IMP Report

Name: 傅伟堡

SID：11812202

Part 1: Preliminaries

1.1 Problem Description

1.1.1 Introduction

A social network is essentially a graph G(V,E,P), where V is a node-set, E is an edge set, and P is the probability set of all the edges. A user is a node V, and the relationship between users is edge E. Each edge has a probability P, and information will be propagated on the graph according to the probability of edge.

1.1.2 Goal

Influence Maximization Problem is the problem of choosing a fixed sized subsets of nodes in a social network which aims to maximize the spread of influence.

1.1.3 Algorithms

There are two kinds of stochastic diffusion models, Linear Threshold(LT) and Independent Cascade(IC) seperately. Besides, the algorithm I used in IMP is *Influence Maximization via Martingale* and this algorithm can solve the problem in near-linear time.

1.1.4 Software & Hardware

This project is written in *Python 3.7* with editor *PyCharm 2020.2.3*.

1.2 Problem Application

There are many applications for IMP, including viral marketing, recommendation systems, time detection, etc. For example, if a company want to promote their products by viral marketing, they will first choose a small number of people needed to make a free trial promotion commodity, elected the user node satisfied with the goods when they want to through the network to colleague friend recommended the goods to make more people buy the goods finally. How to identify these people try out the product such that the ultimate number to buy the product is the most is the IMP problem.

Part 2: Methodology

2.1 Notation

- **RR**: reverse reachable set
- **R**: the set of RR sets generated by IMM's sampling phase
- **node_num**: the node number in the graph
- **seed_size**: the seed size we will choose
- **FR(S)**: the fraction of RR sets in R that are covered by a node set S
- **epsoid, l**: the hyperparameters to control the size of RR set

- **theta, theta_i**: the number of nodes we should sample
- **alpha, beta, lambda_, eposoid_, theta, lambda_star**: some variables to calculate theta and theta_i.

2.2 Data Structure

- **Graph**: the social network
- **Graph_rev**: the social network by reversing edges
- **activaty_set**: nodes have been activated
- **new_activaty_set**: nodes have been activated in current round

2.3 Model Design

2.3.1 Problem Formulation

It is clearly a IMP solution, and there are a lot of related work of IMP. I choose an effective algorithm to solve the problem: IMM. IMM have two main steps: sampling and node selection. The expected time of IMM is $O((k+l)(n+m)log\frac{n}{\epsilon^2})$ and it will return a $(1-\frac{1}{e}-\epsilon)$ approximate solution.

2.3.2 Problem Solution

The main idea is to use reverse reachable set to estimate influence. We sample some nodes in the network and then calculate the fraction of **RR** nodes which overlaps with the initial set to estimate influence. By the way, LT and IC model have different ways to get reverse reachable set. We only need to sample reachable sets and then choose fixed sets, we can get the seed sets which have maximum influence.

2.4 Detail of Algorithms

This project contains two parts: ISE and IMP.

2.4.1 ISE

There are two kinds of stochastic diffusion models, Linear Threshold(LT) and Independent Cascade(IC) seperately.

```
Function IC(Graph,SeedSet)
    Initialize ActivitySet = {SeedSet}
    count = ActivitySet.length
    while(!AcitivitySet.IsAmpty())
        Initialize newActivitySet = {}
        for each seed in ActivitySet
            for each inactive neighbor in seed
                seed tries to activate neighbors with probability
                if(activated)
                    Update the state of the neighbor as Active
                    newActivitySet.add(neighbor)
                end if
            end for
        end for
        count = count + newActivitySet.length
        ActivitySet = newActivitySet
    end while
    return count
```

```
Function LT(Graph,SeedSet)
```

```
    Initialize ActivitySet = {SeedSet}
    count = ActivitySet.length
    while(!AcitivitySet.IsAmpty())
        Initialize newActivitySet = {}
        for each seed in ActivitySet
            for each inactive neighbor in seed
                Calculate the weights of activated neighbors: w_total
                if(w_total >= neighbor.thresh)
                    Update the state of the neighbor as Active
                    newActivitySet.add(neighbor)
                end if
            end for
        end for
        count = count + newActivitySet.length
        ActivitySet = newActivitySet
    end while
    return count
```

During the statistical process, we call functions repeatedly. Since we are not sure that how many times we should call functions in each case, we should use time to judge whether the program exits. Specifically, I allocate $8s$ for program to exit and flush.

```
Function Statistics(time_limit)
    round = 0.0
    sum = 0
    while(current_time - begin_time < time_lime-8)
        round += 1
        sum += IC(Graph,SeedSet) or LT(Graph,SeedSet)
    end while
    return sum/round
```

2.4.2 IMP

IMM have two main steps: Sampling and Node Selection.

```
Function IMM(Graph_rev,seed_size,epsoid,l)
    l = l * (1 + log(2)/log(n))
    R = Sampling(Graph, seed_size, epsoid, l)
    Sk_star = NodeSelection(R, seed_size)
    return Sk_star
```

