Game Theory and Its Application Homework1

Student ID: 311551136　Name: 蔡沅恆

WS model:

1. n represents node number in the graph
2. P represents the probability of rewire
3. K the number edge that one node connect to adjacent node
4. arr represents 2-d array of graph and it's empty(all 0)

In the first, the graph has n nodes and each node has k edges connecting to adjacent node.in my ws function, I do not mark "1"(there's edge) and "0"(there's no edge) for initial graph. I know where will have edge(it will like a step around the diagonal of graph) ,so I just need to decide with probability-p in the place it should be marked as "1" in initial if it would be rewired. if no I marked "1" ,else, when my position is at arr[i][j], using the random method to decide new j, and marking arr[i][new j] as "1"(i keeps the original value).Please note, In this ws model function, the total of edge may decrease because the rewired edge may connect to same edge.

```
void ws(int n,double p,int k,vector<vector<int>> &arr){
    //builde ring lattice
    for(int i{0};i<n;i++){
        for(int l{0};l<(k+1);l++){
            int j=((int)(n+l+i-ceil(k/2)))%n;
            if(i<j){
                if((double)rand()/(RAND_MAX+1.0)<p){
                    int new_{i};
                    while(new_==i or new_==j){  //new_==j equal to origin
                        new_=rand()%n;
                    }
                    arr[i][new_]=1;
                    arr[new_][i]=1;
                }
                else{
                    arr[i][j]=1;
                    arr[j][i]=1;
                }
            }
            else if(i==j){
                arr[i][j]=0;
            }
        }
    }
}
```

Assignment 1 : maximal independent set

In this assignment, we need to use game theory to solve maximal independent

set problem. And the assignment requires the average consequences of 100 times of the probability of 0,0.2,0.4,0.6 and 0.8 for ws model to rewire edge. In below code, ws-function is called to generated new graph based on ws model, and mis-function is called to decide the NE(maximal independent set) of the graph.

```
int main(){
    int n{30};
    int k{4};
    srand(time(NULL));

    for(double p{0};p<1.0;p=p+0.2){
        int number{0};
        int move{0};
        for(int b{0};b<100;b++){
            vector<vector<int>> arr(n,vector<int>(n,0));
            ws(n,p,k,arr);
            ans aa=mis(n,arr);
            number=number+aa.number;
            move=move+aa.move_count;
        }
        cout<<"move_count:"<<(double)move/100<<"  cadinality:"<<(double)number/100<<endl;
    }
```

First, I model utility function as:

$$u_i(C) = \sum_{p_j \in N_i} \omega(c_i, c_j) + c_i$$

N_i: p_i's open neighbors

where

$$\omega(c_i, c_j) = -\alpha c_i c_j \qquad \alpha > 1\text{: constant}$$

The utility of i is the sum of w-value of each node j which is adjacent to i, and w-value Is calculated by the -a*i's strategy*j's strategy.

```
int utility(vector<vector<int>> & arr,vector<bool> &strategy,int i,int choose,int n){
    int u{choose};
    const int a{2};
    for(int j{0};j<n;j++){
        if(j!=i) u=u-arr[i][j]*strategy[j]*choose*a;
    }
    return u;
}
```

To be convenient, I do not care the node if it has edge with i, but I multiple arr[i][j] directly, if there is no edge between i and j ,then the arr[i][j] equals to zero, the contribution of the node which is not adjacent to i is zero.

Second, after knowing the utility of each node in specific strategy space, we can decide the best respond of node iteratively and finally get NE of game.

To find the NE, I use mis function as below description.

In mis-function, First, Randomize the initial state of strategy space :

```
//randomize initial game state
for(int j{0};j<n;j++){
    strategy[j]=rand()%2;
}
```

Second, use while loop until all node reach their best respond

In while loop, for each randomly selected node, using utility function explained before to get the utilities of “choose” and “not choose” and compared them to decide I’s best respond:

```
if(utility(arr,strategy,i,0,n)>utility(arr,strategy,i,1,n)){
```

If choose(same as not choose),we need to discuss two possible situation. one, the node have different strategy with original strategy, so we need to record there’s a move of game happens. Moreover, because the situation of game would be different(ie. we need to check n times if all no node reach its best respond again) ,we ne to set “ne” counter to zero to ensure re-check n times.

