Report of Influence Maximization Problem Using a Martingale Approach

Name ID: Yutong Wang 11611808
Department: Computer Science and Technology
Institution: Southern University of Science and Technology
Email: 11611808@mail.sustc.edu.cn

I. Preliminaries

A. Problem Description

The problems of this project include two tasks for Influence Maximization Problems (IMPs) in social networks. First, given a specific seed set S and a social network G, implement an estimation algorithm to calculate the influence spread of S. Second, given a positive integer k and a social network G, find a small subset of nodes (referred to as seed set) of size k in the social network G that could maximize the spread of influence.

B. Problem Applications

IMP has widespread applications including viral marketing, rumor control and so on. This problem originates from viral marketingin social networks (e.g., Facebook), which has become a common ground for businesses to target potential customers. Viral marketing aims to select a small number of influential users to adopt a product, and subsequently trigger a large cascade of further adoptions by utilizing the Word-of-Mouth effect in social networks [1][2] .

II. Methodology

A. Notation

Notation	Meaning
G	Network, a directed graph.
S	Seed set.
n	Number of nodes.
m	Number of edges.
RR set	The set generated during smapling.
R	The set of RR sets.
IC	Independent cascade model.
LT	Linear threshold model.
k	Size of seed set in IMP.
$FR(S)$	Fraction of RR sets in R covered by S.

Table 1. Notations table.

B. Data Structure

1. **ISE**:

- *2d-array* : I use a 2d-array with size $n * n$ to store the directed edges with their weights.
- *dictionary(map)* : I use a dictionary(map) to store a node's children. The key of the dictionary is the nodes which have children. And the value of the key is a list containing the node's children.
- *set* : I use two sets during sampling. One set is to store the activity set of each new iteration. The other one is to store all the nodes which have been activited.

2. **IMP**:

- *2d-array* : I use a 2d-array with size $n * n$ to store the directed edges with their weights.
- *dictionary(map)* : I use a dictionary(map) to store a node's father nodes. The key of the dictionary is the nodes which have father nodes. And the value of the key is a list containing the node's father nodes.
- *set* : I use a set (aka. RR set) to store nodes during. And another set is used to store the activity set of each new iteration.

C. Model Design

1. **ISE**:

- *How to fomulate the problem?*
Because every time of node spreading has randomness, so it will not achieve an accurate result if there is only one time of samlping. So the ISE problem is to sample enough times and obtain an average result in the end. The main problem is how to sample under different spreading models(aka. Independent Cascade and Linear Threshold).

- *How to solve the problem?*
Step 1: Check out the type of the model, if the model type is IC, call the function *one_IC_Sample* to get the number of spreading nodes for one time. If the model type is LT, call the function *one_LT_Sample* to get the number of spreading nodes for one time.
Step 2: Repeat Step 1 for 10000 times, add result every time together and finally calculate the average number of spreading nodes.

2. **IMP**:

- *How to fomulate the problem?*
The problem is to find k seeds which can influence the maximum number of nodes. Traditional greedy algorithm works in small network but is not efficient and cost long time to get result in large graph, e.g. NetHEPT network and so on. So the problem is how to implement an efficient algorithm to find k seeds which can influence the maximum number of nodes whihch is also pretty practical in huge networks.

- *How to solve the problem?*

Step 1: Sampling enough times to generate RR sets and store all the RR sets in a R set. During each time of iteration, I select a node randomly in the graph and generate a RR set for this selected node to see which nodes can make influence on this node. This step uses the algorithm in function *Sampling*, *generateRR_IC*, *generateRR_LT*.

Step 2: After generating all the RR sets and put them in one R set, find the top k nodes which can influence the maximun number of nodes. This step uses the algorithm in function textitNodeSelection.

D. Details of Algorithm

1. **ISE**:

Algorithm 1 *one_IC_Sample*

This algorithm is used when the model type is IC and is to get the influence spread of S in one sampling. To improve the performance of my algorithm in speed, I use the data structure set and dictionary(map) for most time.

The **pseudo-code** is as followed:

```
function one_IC_Sample:
    Initialize ActivitySet <== SeedSet
    Initialize Activited <== SeedSet
    Initialize count <== length of ActivitySet

    while(!ActivitySet.IsEmpty())
        Initialize newActivitySet <== NULL
        for each seed in ActivitySet
            for each neighbor of seed
                if neighbor not in Activited
                    //seed try to activate neighbor:
                    rand_num <== a random number in range(0,1)
                    if edge's weight > rand_num:
                        //sucsessfully activate neighbor
                        add neighbor to the set Activited
                        add neighbor to the set newActivitySet
        count = count + length of newActivitySet
        ActivitySet = newActivitySet

    return count
```

Algorithm 2 *one_LT_Sample*

This algorithm is used when the model type is LT and is to get the influence spread of S in one sampling. To improve the performance of my algorithm in speed, I use the data structure set and dictionary(map) for most time. What's more, I also use two dictionaries(maps) to store nodes and corresponding threshold and weights sumed up by its father nodes.

The **pseudo-code** is as followed:

```
function one_LT_Sample:
    Initialize ActivitySet <== SeedSet
    Initialize Activited <== SeedSet
    Initialize count <== length of ActivitySet
    Initialize node_thresh_dict <== {}
    Initialize w_total_dict <== {}

    while(!ActivitySet.IsEmpty())
        Initialize newActivitySet <== NULL
        for each seed in ActivitySet
            for each neighbor of seed
                if neighbor not in Activited
                    //seed try to activate neighbor:
                    if neighbor not in node_thresh_dict
                        add neighbor to node_thresh_dict
                        give neighbor a random threshold value
                        add neighbor to w_total_dict
                        initialize its value as 0
                    w_total_dict[neighbor] += edge weight
                    if w_total_dict[neighbor] >= node_thresh_dict[neighbor]
                        //sucsessfully activate neighbor
                        add neighbor to the set Activited
                        add neighbor to the set newActivitySet
        count = count + length of newActivitySet
        ActivitySet = newActivitySet

    return count
```

2. **IMP**:

Algorithm 3 *IMM*

IMM algorithm first invoke Algorithm 4 *Sampling* to sample enough RR sets and then invoke Algorithm 5 *NodeSelection* to select k node as seed set which can achieve maximum influence.

