Annals of Pure and Applied Mathematics
Vol. 7, No. 1, 2014, 77-82
ISSN: 2279-087X (P), 2279-0888(online)
Published on 9 September 2014
www.researchmathsci.org

An Advanced Approach to Solve two Counterfeit Coins Problem

***Joydeb Ghosh*[1], *Lagnashree Dey*[2], *Ankita Nandy*[2], *Arpan Chakrabarty*[2]**
***Piyali Datta*[2], *Rajat Kumar Pal*[2] and *Ranjit Kumar Samanta*[3]**

[1]Department of Mathematics, Surendra Institute of Engineering and Management, New Chamta, Siliguri, Darjeeling – 734 009, West Bengal, India
Email: joydeb009@gmail.com
[2]Department of Computer Science and Engineering, University of Calcutta
92, A. P. C. Road, Kolkata – 700 009, West Bengal, India
Email: {rose2009mail, yoursankita.nandy09, arpan250506, piyalidatta150888, pal.rajatk}@gmail.com
[3]Department of Computer Science and Application, North Bengal University
Darjeeling – 734 013, West Bengal, India
E-mail: rksamantark@gmail.com

Received 21 July 2014; Accepted 21 August 2014

Abstract. Though the counterfeit coin problem is well known as a fascinating puzzle it claims great importance in Computer science, Game theory, and Mathematics. In terms of the puzzle the objective is to detect the counterfeit coins which are identical in appearance but different in weight. The word *counterfeit* not only describes forgeries of currency or documents, but can also be applied to software, pharmaceuticals, clothing, and more recently, motorcycles and cars, especially when these result in patent or trademark infringement. Furthermore, the goal in this problem is to minimize the number of weighing, i.e., the number of comparisons required to find the false coin/s and their type (whether heavier or lighter than the original coin). Finding one counterfeit coin among *n* coins is complex and tricky enough. The problem gets more complicated when the set of *n* coins contains two false coins as the false coins pair may appear in several different combinations. In this paper, we have developed a new algorithm for solving two counterfeit coin problem in O(log*n*) time, where *n* is the total number of coins.

Keywords: Counterfeit coin problem, equal arm balance, desgin of algorithm, decision tree, complexity

AMS Mathematics Subject Classification (2010): 05C78

1. Introduction

Computing a solution of the counterfeit coin problem has huge significance in both theoretical and commercial sphere as well as to prevent forgery in different fields. The objective is to minimize the number of weighing for which it is sufficient to determine the defective coin(s) in a set of n coins using only an equal arm balance, when the number of odd coins is precisely known and they are identical in appearance but different in weight (either heavier or lighter) than a true coin. The complexity of the problem increases with the increment of the number of counterfeit coins in a set. If P is the number of counterfeit coins in a set of n coins, it is not only sufficient to consider whether the counterfeit coins are heavier or lighter in comparison to a genuine coin individually, but we must also take into account their mutual relation like equally heavier or lighter, unequally heavier or lighter, etc. In this paper, we consider that there are two false coins in a set of n coins which are equally heavier (or equally lighter) in comparison to a genuine coin. The objective is to identify the counterfeit coins using a minimum number of comparisons (or weighing).

2. Literature survey

In paper [4], the problem has been introduced in two ways. In the first case, we do not know if there is a fake coin in the given set. The process is supposed to check it first, and if yes, then identify the targeted coin by means of a minimum number of weighing. In the second case, it is told that there is a counterfeit coin and the objective is to find the coin through minimum number of weighing. At times, a standard coin may also be given. In the first case, if a lighter coin is there in the given set S of coins, then it is proved that the least number of weighing to find out the fake coin satisfies $3^{n-1} < |S| \leq 3^n$ for some unique value of n, where |S| is the cardinality of set S. In the second case, we are given a set S of coins plus a standard coin, where only one coin in S is of different weight. Then it is proved that $(3^{n-1}-1)/2 < |S| \leq (3^n-1)/2$.

In paper [1], the problem has been addressed as an application of dynamic programming and the associated analysis has been made through optimal and suboptimal testing policy. Here also only one coin is defective out of n given coins. This technique always assumes $k < n$ coins in each pan for each weighing, where the value of k essentially depends on the value of n. If the two groups balance, the defective coin must be in the remaining $n-2k$ coins; otherwise, the false coin is in one of the k groups. After each weighing, the number of coins to be examined reduces, but the problem remains the same. This allows the authors to apply dynamic programming to this problem.

