Prisoner's Names in Boxes

Ryan Williams

December 9, 2015

1 Introduction

When Danish computer scientist Peter Bro Miltersen first devised this problem, he didn't believe there to be a solution. That was until colleague Sven Skyum's suggestion over lunch one day[2].

The Concept is simple: there are 100 prisoners, each with a box containing their name. One by one they enter the room and try to find their own box. They may look in a maximum of 50 boxes and if they find their allocated box, they leave the room exactly as they found it. They are not allowed to communicate with other prisoners once they have entered the room but they may discuss strategies beforehand. If one prisoner fails to find his box then all prisoners are to be executed, therefore they must all find their names to survive.

At first sight you may believe that they stand a decent chance just choosing a random 50 boxes each. Using code to model the situation and maths to calculate the probability, I will prove why this is not the case and share a strategy that has been proven to have the highest chance of survival[2].

2 Random Box Selection

2.1 Coding the Random Approach

I have written a code which contains a function that replicates the scenario where each prisoner enters the room and selects a random 50 boxes each. We know that their survival is very unlikely but how unlikely is it? I decided to find the rate of success by running the function 10 thousand times and find the average. However, in all those trials there were precisely 0 successes. Even when run 10 million times all trials failed.

```
def randombox():
    """
    a function to model prisoners choosing boxes at random

    arguments: i = prisoner

    output: True (success) or False (failure)
    """
    boxes = list(range(1, 101))  # creates list 'boxes' containing 1-100
    i = 1  # set variable 'i', start at first prisoner
    while i <= 100:  # whilst there are prisoners still to open boxes
        if i in random.sample(boxes, 50):  # is 'i' in a random 50 numbers
            i += 1  # increment 'i' - next prisoner
        else:
            return False  # if it is not contained they have failed
    return True  # if all 100 prisoners make it through they have succeeded
```

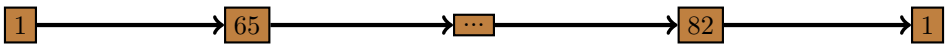
2.2 Calculating the Random Approach

Calculating the probability of success from choosing random boxes would take far too long for the computer using the code above. So let's take a look at the mathematical side of this. Each prisoner that enters the room has a 0.5 chance at finding his box, however the chance that 2 consecutive prisoners find their box would be 0.5×0.5 or 0.5^2. Therefore, the chance of this happening 100 times consecutively would be $0.5^{100} = 7.886... \times 10^{-31}$. Not very favourable. Luckily for the prisoners, a solution with considerably more chance of survival exists...

3 Strategic Approach

3.1 The Solution

Before entering the room the prisoners assign themselves a unique number between 1 and 100. That will be their box. When they enter the room they begin by opening their own box. For example, prisoner 1 would open the first box and if it contained a number that was not his own, e.g. 65, he would open box 65, then the box of that contained number and so on until he opens the box that holds his number, in this case 1. If he manages to find his own number in the first 50 attempts then he has succeeded.



3.2 Coding the Solution

Just like in the random box selection scenario, I found it appropriate to write a code which contains a function that replicates the prisoners situation, this time following the solution above. I also added the chance to change the amount of boxes/prisoners, which you can do by changing variable 'm' which is set as 100 by default.

Firstly, we define the list of numbers before we then shuffle them into a random order. After setting the variables, a while loop begins that continues until every prisoner has opened their boxes. If the prisoner has opened no boxes then 'a' takes the value of the number in the shuffled list with index 'i', then we increment 'b' as a box has been opened. If 'a' does not equal 'i', 'a' then takes the value of the number with index 'a', and once again we increment 'b'. If 'a' = 'i' then the prisoner has found his own number and we increment 'i', meaning we move onto the next prisoner, and reset 'b' to 0. If 'b' ever reaches half the amount of boxes and we have not found the prisoners number then the prisoners have failed and the function returns False. However, if we pass through the loop repeatedly until 'i' reaches 'm' then all of the prisoners found their number and the function returns True.

```
def orderedbox(m = 100):
    """
    a function to model the strategic solution

    arguments: i = prisoner number;
               a = value in box;
               b = number of boxes opened;
               m = amount of boxes/prisoners.

    output: True (all prisoners succeed) or False (one prisoner fails)
    """
    boxes = list(range(m))  # creates a list with 'm' integers
    random.shuffle(boxes)  # moves the list of boxes into a random order
    i, a, b = 0, -1, 0  # set variables
    while i < m:  # while we haven't run out of prisoners
        if b == 0:  # if the prisoner hasn't opened a box yet
            a = boxes[i]  # the value in the prisoners own box
            b += 1  # add 1 to the amount of boxes they've opened
        if a != i:  # if they haven't yet found their number
            a = boxes[a]  # opens the box of the number they just found
            b += 1  # add one to the amount of boxes they've opened
        if a == i:  # if box they've opened contains their number
            i += 1  # move onto the next number/prisoner
            b = 0  # reset amount of boxes opened for the next prisoner
        if b == m / 2 and i != m:  # all 50 boxes opened without success
            return False
    if i == m:  # if all prisoners make it through the steps of the function
        return True
```

Using a separate piece of code I then ran the function multiple times and calculated the rate of success. It averaged around 0.31 which is a dramatic improvement to our first attempt with random selection.

