Monte Carlo Simulation of 100 Prisoners Cell Problem

Md. Sakibur Rahman
dept. computer science and engineering
BRAC University
Dhaka, Bangladesh
md.sakibur.rahman3@g.bracu.ac.bd

Abstract—**Mathematical problems are always fascinating and among them 100 prisoners' cell problem seemed to be impossible to solve at the beginning but with time mathematicians have solved them with different approaches. Among them the naive approach and the pointer following loop strategy are simulated in this paper with Monte Carlo modeling and the difference was remarkable. The pointer following loop strategy approach showed 30-32% success probability where the naive approach resulted in a very low probability. In this paper both of the simulations are shown with nodal visual representation and graphs.**

I. Introduction

The 100 prisoners problem is a pretty famous problem in probability theory and combinatorics. It has grown in popularity since 2003, because of its elegant and surprisingly efficient solution to a seemingly impossible riddle. The 100 prisoners problem has different renditions in mathematical literature, but the principle is always the same. We have implemented the simulation in Monte Carlo model where we simulate the event thousands of times to get the resultant. The problem statement is the following

"The director of a prison offers 100 death row prisoners, who are numbered from 1 to 100, a last chance. A room contains a cupboard with 100 drawers. The director randomly puts one prisoner's number in each closed drawer. The prisoners enter the room, one after another. Each prisoner may open and look into 50 drawers in any order. The drawers are closed again afterwards. If, during this search, every prisoner finds his number in one of the drawers, all prisoners are pardoned. If just one prisoner does not find his number, all prisoners die. Before the first prisoner enters the room, the prisoners may discuss strategy—but may not communicate once the first prisoner enters to look in the drawers. What is the prisoners' best strategy?"

II. Related Works

In this article [6], the authors described the probability of success and explained the possible optimal solution for two similar yet little different situations. Firstly, they calculated the success probability of the 100 people. Since all of the prisoners only get to open n boxes from a total of 2n boxes, their rate of succession is $(½)^{100}$. Which is a really really small rate. They must be extremely lucky to be successful in this test without any algorithm. To overcome this rare situation they tried an algorithm on a smaller scale with 10 boxes. The first idea they tried was that team members 1-5 examined lockers 1-5, and team members 6-10 examined lockers 6-10, they would succeed provided numbers 1-5 are placed in lockers 1-5. But in this case the probability for member 6-10 is influenced by the probability of success of member 1-5. The ideal strategy would be the one where member 1 succeeds then everyone else does too. Their good strategy is simple to implement and the choice of the next locker does not depend on the entire sequence of numbers seen but only on the most recent number. The good strategy has player Bi start by opening locker i. Then if he finds number k at any stage and k != i, he opens locker k next. Notice that player Bi, never opens a locker (other than locker i) without first finding its number, so each time he opens a new locker he must find either his own number or the number of another unopened locker. This creates a cycle for the players and after calculating the permutations of 10 numbers and their permutations of created cycles, the probability of players winning hits overall of 35%, and if we do the same calculations for 100 players it results around 30-31%. Which is a far better outcome than the naive approach to find the specific numbers of the players.

In this paper [3], the author's algorithm was that Player Pi starts at the beginning of the bin Bi at box Di. Pi sequentially checks the next boxes, keeping track of the surplus since Di until the surplus becomes non-negative (at box mi). Pi opens box mi, if $\pi(i) = i$, then the player is done. Otherwise, Pi starts again at the beginning of the bin B$\pi(i)$ at box D$\pi(i)$, resets the surplus and does step 2 and 3 until he finds his number or has opened b/k boxes. The general idea behind that strategy is close to the special case in [6]. They could find some better bounds by tweaking the calculation as Y. Wang did in his paper [5] by finding a new lower bound on the probability of success.

In the paper [1], the authors tested the execution time of an existing optimal strategy to solve the 100 prisoners problem in both sequential and parallel implementation with graphics processing units. And they compared the execution times to see how they vary when the problem size increases.They have shown that the increased problem size results in improved GPU usage for specific factors. These factors include memcpy size and compute utilization. The speed up increases by increasing the problem size. This relationship is not the result of the process of a larger data to be processed and we are using the GPU efficiently. Rather, this is because the O(n2) sequential algorithm's execution time grows faster than the parallel algorithms. On average the parallel user defined execution performed better than the sequential and parallel thrust execution.

