

Influence Maximization Problem

Yue Leng 11510213

1. Preliminaries

Influence maximization is the problem of finding a small subset of nodes (seed nodes) 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. The social networks are large-scale, have complex connection structures, and are also very dynamic, which means that the solution to the problem needs to be very efficient and scalable.

In this project, I study efficient solutions for two computational tasks for Influence Maximization Problems (IMPs) in social networks. 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.

In the task of ISE, I improve the data structure to finish the task more quickly in both IC and LT model. Also, in the task of IMP, I combine the heuristics and CELF algorithms, and improve them by pruning more unnecessary branches to speed up the process.

According to the experiment results, **ISE.py** give an accurate estimation of the influence spread in a short time. And **IMP.py** find a high-quality solution to IMPs as fast as possible within a limited time.

2. Methodology

2.1. ISE

2.1.1. Overview

The ISE is to implement an estimation algorithm for the influence spread in both IC and LT model. In this algorithm, firstly, read the data to get the information, and then calculate influence spread 10000 times by divided is into 1000/7 times by using 7 multiple processes.

2.1.2. Outline

```
▼ ISE.py
  ► Graph(dict)
    f IC()
    f ise(q, times, model, random_seed)
    f LT()
    f read_file(datafile, seedfile)
```

2.1.3. Algorithm

(i) Graph

A graph is an ordered pair $G = (V, E)$, where V is a set of nodes (vertices, points) and E is a set of edges (the weight of edge). This is a directed edge and in that case G is called a directed graph. The graph is represented as adjacency list by using nested dictionary data structure, which ensure the high efficient of researching in the graph.

(ii) Read_file

Read the network data file and seed data file, to get the graph and seed set.

(iii) IC

Use set to maintain the state of every node, the time complexity of searching node in set in $O(1)$.

The pseudocode is as follows:

```

初始化ActivitySet为SeedSet
count = ActivitySet.length
while (!ActivitySet.isEmpty())
    newActivitySet 初始化为空
    for each seed in ActivitySet
        for each inactive neighbor in seed
            seed 以自己和邻居间的权重为概率尝试去激活邻居
            if (激活成功)
                更新邻居状态
                newActivitySet.add(neighbor)
            endif
        end for
    end for
    count = count + newActivitySet.length
    ActivitySet = newActivitySet
end while
return count

```

(iv) **LT**

Speed up the algorithm by:

- (a) Use dictionary to maintain the threshold and accumulated weights of every node, the time complexity of searching threshold in set in $O(1)$.
- (b) Initialize the threshold and accumulated weights of a node only when it is needed.

The pseudocode is as follows:

```

初始化ActivitySet为SeedSet
初始化每个结点的阈值（随机产生），如果产生0.0的阈值，加入ActivitySet中
count = ActivitySet.length
while (!ActivitySet.isEmpty())
    newActivitySet 初始化为空
    for each seed in ActivitySet
        for each inactive neighbor in seed
            计算该邻居结点的所有激活状态邻居的权重总和w_total
            if (w_total >= neighbor.thresh)
                更新邻居状态为Active
                newActivitySet.add(neighbor)
            endif
        end for
    end for
    count = count + newActivitySet.length
    ActivitySet = newActivitySet
end while
return count

```













2.2. IMP

2.2.1. Overview

After reading the file, the IMP invokes the “heuristics_CELF _improved” method. In this method, firstly, use “heuristics” method to find 8 times seeds set, and then take it as the input of “CELF_improved” to find the final seedset.

If the time is up before “CELF_improved” find all the seeds, take the result of “Heuristics_improved” as the residuals.

2.2.2. Outline

- ▼  IMP.py
 - ▶  Graph(dict)
 - ▶  timeout
 -  CELF_improved(k, seedset)
 -  Heuristics(k)
 -  heuristics_CELF_improved(k)
 -  Heuristics_improved(k)
 -  IC(seedset)
 -  ise(random_seed, model, q_in, q_out)
 -  ise_finalresult(model, seedset)
 -  LT(seedset)
 -  read_file(datafile)

2.2.3. Algorithm

(i) Timeout

The timeout module is used to control the running time of this program. If the termination type is 1 and the running time is larger than the requirement, this program with exit properly.

(ii) Heuristics_CELF_improved

First, use heuristics algorithms to find $8*k$ (8 times) seeds, and then take these seeds as input of CELF_improved method. By using this strategy, the run time decreases a lot because the less number of initial seeds. In addition, the quality of seed set rarely loss.

(iii) Heuristics_improved

The heuristics function represents the expectation of influence of a node.

$$H(n) = E [\text{weight} * \text{outdegree} (\text{every neighbor of } n)]$$

When a node is chosen as seed, the influence ability of its neighbors will be affected.

