Report for IMP

Jiayu Zhang
SID:11812425
Department of Computer Science and Engineering
CS303 Artificial Intelligence
Email: 11812425@mail.sustech.edu.cn

1. Preliminary

1.1. Problem Description

In recent years, the rise of social software such as Weibo, QQ, Tiktok, etc. has led to the rapid development of social networks. In social networks, the speed and quality of information flow determines the development of this social network. When distributing business information, some merchants often choose users with greater influence in order to obtain higher benefits at lower costs. The problem of influence maximization is to study such a situation.

In the problem of influence maximization, we need to select a certain number of nodes as the initially activated seed set in a given network, so that this seed set has the greatest influence in this network.

1.2. Problem Applications

The problem of influence maximization is a difficult problem to solve, and its algorithm can only be continuously optimized over time. When continuously optimized, such an algorithm can be applied to merchants to place business information on social networks. Of course, when encountering more serious problems, the government can also notify the largest number of people of emergency messages at the lowest time cost with the help of the algorithm.

2. Methodology

2.1. Notation

- n: the number of nodes in G
- m: the number of edges in G
- k: the number of the seed set for influence maximization
- θ: the number of rr sets in R
- R: the set of rr sets sampled
- $F_R(S)$: the fraction of rr sets in R covered by node set S

2.2. Data structure

- $pool$: process pool, used for calculate results with multiple process
- $nodes$: a dictionary used to represent the graph, the key is the node id, the value is a two-dimensional list, where the first dimension represents the end node of the edge from this node as the starting point and the second dimension represents the corresponding weight
- $async\_result$: multiprocessing running list, which contains all the running and waiting functions in the process pool
- $results$: the multiprocessing calculation result list contains the calculation results of all completed processes
- $seeds$: initially activated seed set in ISE or the result seed set in IMP
- $activity\_set$: list of nodes activated for each iteration
- $total\_set$: list of nodes activated each time the complete sampling process
- R: A list of rr sets
- S_k: The k best result nodes selected in R

2.3. Model design

In the IMP problem, we need to calculate the influence of the seed. Generally speaking, there are two basic flow models for social networks, one is the independent cascade model(IC model), and the other is the linear threshold model(LT model).

2.3.1. ISE.

- $ICmodel$: In social networks, there is a propagation probability between two connected nodes, and the activated node has the probability to activate the end node of its outgoing edge, and this probability has nothing to do with history. No matter the activation is successful or not, the node that tries to activate the end node of the out-edge will not make another activation attempt.
- $LTmodel$: A node is affected by all neighboring nodes. If the combined value of the influence is

greater than the initial threshold of the node, the node is activated. Such models have historical effects.

In order to obtain more accurate results, it is necessary to sample the seed set many times to average. Therefore, such a problem can be calculated by using multiple processes to calculate as many results as possible within a specified time.

2.3.2. IMP. In the IMP problem, we use the IMM algorithm to sample the graph first, calculate the rr set within the specified size range and add it to R, and then go through a round of selection to finally obtain the desired seed set result. Similarly, we can also use multi-processes in IMP to calculate the results of multiple imm concurrently and put them in the results, and then iterate the results to obtain the k nodes with the most occurrences as the final result set.

2.4. Detail of algorithms

2.4.1. ICsample. The initial seed set is used as the activated seed set. In each round, the last activated seed set is used to activate the end nodes of the edges starting from them. For each out edge, a floating point number of 0 1 is randomly generated. If the floating point number is less than the weight of this edge (the influence probability), then the end node is activated and added to the list of newly activated nodes. The final return value is the total number of activated nodes, or the list of nodes in the activated state (pseudocode shows the former example).

Algorithm 1 ICsample($seeds$)

```
activity_set ← seeds
count ← activity_set.length
while activity_set is not empty do
    new_activity_set ← ϕ
    for seed in activity_set do
        for each inactive neighbor in seed do
            seed tries to activate neighbors with probability
            if activated then
                update state of neighbor as active
                add neighbor to new_activity_set
            end if
        end for
    end for
    count+ = new_activity_set.length
    activity_set ← new_activity_set
end while
return count
```

2.4.2. LTsample. Use the initial seed set as the activated seed set. At the beginning of the algorithm, an initial threshold value of 0 1 is randomly generated for each node. When each iteration reaches the current node, the sum of the weights of the edges whose starting node is activated is counted. If it is greater than the initial threshold value, it will be activated. And it is added to the list of newly activated nodes. The final return value is the total number of activated nodes, or the list of nodes in the activated state (pseudocode shows the former example).

Algorithm 2 ICsample($seeds$)

```
activity_set ← seeds
randomly sample thresholds for all nodes
count ← activity_set.length
while activity_set is not empty do
    new_activity_set ← ϕ
    for seed in activity_set do
        for each inactive neighbor in seed do
            w_total ←the weights of activated neighbors
            if w_total ≤ neighbor.threshold then
                update state of neighbor as active
                add neighbor to new_activity_set
            end if
        end for
    end for
    count+ = new_activity_set.length
    activity_set ← new_activity_set
end while
return count
```

2.4.3. IMM. Sampling the graph to generate a list R containing the rr set, and then use the NodeSelection function to select the nodes in it to obtain the final result set.

