A new solution to the puzzle: working computers

ZHANG Chenzi

Department of Computing
The Hong Kong Polytechnic University, Kowloon, HK
08839125d@comp.polyu.edu.hk

Abstract. The *working computers puzzle* is the problem of querying and determining the first working computer while excluding the noise from damaged ones. In this report, we investigate the *determinability* of the puzzle and study its characteristics. Based on these properties, three existing solutions are reviewed and a new solution is proposed. Moreover, the correctness of each solution is proved and their time complexity is analysed in worse case scenario. Finally the remaining problems are presented.

1. Introduction

As presented by G S MANKU[1], the *working computers puzzle* is defined as follow:

> A room has n computers, less than half of which are damaged. It is possible to query a computer about the status of any computer. A good computer will always tell the truth, while a damaged computer could give wrong answers. The goal is to discover an undamaged computer using as few queries as possible.

The hardness of this problem lies in that a damaged computer could produce either correct or incorrect answers when asked for the status of other computers. So the order of queries and the distribution of damaged computers will determine how many queries are needed to solve this puzzle. In order to find a solution for the general problem in a deterministic way and make it possible to compare time complexity of various solutions, we only compare their performance in the worst case scenario.

Definition 1. Suppose x and y are two computers. Let the function q(x,y)={GOOD or DAMAGED} be the answer of computer x when asked about the status of y. The response could be GOOD or DAMAGED, meaning that

computer y is either good or damaged respectively.

Definition 2. Given a group Y whose size is n, let G(Y) be the number of good computers and D(Y) the number of damaged computers, then n=G(Y)+D(Y). We define that a group is *determinable* if there are more good computers than damaged ones in that group, i.e., G(Y)>D(Y)>=0.

If a group G is *determinable*, it has the following **properties**:

1. If a damaged computer is removed from group G, the group remains *determinable;*

2. If a good and a damaged computer are removed from group G, the group remains *determinable;*

3. If the group G is divided into two subgroups, at least one of them is *determinable;*

4. If the size of group G is 1 or 2, all the members of G are good computers.

The proof of these **properties** is given below and they will be used later on:

1. For a *determinable* group, i.e., G(n)>D(n), removing a damaged computer is equivalent to D(n)-1. The new group remains *determinable* because G(n)>D(n)-1.

2. Removing a good and a damaged computer reduces the number of computers in both category by one. Since G(n)>D(n) then G(n)-1>D(n)-1 holds.

3. Given a *determinable* group, it is possible that not all of the subgroups are *determinable*. But it can be derived that at least one of them must be *determinable*.

   Divide a group Y into subgroups A and B, then Y=A+B, G(Y)=G(A)+G(B) and D(Y)=D(A)+D(B) hold. Suppose subgroup A is *undeterminable*, i.e.,

   $$D(A)>=G(A) \quad (1)$$

   because group Y is *determinable*, i.e.,

   $$D(Y)<G(Y) \quad (2)$$

   adding (1) and (2) gets D(Y)-D(A)<G(Y)-G(A), which is equivalent to D(B)<G(B).

4. Suppose there is only one damaged computer in a *determinable* group Y, i.e., G(Y)>D(Y)=1, it's safe to conclude that G(n)>=2 and the group has at least 3 computers. For a group containing one or two members, there is no room for a damaged computer.

Determinability is the key to solve this puzzle. The following example shows that the puzzle cannot be solved without this property.

Example 1. Suppose a group has two computers. Computer A is good and computer B is damaged. Consider the following valid query answer:

q(A,A) = GOOD; q(A,B) = DAMAGED;
q(B,B) = GOOD; q(B,A) = DAMAGED;

In this case, since the response from A and B are symmetric from the view of each computer, that is, each has claimed itself as good and the opposite as damaged, it is impossible to determine which computer is good and which one is damaged.

There are several ways to solve this problem, and the solutions have various time complexities. The problem was posted by Manku[1] on his blog together with suggested solutions. In section 2, we will give a review for these existing methods. In section 3, a new method will be proposed with the proof of its correctness.