One classical solution is available in the form of a *decision tree* that represents a set of all possible decisions by which we can acquire the desired solution(s) of the problem [2, 3]. In this solution, each internal vertex (that is not a leaf vertex) symbolizes a comparison between a pair of equal sets of coins using an equal arm balance. Here the problem under consideration is more generalized; the fake coin can either be heavier or lighter. So, for n given coins, there are $2n$ leaf vertices in the tree as probable solutions.

In paper [5], the problem of ascertaining the minimum number of weighing which suffice to determine the counterfeit (heavier) coins in a set of n coins of the same appearance, given a balance scale and the information that there are exactly two heavier coins present, has been considered. Both of heavier coins are of equal weight and they are not heavier than 3/2 times than the true coin. If p is the maximum number of comparisons required to find out two false coins (equally heavier), the paper introduces an algorithm which has the lower bound $\lceil \log_3({}^nC_2) \rceil$. In this paper, an infinite set of n has been determined for which this lower bound is reached, whereas the upper bound is only one unit more than the lower bound.

3. Problem formulation and algorithm

In this section, we formulate and develop an efficient algorithm to solve two counterfeit coins problem. The algorithm finds both the counterfeit coins among *n* number of coins, which are indexed sequentially from 1 to *n*. Finding out two false coins introduces several cases, i.e., different combinations of false coin pairs, such as heavier-heavier, lighter-lighter, heavier-lighter, etc. The behaviour of the problem changes with the change in the specification of the problem. In this paper, we consider the case where both false coins are equally heavier (or equally lighter) than the true coin and any two coins among *n* coins may be false. We assume that the heavier (lighter) and the true coin are denoted by H (L) and T, whereas the weight of those are w(H) (w(L)) and w(T) respectively. The issue to be considered in this problem is the minimum value of *n* such that we can find two false coins among them without using any extra standard coin.

Lemma 1. To find *p* counterfeit coins among *n* coins without taking help of an additional genuine coin, the value of *n* has to be at least of $2p+1$.

Proof: If there is one false coin among two coins, a standard coin is necessary to detect the false coin unless the weight of the correct coin is given. So if the number of false coins, $p = 1$, the minimum value of $n = 3 = 2p+1$. Now, if there are two false coins we can identify them from four coins if and only if two false coins are of different weight. If they are of same weight, we cannot conclude which set of coins is true, as there are equal numbers of false and true coins. So, for $p = 2$, the minimum value of $n = 5$, i.e., $2p+1$. In general, if there are *p* counterfeit coins, we can detect them from $p+2$ coins if all the *p* coins are of distinct weights. But if at least two false coins are of same weight, we cannot distinguish them from the set of all coins. So, to satisfy the above cases specially the case where all the false coins are of same weight, we need at least one more true coin than the false coin, i.e., the minimum number of total coins required is $p+p+1 = 2p+1$. □

The algorithm proceeds by dividing the coins into three sets, say *K*1, *K*2, and *K*3. The results of division depends on three cases: (i) *n* is divisible by 3, i.e., $n|3$, (ii) $n+1$ is divisible by 3, i.e., $(n+1)|3$, and (iii) $n–1$ is divisible by 3, i.e., $(n–1)|3$. Thus, depending on the value of *n* it can easily be decided to which group the set of *n* coins belongs to and precisely which variant of the algorithm can be applied on the given set of coins. We assume that the coins are indexed by natural numbers 1 through *n*. For the first case, $|K1| = |K2| = |K3| = n/3$. For the second case, $|K1| = |K2| = (n+1)/3$, and $|K3| = n–2(n+1)/3 = (n–2)/3$. So, there is a difference of one coin between *K*1 and *K*3, or *K*2 and *K*3. For the third case, $|K1| = |K2| = (n–1)/3+1 = (n+2)/3$, and $|K3| = n–2(n+2)/3 = (n–4)/3$. There is a difference of two coins between *K*1 and *K*3, or *K*2 and *K*3.