The **pseudo-code** is as followed:

```
function IMM(G, k, e, l):
    l = l * (1 + log2/logn)
    R = Sampling(G, k, e, l)
    S_k = NodeSelection(R, k)
    return S_k
```

Algorithm 4 *Sampling*

In sampling phase, IMM algorithm uses certain method to calculate how many times RR set should be sampled and generated. After the first phase of generating RR sets (aka. sampling), the times of sampling in the second phase is computed and the second phase of sampling begins. Finally, a R set which contains many RR sets generated from both two sampling phases is returned.

The **pseudo-code** is as followed:

```
function Sampling(G, k, e, l):
    Initialize a set R = NULL and an integer LB = 1
    Let e_ = squareRoot(2) * e
    for i = 1 to (log2(n))-1 do
        Let x = n/(2^i)
        Let sita_i = lmd_/x
        while |R| <= sita_i do
            Select a node v from G uniformly at random
            Generate an RR set for v, and insert it into R
        Let S_i = NodeSelection(R)
        if n * FR(S_i) >= (1 + e_)*x then
            LB = n * FR(S_i)/(1+e_)
            break
    Let sita = lmd*/LB
    while |R| <= sita do
        Select a node v from G uniformly at random
        Generate an RR set for v, and insert it into R
    return R
```

Algorithm 5 *NodeSelection*

R is the set of random RR sets generated by the sampling phase of IMM, and $FR(S)$ is the fraction of RR sets in R that are covered by a node set S. Algorithm 5 shows the pseudocode of the node selection phase of IMM, which takes as input R and a positive integer k, and returns a size-k node set S_k. As can be seen, the algorithm implements a greedy approach.

The **pseudo-code** is as followed:

```
function NodeSelection(R, k):
    Initialize a node set S_k = NULL
    Initialize a node set V containing all the nodes in R set
    for i = 1 to k do
        Identify the vertex v in V that maximize (FR(S_k + v) - FR(S_k))
        Insert v into S_k
    return S_k
```

III. Empirical Verification

A. Dataset

Extra datasets are used during the experiment:

1. *NetHEPT.txt*: n = 15233, m = 32235
2. *epinions-d-5.txt*: n = 39008, m = 84002
3. *twitter-d.txt*: n = 81306, m = 1768149

B. Performance measure

Test environment: MacOS Sierra 10.12.6; Processor 2.9 GHz Intel Core i5; Memory 8 GB 2133 MHz LPDDR3.

How to meassure:

i. For ISE, I use the total time and the value of the estimation to evaluate.

ii. For IMP, I use total time and the estimation influence of the output seed set to measure the performance. I pass the result seed set of IMP as the parameter of ISE and get the spread influence of that result.

C. Hyperparameters

i. For ISE, the hyperparameter refers to the sampling time. If the sampling time is too little, the bias will get larger. If it is too large, the time will cost a lot. From experience, I find the influence spread will be stable around a value when the sampling time is 10000 and the cost time is also acceptable. So I set the hyperparameter – sampling time to be 10000.

ii. For IMP, the hyperparamers refer to e which is the fault tolerance of the result. The smaller the e is, the more accurate the result will be. Under the trade off between time consuming and accuracy, I set the hyperparameter – e to be 0.1.

D. Experimental results

1. IC Model: Let seed set size k = 5. For IMP, e = 0.1; for ISE, sampling time = 10000.

ISE:

Network	Time(s)	Influence
NetHEPT	0.82	43.19
epinions-d-t	1.91	74.43
twitter-d	55.3	1901.77

Table 2. ISE results under 3 networks in IC model.

IMP:

Network	Time(s)	Influence
NetHEPT	8.98	43.19
epinions-d-t	20.77	74.43
twitter-d	307.46	1901.77

Table 3. IMP results under 3 networks in IC model.

2. LT Model

ISE:

Network	Time(s)	Influence
NetHEPT	0.72	40.93
epinions-d-t	2.71	76.19
twitter-d	59.76	1892.95

Table 4. ISE results under 3 networks in LT model.

IMP:

Network	Time(s)	Influence
NetHEPT	13.34	40.93
epinions-d-t	31.13	76.19
twitter-d	428.42	1892.95

Table 5. IMP results under 3 networks in LT model.

E. Conclusion

Different implements of ISE cause different results. Under some improvement in data structure and algorithm as well as making use of multiprocessing, ISE can also improve a lot in both speed and accuracy. As for IMP, the re-implement of IMM algorithm makes my code faster than before in a large amount of times. As can be seen from the experiments, the smallest network among the three has more than 15000 nodes but running this graph only takes around 10 seconds under both IC model and LT model. For the largest network which has more than 1760000 edges, it run aound several minutes, which is acceptable comparing traditional algorithms.

References

[1] F. Bass, "A new product growth model for consumer durables", Manage. Sci., vol. 15, no. 5, pp. 215-227, 1969.

[2] V. Mahajan, E. Muller, F. M. Bass, "New product diffusion models in marketing: A review and directions for research", J. Market., vol. 54, no. 1, pp. 1-26, 1990.