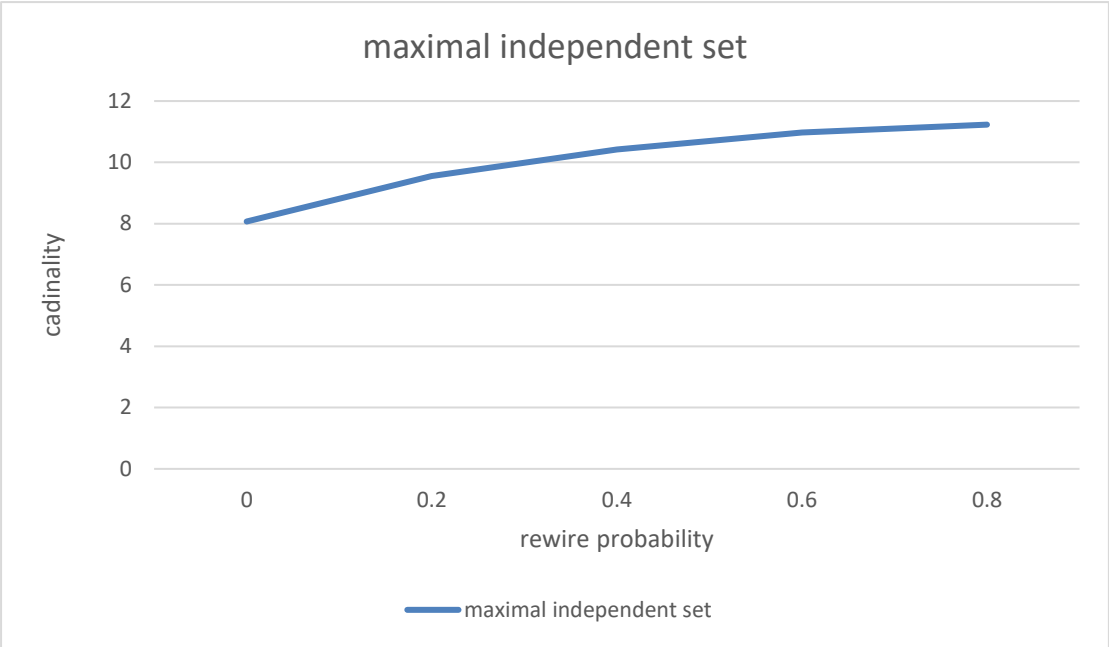
```
if(recent_strategy!=false){
    for(int j{0};j<n;j++){   //initialize if_ne
        if_ne[j]=false;
    }
    ne=1;   //(true->false)entire condition change re
    if_ne[i]=true;
    move_count++;
    strategy[i]=false;
}
```

In the other situation, because there’s no change in the game, we just need to do “ne++” to record another node which reaches its best respond.After n times checks, it will be out of while loop to NE(maximal independent set) .

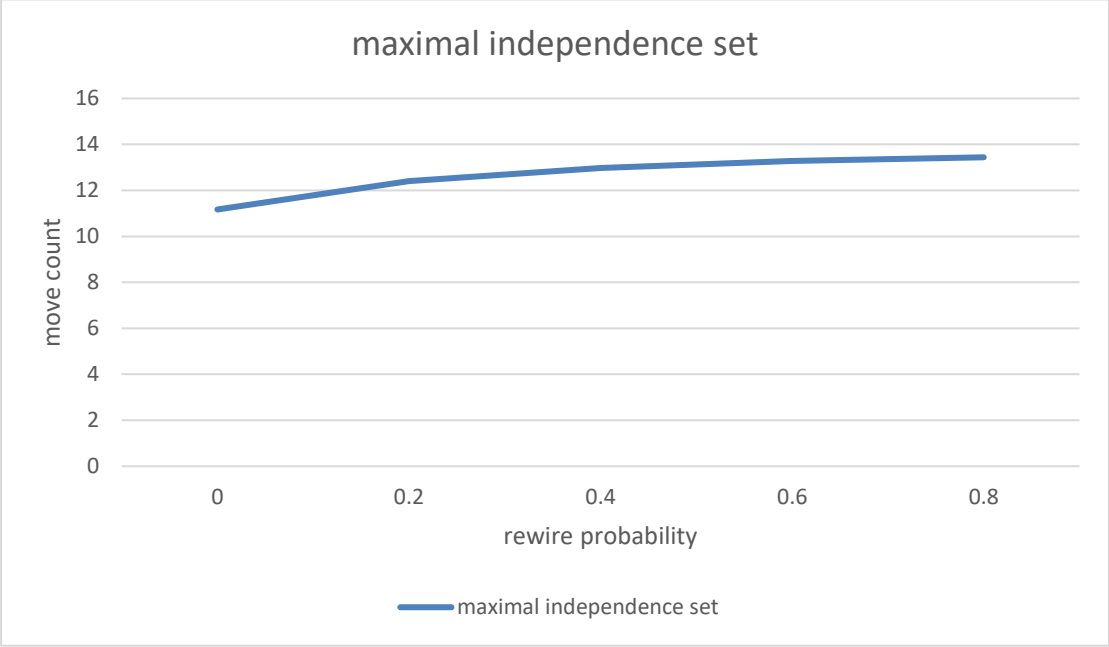
Third, we need to vertify the consequence of our algorithm. When the node in maximal independent set has adjacent node which strategy is also “1” then output ” False”.(not independent23w)

```
//vertify that the game state is a valid solution
for(int i{0};i<n;i++){
    for(int j{0};j<n;j++){
        if(arr[i][j] and strategy[i] and strategy[j]){
            cout<<"false"<<endl;
        }
    }
}
```