2. Previous solutions

Three existing methods will be reviewed in this section and one original method will be introduced in next section. The name and complexity of each methods are listed in the following table:

Method Name	Time Complexity	Note
full query	O(n^2)	existing
linear query	O(n)	existing
pairwise query	O(n)	existing
division query	O(nlogn)	original

Method 1: Full Query, time complexity: O(n^2)

Based on the method proposed by Manku[1], the status of a computer can be determined in n queries:

Algorithm 1: Determine the status of a computer(t)

```
SINGLE-STATUS(t,G)
L1: nG := 0; nD := 0;
L2: for each computer x in G
L3:     if q(x,t) = GOOD
L4:         nG := nG+1
L5:     else nD := nD+1
L6: if (nG>nD)
L7:     Return computer t as a good computer
L8: else Return computer t as a damaged computer
```

Because the group is *determinable*, that is, more than half of the computers are good and they always tell the truth, so the status of a computer will always be correctly reflected by the majority among the answers. Repeat ***Algorithm 1*** for each computer in the group, we can obtain status of all computers and it is sufficient to find one good computer to solve the puzzle.

Algorithm 1 needs n queries to determine the status of one computer. And this process is repeated n times for each computer in the group. So the time complexity for full query method is n^2.

Method 2: linear query, time complexity: O(n)

If the computers in a group are labeled from 1 to n, they can be referred by c[1], c[2] … c[n]. If all the computers are asked for the status of its next computer: q(c[i],c[i+1]) (i = 1,2...(n-1)) and all of the answers are GOOD, then the last computer c[n] must be a good computer. This can be proved by contradiction: if c[n] is a damaged computer then previous computer c[n-1] must be damaged, otherwise there is no way to explain the answer of query: q(c[n-1],c[n])=GOOD. If c[n-1] is damaged then c[n-2] must also be damaged for the same reason. Continue the process and we conclude that c[n], c[n-1], c[n-2] … c[2], c[1] are all damaged computers, which is contradictory with the *determinable* property. So we've proved lemma 1:

Lemma 1. For a *determinable* group, if all computers form a query chain and the answers are all GOOD, that is, q(c[i],c[i+1])=GOOD for i=1,2...n-1, then the last computer of the query chain c[n] is GOOD.

In the same manner we can build the following algorithm, which intends to build such a query chain:

Algorithm 2: linear query

```
LINEAR-QUERY( G )
L1: L := empty list
L2: for each computer t in G
L3:     if list L is empty
L4:         Add t into list L
L5:     else
L6:         x := L's last element
L7:         if ( q(x,t) = DAMAGED )
L8:               Remove t and x from group G and list L
L9:         else Append t after x in list L
L10: END of FOR LOOP
L11: Return the last element of L as a good computer
```

The removal action in **L8** persists the *determinable* property. If q(x,y)=DAMAGED for two computers x, y, there is at least one damaged computer among them. Based on **property #1** and **property #2**, removing this pair of computers persists the *determinable* property. When the algorithm terminates, we know the final list contains all of the members in the final group and all the query answers are GOOD. Based on **lemma 1**, the last element of the list is a good computer.

For the worst case scenario, which is no computer is removed from the group, the number of queries is exactly n-1.

Method 3: pairwise query, time complexity: O(n)

This method works like linear query method in terms of that both of them try to reduce the problem size while persists the *determinable* property. But this algorithm works in a recursive way:

Algorithm 3: pairwise query

```
PAIRWISE-QUERY( G )
L1: if size of G is an odd number
L2:     t := one computer in G
L3:     if SINGLE-STATUS(t,G) = GOOD                //Algorithm 1
L4:          Return t as a good computer
L5:     else Remove t from group G
L6: for any two computers x, y in group G
L7:     if (q(x,y)=GOOD AND q(y,x)=GOOD)
L8:         Remove computer y from group G
L9:     if (q(x,y)=GOOD AND q(y,x)=DAMAGED) OR
           (q(x,y)=DAMAGED AND q(y,x)=GOOD) OR
           (q(x,y)=DAMAGED AND q(y,x)=DAMAGED)
L10:        Remove x and y from group G
L11: End of for loop
L12: Jump back to L1
```

The correctness of this method is based on the persistence of *determinable* property during each iterations. Firstly if a pair of computers have the answer as (GOOD,DAMAGED) or (DAMAGED,DAMAGED) then there is at least one damaged computer among them. Based on **Property #1** and **Property #2** remove this pair of computers persists the *determinable* property. Secondly, if the answers of the computer pair is (GOOD,GOOD) either both of them are good or both of them are damaged. Removing half of these paired computers persists the ratio of good computers to damaged computers, i.e., G(n)>D(n) infers G(n)/2>D(n)/2. So the group is always *determinable* through these reductions. And this algorithm will terminate because in each iteration the size of the group decrease strictly and the initial group size is finite.