3.3 Calculating the Solution

Let's have a look at why the probability of survival is so much higher using this technique. When a prisoner succeeds in finding his own number it is because the path of boxes he opened has sent him to his own box. The only way that he could fail is if the

cycle containing his number is larger than 50. So to calculate the probability of success we must first find the probability that the permutation will contain a cycle larger than 50 then subtract that from 1.

This calculation has been very well demonstrated on "The Simon Fraser University" website[1]. There it states that there are $\binom{2n}{k}$ many ways to pick the entries of the cycle, where n is half the amount of boxes and k is the length of a cycle. It then goes on to say there are $(k-1)!$ ways to order them in the cycle and $(2n-k)!$ ways to permute the rest. Multiplying all of them together will give:

$$\binom{2n}{k} \cdot (k-1)! \cdot (2n-k)! = \frac{(2n)!}{k}$$

We know that we can have at most one cycle greater than n, so the number of permutations that contains such a cycle is $\sum_{k=n+1}^{2n} \frac{(2n)!}{k}$. And therefore to calculate the probability we divide that by $(2n)!$:

$$\frac{1}{(2n)!} \sum_{k=n+1}^{2n} \frac{(2n)!}{k} = \sum_{k=n+1}^{2n} \frac{1}{k}$$

As we previously stated we need to subtract this from 1 as it is currently calculating the probability of failure. If we now test this with the example we've been using (where $n = 50$):

$$1 - \sum_{k=n+1}^{2n} \frac{1}{k} \approx 0.31$$

So the result of these calculations taken from "SFU"[1] agree with the results from the code above.

4 Altering the Number of Boxes

It is important to consider how changing the number of boxes/prisoners affects the outcome. You may think that the more you add, the more opportunities there are to make a mistake. Note that in this example as we increase the amount of boxes we also increase how many they may open so that they can still only open half. You may also believe that this increases the amount of permutations with a chance of success. However, using a code to obtain the results after changing the quantity of boxes and another code to plot the graph shown in Figure 1 it appears that the probability does not change, however my code wasn't run enough times to give a reliable result.

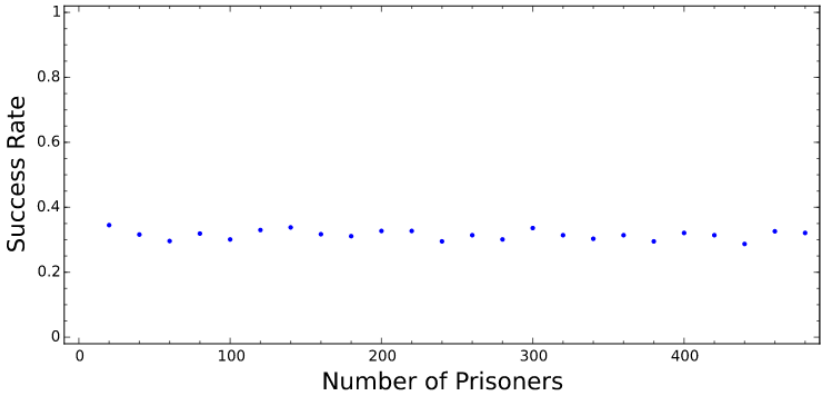


Figure 1: A graph to show the correlation between number of prisoners/boxes and success rate

5 Conclusion

This report has investigated and provided a code for two approaches to this problem and backed up its results with rigorous calculations. Both codes would need to be run many more times for a truly accurate result, however that would require a computer far superior to my own. This may suggest that Figure 1[4] may change after more iterations.

This problem can be investigated much further by changing some of the conditions. For example, we could test how the probability would change when we add more prisoners but we still only allow them to open 50 boxes. Another part of the problem we could look into is how many prisoners must attempt it before we can know whether there is a large cycle in the permutation. I've tried running the code and have it print which prisoner it reaches, and when there is a trial that fails it almost always happens within the first few prisoners. There are an infinite amount of ways this problem can be altered and analysed.

References

[1] Jamie Mulholland. Puzzle: Prisoners Names in Boxes. http://people.math.sfu.ca/~jtmulhol/math302/notes/25-Prisoners-in-Boxes.pdf. Accessed: 2015-12-07.

[2] Peter Winkler. Seven Puzzles You Think You Must Not Have Heard Correctly. https://math.dartmouth.edu/~pw/solutions.pdf. Accessed: 2015-12-07.