Consequence:



Looking at the line chart above, we will find that as the probability value increases, the cardinality of IDS will also increase. It indicates that as long as rewiring, the cardinality of IDS may be influenced. The reason of result is that when probability increase, it may decrease in total number of edge. And then the set will be bigger because less constrains(edge) for independence.



Next, this line chart shows that as the probability value increases, the average number of moves per node will also increase. It indicates that as long as rewiring, it

would take more steps to find out the Nash equilibria as our strategy profile. Similarly, although the result in this diagram doesn't grow dramatically, it still shows the specific relationship between both of them.

Assignment 2: symmetric MDS-based IDS

In this assignment, we need to use game theory to solve maximal independent dominant set problem. The problem is descripted as follower, to find the maximal dominating set of graph which keeps with independent property. The assignment requires the average consequences of 100 times of the probability of 0,0.2,0.4,0.6 and 0.8 for ws model to rewire edge.The algorithm of IDS problem is similar as maximal independent set problem mentioned before. the only different between them is their utility function, so in this report I don't explain code and theory again exclude the part of "utility function".In this problem, the concept of the utility of game is as below:

If strategy of is "not choose", then its utility is zero.

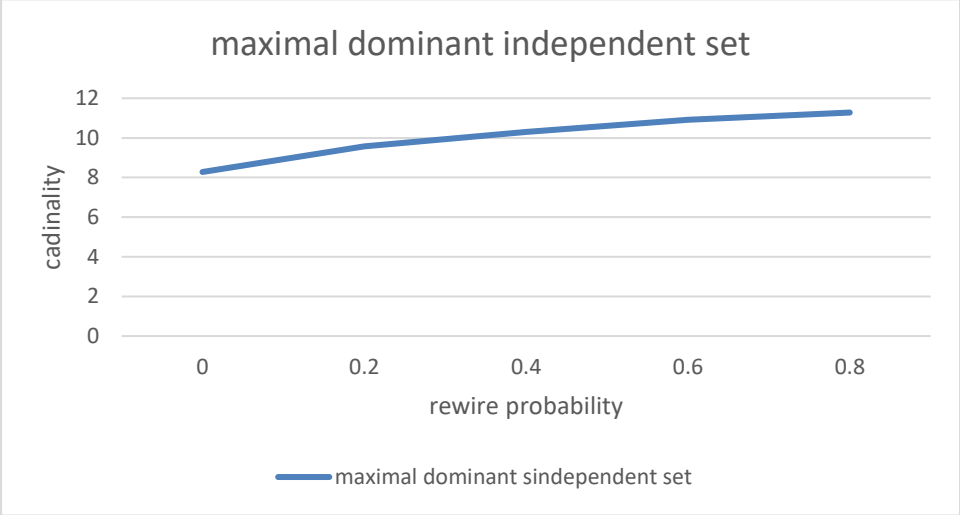
Else, i is "choose". We need to avoid there exist dependence, so when there are node which are adjacent to i, we let i's utility be very bad(less than zero),then we can ensure its strategy wouldn't be "choose". Otherwise, if there's no node which is selected, then we must select I to satisfy "maximal" and "dominant" concurrently. So its utility is greater than zero.

```
if(choose==0) return 0; //if "choose" return 0
for(int j{0};j<n;j++){
    if(strategy[j]==1 and i!=j and arr[i][j]) return -100;
}
return 100;
```