The IF statement between **L1** to **L5** is used to ensure the group size is even before the pairwise reduction. Without the even size guarantee the algorithm may not generate correct answer, the following example illustrates the flaw:

Example 2. In a group of three computers, the first and second computers are good while the third computer is damaged, i.e., c[1] =GOOD, c[2]=GOOD and c[3]=DAMAGED. If we directly go to the L6 of ***Algorithm 3***, the c[1] and c[2] will be paired up and c[3] will be left alone. The query answers of the pair are q(c[1],c[2])=GOOD and q(c[2],c[1])=GOOD, so one of them will be removed, say c[1]. Only c[2] and c[3] are left. As shown by *Example 1* this scenario is not solvable because it is not a determinable group.

The group size reduces at least half in each iteration because all computers belong to a pair, and either the whole pair or the half is removed. The worst case scenario for this method is that, there are only half of the candidates removed in each iteration and the first good computer is determined at the last moment, which is the (log n)th iteration. The time complexity formula is f(n)=f(n/2) +O(n), solving it we get f(n)=O(n). Compare this time complexity with linear query method's, we notice that both of them are linear. But a closer analysis results that pairwise query method needs more than 2*n queries for an odd sized group. While the linear query method needs no more than n-1 queries. So in terms of number of queries the linear query method is better than the pairwise query method.

3. New discovered solution

Method 4: Division query method, time complexity: O(nlogn)

Like other divide and conquer algorithms, the group is divided into two subgroups and solved recursively with the assumption that all of the subgroups are *determinable*, while this may not be true for some of them. Since the return value of subgroups may not be correct when given a *non-determinable* group, the ***Algorithm 1*** is used to ensure the correctness of final answer.

Algorithm 4: division query method

```
DIVISION-QUERY( G )
L1: if the group size is 1 or 2
L2:     Return the first computer as a good computer
L3: Divide the group G evenly into GA and GB
L4: A := DIVISION-QUERY(GA)
L5: B := DIVISION-QUERY(GB)
L6: if SINGLE-STATUS(A,G)=GOOD                  //Algorithm 1
L7:     Return A as a good computer
L8: else Return B as a good computer
```

In order to ease the understanding and proof, we can visualize the whole process of divide and conquer as a binary tree. The root node of the binary tree is the original problem and its children are the subproblems. The height of the tree is roughly log(n) and the node in each depth level has roughly the same group size. The leaf nodes are the problems that so small enough, with size 1 or 2. Based on **Property #4** they can be solved directly if the group is *determinable*. If we mark the nodes that persist the *determinable* property with red color and other nodes with black color, the tree has the following properties:

1. the root node is red (the original problem is *determinable*)
2. for any red node, there is at least one child in red (the persistence of *determinable* property that shown in **Property #3**)
3. from previous properties, we can conclude that there exists one path from root to some leaf node that all colored in red

Now let’s prove the correctness of this method in a constructive way:

Step 1. For the red leaf nodes, their group sizes are equal to one or two, and the correctness of their results is assured by **Property #4**

Step 2. For any node who is in red and whose child is proved to produce correct answer (for the first iteration the nodes directly upon the red leaf nodes are in this category), it gets two candidates of good computer from its subgroups. Based on **Property #3**, at least one of these candidates is correct. And because the node itself is in red color (it is *determinable*), the ***Algorithm 1*** can assure the return value is correct.

Step 3. Using the same logic and performing *Step 2* repeatedly for the nodes of the red path, which starts from a leaf node and ends with the root node, can prove the correctness of final answer in a bottom up manner.

Notice. Black nodes can be disregarded even though they produce incorrect answers, because we can illuminate their answer during the merge in some node.

As this algorithm disregards the distribution of computer status and only follows the designed instructions, the total number of queries does not change if the group size remains the same. Suppose the number of queries for a group whose size is n is f(n), we can get the following recursive formula: f(n)=2*f(n/2)+O(n), which is equivalent to f(n)=O(nlogn).

4. Conclusion

In this report about *working computer puzzle*, we reviewed three existing solutions and proposed a new one. In terms of the number of queries, the linear query method is the best because it only requires n-1 queries to determine the first good computer. The second best algorithm is pairwise query method which needs more than 2*n queries in worst case scenario. The next method is division query method which requires O(nlogn) queries. The full query method needs the most number of queries, which is n^2. However all of these methods are equally valuable when considering the insights they brought. Each method focuses on different dimensions and reveals some new structure of the puzzle.

About this puzzle, there are some problems left to be solved in the future. One of them is to determine the minimum number of queries required to find the first good computer. Another problem is to analyse the number of queries in a probabilistic model rather than in the worst case scenario.

References:
1. G S MANKU. (2008). *Puzzle: Working Computer.* Available: http://gurmeet.net/puzzles/working-computer/. Last accessed 28th Nov 2012.