# IMP

# Project 3 Report of

# CS303 Artificial intelligence

# Department of Computer Science and

# Engineering BY


# Hongxin Peng

# 11710716

# 1. Preliminaries

## 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.

**Social network and influence spread**

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 (u,v) ∈ [0,1] which indicates the probability that u influences v.

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

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.

**Goal of IMP**

To find a seed set S that maximizes $\sigma(S)$, subject to |S| = k .

## Problem Applications

IMP algorism can measure the influence of some spreading things, such as frog, flow, pestilence, advertisement and so on. Then if you want to put advertisement to influence whole country, but it is cost too much that put in every city. We can use IMP to select one seed set as city to make influence most.

# 2. Methodology

## Notations

**LT()** to get the score in LT model.

**IC()** to get the score in IC model.

**Select_seed()** to select better node as initial seed set.

**Initial_celf()** to traverse initial seed set to get score and put into PriotyQueue, then pop the best one as the first seed.

**Get_Seed()** to select the final seed from initial seed set.

## Data Structure

**Network = []** to store the graph.

**initial_seed = []** to store the better seed by selected in first time.

**degree_rank = heapq** to store the rank of each node with degree.

**label = []** to store the which router we pop the node .

**Seeds = []** to store the final seed set.

**score_set = heapq** to store the rank of initial seed with score in IC or LT model.

**node()** to store node's state and neighbors.

**outnei=[] in node** to store out neighbor of node.

**innei=[] in node** to store in neighbor of node.

## Model design

In order to select the best seed set, I use the celf++ method, but it cost too many time if traverse every node in graph. So I optimal it by selecting initial seed set.

First, rank the node by there out degree

Second, select the top (final seed size * 8) seed as initial set.

Third, rank the initial seed with score of IC/LT model.

Forth, select most score node and into final seed size then get the score. If it is still the top one in the score_rank, then we choose it. Else remove it from final seed set.

Fifth, do Forth step until the seed size equal the final seed set size.

## Detail of Algorism

Build_network

```
Bulid_network():
    read evey line in the network.txt as line
    split line
    if first line:
        total node number = line[0]
        total edge number = line[1]
    else:
        node[line[0]] set out neighbor node[line[1]] and out weight
        node[line[1]] set in neighbor node[line[0]] and in weight
```

Select_seed

```
Select_seed():
    degree_rank = headq //优先队列 极小值优先
    add every node into degree_rank with out degree
    select top (seed size * 8 ) node as initial seed set
```

Initial_celf

```
Initial_celf():
    traverse every one in initial seed set:
        add into score_rank with IC/LT model score
        label[node]=0 //标记每个点计算的回合
```

Get_final_seed

```
Get_final_seed():
    while final_seed.size < require_size
        last_score=score of final_seed
        node=score_rank.pop()
        final_seed.append(node)
        label[node] = final_seed.size
        get now_score of final_seed
        put now_socre-last_score into score_rank
        if the label[score_rank.top] =final_seed:
            //放进去之后还是得分最优
            then we choose this node
        else:
            final_seed.remove(node)
            //将他移除
    print final_seed as result
```

main

```
main():
    Bulid_network()
    Select_seed()
    Initial_celf()
    Get_final_seed()
```

IC

```
        When a node u gets activated, initially or by another node, it has a
single
chance to activate each inactive neighbor v with the probability proportional to
the edge weight w(u,v).
        Afterwards, the activated nodes remain its active state but they have no
contribution in later activations.
        The weight of the edge (u,v) is calculated as w(u,v)=1/d(v), where
d(n)denotes the in-degree of node v.
```

LT

```
        At the beginning, each node v selects a random threshold uniformly at
random in range [0,1].
        If round t ≥ 1, an inactive node v becomes activated if total in weight
of v >=its threshold
        The weight of the edge (u,v) is calculated as w(u,v)=1/d(v), where
d(v)denotes the in-degree of node v
```

---

# 3. Empirical Verification

## Dataset

Use the  network.txt and NetHEPT.txt.

## Performance Measure

n: node number in network  s: require seed size

**Time complexity :**

Firstly, traverse every node to select initial seed set put into heap cost O(nlog(n)) choose 8*s initial seed set.

Second, for every node wo calculate 500 times IC/LT scores and put into heap cost O(500 * 8  * s * log(8s) * model cost)

Final, get all the seed until the size equal to s. To the worse time:

we need to traverse every node in heap and calculate 500 times.

Then the worse time complexity is O(8s*s *500 *model cost)

**Accuracy :**

| type | result | best |
|---|---|---|
| network-5-IC | 30.6756 | 30.7222 |
| network-5-LT | 36.1613 | 37.5412 |
| NetHEPT-5-IC | 314.0349 | 323.6857 |
| NetHEPT-5-LT | 349.9411 | 392.9750 |
| NetHEPT-50-IC | 1119.3476 | 1297.8358 |
| NetHEPT-50-LT | 1 501.0742 | 1701.9953 |

we can find the final result will not be optimal  with this celf++  because the time of operator not enough.

## Hyperparameters

The initial seed size is depend on my decision. The times of s will decide the size which we can choose seed from.  Bigger is better. But it influence the time complexity so much. So I need to balance the time and the initial seed size.

The time of calculate model score also influence whether we choose the node, because it influence the accuracy of model score. Also, this parameter will decide the time so much.

So I balance these two parameter to get better score and not exceed time.

Finally I use s*8  and every time select node calculate 500 time model score and average.

## Experimental results

I can pass all the test.

| type | result | best |
|---|---|---|
| network-5-IC | 30.6756 | 30.7222 |
| network-5-LT | 36.1613 | 37.5412 |
| NetHEPT-5-IC | 314.0349 | 323.6857 |
| NetHEPT-5-LT | 349.9411 | 392.9750 |
| NetHEPT-50-IC | 1119.3476 | 1297.8358 |
| NetHEPT-50-LT | 1 501.0742 | 1701.9953 |

## Conclusion

**Disadvantage:**

- the accuracy of this algorism in limited time wont be 100%
- this algorism cost so much time
- I need to using different multipliers when different situation

**Advantage:**

- this algorism is greedy algorism, easy to understand
- optimal the celf++ reduce the time cost
- use headq, no need to traverse whole score_list reducing time cost.

# 4. Reference


[1] J. Leskovec, A. Krause. Cost-effective Outbreak Detection in Networks. PA, USA, June 2007.

[2]D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In KDD, 2003.

[3] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. Inf. Proc. Let., 1999.