\overrightarrow{m_1}.c\overrightarrow{(c_1 + ... + c_n)})$. Let's say $\overrightarrow{A} = \overrightarrow{m_1}.c\overrightarrow{(c_1 + ... + c_n)}$ and $A \in \{0,1\}^n$. So A will go the same leaf as $\{\overrightarrow{c_1}, ..., \overrightarrow{c_n}\}$. But $\overrightarrow{A}$ has number $1's$ more than k. Therefore $\overrightarrow{A}$ can't go to that same leaf. Hence contradiction $\{\overrightarrow{c_1}, .., \overrightarrow{c_n}\}$ can't go to the same leaf. ∎

COROLLARY 4.1.3. *Let $\overrightarrow{c_1}, ..., \overrightarrow{c_{k+1}} \in \{0,1\}^n$. For any i length of c_i is n and number of $1's$ in any c_i is k. $\forall i \in (1, .., k+1)$ and $j \in (1, .., n)$ $\overrightarrow{c_i}$ has 1 at jth position and $\exists$ exactly one $m \in (1, .., k+1)$ such that $\overrightarrow{c_m}$ does not have 1 at jth position . Then $\overrightarrow{c_1}, ..., \overrightarrow{c_{k+1}}$ can't go to the same leaf.*

PROOF. Let $\overrightarrow{c_1}, ..., \overrightarrow{c_{k+1}}$ goes to same leaf. $\overrightarrow{m_1}, .., \overrightarrow{m_n}$ be the set of measurements applied on $\overrightarrow{c_i}$ where $i \in (1, .., k+1)$. Let $\overrightarrow{A} = \overrightarrow{(c_1 + ... + c_{k+1})}$ and $\overrightarrow{A} \in \{0, k\}^n$. So $\frac{1}{k}\overrightarrow{A} \in \{0, 1\}^n$. Hence using the above proposition we can conclude that $\{\overrightarrow{c_1}, ..., \overrightarrow{c_{k+1}}\}$ can't go to the same leaf. ■

Example: Let $n = 6$, $k = 4$ then $\{011110, 111100, 111010, 110110, 101110\}$ will not go to same leaf.

PROPOSITION 4.1.4. *Let the number of heavy coin be k and n be the total number of coins. Let $\overrightarrow{c_1}, ..., \overrightarrow{c_n}$ goes to same leaf. Then compliment of these Boolean-vectors will also go to the same leaf and when the number of heavy coin are k and $n - k$ then the corresponding number of leaves are equal.*

PROOF. Here complement of Boolean-vectors implies replace 1 with 0 and 0 with 1. Lets denote complement of a vector $\overrightarrow{c}$ as $\overrightarrow{\overrightarrow{c}}$. Let's suppose length of the vectors are n and number of 1 in each vector is k. So the complement of the vectors will have $n - k$ number of 1 in them. Let $\overrightarrow{c_1}, ..., \overrightarrow{c_n}$ goes to same leaf. $\overrightarrow{m_1}, ..., \overrightarrow{m_r}$ are the required measurement to reach the leaf. So $\forall i \in (1, ..n)$ and $\forall j \in (1, ..r)$ $sign(\overrightarrow{m_j}.\overrightarrow{c_i})$ is same. But then $\forall i \in (1, ..n)$ and $\forall j \in (1, ..r)$ $sign(\overrightarrow{m_j}.\overrightarrow{\overrightarrow{c_i}})$ will be same. Hence $\overrightarrow{\overrightarrow{c_1}}, ..., \overrightarrow{\overrightarrow{c_n}}$ will also go to same leaf. We can partition $c = \{\overrightarrow{c_1}, .., \overrightarrow{c_n}\}$ as the set of vectors which goes to same leaf. As we can see $\exists$ a bijection between the partition of $\overrightarrow{c}$ and $\overrightarrow{\overrightarrow{c}}$. Hence the number of leaves are same when the number of heavy coins is k and $n - k$. ■

