Theoretical Computer Science 181 (1997) 347–356

Theoretical Computer Science

A $\frac{3}{2}\log 3$-competitive algorithm for the counterfeit coin problem

Peng-Jun Wan[a,*], Qifan Yang[a,1], Dean Kelley[b]

[a] *Computer Science Department, University of Minnesota, Minneapolis, MN55455, USA*
[b] *St. Mary's University*

Abstract

We study the following *counterfeit coin problem*: Suppose that there is a set of n coins. Each one is either *heavy* or *light*. The goal is to sort them according to weight with a minimum number of weighings on a balance scale. Hu and Hwang gave an algorithm with a competitive ratio of $3\log 3$ (all logarithms are base-2). Hu, Chen and Hwang also gave an algorithm with a competitive ratio of $2\log 3$. In this paper we give an improved algorithm whose competitive ratio is $\frac{3}{2}\log 3$.

1. Introduction

Consider a set of n coins C which contains d light coins and $n-d$ heavy coins. The cases where d is known and where d is unknown are considered as different problems. We want to sort the coins by using a balance scale and perform a minimum number of weighings to identify the d light and the $n-d$ heavy coins.

Let $M_A(n:d)$ denote the maximum number of weighings required by algorithm A to sort the n coins when d is *unknown* and let $M_A(n,d)$ denote this maximum when d is *known*. Let $M(n:d)=\min_A M_A(n:d)$ and let $M(n,d)=\min_A M_A(n,d)$. An algorithm A is a *competitive algorithm* if there exist constants c and b such that for all $n>d>0$ we have

$$M_A(n:d)\leqslant cM(n,d)+b.$$

The constant c is called the *competitive ratio.*

Hu and Hwang [4] first proposed a bisecting algorithm with a competitive ratio $3\log_2 3$. Soon after, Hu et al. [3] discovered a doubling algorithm with competitive ratio $2\log_2 3$. In this paper we give an improved algorithm and show that it has a competitive ratio of $\frac{3}{2}\log 3$.

* Corresponding author. E-mail: wan@cs.umn.edu.
[1] Visiting scholar in the Department of Computer Science, University of Minnesota. Supported by Pao Yu-kong and Pao Zhao-long scholarship and NSF grant CCR-9208913.

2. Preliminaries

The analysis of competitive ratio involves both lower-bound and upper-bound problems. In this section, we first list some results about the lower-bound for $M_A(n:d)$ and $M_A(n,d)$.

Hu and Hwang [4] gave a lower-bound for $M_A(n,d)$:

Lemma 2.1.

$$M_A(n,d) \geqslant \frac{d}{\log_2 3}\left(\log_2 \frac{n}{d} + \log_2 \frac{e\sqrt{3}}{2}\right) - \frac{\log_2 d}{2\log_2 3} - \frac{0.567}{\log_2 3} - \frac{1}{2}.$$

Cairns [1] discovered an optimal algorithm to find the only one counterfeit in a set of coins, when it is known before testing there is only one counterfeit.

Lemma 2.2. $M(n,1) = \lceil \log_3 n \rceil$.

Hu and Hwang [4] also gave the value of $M(n:1)$.

Lemma 2.3. $M(n:1) = \lceil \log_2 n \rceil$.

For convenience we assume that the value of function $d \log_2 n/d$ at $d=0$ is 0 because $\lim_{d\to 0} d\log_2 n/d = 0$. The following lemma, given by Du and Park [2], is an important tool for analysis.

Lemma 2.4. *Let* $d = d_1 + d_2$ *and* $n = n_1 + n_2$ *where* $d_1 \geqslant 0$, $d_2 \geqslant 0$, $n_1 > 0$ *and* $n_2 > 0$. *Then*

$$d_1 \log_2 \frac{n_1}{d_1} + d_2 \log_2 \frac{n_2}{d_2} \leqslant d \log_2 \frac{n}{d}.$$

