IMP Project

Yue Gong 11611908
Department of Computer Science and Engineering
Southern University of Science and Technology
Email: ~~11611908@mail.sustc.edu.cn~~
11611908@mail.sustech.edu.cn

1. Preliminaries

1.1. Problem Description

Influence Maximization Problem (IMP) is the problem of finding a small subset of nodes (referred to as seed set) in a social network that could maximize the spread of influence. The influence spread is the expected number of nodes that are influenced by the nodes in the seed set in a cascade manner.

Network

A social network is modeled as a directed graph $G = V$, E with nodes in V modeling the individual in the network and each edge u, $v \in E$ is associated with a weight w(u, v)$\in$[0,1] which indicates the probability that u influences v.

Seed Set

Let S $\subset$ V to be the subset of nodes selected to initiate the influence diffusion, which is called the seed set.

Stochastic Diffusion Model

The stochastic diffusion models specify the random process of influence cascade from S, of which the output is a random set of nodes influenced by s. The expected number of influenced nodes $\sigma(S)$ is the influence spread of S.

1.2. Problem Applications

A social network the graph of relationships and interactions within a group of individuals − plays a fundamental role as a medium for the spread of information, ideas, and influence among its members. An idea or innovation will appear − for example, the use of cell phones among college students, the adoption of a new drug within the medical profession, or the rise of a political movement in an unstable society − and it can either die out quickly or make significant inroads into the population. If we want to understand the extent to which such ideas are adopted, it can be important to understand how the dynamics of adoption are likely to unfold within the underlying social network.

2. Methodology

2.1. Notation

- **node_num**: number of nodes in the graph
- **edge_num**: number of edges in the graph
- **k**: the number of seeds
- **graph**: a graph stores the network
- **pgraph**: a graph stores the inverse network
- **weight**: the weight of an edge
- **seed_set**: the set of seeds
- **seed_size**: the size of seed set
- **model**: type of model, can be IC or LT
- **worker**: a array maintains all processes
- **R**: the set of RR sets generated by IMM sampling phase
- θ:the number of RR sets in R

2.2. Data Structure

In this problem, my data structure includes list, set, dictionary, graph and heap. To implement a graph, I use adjacency list instead of adjacency matrix which can save space significantly when dealing with large graphs. Besides, when I always use list instead of set when I make sure that no element will be repeated. Because if we use set, it will check whether an element has already in it which will decrease efficiency.

2.3. Model Design

To solve the influence maximization problem, I implement the algorithm proposed by paper called "IMM". If we use the traditional greedy algorithm, we need to do ISE every time we select new node, which is quite time_consuming because ISE needs to sample 10000 times. IMM has a totally different idea. First of all, we have the concept of RR(reverse reachable)set. For example, we can take node v as the seed, its rr set is all the nodes which can be influenced by node v in the reverse graph. Finally we have sampled lots of nodes and got lots of corresponding rr sets, which can be put into a large set R. Intuitively, we know that the node that has most appearances in the R would be the node which is most likely to influence other nodes. Following this thought, we can select top k nodes in the R.

2.4. Detail of algorithm

First, I'll introduce the IC(independent cascade) model in the ISE.

Algorithm 1 ise_IC

```
count ←the length of the seed set
activity_set ← seed_set
active_node ← seed_set
while activity_set is not empty do
  new__activity_set ← ∅
  for each seed in activity_set do
    for each neighbor of seed do
      if neighbor is not active and probability is lower than the weight then
        new_activity_set ← new_activity_set ∪ neighbor
        active_node ← active_node ∪ neighbor
      end if
    end for
  end for
  count ← count + thelengthoftheseedset
  activity_set ← new_activity_set
end while
return count
```

Second, I will introduce the LT(linear threshold) model in the ISE

Algorithm 2 ise_LT

```
count ← the length of the seed set
activity_set ← seed_set
active_node ← seed_set
node_threshold ← empty_dictionary
node_weights ← empty_dictionary
while activity_set is not empty do
  for each seed in active_node do
    for each node of seed do
      if node is not active then
        if node is not in node_threshold then
          intialize the node threshold with random value
          initialize node in the node_weight with 0
        end if
        add weight to the node's weight
        if the total weight exceeds the node threshold then
          new_set ← new_set ∪ neighbor
          active_node ← active_node ∪ neighbor
        end if
      end if
    end for
  end for
  count ← count+ the length of the seed set
  activity_set ← new_activity_set
end while
return count
```

Then, I will introduce the framework of IMM algorithm. It has two phases, one is the sampling phase, the other is the node_selection phase.

Algorithm 3 IMM

```
Input: k, eps, l
l ← l * (1 + 1/(log2*node_num))
R ← sampling(k, eps, l)
S*_k ← node_selection(R, k)
return S*_k
```

Next, I will introduce how to calculate RR sets under the IC model and LT model respectively.