PROPOSITION 4.1.5. *Let k be the number of 1's of the Boolean-vectors and n be the length of the vectors. Among them set of vectors whose ith co-ordinate is 1 can't go to the same leaf.*

PROOF. We will prove this result also using contradiction. Let $c = \{\overrightarrow{c_1}, ..., \overrightarrow{c_n}\}$ set of all Boolean-vectors whose ith position takes the value 1. Set of all vectors in c goes to same leaf. Then $\exists$ a measurement $\overrightarrow{m}$ such that $sign(\overrightarrow{m}.\overrightarrow{c_i})$ is same $\forall i \in (1, .., n)$. Let $\{\overrightarrow{c_i}, \overrightarrow{c_j}\} \in c$ and x be total number of nonzero entries of $\overrightarrow{m}$. Let $\overrightarrow{c_i}$ be a vector which takes 1 on those co-ordinates where co-ordinate of $\overrightarrow{m}$ is -1 and 0 at-least one of those co-ordinates where the co-ordinate $\overrightarrow{m}$ is 1. Similarly $\overrightarrow{c_j}$ be a vector which takes 1 on those co-ordinates where co-ordinate of $\overrightarrow{m}$ is 1 and 0 at-least one of those co-ordinates where the co-ordinate $\overrightarrow{m}$ is -1. $\{\overrightarrow{c_i}, \overrightarrow{c_j}\}$ both have 1 at ith co-ordinate. But then $sign(\overrightarrow{m}.\overrightarrow{c_i}) \neq sign(\overrightarrow{m}.\overrightarrow{c_j})$, so set of all vectors of c can't go to the same leaf. ■

4.2 Number of leafs in the decision tree

In this section we will count the total number of leafs in the decision tree using the properties of the Boolean-vectors.

PROPOSITION 4.2.1. *Total number of leafs are at-least* $2nlogn$.

PROOF. Let the length of vectors be n. Let k be total of $1's$ in each of the boolean-vectors. k has at-least n many choices therefore, number of leafs are at-least n.
Using Proposition 4.1.1 we know that the vectors which has 1 in distinct co-ordinates goes to different leaf. So for each k there are at-least n/k leafs. The total number of leafs are at-least $\Sigma_{k=0}^{n} \frac{n}{k}$. But using Proposition 4.1.4 we also know for $k = i$ and $k = n - i$ for both the cases number of leafs are same so the above summation becomes $2 * \Sigma_{k=0}^{n/2} \frac{n}{k}$
So the total the number of leafs are at-least $2 * \Sigma_{k=0}^{n/2} \frac{n}{k} = 2.n(1 + \frac{1}{2} + ... + \frac{1}{\frac{n}{2}}) \approx 2 * nlogn$. ■

Corollary: Adaptive lower bound of coin counting problem is $log2n + loglogn$.

PROOF. This finally proves the Theorem 1.0.1. ■

CHAPTER 5

Conclusion

The main aim of this thesis is to improve the lower bound of "coin counting problem". Idea is to increase the number of leaves of the decision tree which automatically increases the depth of the tree and hence the lower bound. We partition the Boolean-vectors so that each part can be assigned to a unique leaf node of the tree. All Boolean-vectors from the same partition goes to the same leaf node. Therefore there is a bijection between number of leaf node and number of partition. So by counting the number of partitions of the vectors we have found the total number of leaf nodes is at-least $2nlogn$. Therefore the improved adaptive lower bound of coin counting problem is $log2n + loglogn$.

5.1 Future outlook

In chapter 3 we have given a $O(log^2n)$ adaptive algorithm of coin counting problem. The current adaptive lower bound is $log2n + loglogn$. As we can see the adaptive bound is not tight. Further work can be to improve the lower bound or to give a $O(logn)$ adaptive algorithm of the coin counting problem. So that the adaptive coin counting problem becomes tight. Similarly oblivious bound is also not tight. So in future, people can also try to improve that bound too.

References

[1] PURDY, E. Lower bounds for coin-weighing problems. *ACM Transactions on Computation Theory (TOCT) 2*, 2 (2011), 1–12.