The Sampling part in the paper is as below:

```
Function Sampling(Graph_rev, seed_size, epsoid, l)
    Initialize a empty set R and an integer LB = 1
    Let epsoid_ = Sqrt(2) * epsoid
    for i = 1 to round_down(log(n)/log(2))
        x = node_num / pow(2,i)
        lambda_ = ((2 + 2 * epsoid_ / 3.0) * (log_binomial(n,k) + l *
math.log(n) + math.log(math.log2(n))) * n) / pow(epsoid_,2)
        theta_i = lambda_ / x
        while (R.length <= theta_i)
            Select a node v from Graph_rev uniformly at random
            Generate an RR set for v, and insert it into R
        end while
        Let Si, FRSi= NodeSelection(R)
        if node_num * FRSi >= (1 + epsoid_) * x
            LB = node_num * FRSi / (1 + epsoid_)
            break
        end if
    end for
    alpha = Sqrt(l * log(node_num) + log(2))
    beta = Sqrt((1-1/e)*(log_binomial(node_num,seed_size) + l*log(node_num) +
log(2)))
    lambda_star = 2 * n * pow(((1-1/e)*alpha + beta),2) * pow(epsoid,-2)
    theta = lambda_star / LB
    while (R.length <= theta)
        Select a node v from Graph_rev uniformly at random
        Generate an RR set for v, and insert it into R
    end while
    return R
```

The NodeSelection part in the paper is as below

```
Function NodeSelection(Graph_rev,seed_size)
    Initialize an empty node set Sk_star
    for i = 1 to seed_size
        Identify the vertex v that maximizes FR(Sk_star union v)-FR(Sk_star)
        Insert v into Sk_star
    end for
    return Sk_star
```

Specifically, the method of getting the reverse reachable set of node is as below.

```
Function generate_rr_IC(Graph_rev, node)
    Initialize an empty set activateSet
    activateQueue = {node}
    while(!activateQueue.IsEmpty())
        Initialize an empty set newActivateSet
        for each seed in activateQueue
            for each inactive neighbor in seed
                seed tries to activate neighbors with probability
                if(activated)
                    activateSet.add(neighbor)
                    newActivitySet.add(neighbor)
                end if
            end for
        end for
        activityQueue = newActivitySet
    end while
    return activateSet
```

```
Function generate_rr_LT(Graph_rev, node)
    activateSet = {node}
    activateQueue = node
    while(activateQueue != None)
        Initialize an empty set newActivateSet
        select a neibor v of activateQueue at random
        if(v not in activateSet)
            activateSet.add(v)
            newActivitySet.add(v)
        end if
    end while
    return activateSet
```

In order to prevent time out and out of memory, I revise the Sampling part as below. I use half time to do Sampling part to prevent time out and limit the size of R to prevent out of memory.

```
Function Sampling(Graph_rev, node_num, time_limit)
    Initial an empty set R
    while(current_time - begin_time < time_limit/2 && len(R)<4000000)
        choose a node v at random
        RR = generate_rr_LT/IC (Graph_rev, v)
        R.append(RR)
    end while
```

Part 3: Empirical Verification

3.1 Dataset

Actually, two simple datasets are given to us and I use them to test my program. Besides, I test with random-graph with different size.

3.2 Performance Measure

- Time using: We have explained before, my program will takes the nearly full time.
- Usability test: Basically, the measurement is related to the given platform. If our program is not usable, we cannot pass the test
- Accuracy: I compare the result of my program with the result on the ranking to judge the program performance.

3.3 Hyperparameter

The initial implement have two hyperparameters e, l to control accuracy and time spend. IMM runs on $O((k+l)(n+m)log\frac{n}{\epsilon^2})$ expected time and returns a $(1-\frac{1}{e}-\epsilon)$ approximate solution. Then I set $e=0.1$ and $l=1$

In the final implement, the only two hyperparameters are the time used in Sampling and the maximum size of R. I set $timeusedinSampling = time\_limit/2$ and the maximum size is 4000000.

3.4 Experimental Result

In ISE: I passed all testcases and get the full points.

DataSet	Runtime	Result
random-graph50-50-seeds-5-IC-new	53.95	30.73
random-graph50-50-seeds-5-LT-new	53.81	41.31
random-graph500-500-seeds-10-IC-new	54.37	167.72
random-graph500-500-seeds-10-LT-new	54.02	209.47
random-graph1000-1000-seeds-10-IC-new	53.66	176.50
random-graph1000-1000-seeds-10-LT-new	55.01	190.81
random-graph5000-5000-seeds-50-IC-new	54.90	945.27
random-graph5000-5000-seeds-50-LT-new	53.99	1038.34
random-graph15000-15000-seeds-50-IC-new	53.62	1586.64
random-graph15000-15000-seeds-50-LT-new	55.23	1712.38
random-graph50000-50000-seeds-100-IC-new	114.22	4098.73
random-graph50000-50000-seeds-100-LT-new	114.02	4315.46

In IMP: all testcases have good results.

DataSet	Runtime	Result
network-5-IC	38.149	30.7222
network-5-LT	37.171	37.5412
NetHEPT-5-IC	37.551	323.2786
NetHEPT-5-LT	36.389	392.975
NetHEPT-50-IC	74.620	1297.8358
NetHEPT-50-LT	72.814	1701.9953
NetHEPT-500-IC	63.331	4331.928
NetHEPT-500-LT	71.415	5586.0506

3.5 Conclusion

- Advantage: IMM have near-linear time spend and reliable accuracy and we do not need to worry about the time out.
- Disadvantage: If the network is large, it will be slower than heuristic algorithm.
- Experience: In this project, I learned the process of Influence Maximization Problem. IMM can be used in IMP which have many nodes and have strict request for accuracy.

Part 4: References

[1] Tang, Y., Shi, Y., and Xiao, X. (2015, May). Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (pp. 1539-1554). ACM.

[2] Kempe, D., Kleinberg, J., and Tardos, É. (2003, August). Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 137-146). ACM.