After dividing the set of *n* coins into *K*1, *K*2, and *K*3, at first *K*1 and *K*2 are put on the arms (or pans) for weighing. Depending on the outcome of the weighing three versions of the algorithm proceeds towards the next weighing taking different sets. Considering the result of its parent nodes some weighing is performed at each internal node. At the leaf nodes we perform either one TCP(K_i) operation or two OCPH(K_i) operations (OCPL(K_i) in case of equally lighter coins) to find out both the counterfeit coins in set K_i. Whenever we are sure that there is only one heavier (lighter) false coin in K_i, we call OCPH(K_i) (OCPL(K_i)). When we are convinced that there are two false coins in K_i, the algorithm is recursively applied to K_i, considering its version. We denote this operation as TCP(K_i) in general. Figure 1 shows the decision tree for the version of the algorithm when $n|3$ (in case of H1 = H2). At the root node, $K1$ and $K2$ are compared and the result is analyzed as follows. If w($K1$) = w($K2$), either $K1$ and $K2$ contain one false coin each or $K3$ contains both the counterfeit coins. So next we compare $K2$ and $K3$. There are two possibilities.

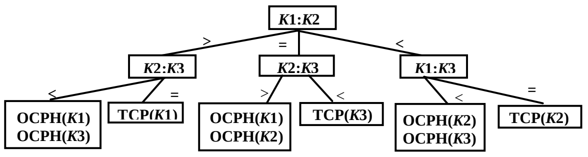
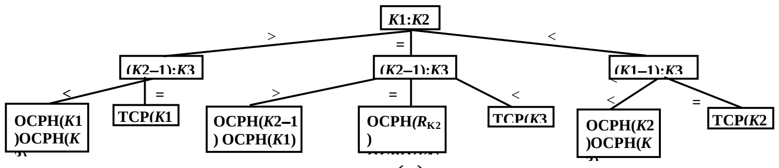


Figure 1: Decision tree of the algorithm for


(a)

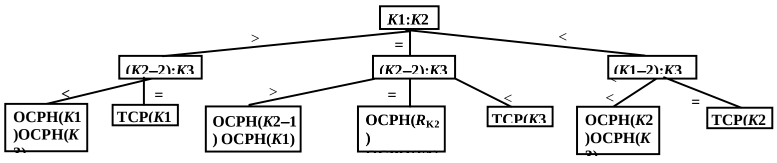

(b)

Figure 2: (a) Decision tree for $(n+1)|3$. **(b)** Decision tree for $(n-1)|3$.

If $K2 > K3$, then we are certain that one false coin is in $K1$ and the other is in $K2$. So, we perform OCPH($K1$) and OCPH($K2$). If $K2 < K3$, then we are sure that both the false coins are in $K3$; thus, we perform TCP($K3$). If w($K1$) > w($K2$), then either both the false coins are in $K1$ or one false coin is in $K1$ and another is in $K3$. So, we compare $K2$ and $K3$ in the next step. The former case arises when w($K2$) = w($K3$). So, TCP($K1$) is applied. But if w($K2$) < w($K3$), then OCPH($K1$) and OCPH($K3$) are performed. Figure 2(a) shows the decision tree when $(n+1)$ is divisible by 3. In this case the algorithm does the same thing as for the version $n|3$ except the second level of comparisons. It takes $(|K2|-1)$ coins instead of $|K2|$ and $(|K1|-1)$ coins instead of $|K1|$. For the sake of determinism, we always prefer the coins from set K_i except the last coin in the set. The subsequent operations in the levels followed are shown in the decision tree.

For $(n-1)|3$, the algorithm proceeds in the same way as for the version $(n+1)|3$ with a little difference in the second level of comparisons in the all the branches. As $K3$ contains two coins less than that of $K1$ or $K2$, it receives $(|K2|-2)$ coins instead of $|K2|$ and $(|K1|-2)$ coins instead of $|K1|$. The operations in the subsequent levels are shown in the decision tree in Figure 2(b). When we call OCPL(K_i) or OCPH(K_i), the algorithm proceeds dealing with that set K_i of coins and we divide it into two equal subsets for further weighing. If this set contains even number of coins we put half of them on the left pan and the remaining half on the right pan and weigh, i.e., comparison is performed. Again at this stage, if we are searching for a heavier coin, i.e., in case of OCPH(), after the weighing we deal only with the coins in the heavier pan. If the number of coins is odd, we divide them into two equal halves and one coin remains out of weighing. If the weighing results in inequality, we focus on either the left pan or the right pan depending on the outcome we examine.

So far we have developed an algorithm to find out two false coins among a set of n coins where both the coins are equally heavier, i.e., w(H1) = w(H2). For the variant where two coins are equally lighter, i.e., w(L1) = w(L2), the algorithm works as well.

