Block-Chain: Double Spending Problem
Mobile Internet Hw2

Liu Hanwen (517030910294)
Shanghai Jiao Tong University IEEE Honer Class
hwLiu2017@sjtu.edu.cn

April 28, 2020

1 Introduction

Double-spending is a potential flaw in a digital cash scheme in which the same single digital token can be spent more than once. For block-chain, it will not happen under normal condition. It is because that we have some mechanisms to grantee such as Unspent transaction output(UTXO) in BTC. While if someone have 51% calculation ability, he will has the opportunity to launch an attack and gain large number of illegal money.

For example, if the hacker make a transaction to the trading venue, he pay with Bitcoin and he get real money. When the deal is concluded, he disclose his previously created attack chain which is longer than the current main-chain. The former deal will withdraw, his Bitcoin return to his account. In this case, he receives both real money and doesn't lose any his origin Bitcoin.

2 Model

The question says that: The attacker has 51% calculation ability, in this case, each new block will has 51% probability to extend in the specific branch during each discrete unit time.

To withdraw the former transaction, the hacker needs to create more than 1+k blocks with the competition with the origin master chain. That is the target position s needs to satisfy: $1+k+1 > s$.

Based on these descriptions, we can construct our model. Considering the condition that two chain has different probability to grow, we had better change the problem which concentrates on the attack-chain. If the attacker successfully create a new block, the length of his attack chain will increase 1, and if he fails, the length will decrease 1.

In this case, we summarize these properties:

1. The attacker will start from position 0, and he will withdraw the deal at position s.
2. For each step, the attacker may step forward with the probability as 0.51, and he may step backward with the probability as 0.49. Among all the steps, they are independent.
3. Motivate from the value of the transaction, the maximum step he will tray is 100.

To calculate the expect value, we definite the following two cases:

- If the attacker reach the position s with i steps, the final income is $100 - i$.
- If the attacker can't reach the target position before 100steps, he will lose i.

What we want to find out is the maximum k, which makes the attacker will not reach further than s=k+2 in 100 steps,that is the opportunity is small enough that we can regard it as an impossible condition.

3 Calculation

If we use f(i,x) represents the possibility of location is x when the i^{th} step, the origin condition and the transfer equations are as follows:

$$f(0,0) = 1 \tag{1}$$

$$f(i,x) = 0.51 * f(i-1, x-1) + 0.49 * f(i-1, x+1) \tag{2}$$

And we can also give the property of possibility at each step.

$$\sum_{x=-\infty}^{\infty} f(i,x) = 1, \forall i \tag{3}$$

In this case, we can calculate the probability condition as the following programming part4.

4 Programming

4.1 Probability for each position of each step

The first part is used to calculating the possibility for each position of each step.

```
for i in range(steps):
    prob.appendleft(0)
    prob.append(0)
    length = len(prob)
    newProb = deque()
    for pos in range(length):
        leftProb = prob[pos - 1] if pos - 1 >= 0 else 0
        rightProb = prob[pos + 1] if pos + 1 < length else 0
        newProb.append(0.51 * leftProb + 0.49 * rightProb)
    prob = newProb
    resultProb.append(list(prob))
```

The results of the first fewer lines:

```
[0.49, 0.0, 0.51]
[0.24, 0.0, 0.5, 0.0, 0.26]
[0.118, 0.0, 0.367, 0.0, 0.382, 0.0, 0.133]
[0.058, 0.0, 0.24, 0.0, 0.375, 0.0, 0.26, 0.0, 0.068]
[0.028, 0.0, 0.147, 0.0, 0.306, 0.0, 0.318, 0.0, 0.166, 0.0, 0.035]
```

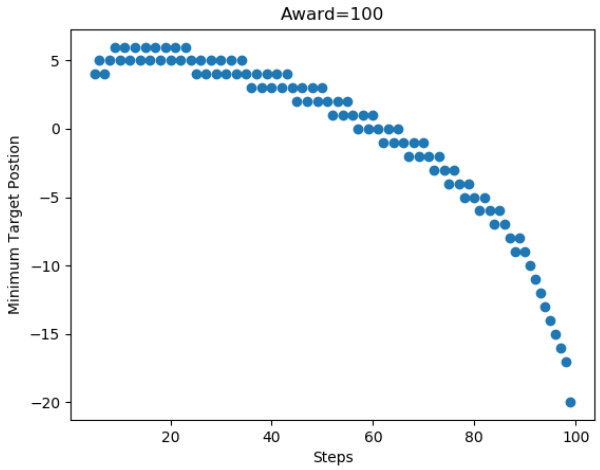
4.2 ExpectMoney depending on target position

The second part is used to calculate the expectMoney for each target position for each step. To describe more detailed, in the first loop, I extract the probability of each position in this step, and in the second loop, I set the target position where the attacker can withdraw the deal and win 100(actually 100 * 10 thousand). And then I have noticed that if the target is farther, it will has less possibility to reach the target in the condition of same steps. So I choose to output the minimum target position when the expectMoney is less than 0, which is also the condition the question want.

```
for step in range(1, steps):
    targetProbe = resultProb[step]
    for target in range(1, 2 * step + 1):
        probWin = sum(targetProbe[target:])
        expectMoney = 100 * probWin - step
        if expectMoney < 0:
            print(step, target - step, expectMoney)
            break
```

4.3 Results

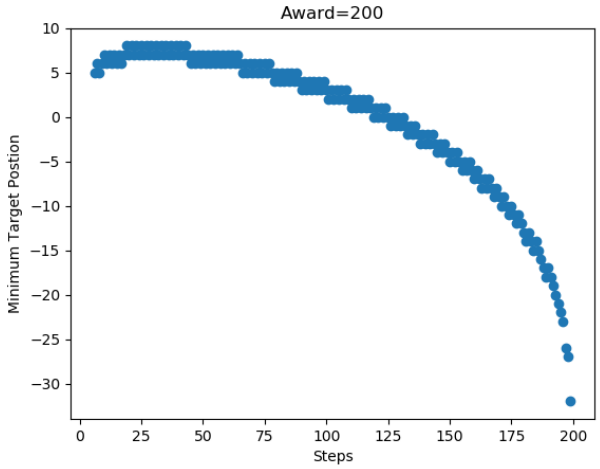
The following picture shows that the minimum target position where the expectMoney is less than 0 for each step.



From this picture, we can see the largest target position is 6. This means that we start from the position 0, and for position 6 the expectMony starts to be less than 0. **Return to the question: $k + 2 > s$, my result for minimum of k is 4.**

5 Disscussion

How about we increase the value of the transaction? The higher value it is, more motivation the attacker will have. In this case, I improve the value of transaction to 200.



In this case, the final value of k needs to be 6.

6 Conclusion

In this paper, I discuss from the definition of double-spending problem. And give the model from the descriptions and program with python to get a relatively accurate value.

References

[1] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[R]. Manubot, 2019.