EE447 Mobile Internet Homework2

Prof. Luoyi Fu, EE447 Mobile Internet

Spring semester, 2020

Fan Zhou(周凡)
517030910305

1 Modeling the Problem

At first thought I think it is very similar to random walker problem. When the attacker starts attacking, it is already $k+1$ blocks behind the main chain. So, like a random walker, it has to travel $k+1$ steps to the right(say it is the correct direction). Once it arrive the final point, it will stop. Thus, the success probability of the attack is turned into the chance of the walker take a step towards the destination.

2 Solution

I then thought of one simple solution: for every step the walker takes, it has $p = 0.51$ to go to the right direction and has $p = 1 - 0.51 = 0.49$ to go to the wrong direction. Thus, the expectation of one step size a random walker takes will be:

$$E(step) = 1 * 0.51 - 1 * 0.49 = 0.02$$

this means, every time the walker takes a step, it will in average takes 0.02 step size to the right direction. To make it to the destination, it has to walk:

$$(k+1)/0.02$$

Now, think about attakcer again. It can't spend over one million on attacking. So, we can have the following inequality:

$$\begin{aligned} & 100 \leq (k+1)/0.02 \\ i.e. \quad & k \geq 1 \end{aligned}$$

Notice the attakcer still need to attack once more to be longer than the main chain. So, we can choose $k = 2$ here.

3 About First Passage

When thinking about the solution to the homework, I also think of first passage problem in Markov chain. Personally, I think it is another way to solve this problem.

We know that as long as the walker arrives the destination, it will stop. Thus, if we can estimate or calculate the probability of $f_{0,k+1}^{(n)}$, which means walker begins at 0 and takes n steps to arrive at $k+1$ for the first time.

To calculate this, We also need to compute $p_{i,j}^{n}$, which means walker begins at i and takes n steps to arrive at j(no matter the first time or second time...).

We can see p is easy to compute, it can be computed through binomial distribution. f is derived from p,

Actually:

$$\begin{aligned} f_{0,k+1}^{(n)} = p_{0,k+1}^{(n)} &- f_{0,k+1}^{(n-1)} p_{0,k+1}^{(1)} \\ &- f_{0,k+1}^{(n-2)} p_{0,k+1}^{(2)} \\ &- f_{0,k+1}^{(n-3)} p_{0,k+1}^{(3)} \\ &\dots \\ &- f_{0,k+1}^{(1)} p_{0,k+1}^{(n-1)} \end{aligned}$$

We will only be able to calculate finite situations, say we compute total n steps.
The attack cost $= f_{0,k+1}^{(n)} * n *$ one time attack cost.
Up to n steps, the probability of success attack is $p_{succ} = \sum_i f_{0,k+1}^{(i)} * 0.51$.
The average benefit $= 100 * p_{succ}$.
Once the attack cost is greater than benefit,
i.e. $f_{0,k+1}^{(n)} * n *$ onetimeattackcost $\geq 100 * p_{succ}$, we think double spending can be preserved.
I implement the calculation in Python.

```python
def C(x, k):
import math
return int( math.factorial(x) / ( math.factorial(x-k) * math.factorial(k) ))

# Markov N-step First Passage probability calculation
# reference http://web2.uwindsor.ca/math/hlynka/wuqianying.pdf
p_ = 0.51
cost_ = 1
deal = 100
for k in range(1, 100):
    tot_p = 0
    # base situations for computing f_{0, k+1}^{n}
    f, p = dict(), dict()
    cost = 0
    for steps in range(2, 1000, 2):
        pr = C(steps, steps//2) * p_**(steps//2) * (1-p_)**(steps//2)
        p[(k+1, k+1, steps)] = pr
        p[(k+1, k+1, steps-1)] = 0
    for steps in range(k+1, 1000, 2):
        l = (steps + k + 1) // 2
        pr = C(steps, l) * p_**(l) * (1-p_)**(steps-l)
        p[(0, k+1, steps)] = pr
        p[(0, k+1, steps-1)] = 0
    for n in range(k+1, 1000, 2):
        # use n steps to achieve same length as the main chain
        # calculate f_{0, k+1}^{n}
        # f(n) = p(0, k+1, n) - f(0, k+1, k+1)p(k+1, k+1, n-(k+1))
        #                     - f(0, k+1, k+2)p(k+1, k+1, n-(k+2))
        #                     - f(0, k+1, k+3)p(k+1, k+1, n-(k+3))
        #                     ... - f(0, k+1, n-1)p(k+1, k+1, n-(n-1))
        pr = p[(0, k+1, n)]
        for j in range(k+1, n, 2):
            pr -= f[(0, k+1, j)] * p[(k+1, k+1, n-j)]
        f[(0, k+1, n)] = pr
        # additional multiply p to be longer than main chain
        cost += pr * n * cost_
        tot_p += pr
        print(k, n, pr, cost)
        # break
    # check the rightness of calculation
```

```
41      print(tot_p, cost, tot_p*100*p_, '\n')
42      if cost > 100 * tot_p * p_:
43          break
44  print(k)
```

The final result is $k = 2$, *i.e.* when we set $k = 2$, the attack cost is more than the expected benefit.

4 External links

Here I list some of the reference materials I looked into.
First Passage computing

- `http://web2.uwindsor.ca/math/hlynka/wuqianying.pdf`
- `https://www.math.ucdavis.edu/~tracy/courses/math135A/UsefullCourseMaterial/firstPassage.pdf`