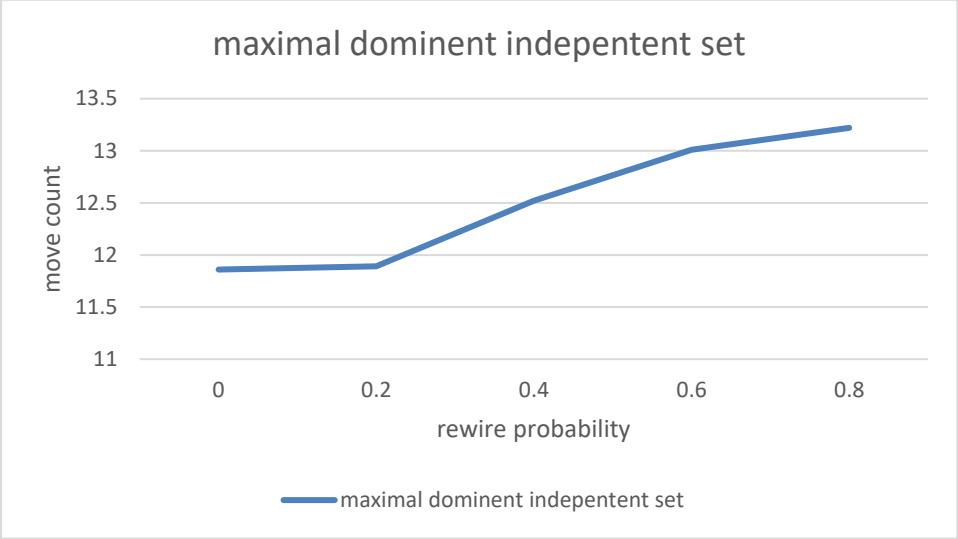
Second, we need to vertify the correctness of the game. The consequence should ensure "dominant" and "independence". The method of checking "independent" is similar to previous assignment. we use "check" array to record which node has been dominated, and finally check this array if there is "0" in it(represent be not dominated)

```
//vertify that the game state is a valid solution
vector<bool> check(30);
for(int i{0};i<n;i++){
    for(int j{0};j<n;j++){
        if(arr[i][j] and strategy[i] and strategy[j]){
            cout<<"not independent"<<endl;
        }
        if(arr[i][j] and strategy[i]) check[j]=true;
        if(strategy[i]) check[i]=true;
    }
}
for(int i{0};i<n;i++){
    if(check[i]==0) cout<<"not dominanted"<<endl;
}
```

Consequence:



Looking at the line chart above, we will find that as the probability value increases, the cardinality of DIS will also increase. It indicates that as long as rewiring, the cardinality of DIS may be influenced. The reason of result is that when probability increase, it may decrease in total number of edge. And then the set will be bigger because less constrains(edge) for independence.



Next, this line chart shows that as the probability value increases, the average number of moves per node will also increase. It indicates that as long as rewiring, it would take more steps to find out the Nash equilibria as our strategy profile. Similarly, although the result in this diagram doesn't grow dramatically, it still shows the specific relationship between both of them.

Assignment 3: maximal matching

In this assignment, we need to use game theory to solve maximal matching problem. And the assignment requires the average consequences of 100 times of the probability of 0,0.2,0.4,0.6 and 0.8 for ws model to rewire edge. In below code, ws-function is called to generated new graph based on ws model, and mis-function is called to decide the NE(maximal independent set) of the graph.