3. The algorithm

A set of coins are called *uniform* if they are all of the same type, and called *unique* if there is only one exception. We also use the modifier "*heavy*" and "*light*" to specify the type of the majority of coins in a uniform or unique set. Assume that the set of coins to be identified is $C = \{c_1, c_2, \ldots, c_n\}$ for $n \geqslant 2$. In the algorithm and in the discussion that follows we employ the notation $|A|$ to denote the number of elements in the set A and $\|A\|$ to denote the weight of the elements of A. Weight of a set of elements is relative. That is, we are only interested in comparison weighings as would be done with a balance scale. Let X and Y be two nonempty sets of coins, then a comparison between X and Y means to compare $X' \subseteq X$ with $Y' \subseteq Y$ such that $|X'| = |Y'|$ and either $X' = X$ or $Y' = Y$. In other words, we compare two largest equinumerious subsets of X and Y. A comparison can have three possible outcomes:

$\|X'\| = \|Y'\|$, $\|X'\| > \|Y'\|$, $\|X'\| < \|Y'\|$. We say the comparison yields equality for the first outcome, and yields inequality for the other two outcomes.

An important component of Hu's and Hwang's $3 \log 3$-competitive algorithm is a halving procedure or *binary search*. In searching for a heavy coin this procedure repeatedly splits a set which is known to contain a heavy coin until that coin is identified. Each time the set is split (adding a light coin if the set is of odd size) and the two halves are compared on the balance scale. If one half weighs more it is selected to be split at the next step. In case the halves are of equal weight (i.e., they balance) the half to be split in the next step is chosen arbitrarily. Note that in this case both halves contain equally many heavy coins.

There are several important informations we can tell from the binary search path. Suppose that the binary search is to seek a heavy coin from a set U.

- There are no equal weighing in the binary search path if and only if U is light unique.
- If there is any equal weighing in the binary search path, consider the last one. Suppose that A_1 and A_2 are the two sets involved in that weighing. Then both A_1 and A_2 are light unique. If there are more than one equal weighings in the binary search path, then the set $U - (A_1 \cup A_2)$ has at least two heavy coins.

The same holds in case the binary search is for a light coin.

Based on the above observation, we develop a multi-phase algorithm. Each phase begins with a set which either contains only a single coin or is a unique set of size at least four. If a phase begins with a a unique set, then at the beginning of the phase, a binary search is used to identify the types of all coins in the unique set. Each phase employs doubling together with binary searches. Except the last phase, each phase find two heavy (or light) coins and more than two coins of the other type. At the end of each phase except the last one, a set which either contains only a single coin or is a unique of size at least four is found for the next phase to begin with.

Before presenting the algorithm, we first describe some variables used in the algorithm:

U: all the coins of unknown type (initially is C).

X: all identified coins in a phase;

Y: a subset of coins from U to be used test against X in a weighing;

ToSeek: the type of coins to seek in a phase. It's value can be *unknown, light, heavy*.

Found: the number of coins of type ToSeek identified in a phase.

Our algorithm can be described as follows.