4. Experimental results

In this section, we choose some values of n so that it would cover all the three categories for the subdivision of n and calculate the average number of comparisons required. To compute the average case complexity, we have to consider pair of false coins in all possible pairs of indexed locations. Hence for a given value of n, there are ${}^{n}C_2$ combinations as the combination of ith heavier and jth heavier is same as the combination of jth heavier and ith heavier being the two false coins equally heavier, we cannot distinguish them. Hence, there are $O({}^{n}C_2)$ leaves in the decision tree. The average number of comparisons required against the number of given coins are shown in Table 1 and the same is plotted in Figure 3.

Table 1: Average number of comparisons for some values of n.

Number of coins (n)	**Total number of comparisons (S)**	**Possible number of false coin combinations ($C = {}^{n}C_2$)**	**Average number of comparisons (AVG = S/C)**
9	171	36	4
11	277	55	5
20	1254	190	6
27	2565	351	7
29	2939	406	7
36	5508	630	8
46	9201	1035	8
54	13365	1431	9
72	27396	2556	10
82	34267	3321	10
100	54926	4950	11
108	65232	5778	11
144	130842	10296	12
198	251883	19503	12
200	254788	19900	12

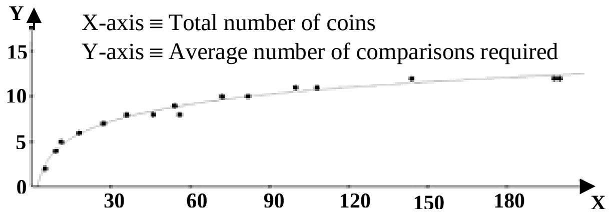


Figure 3: Plot of average number of comparisons required against the number of coins.

5. Computational complexity

At each iteration, n is divided into nearly three equal parts and the cardinality of the set on which the operations are actually performed, always reduces by a factor of three. Let us consider the case $n|3$. As we see at the ith level, each set is of cardinality $n/3^i$. Now if we reach a set with five coins, then we can solve it using exactly four comparisons. So, let at the ith level of comparison the cardinality of the set reduces to five. Thus, $n/3^i = 5$, i.e., $3^i = n/5$. Hence, $i = \log_3(n/5)$. Again, if TCP(K_i) is applied at each iteration before reaching a set with five coins, $2\times i$ comparisons are required resulting in $2\times i+4$ comparisons in total. If OCPH(K_i) or OCPL(K_i) is applied at kth level of comparison, it is definite that before that iteration TCP() is applied for (K–1) times. We know that OCPH() or OCPL() requires $\lceil \log_2 n \rceil$ comparisons and at kth level it is to be applied on $n/3^k$ coins. Hence, it requires total number of $2|K|+2\log_3(n/3^k)$ comparisons. So, in the worst case it would take $O(2|K|+2\log_3(n/3^k)) + O(2\times\log_3(n/5)+ 4)$, i.e., $O(\log n)$ comparisons as a whole.

6. Conclusion and applications

The raising issue of counterfeits violates intellectual property right and also causing damage to both producer and consumer. To identify the counterfeit goods like pirated electronic gadgets, counterfeit batteries used in a digital camera, pharmaceuticals, valuable ornaments, the solution of counterfeit coin problem are used. In this paper, we have developed an algorithm to identify two false coins among a set of n coins that are identical in appearance. In this case we have assumed that both the false coins are equally heavier (or lighter) than the weight of a true coin, and developed algorithms for identifying the same. The algorithm solves the problem with time complexity $O(\log n)$. The most important fact is that the decision tree structure can be used to solve such problems of large size, by eliminating a part of the solution domain after each step of decision making. Especially, as our algorithm works for any value of n, it does not matter if the value of n is not known a priori.

REFERENCES

1. R.Bellman and B.Gluss, on various versions of the defective coin problem, *Information and Control*, 4(2-3) (1961) 118-131.
2. J.Ghosh, P.Senmajumdar, S.Maitra, D.Dhal and R. K. Pal, A generalized algorithm for solving n coins problem, *Proc. of 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE 2011)*, Shanghai, China, vol. 2, pp. 411-415, Jun. 10-12, 2011.
3. J.Ghosh, P.Senmajumdar, S.Maitra, D.Dhal and R.K.Pal, Yet another algorithm for solving n coins problem, *Assam University Journal of Science & Technology: Physical Sciences and Technology*, 8(II) (2011) 118-125.
4. B.Manvel, Counterfeit coin problems, *Mathematics Magazine, Mathematical Association of America*, 50(2) (1977) 90-92.
5. R.Tošić, Two counterfeit coins, *Discrete Mathematics*, 46 (1995) 295-298.