IMP Report

Yilin Zhang 11510280

School of Computer Science and Engineering
Southern University of Science and Technology
11510280@mail.sustc.edu.cn

December 15, 2018

1. Preliminaries

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.

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. Let $S \subseteq V$ 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. And the goal of IMP is to find a seed set S that maximizes $\sigma(S)$, subject to $|S| = k$.

In this project, we need to implement an Influence Spread Estimator (ISE) and an IMP slover named ISE.py and IMP.py, respectively. The first is to implement an estimation algorithm for the influence spread and the second is to design and implement a search algorithm for IMPs. The IMP is NP-hard and the influence spread computation is #P-hard under the definitions shown in the above.

1.1. Software

This project is written in Python. The libraries being used includes *argparse*, *sys*, *Numpy* and *Random*. The local test environment is as follows:

- Python version: 3.6.6
- Operation System: macOS
- Processor: 3.5 GHz Intel Core i7
- Memory: 16 GB

2. Methodology

Given the seed set , the diffusion process unfolds in discrete rounds:

- At round 0, all nodes in S are active and the others are inactive.
- In the subsequent rounds, the newly activated nodes will try to activate their neighbors.
- Once a node becomes active, it will remain active till the end.
- The process stops when no more nodes get activated.

Two fundamental diffusion models Linear Threshold (LT) and Independent Cascade (IC) are specified in this project. I estimate the influence spread by these two models, and search seeds by Cost-effective Lazy-forward (CELF) method.

2.1. Representation

Both ISE and IMP problem need to parse arguments from commend line and read *.txt files including the information of graph or seeds. I used a dictionary to store general informations such as **NETWORK_FILE_PATH**, **DIFFUSION_MODEL**, **TERMINATION** *etc*. As for the graph, I used a 2-dimensional matrix denotes the influence weights among them.

The format of solvers call are as follows (the test environment is Unix-like):

Task 1:

python ISE.py −i <social network> -s <seed set> -m <diffusion model> -t <time budget>

Output: the value of the estimated influence spread.

Task 2:

python IMP.py −i <social network> -k <predefined size of the seed set> -m <diffusion model>

Output: the seed set found by the algorithm.

- **<social network>:** absolute path of the social network file.
- **<seed set>:** absolute path of the seed set file.
- **<predefined size of the seed set>:** a positive integer.
- **<diffusion model>:** IC or LT.
- **<time budget>:** a positive number which indicates how many seconds the algorithm can spend on this instance.

Some constants are as follows:

- **INFO:** a dict stores informations from args.
- **graph:** two-dimensional matrix to present the directed graph.
- **precision:** determine estimate times.

2.2. Architecture

The executable files are *ISE.py* and *IMP.py*, which contain parameter parsing, flie reading, and result printing. There are four modeules *IC.py*, *LT.py*, *CELF.py* and *utils.py* in the *submodule* folder, offer functions IC, LT, CELF and other tools, respectively.

- **IC:**
  - **IC:** This program is to estimate the influence, and output the value of the estimated influence spread. 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. After there is no new activited node, return the count of activited nodes as influence.
- **LT:**
  - **LT:** At the beginning, each node v selects a random threshold θ_v uniformly at random in range [0,1]. If round $t1$, an inactive node v becomes activated if for all activated neighboors u, $\Sigma w(u, v) \geq \theta_v$. After there is no new activited node, return the count of activited nodes as influence.
- **CELF:**
  - **influence:** Calculate the marginal influence and return it.
  - **CELF:** Based on greedy algorithm and exploiting submodularity, significantly reduce # influence spread evaluations. Return the set of seeds.
- **utils:**
  - **count:** Calculate k times seed set influence and return the average value.
  - **model_adapter:** Transfer diffusion model from string to int to simplified judgment.