The overall concept of this algorithm is that if when the node turn to decide its best respond, it can discard its original matching pair if it has, and choose others node which haven't have matching pair. To reach goal of " maximal" matching, we first define the "weight" as the inverse of node's degree. The lower degree node has higher weight, otherwise has lower weight. And the node prefers the highest weight node as its matching pair

In the below code, ws-function is called to generated new graph based on ws model, degree-function is called to decide the weight of the node to find the "maximal matching", and matching-function is called to decide the NE(maximal matching) of the graph.

```
int main(){
    int n{30};
    int k{4};
    srand( time(NULL) );

    for(double p{0};p<1.0;p=p+0.2){
        int number{0};
        int move{0};
        for(int b{0};b<100;b++){
            vector<vector<int>> arr(n,vector<int>(n,0));
            vector<int> degree(n,0);
            ws(n,p,k,arr);

            degreeing(arr,degree,n);
            ans aa{matching(arr,degree,n)};

            number=number+aa.number;
            move=move+aa.move_count;
        }
        cout<<"move_count:"<<(double)move/100<<"  cadinality:"<<(double)number/100<<endl;
    }
```

In the matching-function, first, we need generate the random initial state.

```
for(int i{1};i<n;i++){
    for(int j{i+1};j<n;j++){
        if(arr[i][j] and rand()%2 and match[i]==-1 and match[j]==-1){
            match[i]=j;
            match[j]=i;
        }
    }
}
```

Second, in the while loop ,we find the NE

In the begin of every iteration, we need to selected node which is not checked if it reach its best respond randomly

```
while(if_ne[i]){  //select node which is not ne randomly
    i=rand()%n;
}
```

And for each node, we need to find its matching pair node(best respond) which is not matched and have higher weigh than original pair node.

```
//fin best respond
int best_respond{match[i]};  //if -1 represent that node have not match
int best_weight{(match[i]>-1)?degree[match[i]]:-1};
for(int j{0};j<n;j++){
    if(arr[i][j] and match[j]==-1 and best_weight<degree[j]){ //have edge and do not have match yet and have higher weight
        best_respond=j;
        best_weight=degree[j];
    }
}
```

After find out best respond, there are two situations. One is that its best respond is equal to original strategy. We just need to do “ne++” :

```
//already find best respond and check id change
if(best_respond==match[i]){ //do not change
    if_ne[i]=true;
    ne++;
}
```

In the other, Its best respond is different with original strategy. The situation of game is changed, so we need to set ne to zero. Furthermore, if node i has matching pair originally. We also set its strategy to -1(represent it doesn’t have

matching)

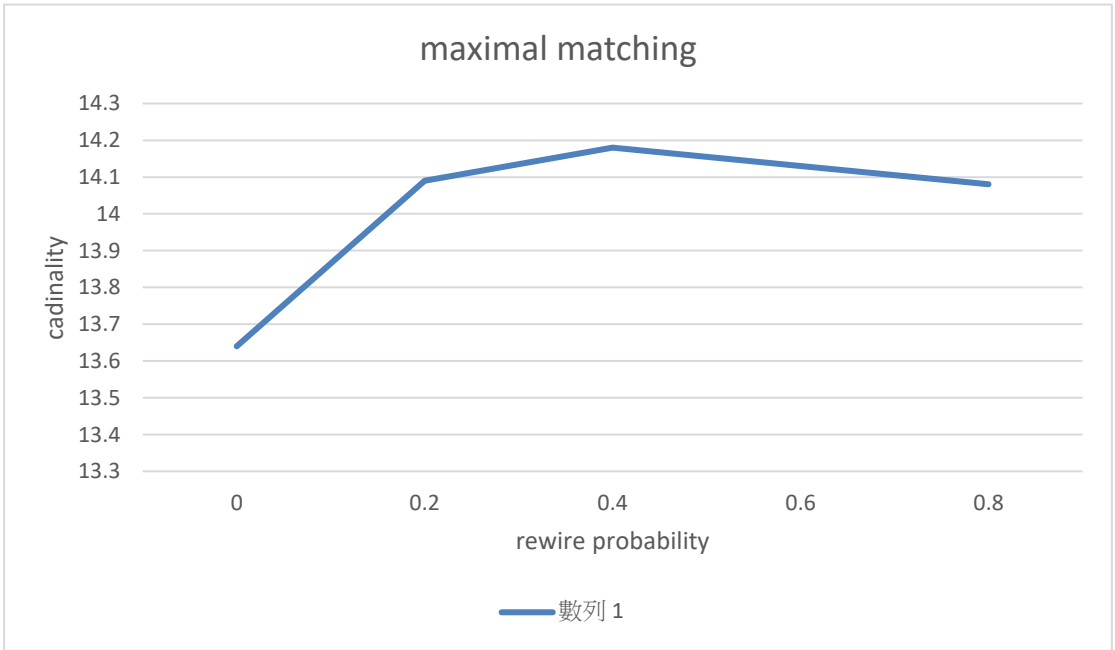
```
else{  //change: delete old match edge ,reset ne and add new match
    move_count++;
    ne=1;
    for(int j{0};j<n;j++){
        if_ne[j]=false;
    }
    if_ne[i]=true;

    int old=match[i];
    if(old!=-1){
        match[old]=-1;
    }
    match[i]=best_respond;
    match[best_respond]=i;
}
```