Algorithm A

```
$X := \{c\}$  where c is any coin in $C$, $U := C - X$;
repeat
  $Found := 0$;
  if ($|X| = 1$) then
    $ToSeek := unknown$;
```

```
if ($|X| > 1$) then
  binary search $X$; /* One weighing could be saved if $|X| \leqslant 3$ or if two haves of
                        $X$ has been compared in the previous phase */
  $Found := Found + 1$;
  $X := 2^{\lfloor \log |X| \rfloor}$ coins from $X$ containing the unique coin;
repeat
  $Y :=$ min$\{|U|, |X|\}$ coins from $U$;
  compare $X$ and $Y$;
  if $Y$ is pure, then
    $X := X \cup Y$, $U := U - Y$;
  else
  binary search $Y$;
  $Found := Found + 1$;
  if $found = 1$ then
    $ToSeek :=$ the type of the coin the binary search looks for;
  if $Y$ is unique, then
    $X := X \cup Y$, $U := U - Y$;
  else
    Let $A_1$ and $A_2$ be the two halves involved in the last equal weighing and $A_1$
      is identified in the binary search.
    if $Found = 2$ then
      $X := A_2$, $U := U - A_1$;
    else
      binary search $A_2$;
      $Found := Found + 1$;
      $U := U - (A_1 \cup A_2)$;
      if $Y - (A_1 \cup A_2) \neq \emptyset$ and there is only one equal weighing in the path,
        then compare $Y - (A_1 \cup A_2)$ against a unique set of same size;
      if $Y - (A_1 \cup A_2)$ is empty or pure, then
        $X := X \cup Y$, $U := U - (Y - (A_1 \cup A_2))$;
      else
        if $Y - (A_1 \cup A_2)$ is unique, then
          $X := Y - (A_1 \cup A_2)$, $U := U - (Y - (A_1 \cup A_2))$;
        else
          $X := \{c\}$ where $c$ is any coin in $U$, $U := U - \{c\}$;
until $found = 2$ or $U = \emptyset$;
/* Now $X$ either contains a single coin or is a unique set or contains exactly
  two coins of the ToSeek type. */
if $X$ contains two coins of the ToSeek type, then
  repeat
    $Y :=$ min$\{|U|, |X|\}$ coins from $U$;
    compare $X$ and $Y$;
```

```
        if $Y$ contains at least two coins of the ToSeek type, then
          $X := \{c\}$ where $c$ is any coin in $U$, $U := U - \{c\}$;
        else
          split $Y$ into two halves and compare them;
          if $Y$ is unique, then
            $X := Y$, $U := U - Y$;
          else
            $X := X \cup Y$, $U := U - Y$;
      until $X$ is either unique or contains a single coin or $U = \emptyset$;
  until $U = \emptyset$;
```

Now we give a brief explanation of algorithm A. Each phase corresponds to the outer *repeat-until* loop. Each phase begins with a set of coins X which either contains a single coin or is a unique set. If X is a unique set, then the type of the unique coin is given by the variable *ToSeek*, from the previous phase. A binary search is used to identify the type of coins in X when X is a unique set. The first inner *repeat-until* loop performs the doubling process. Each time a set Y of coins are picked from the set of unidentified coins. The doubling process continues until a Y which contains the coin of different type from the majority type of X. Then the algorithm takes a binary search on Y. As we mentioned before, each phase except the last one find exactly two heavy (or light) coins. After the binary search on Y, if the number of heavy (or light) coins founded so far in the current phase, given by *Found*, is one, then it will either find another one from Y if Y has any or it will start another this inner loop. Once the phase finds two heavy (or light) coins, it will prepare the set X for the next phase to begin with. The set X is chosen in two ways. One way is to choose any unidentified coin if the algorithm can determine that the remaining unidentified coins contains at least two coins of the type given by *ToSeek*. The other way is to find a unique set in which the unique coins is of type given by *ToSeek*. The second inner *repeat-until* loop performs such kind of preparing.

In the next section, we will give an analysis of the algorithm.

4. The competitive ratio of Algorithm A

We will count the number of weighings in each phase in term of the number of coins identified in this phase. Lemma 4.1 will analyze the phases except the last one and Lemma 4.3 will analyze the last phase.

Lemma 4.1. *If all together $\bar{n}$ coins are identified in a phase other than the last one, then the number of tests in this phase is at most* $3\log\bar{n}$.

Proof. Suppose that algorithm A finds two light coins and at least two heavy coins in this phase and without loss of generality suppose that the phase starts with X containing a single coin (or else fewer weighings than the following are needed). There are four cases regarding how the two light coins are found.

Case 1: One light coin is found in the ith doubling and in the jth doubling it is found that Y contains only one light coin.

If $\bar{n} = 2^{k-1}$ for some $k > j$, then the number of tests in this phase is at most

$$(2i-1)+(j-i)+(j-1)+2(k-j)=2k+i-2\leqslant 3k-3=3\log(2^{k-1}).$$

If $\bar{n} = 2^j$ and $n = 2^j + n_1$ with $n_1 < 2^j$, then the number of tests in this step is at most

$$(2i-1)+(j-i)+(j-1)+2=i+2j\leqslant 3j-1<3\log(2^j).$$

Case 2: One light coin is found in the ith doubling and in the jth doubling it is found that Y contains at least two light coins.