Algorithm 1 IC

Input: *graph*, *SeedSet*
Output: count of activited nodes (influence)

initial $ActivitySet$ = $SeedSet$
$count = ActivitySet.length$
while !$ActivitySet.IsAmpty()$ **do**
    initial $newActivitySet = \emptyset$
    for all $seed \in ActivitySet$ **do**
        for all inactive neighbor $\in seed$ **do**
            try to activate by $w(seed, neighbor)$
            if activate successed **then**
                update neighbor's state
                $newActivitySet.add(neighbor)$
            end if
        end for
    end for
    $count = count + newActivitySet.length$
    $ActivitySet = newActivitySet$
end while
return *count*

2.3. Algorithm

ISE program simply use IC or LT model to estimate the influence. It prefer an average value

Algorithm 2 LT

Output: count of activited nodes (influence)

initial $ActivitySet$ = $SeedSet$
for all node **do**
  initial $threshold \in [0, 1]$
  if $threshold = 0$ **then**
    add $node$ to $ActivitySet$
  end if
end for
$count = ActivitySet.length$
while $!ActivitySet.IsAmpty()$ **do**
  initial $newActivitySet = \emptyset$
  for all $seed \in ActivitySet$ **do**
    for all inactive neighbor $\in seed$ **do**
      calculate the sum w_total of all active neighbors' weight of $neighbor$
      if $w\_total \geq neighbor.threshold$ **then**
        update neighbor's state
        $newActivitySet.add(neighbor)$
      end if
    end for
  end for
  $count = count + newActivitySet.length$
  $ActivitySet = newActivitySet$
end while
return $count$

Algorithm 3 influence

Input: $graph, current\_seed, model, precision$

Output: current influence of $current\_seed$

initial $infs = 0$
if model is IC **then**
  repeat
    $infs = infs + IC(graph, current\_seed)$
  until $precision$ times
else
  repeat
    $infs = infs + LT(graph, current\_seed)$
  until $precision$ times
end if
$infs = infs/precision$
return $infs$

by repeatedly estimated when time is limited, so we define a number by **formula 1** named *precision*, which is inversely proportional to the graph size. The result need to plus 1 to avoid value 0, and have an upper bound $MAX\_precision$ to guarantee execution efficiency.

IMP program also used the *precision*, but it has different meaning with it in ISE program. IMP program using CELF algorithm to get seeds. CELF function just running once in this process, but for estimating emarginal influences (Δinf) **formula 2**, average assessment is essential to guarantee the reliability of each single seed choice.

Here are details of these algorithms.

$$precision = MIN(\frac{scale\_factor}{node\_number} + 1, MAX_{precision}) \tag{1}$$

$$node_{\Delta inf} = influence_{SeedSet \cup node} - influence_{SeedSet} \tag{2}$$

3. Empirical Verification

3.1. Experiments

A basic test (a graph with 62 nodes and 159 edges) has been given to test the usability of ISE and IMP code, and both of them running well. Also, my code can handle further tests on online-judge platform in limited time and get good solutions except LT model of the bonus dataset named NetHEPT-50 which has more than 15,000 nodes. But considering I just using CELF rather than more advanced algorithm, this result indicates that the code has a good performance.

3.2. Data and data structure

Data being used in this project is a directed graph and the influence number between its nodes. Data structure used in this project includes dictionary, array, tuple, list, and set.

3.3. Performance

CELF's theoretic running time, computed by time complexity of functions is about $O(krn'\sigma)$, where k is size of seeds, σ is the influence spread

Algorithm 4 CELF

Input: $graph, size, model, precision$
Output: a k size seed set
initial $SeedSet = \emptyset$
initial $queue = \emptyset$
$current\_influence = 0$
add all nodes with nonzero out-degree to queue
for all $node_i \in queue$ **do**
    initial $\Delta inf_i = \infty$
end for
while $SeedSet < size$ **do**
    initial $best = 0$
    new_queue = reverse sorted $queue$ by marginal influence
    while exit $\Delta inf_{node} \in new\_queue > \Delta inf_{best}$ **do**
        update Δinf_{node}
        if n **then**ew $\Delta inf_{node} > \Delta inf_{best}$
            $best = node$
        end if
    end while
    add $best$ to $SeedSet$
    update $current\_influence$
end while
return $SeedSet$

of the seed set. n' is the number of simulations at each iteration.

3.4. Analysis

IC and LT are two of classical models to analyze IMP problem, and there are some other models are not considered in this project such as the triggering model (TR) and the Continuous-Time (CT) IC model[1] *etc.* Besides, CELF is a simulation method mentioned in 2007. There are some better approaches now considering IMP problem by improving *theoretical efficiency* of the simulation based methods while preserving the approximation guarantee, these new approaches are able to handle bigger and more complex network graph than CELF.

References

[1] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan, *Influence Maximization on Social Graphs: A Survey*, IEEE Transactions on Knowledge and Data Engineering, pp. 1852-1872, 2018.