Algorithm 3 IMM($\varepsilon, l, k, nodes$)

```
R ←SAMPLING(ε, l, k, nodes)
S_k ←NODESELECTION(R, k)
return S_k
```

2.4.4. Sampling. Sampling function, sample the graph to get the list of rr set R.

$$\lambda' = \frac{(2 + \frac{2}{3}\varepsilon') \cdot (log\binom{n}{k} + l \cdot log\ n + log\ log_2\ n) \cdot n}{\varepsilon'^2} \quad (1)$$

$$\alpha = \sqrt{l \cdot log\ n + log\ 2}$$

$$\beta = \sqrt{(1 - 1/e) \cdot (log\binom{n}{k} + l \cdot log\ n + log\ 2)}$$

$$\lambda^* = 2n \cdot ((1 - 1/e) \cdot \alpha + \beta)^2 \cdot \varepsilon^{-2} \quad (2)$$

2.4.5. NodeSelection. Iteratively calculate the marginal benefits of the selected nodes according to R, and select the k nodes with the highest marginal benefits.

Algorithm 4 Sampling($\varepsilon, l, k, nodes$)

$R \leftarrow \phi$
$LB \leftarrow 1$
$\varepsilon' \leftarrow \sqrt{2} \cdot \varepsilon$
for $i \leftarrow 1$ *to* $log_2 n$ **do**
    $x \leftarrow n/2^i$
    $\theta \leftarrow \lambda'/x$($\lambda'$ defined in Equation (1))
    while |R|$\leq \theta$ **do**
        $v \leftarrow$ randomly choose one node
        generate rr set for v and add it to R
    end while

    $S_i \leftarrow$NODESELECTION(R, k)
    if $n * F_R \leq (1+\varepsilon') \cdot x$ **then**
        $LB = n * F_R/(1+\varepsilon')$
        break
    end if
end for

$\theta \leftarrow \lambda^*/LB$($\lambda^*$ defined in Equation (2))
while |R|$\leq \theta$ **do**
    $v \leftarrow$ randomly choose one node
    generate rr set for v and add it to R
end while
return R

Algorithm 5 NodeSelection(R, k)

$S_k^* \leftarrow \phi$
for $i \leftarrow 1$ *to* k **do**
    find v to maximize $F_R(S_k^* \cup v) - F_R(S_k^*)$
    add v to S_k^*
end for
return S_k^*

3. Empirical Verification

3.1. Dataset

In the ISE stage, only network.txt, NetHEPT.txt, and network_seeds.txt were used for testing. In the IMP stage, in addition to the three files provided, I wrote a file test_data.py that can randomly generate n nodes and m edges and the edge weights meet the requirements of the problem. It is used to calculate the performance of the program under different data sets.

3.2. Performance measure

In the ISE stage, the performance of the program can be judged only by comparing the calculated results and the data on the rankings.

In the IMP stage, there are many factors that affect the speed and quality of seed generation. One of them is the complexity of the graph. The number of nodes and the number of edges in the graph will affect the calculation speed. Another is the size of k, which will affect the speed of selection.

Dataset	Runtime	Result
random-graph50-50-seeds-5-IC-new	2.56	30.602
random-graph50-50-seeds-5-LT-new	2.35	41.297
random-graph500-500-seeds-10-IC-new	2.66	167.810
random-graph500-500-seeds-10-LT-new	3.81	209.444
random-graph1000-1000-seeds-10-IC-new	4.60	176.921
random-graph1000-1000-seeds-10-LT-new	3.46	190.759
random-graph5000-5000-seeds-10-IC-new	15.21	944.01
random-graph5000-5000-seeds-10-LT-new	18.44	1039.438
random-graph15000-15000-seeds-100-IC-new	58.78	1588.297
random-graph15000-15000-seeds-100-LT-new	42.34	1701.817
random-graph50000-50000-seeds-100-IC-new	117.38	4090.954
random-graph50000-50000-seeds-100-LT-new	118.44	4317.123

TABLE 1. ISE RESULT

Dataset	Runtime	Result
network-5-IC	2.37	30.3611
network-5-LT	2.35	37.5412
NetHEPT-5-IC	38.56	323.6857
NetHEPT-5-LT	31.83	392.975
NetHEPT-50-IC	60.80	1296.8626
NetHEPT-50-LT	39.82	1701.75
NetHEPT-500-IC	117.55	4330.6597
NetHEPT-500-LT	117.03	5570.9322

TABLE 2. IMP RESULT

3.3. Hyperparameters

- ε, l: The hyperparameter used to control the length of R in each iteration, in order to ensure that each rr set is sufficiently effective and does not take too much time, so $\varepsilon = 0.5, l = 1$ to obtain better results.

3.4. Experimental results

In ISE, my code passed all test datasets. See results in Table 1.

In IMP, all tests on OJ have good results. See results in Table 2.

3.5. Conclusion

In this project, both ISE and IMP used multiple processes to make full use of resources to calculate more results, which made my results more accurate and effective. However, in IMP, there is no theoretical proof to use multiple processes to calculate the results and select the most k nodes, and sometimes the data is not necessarily better than a single calculation.

After this period of study, I have a preliminary understanding of the influence algorithm and python multiprocess programming, which laid some foundation for future career development. In the future, I may read more literature related to IMP, mainly to study their theoretical basis, to optimize my algorithm instead of relying solely on a large number of calculations to win.

Acknowledgments

I would like to thank Prof. $TangKe$ for his course. Also, I want to thank TA $ZhaoYao$ for her guidance.

References

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

[2] S. Bharathi, D. Kempe, and M. Salek. Competitive influence maximization in social networks. In *WINE*, pages 306–311,2007.

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