Let A_1 be the set of coins in Y identified by the binary search. In this case, $\bar{n} = 2^{j-1} + |A_1|$.

The number of tests in this step is at most

$$(2i-1)+(j-i)+(j-1)=2j+i-2<3\log(2^{j-1}+|A_1|).$$

Case 3: In the ith doubling, Y contains exactly two light coins.

Suppose that $\bar{n} = 2^{k-1}$. Then the number of tests in this step is at most

$$(3i-3)+2(k-i)<3\log(2^{k-1}).$$

Case 4: In the ith doubling, Y contains at least three light coins.

Let A_1 and A_2 be the two halves involved in the last equal weighing in the binary search path of Y. Then $\bar{n} = 2^{i-1} + |A_1| + |A_2|$. The number of tests is at most

$$(3i-4)+1\leqslant 3\log(2^{i-1}+|A_1|+|A_2|). \quad \square$$

In this analysis the phase is assumed to begin with a set X containing only single coin. If a phase begins with a unique set, then even fewer tests are needed. In fact, instead of using $2i-1$ tests to find the first light coin, the algorithm begins by binary searching X. If $|X|\geqslant 4$, then the number of tests used for it is $\lceil \log|X| \rceil = i+1$ and $i+1\leqslant 2i-1$ for $i\geqslant 2$. If $|X|\leqslant 3$, then one test is enough.

Lemma 4.2.

If $0 < d_1 < d_2 \leqslant \frac{n}{2}$

then $d_1 \log \frac{n}{d_1} + d_1 \leqslant d_2 \log \frac{n}{d_2} + d_2.$

Proof. Let

$$f(x) = x\log\frac{n}{x} + x.$$

Then $f'(x) = 2n/ex \geqslant \log\frac{4}{e} > 0$. So $f(x)$ is increasing for $x \leqslant \frac{n}{2}$. $\square$

Lemma 4.3. *Suppose that Algorithm A uses t phases to solve an (n,d) problem and in the ith phase it finds d_i light coins and $n_i - d_i$ heavy coins $(i=1,\dots,t)$. Let $m_i = \min\{d_i, n_i - d_i\}$.*

1. *If $m_t = 2$ then the number of tests used in the last phase is at most $3\log n_t$.*
2. *If $m_t = 1$, consider the last two phases. Let $\overline{n} = n_{t-1} + n_t$. Let $\overline{d}$ and $\overline{n} - \overline{d}$ be the number of light coins and heavy coins respectively and let $\overline{m} = \min\{\overline{d}, \overline{n} - \overline{d}\}$. Then the total number of tests in the last two phases is at most $\frac{3}{2}(\overline{m}\log\frac{\overline{n}}{\overline{m}} + \overline{m})$.*

Proof. (1) There are two possible cases when $m_t = 2$.

Case 1: The algorithm finds one light (or heavy) coin in the ith doubling and then another one in the jth doubling for $j > i$.

1. If $n_t \geq 2^j$ then the number of tests used in this phase is at most
$$(2i-1) + (j-i) + (j-1) + 2(\lceil \log n_t \rceil - j) \leq 2\log n_t + i < 3\log n_t.$$
2. If $n_t = 2^{j-1} + n_t'$, for $n_t' < 2^{j-1}$ then the number of tests used in this phase is at most
$$(2i-1) + (j-i) + \lceil \log n_t' \rceil \leq 3j - 3 < 3\log n_t.$$

Case 2: The algorithm finds two light (or heavy) coins in the ith doubling.

1. If $n_t \geq 2^i$ then the number of tests used in this phase is at most
$$(3i-3) + 2(\lceil \log n_t \rceil - i) = 2\log n_t + i - 1 < 3\log n_t.$$
2. If $n_t = 2^{i-1} + n_t'$ and $n_t' < 2^{i-1}$ then the number of tests used in this phase is at most
$$i + 2\lceil \log n_t' \rceil - 1 \leq 3i - 3 < 3\log n_t.$$

(2) If $m_t = 1$, we consider the last two phases. We first consider the case $\overline{m} = 3$. Without loss generality, we suppose that $\overline{m} = \overline{d}$. The second to last phase does not end until the algorithm finds a set X containing exactly one light coin and the tth phase begins with X.