III. Methods And Analogy

A. The Naive Approach

If the prisoners start to search for their own boxes in random order without any sort of predetermined strategy or algorithm, each individual will succeed in finding his own number with probability 1/2. If they act independently, they must get lucky 100 times in a row, and the team will win with probability only $(1/2)^{100}$. And if we calculate it we get a probability of 0.00000000000000000000000000000078. Which is a really really small number and will take hundreds of hours of simulation time. So it was convenient to prove it mathematically

B. The Pointer Following Loop Strategy

- The prisoner will go to the box numbered with his own cell number.
- If his cell number matches with the number inside the box, then he will get out.
- Else he will go to the box numbered with the number he found inside the previous box.
- He will keep searching until he opens 50 boxes.
- If the loop he was following ends, he will start again from any random numbered boxes and follow the same strategy until 50 boxes.

C. Analysis of Pointer Following Loop Strategy

According to [two], a permutation of the numbers of the numbers from 1 to 100 can only contain at most one cycle of length $\boldsymbol{l} > 50$, where $\boldsymbol{l}$ is the length of the longest cycle. We want to calculate how many different permutations would have a cycle of length $\boldsymbol{l}$, in order to calculate the probability of having a cycle of that length.

There are $\binom{100}{l}$ ways to select which numbers are in the cycle. Within the cycle, there are ($\boldsymbol{l}$ - 1)! ways of organizing the numbers because of the cyclic symmetry. Lastly, the remaining numbers can be arranged in (100 - $\boldsymbol{l}$)! Ways. By consequence, the numbers of permutations of the numbers from 1 to 100 with a cycle of length $\boldsymbol{l} > 50$ is

$$\binom{100}{l}(l-1)!(100-l)! = \frac{100!}{l} \qquad (1)$$

Since there are a total of 100! possible permutations, then the probability of success of this strategy is

$$1-\frac{1}{100!}(\frac{100!}{51}+...+\frac{100!}{100}) = 1-(\frac{1}{51}+...+\frac{1}{100}) \approx 0.311 \qquad (2)$$

Even if we increase the number of prisoners to 2n and each prisoners can open n boxes each, the probability of success will still always be lower bounded by 1 - ln2 $\approx$ 0.306.

IV. Implementation And Results

A. Languages and Libraries

- **Pandas** - It is used for data manipulation in tabular form.
- **Numpy** - It is used for numerical calculation and randomized number generation and implementation.
- **Tqdm** - It is used for Run time Analysis
- **Networkx** - It is used to visualize the boxes in loops or cycles in form of nodes.
- **Matplotlib** - It is used for plotting the result graphs.
- **Seaborn** - For better visualization for matplotlib

[1]Our Working Project Code
[2]Source code

B. The Naive Approach

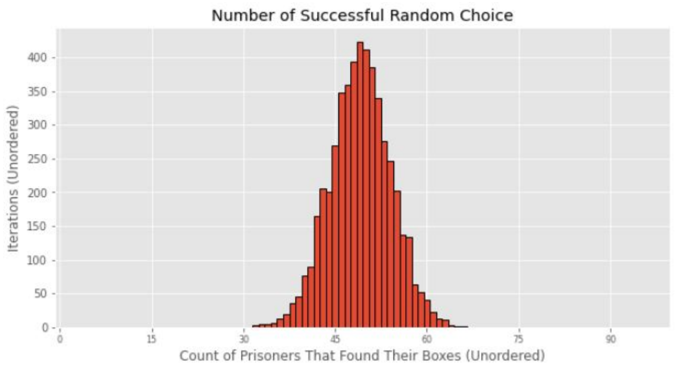

Fig. 1. Number of Successful Prisoners with the Naive Approach

In this approach we have randomly simulated the prisoners to pick boxes in random order and the iteration was 5000 times, However the probability distribution was so low it was very difficult to visualize the plot for 5000 iterations. In Fig. 1 the result is shown and it follows a normal distribution. As it is a randomly generated simulation the values will change each time we run the simulation but the probability will always be very low.The total run time was 36 seconds to simulate the event with 138.88 iterations per second.