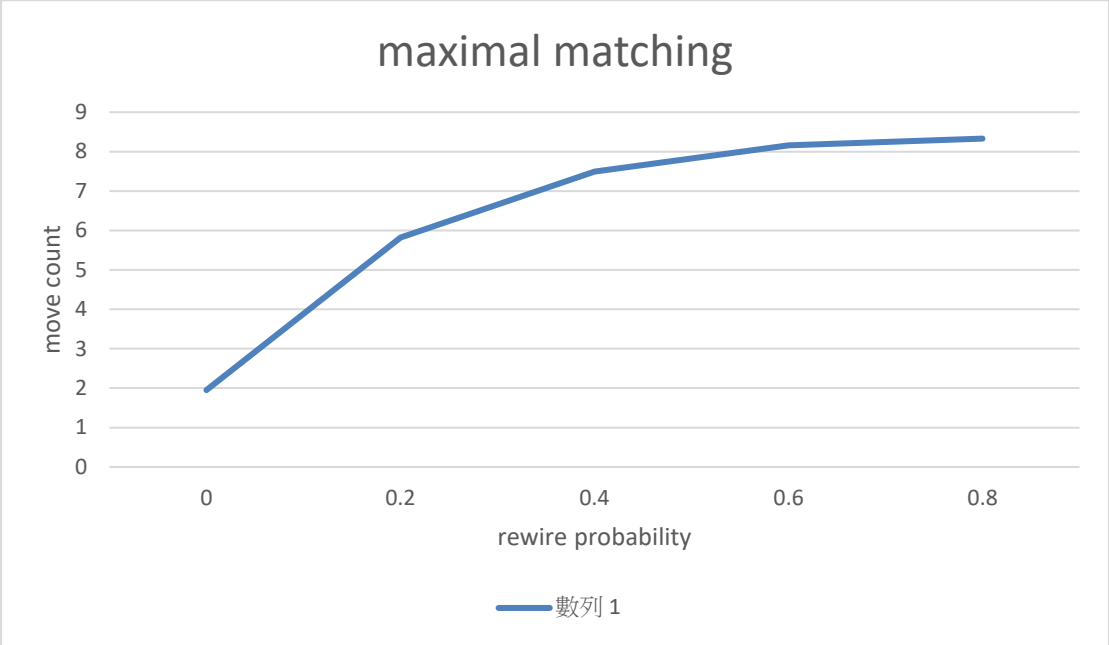
After find NE of the game, we need to vertify “maximal” and “matching” of the consequence. for every node its strategy is bigger than -1 (possible matching strategy is 0~(n-1) and -1 represent no matching) then check the strategy of its matching pair if node I, if not, the consequence is not a matching. Furthermore, check if there are node which don’t have matching and they have common edge to vertify the maximal property.

```
//vertify that the game state is a valid solution
int match_double{0};
for(int i{0};i<n;i++){
    if(match[i]>-1){
        match_double++;
        if(match[match[i]]!=i) cout<<"not pair matching"<<endl;
    }
    else{
        for(int j{0};j<n;j++){
            if(arr[i][j] and match[j]==-1) cout<<"not maximal matching"<<endl;
        }
    }
}
```

Consequence:



Looking at the line chart above, we will find that as the probability value increases, the cardinality of matching will also increase. However, after 0.4, the cardinality of matching will decrease. The reason of the trend is that before 0.4, the probability of rewire still small enough to maintain the original graph which is easier to reach bigger matching. And after probability is larger than 0.4, the rewired graph is more likely to have ones node with very large degree and ones with very small degree which make the number of matching decrease.



Next, this line chart shows that as the probability value increases, the average number of moves per node will also increase. It indicates that as long as rewiring, it would take more steps to find out the Nash equilibria as our strategy profile. Similarly, although the result in this diagram doesn't grow dramatically, it still shows the specific relationship between both of them.