1. If $|X| \geq 4$ the total number of tests used in the last two phases is at most
$$\begin{aligned}
&3\log n_{t-1} + (\lceil \log n_t \rceil - \lfloor \log |X| \rfloor) + \lceil \log |X| \rceil \\
&\leq 3\log n_{t-1} + \log n_t + 2 \\
&= \frac{3}{2}\left(2\log\frac{n_{t-1}}{2} + 2 + \log n_t + 1\right) - \frac{1}{2}\log n_t + \frac{1}{2} \\
&< \frac{3}{2}\left(3\log\frac{\overline{n}}{3} + 3\right) \\
&= \frac{3}{2}\left(\overline{m}\log\frac{\overline{n}}{\overline{m}} + \overline{m}\right).
\end{aligned}$$
2. If $|X| \leq 3$, then the total number of tests used in the last two phases is at most
$$3\log n_{t-1} + \lceil \log n_t \rceil + 1 \leq 3\log n_{t-1} + \log n_t + 2 \leq \frac{3}{2}\left(\overline{m}\log\frac{\overline{n}}{\overline{m}} + \overline{m}\right).$$

Now we assume that $\overline{m} \geqslant 4$. Suppose we seek two light coins in the second to the last phase. Since $m_t = 1$ and there are at least two light coins remaining, the last phase should have to start beginning with X containing only a single coin, and the algorithm will seek a heavy coin. The total number of tests used in the last two phases is at most $3\log n_{t-1} + 2\log n_t + 1$ by Lemma 4.1.

Case 1: If $n_t \leqslant 16$ then

$$\begin{aligned} 3\log n_{t-1} + 2\log n_t + 1 &= \frac{3}{2}\left(2\log\frac{n_{t-1}}{2} + 2 + \log n_t\right) + \frac{1}{2}\log n_t + 1 \\ &\leqslant \frac{3}{2}\left(3\log\frac{\overline{n}}{3} + 2\right) + 3 \\ &= \frac{3}{2}\left(3\log\frac{\overline{n}}{3} + 4\right) \\ &\leqslant \frac{3}{2}\left(4\log\frac{\overline{n}}{4} + 4\right) \\ &\leqslant \frac{3}{2}\left(\overline{m}\log\frac{\overline{n}}{\overline{m}} + \overline{m}\right). \end{aligned}$$

The last inequality holds since $\overline{m} \geqslant 4$ and $\overline{n} \geqslant 2\overline{m}$ (by Lemma 4.2).

Case 2: If $n_t > 16$ then

$$\begin{aligned} 3\log n_{t-1} + 2\log n_t + 1 &= \frac{3}{2}\left(2\log\frac{n_{t-1}}{2} + \log n_t + 2\right) + \frac{1}{2}\log n_t + 1 \\ &\leqslant \frac{3}{2}\left(3\log\frac{\overline{n}}{3} + \log n_t + 2\right) - \log n_t + 1 \\ &\leqslant \frac{3}{2}\left(4\log\frac{\overline{n} + n_t}{4} + 2\right) - \log n_t + 1 \\ &\leqslant \frac{3}{2}\left(4\log\frac{\overline{n}}{4} + 6\right) - \log n_t + 1 \\ &\leqslant \frac{3}{2}\left(4\log\frac{\overline{n}}{4} + 4\right) - \log n_t + 4 \\ &\leqslant \frac{3}{2}\left(4\log\frac{\overline{n}}{4} + 4\right) \\ &\leqslant \frac{3}{2}\left(\overline{m}\log\frac{\overline{n}}{\overline{m}} + \overline{m}\right). \quad \square \end{aligned}$$

Now we can bound the number of weighing in our algorithm as follows.

Theorem 4.4. *Let* $m = \min\{d, n-d\}$. *Then* $M_A(n : d) \leqslant \frac{3}{2}(m\log\frac{n}{m} + m)$.