C. The Pointer Following Loop Strategy

After applying the Pointer Following Loop strategy, the probability of success for 100 prisoners increased significantly. We can notice in Fig. 2, among 5000 iterations, 45-50 prisoners have found their boxes less than 4% of the time. On the other hand, 100 out of hundred prisoners have found their boxes around 1500-1600 iterations. So the probability of success is between 31-32%. To be more precise, the probability was 0.3124. It will change every time we re-run the simulation but the change will be very insignificant. The

[1]Simulation of 100 Prisoners Problem with Monte Carlo Modeling
[2]Source of Our Code: Kaggle Notebook

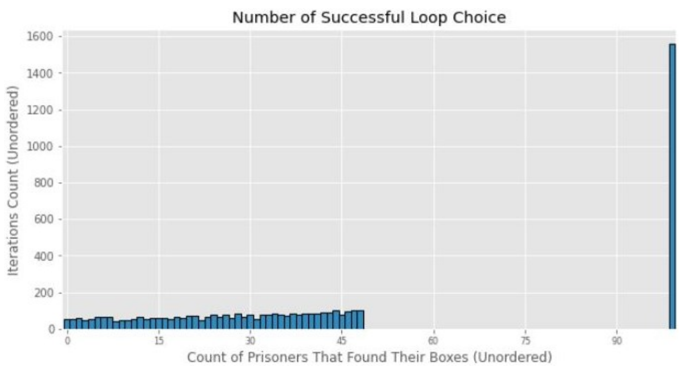

Fig. 2. Number of Success with Pointer Following Loop Strategy

total run time for this strategy was close to 3 seconds with 1746.42 iterations per second. Which is very efficient as well compared to the naive approach.

V. Monte Carlo Simulation

According to [2] The Monte Carlo method is, broadly,a technique that solves mathematical problems by solving their statistical analogues, by subjecting random numbers to numerical processes. The method uses statistical sampling to obtain solutions to deterministic or stochastic problems that cannot readily be solved using closed-form techniques.

A. *Areas of Applications*

From [7], We get that Monte Carlo methods are especially useful for simulating phenomena with significant uncertainty in inputs and systems with many coupled degrees of freedom.

- **Physical Sciences:** Monte Carlo method is used in computational physics, physical chemistry, and related applied fields. It has diverse applications from complicated quantum chromodynamics calculations to designing heat shields and aerodynamic forms as well as in modeling radiation transport for radiation dosimetry calculations
- **Engineering:** Monte Carlo methods are extensively used in engineering for sensitivity analysis and quantitative probabilistic analysis in process design.
- **Computer Graphics:**In Path tracing or we can also call it ray tracing, repeated sampling of any given pixel will eventually cause the average of the samples to converge on the correct solution of the rendering equation, making it one of the most physically accurate 3D graphics rendering methods in existence. Because of this we can get more accuracy in lighting and shadow effects.
- **Computational Biology:** Monte Carlo methods are used for Bayesian inference in phylogeny, or for studying biological systems such as genomes, proteins, or membranes.
- **Applied Statistics:** In applied statistics, Monte Carlo methods may be used for comparing competing statistics for small samples under realistic data conditions, provide implementations of hypothesis tests that are more efficient than exact tests such as permutation tests (which are often impossible to compute) while being more accurate than critical values for asymptotic distributions and To provide a random sample from the posterior distribution in Bayesian inference.
- **Artificial Intelligence for Games:** Monte Carlo methods have been developed into a technique called Monte-Carlo tree search that is useful for searching for the best move in a game.

B. *Our Implementation*

We have used the traditional and similar approach as coin toss simulation and calculating the value of pi using Monte Carlo simulation in our problem. To prove the mathematical proof we needed to run the even multiple times and almost all the times it showed the similar result.