Algorithm 4 generate_rr_ic

```
Input: node
Output: activity_nodes
activity_set = []
add node to activity_set
activity_nodes = []
add node to activity_nodes
while activity_set is not empty do
  new_set = []
  for seed in activity_set do
    for node in seed's neighbors do
      if node is not active and above the possibility threshold then
        add node to activity_nodes
        add node to new_set
      end if
    end for
  end for
  activity_set = new_set
end while
return activity_nodes
```

Algorithm 5 generate_rr_lt

```
Input: node
Output: activity_nodes
activity_nodes = []
add node to activity_nodes
activity_node = node
while activity_node is not -1 do
  new_activity_node = -1
  if activity_node has no neighbor then
    break
  end if
  neighbors = activity_node's neighbors
  candidate = randomly select one node from neighbors
  if candidate not in activity_nodes then
    add candidate to activity_nodes
    new_activity_set = candidate
    activity_set = new_activity_set
  end if
end while
return rr_set
```

Algorithm 6 sampling

Input: k, eps, l
Output: R
$R \leftarrow \emptyset$
$LB \leftarrow 1$
$eps_{prime} \leftarrow \sqrt{2} \cdot eps$
$n \leftarrow node\_num$
for $i = 1$ to $log_2 n$ **do**
$x \leftarrow n/2^i$
$\theta_i \leftarrow \frac{(2+\frac{2}{3}eps_{new})(\log C_n^k + l\log n + \log\log_2 n)n}{x \cdot eps_{new}^2}$
while $|R| \leq \theta_i$ **do**
Select a node v from G uniformly at random
Generate an rr_set for v, and insert it into R
end while
Let $S_i \leftarrow node\_selection(R)$
if $n \cdot F_R(S_i) \geq (1 + eps_{new}) \cdot x$ **then**
$LB \leftarrow \frac{n \cdot F_R(S_i)}{1+eps_{new}}$
$break$
end if
end for
$\alpha \leftarrow \sqrt{l\log n + \log 2}$
$\beta \leftarrow \sqrt{(1-\frac{1}{e}) \cdot (\log C_n^k + l\log n + \log 2)}$
$lamda\_star \leftarrow 2n \cdot ((1-\frac{1}{e}) \cdot \alpha + \beta)^2 \cdot eps_{new}^{-2}$
Let $\theta \leftarrow \frac{lamda\_star}{LB}$
while $|R| \leq \theta$ **do**
Select a node v from graph uniformly at random
Generation an rr_set for v, and insert it into R
end while
return R

Node selection is used to select top k nodes from the R.

Algorithm 7 node_selection

Input: R, k
Output: S_k
$S_k = \emptyset$
rr_degree = []
node_rr_set = empty dictionary
initialize rr_degree with the occurences of each node in the R
initailize node_rr_degree with the node and the rr_set it belongs to.
while the size of S_k is not k **do**
find the node with the most occurence times and delete all rrsets contain it.
end while

3. Empirical Verification

3.1. Dataset

ISE: $network\_seeds, network\_seeds2, NetHEPT\_5seeds$
$NetHEPT\_50seeds$
IMP: $network\_5, NetHEPT\_5, NetHEPT\_50,$
$epinions\_d\_5, epinions\_d\_50$

3.2. Performance measure

I set $eps = 0.1, l = 1$, and use eight processes.
Test environment is given by my server: intel(R) Xeon(R) W-2125, 4.00GHz, 8 cpus.

3.3. Hyperparameters

$eps = 0.1$
$l = 1$
The paper suggests us to set $eps = 0.5$ and $l = 1$. However, I find that decreasing the value of eps will generate better solution. So I finally set it to be 0.1.

3.4. Experimental results

The code of ISE is about the same, the time and the result are nearly the same, so I will not show the result of ISE, I would like only to show the result of IMP.

Dataset	Vertex	Model	K	ans	time
network	62	IC	5	30.64	0.11
network	62	LT	5	37.56	0.074
NetHEPT	15233	IC	5	323.28	2.40
NetHEPT	15233	LT	5	393.81	2.28
NetHEPT	15233	IC	50	1296.92	3.59
NetHEPT	15233	LT	50	1698.25	3.01
epinions_d	39008	IC	5	1311.56	4.84
epinions_d	39008	LT	5	1422.16	3.09
epinions_d	39008	IC	50	4588.38	12.33
epinions_d	39008	LT	50	5466.88	11.08

I have compared my results with other students' outputs on the OJ. I find that my result can always rank into the top 10 and my time efficiency can also rank into the top 10.

3.5. Conclusion

In conclusion, the algrithm proposed by the paper IMM is quite efficient. When greedy algorithm cannot give a result on the graph NetHEPT, IMM can give the results on the NetHEPT in less than 10 seconds.

However, the algorithm also has some drawbacks. Because it needs to generate rrsets, it has larger memory demand. Besides, if every node in the graph has lots of edges, the efficiency of this algorithm will decay quickly.

References

[1] Y. Tang, Y. Shi, and X. Xiao, "Influence Maximization in Near-Linear Time," in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15, 2015.