Proof. Suppose that Algorithm A takes t phases to solve an (n,d) problem and in the ith phase it identifies d_i light coins and $n_i - d_i$ heavy coins. Let $m_i = \min\{d_i, n_i - d_i\}$

for $i=1,2,\dots,t$. If $m_t=2$ then

$$\begin{aligned}M_A(n:d) &\leqslant 3(\log n_1+\log n_2+\cdots+\log n_t) \quad \text{by Lemmas 4.2 and 4.3(1)}\\ &= \frac{3}{2}\left(2\log\frac{n_1}{2}+2+2\log\frac{n_2}{2}+2+\cdots+2\log\frac{n_t}{2}+2\right)\\ &\leqslant \frac{3}{2}\left(2t\log\frac{n}{2t}+2t\right) \quad \text{by Lemma 2.4.}\end{aligned}$$

Since $m_t=2$, we have $m\geqslant 2t$ and

$$M_A(n:d)\leqslant\frac{3}{2}\left(m\log\frac{n}{m}+m\right)$$

by Lemma 4.2.

If $m_t=1$ then by Lemmas 4.2, 4.3(2), and 2.4

$$M_A(n:d)\leqslant\frac{3}{2}\left((2(t-2)+\overline{m})\log\frac{n}{2(t-2)+\overline{m}}+(2(t-2)+\overline{m})\right)$$

As $m\geqslant 2(t-2)+\overline{m}$, we have

$$M_A(n:d)\leqslant\frac{3}{2}\left(m\log\frac{n}{m}+m\right)$$

by Lemma 4.2 and since $n\geqslant 2m$. $\square$

The next theorem gives the main result of this paper.

Theorem 4.5. $M_A\leqslant\frac{3}{2}(\log 3)M(n:d)+3$.

Proof. Since $M(n:1)=\lceil\log n\rceil$ (Lemma 2.3), we have

$$M_A(n:1)\leqslant 2\lceil\log n\rceil-1\leqslant 2\log n+1<\frac{3}{2}(\log 3)M(n:1)+3.$$

Now, without loss of generality, suppose $m=d$. For $d\geqslant 2$, by Lemma 2.1, we have

$$\begin{aligned}\frac{3}{2}(\log 3)M(n:d)+3 &\geqslant \frac{3}{2}(\log 3)M(n,d)+3\\ &\geqslant \frac{3}{2}\left(d\log\frac{n}{d}+d\right)+\frac{3}{2}\left(\log\frac{e\sqrt{3}}{2}-1\right)d\\ &\quad -\frac{3\log d}{4}-\frac{3}{2}(0.567)-\frac{3}{4}+3\end{aligned}$$

Let

$$h(d)=\frac{3}{2}\left(\log\frac{e\sqrt{3}}{2}-1\right)d-\frac{3\log d}{4}-\frac{3}{2}(0.567)-\frac{3}{4}+3.$$

Then

$$h'(d)=\frac{3}{2}\left(\log\frac{e\sqrt{3}}{2}-1\right)-\frac{3}{4d\ln 2}.$$

Note that $h'(d) > 0$ and $h(d)$ is increasing when $d \geqslant 4$. Moreover, $h(2)$, $h(3)$, and $h(4)$ are all positive. Thus we have that

$$\begin{aligned} M_A(n:d) &\leqslant \frac{3}{2}\left(d\log\frac{n}{d}+d\right) \\ &\leqslant \frac{3}{2}(\log 3)M(n:d)+3. \qquad \square \end{aligned}$$

References

[1] S.S.Cairns, Balance scale sorting, *Amer. Math. Monthly* **70** (1963) 136–148.
[2] D.Z. Du and H. Park, On competitive group testing, *SIAM J. Comput.* **23** (1994) 1019–1025.
[3] X.D. Hu, P.D. Chen and F.K. Hwang, A new competitive algorithm for the counterfeit coin problem, *Inform. Process Lett.* **51** (1994) 213–218.
[4] X.D. Hu and F.K. Hwang, A competitive algorithm for the counterfeit coin problem, preprint, 1992.
[5] D.D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging rules, *Comm. ACM* **28** (1985) 202–208.