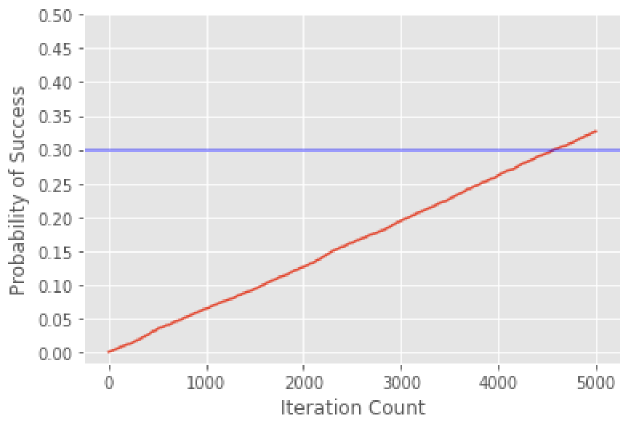

Fig. 3. Success Probability vs Iteration plot in Monte Carlo modeling approach

In Fig. 3, The visualization of success probability has been shown in Monte Carlo simulation. We can notice the probability of success is increasing with the iteration (N=5000) and crossing the threshold of 30% and the probability in this simulation was 0.327. But with every run of the whole operation the success probability will change because the operation is very random.

VI. Discussion

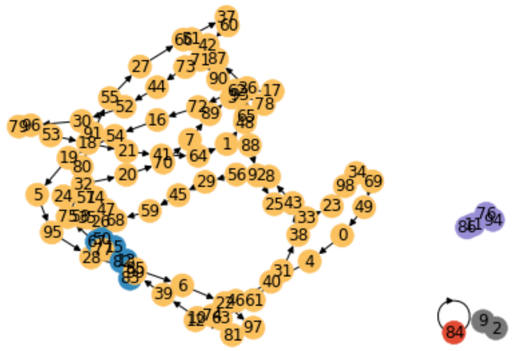

Fig. 4. Generated Loops while simulating in form of Directed Nodes

Among the two approaches we have implemented, the pointer following loop strategy has shown remarkably efficient results with less run time than the naive approach.The underlying idea for loop strategy is the created loops while looking for a specific box. According to [4] the loop strategy works because

- Boxes are ”linked” to each other.
- You can think of this in terms of a graph network where the boxes are nodes and the paper is an edge.
- The graph consists of any number of ”loops”.
- By starting with their own number the prisoner is guaranteed to be in their own loop.
- If the largest ”loop” within the graph is less than 50, all prisoners will succeed.

There is a one percent chance that a random arrangement of slips results in a loop of length 100 and this is a general result the probability that we get a loop of length 99 is 1 over 99. The probability that we get a loop of length 98 is 1 over 98. so the probability that there is a loop longer than 50 is $(1/51+1/52+1/53+...)$ and after adding all these up and it equals 0.69 there is a 69% chance of failure for the prisoners meaning a 31% chance of success where the longest loop is 50 or shorter.

VII. Conclusion

Both of the approaches we have simulated have shown the expected results compared to the mathematical proof. The Loop strategy has shown better results in case of success probability and run time. However it depends on the generated loops , so it has a dependency on previously found nodes. On the other hand, the naive approach was very inefficient in case of success probability and run time but it has no dependencies on the previous or next box because the boxes were selected randomly.

References

[1] F. Jenabi and H.-R. Hamidi. The 100 prisoners problem: Parallel execution using graphics processing unit. *Journal of Telecommunication, Electronic and Computer Engineering*, 9:211–215, 2017.

[2] R. E. Marks. *Monte Carlo*, pages 1–4. Palgrave Macmillan UK, London, 2016. ISBN 978-1-349-94848-2. URL https://doi.org/10.1057/978-1-349-94848-2_709-1.

[3] T. Schoen. *Introduction to the general case of the 100 Prisoners Problem.* May 2018. URL https://math.mit.edu/~apost/courses/18.204_2018/Timothee_Schoen_paper.pdf.

[4] Veritasium. The riddle that seems impossible even if you know the answer, June. 2022. URL https://www.youtube.com/watch?v=iSNsgj1OCLA.

[5] Y. Wang. Project report - the locker puzzle, Nov. 2015.

[6] M. Warshauer and E. Curtin. The locker puzzle. *The Mathematical Intelligencer*, 28(1):28–31, Dec. 2006. doi: 10.1007/bf02986999. URL https://doi.org/10.1007/bf02986999.

[7] Wikipedia. Monte Carlo method — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Monte%20Carlo%20method&oldid=1107916943, 2022. [Online; Accessed 11-September-2022].