The pseudocode is as follows:

Algorithm 1 Heuristics improved (graph, k)

```

1: initialize SeedSet, h
2: for each  $n \in \text{graph}$  do
3:   calculate  $\text{outdegree}[n]$ 
4: end for
5: for each  $n \in \text{graph}$  do
6:    $h[n] = 0$ 
7:   for each  $\text{neighbor} \in N(n)$  do
8:      $h[n] + = \text{weight} * \text{outdegree}[\text{neighbor}]$ 
9:   end for
10: end for
11: while  $|\text{Seedset}| \neq k$  do
12:    $\text{seed} \leftarrow n$  with maximum  $h(n)$ 
13:    $\text{SeedSet.add}(\text{seed})$ 
14:   for each  $\text{neighbor} \in N(\text{seed})$  do
15:      $h(\text{neighbor}) = (1 - \text{weight}) * (h[\text{neighbor}] - |\text{intersection nodes}|)$ 
16:   end for
17: end while
18: return SeedSet

```

(iv) CELF_improved

Improve the traditional CELF:

- (a) When adding first node, only calculate 100 times instead of 1000 times. At the same time, to avoid the loss of accuracy, the confidence interval is used to estimate the upper bound of the increment of influence spread.
- (b) Because of CELF pruning is based on the comparison with last round, use the upper bound of increment of influence spread will not affect the accuracy and reduce more process of ISE calculation, which is the most time-consuming part.
- (c) Use the data structure heap to store the node, the increment of influence spread, and the calculated round. The heap structure can always ensure the node with the greatest increment of influence spread in the top of heap.

The pseudocode is as follows:

Algorithm 2 CELF improved (graph, k, NodeSet)

```
1: initialize nodeHeap, preSpread, SeedSet
2: for each  $n \in \text{NodeSet}$  do
3:   times = 100
4:   [low, high] = the 95% confidence interval of  $ise(100times)$ 
5:   nodeHeap.add(node with high)
6: end for
7: for  $i = 1; i < k; i++$  do
8:    $m = \text{nodeHeap.top}$ 
9:   while  $m \notin \text{calculate in this loop}$  or  $m.\text{times} \neq 10000$  do
10:    if  $m \in \text{calculate in this loop}$  then
11:      times = 10000
12:      spread =  $ise(10000times) - \text{preSpread}$ 
13:      nodeHeap.(node with spread)
14:    else
15:      times = 100
16:      [low, high] = the 95% confidence interval of  $ise(100times)$ 
17:      nodeHeap.add(node with high)
18:    end if
19:  end while
20:  seed = nodeHeap.pop()
21:  preSpread =  $ise(\text{seed}) + \text{preSpread}$ 
22:  SeedSet.add(seed)
23: end for
24: return SeedSet
```

3. Empirical Verification

The experiments are run in two data set:

(1) network.txt: 63 nodes and 159 edges.

(2) NetHEPT.txt: 15233 nodes and 32235 edges.

3.1. ISE

The run time of (IC, LT) model in small dataset -- network.txt and big dataset -- NetHEPT.txt, the approximate results are shown below.

IC, LT (s)	network.txt	NetHEPT.txt
k = 1	0.08, 0.10	0.45, 0.73
k = 5	0.13, 0.17	1.29, 2.10
k = 10	0.15, 0.17	1.58, 3.25
k = 20	0.15, 0.20	2.25, 4.70
k = 50	0.16, 0.17	3.40, 7.50

Table 1. Run time of ISE

According to the run time, even though the graph has 15233 nodes, the ISE calculation can be finished in less than 10s. Therefore, the ISE algorithm is very fast.

3.2. IMP

3.2.1. Compare performance

For network.txt, the spread of influence:

IC, LT	Greedy	Heuristics_CELF_improved
k = 1	11, 12	11, 12
k = 5	30, 37	30, 37
k = 10	42, 54	42, 54
k = 20	52, 62	51, 62
k = 50	62, 62	62, 62

Table 2. Performance Comparison of IMP in network.txt

For NetHEPT.txt, the spread of influence:

IC, LT	CELF	Heuristics_CELF_improved
k = 1	91, 101	91, 101
k = 5	323, 395	323, 393
k = 10	510, 634	508, 634
k = 20	773, 986	773, 984
k = 50	1295, 1689	1291, 1687

Table 3. Performance Comparison of IMP in NetHEPT.txt

It is shown that, the performance of IMP algorithms developed in this project is almost as good as greedy algorithms. Therefore, the quality of this algorithm is proved.

3.2.2. Compare time

For network.txt the time:

IC/ LT	Greedy(total)	Heuristics_CELF_improved
k = 1	8.6	<1
k = 5	1000+	1.6/ 1.5
k = 10	2000+	3.5/ 3.2
k = 20	4000+	15/ 9
k = 50	17000+	29/ 17

Table 4. Time Comparison of IMP in network.txt

For NetHEPT.txt the time:

IC/ LT	Greedy	CELF(total)	Heuristics_CELF_improved
k = 1	NaN	2002	2/ 4
k = 5	NaN	2030	10/ 19
k = 10	NaN	2368	40+/ 80+
k = 20	NaN	4406	170+/ 270+
k = 50	NaN	17234	1600+/1900+

Table 5. Time Comparison of IMP in NetHEPT.txt

It is shown that the “Heuristics_CELF_improved” algorithm is much faster than greedy and traditional CELF in both small size and big size data.

References

- [1] Wei Chen, Yajun Wang, and Siyu Yang. 2009. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD '09). ACM, New York, NY, USA, 199-208.
- [2] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 420–429, 2007.