Influence maximization Report

Zhongwei Wan
11612201
School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11612201@mail.sustc.edu.cn

1. Preliminaries

1.1 Problem Description

The issue of maximizing impact is based on the social network model. The problem is trying to find the most influential group of nodes in a huge social network. Then use two models, one is the Independent Cascade Model and the other is the Linear Threshold Model, to verify the maximum impact of this group of points. Some scholars have demonstrated in their research that under both models, the problem is NP-hard and a simple greedy algorithm successively based on the nice properties of submodularity[1]. In this project, we first use two ISE models, one is the LT model, and the other is the IC model, which calculates their maximum and stable influence based on given network graph and seeds. Secondly, in the network diagram, the CELF++ algorithm is used to find the most influential seeds in the specified time.

1.2 Problem Application

The issue of maximizing impact can be applied to business assessment analysis and finding the most influential people in social networks. For example, in the facebook network, what kind of stars can spread social media information the farthest, or in politics, what kind of senators and congressmen can drive more people in the campaign speech. Therefore, in these examples that can form a network model, it is important to use a more efficient algorithm to optimize the problem to get a better solution.

2. Methodology

2.1 Notation

In my ISE project, The variables are vertex_num, edge_num, graph, seeds, node, total_influence. In my IMP project, the variables are a little different from ISE project, they are out_degree, prev_best, current_seed. Next, I will give more details about these normal variables.

Variables:

-vertex_num: the number of vertex in network.
-edge_num: the number of edge in network.
-graph: the model of network.
-node: the every node in the graph
-total_influence: result of influence evaluated by seeds
-seeds : the seeds are used to evaluate the influence
-out_degree: out degree of each node.
-prev_best: the node has the best marginal benefit .
-current_seed: the current node has best marginal benefit.

2.2 Software and data structure

In the project, I used pycharm IDE and python 3.6 to implement my project, in addition, the Numpy is the library I used in the project. In programming, I use two kind of data structure in python, one is list, the other is default-dict, which is very convenient to store the data according to index.

2.3 Model design

2.3.1 Formulate the problem

In ISE, I first made it clear that under the given network diagram, the seeds set was used to calculate the impact results based on the two models LT and IC. In IMP, according to the definition of the problem, I will use the greedy algorithm and CELF++ to find the seeds in the K-size case, so that these seeds can make the most influential.

2.3.2 Method to solve the problem

In the ISE problem, I use both LT and IC models. The IC compares the randomly generated values of the nodes with the values of the edges, and is activated if it is less than the edge. The LT model adds the adjacent edges of the node's activated parent nodes, and if the weight value is greater than the valve randomly generated by the node, the modified node is activated. Both algorithms use breadth-first search.

In the IMP question, I will calculate the greedy algorithm based on the marginal benefit, and then compare it with the greedy algorithm using the CELF++ optimized algorithm.

2.3.3 Algorithm step by step

In ISE project:

First, in the data preprocessing stage, I store network.txtand seeds.txt in defaultdict and list respectively. **Second**, I will use the breadth-first search principle, according to the LT and IC models mentioned above, respectively. Using a given seeds to search and calculate impact values. **Third**, in a function, 10,000 averaging is calculated and the average result is returned.

In IMP project:

First, pre-process data like the ISE project. **Second,** according to the nature of the sub-module, calculate the marginal benefit after adding a seed, add it to the set, and select the point that maximizes the marginal benefit from the set each time and add it to the seeds.**Third**, according to the CELFII algorithm, the greedy algorithm is optimized, and the calculation amount using the LT and IC models is reduced according to the nature of the algorithm. In the next pseudo code, I will describe the CELFII algorithm in detail. **Fourth**, return the seeds value of the algorithm.

2.3.4 Detail of algorithm

2.3.4.1 Architecture

Main function in ISE and IMP:

--**get_args:** to get the data from command line.
--**read_data**: read the data to corresponding structure.
--**IC_model**: algorithm according to IC model.
--**LT_model**: algorithm according to LT model.
--**sub-modular_greedy**: the basic greedy algorithm.
--**CELFII_IC**: the CELF++ algorithm in IC model
--**CELFII_LT**:the CELF++ algorithm in LT model
--**get_final_seeds**: return the optimal seeds result.

2.3.4.2 pseudo codes of Algorithm

In this part, I will detail the main four algorithm pseudo-codes, they are **IC_model**, **LT_model**, **sub_modular_greedy**, and **CELFII** respectively.

IC_model algorithm: the algorithm is mainly used to calculate the influence according to IC principle. And it used basic BFS algorithm to implement the whole process.

Algorithm 1 : IC_model

```
Input:graph[v][v], seeds[k]
Output:influnce_num
1:  influences = seeds
2:  queue = influences
3:  while queue is not null:
4:      node =  queue.pop()
5:      for i in graph[node]:
6:          random_num = random.random()
7:          if random_num <=  graph[node][i]:
8:              influences.append(i)
9:  influence_num = length(influences)
11: return influence_num
```

LT_model algorithm: This algorithm is very different from the IC model. First, it is based on the result of adding the weights of the edges to which the activated parent node is connected, and comparing with randomly generated valve of the node

Algorithm 2 : LT_model

```
Input: graph[v][v], seeds[k], in_degree[n]
Output:influence_num
1: influence = seeds
2: queue = influences
3: while queue is not null:
4:      node = queue.pop()
5:      for i in graph[node]:
6:          random_num = random.random()
7:          for j in in_degree[n]:
8:          if j in influences:
9:              value += graph[j][i]
11:         if value > random_nun:
12:             influences.append(i)
13:             queue.append(i)
14: influence_num = length(influences)
15: return influence_num
```

submodular_greedy algorithm: This is the basic greedy algorithm for finding the most influential nodes. It is the node with the largest marginal influence added to the seeds every time. Until the K size required by the problem.

Algorithm 3: submodular_greedy

```
Input: graph[v][v], vertex_num, seed_size, out_degree
Output: seeds[k]
```

```
1: seeds = [k]
2: s = dict{ }
3: for i in range(seed_size):
4:     for node in range(1, vertex_num + 1):
5:         s[node] = 0
6:         if node not in seeds:
7:             if node in out_degree:
8:                 for i in range(1000):
9:                     new_seeds.add(node)
10:                    s[node] +=  IC_model(new_seeds)
11:                                - IC_model(seeds)
12:                s[node]/=1000
13:        seed = max(element in s)
14:        seeds.append(seed)
15: return seeds
```

CELF++ algorithm: The algorithm optimizes the number of calculations based on the greedy algorithm, so that the time is shortened when the results of the greedy algorithm are similar.

Algorithm 4 : CELF++

Input: graph
Output:graph[v][v], vertex_num, seed_size, out_degree

```
1: test_count = 0
2: seeds = [k]
3: s = dict {}
4: while seeds.length < seed_size:
5:     if seeds.length = 0:
6:         for node in range(1, vertex_num + 1):
7:             s[node] = 0:
8:             if node in out_degree.keys():
9:                 for i in range(1000):
10:                    s[node]+= IC_model(node)
11:                s[node]/=1000
12:        max_seed = choose the max value from s{}
13:    else if seeds.size ! = 0:
14:        prev = choose the max value from s{}
15:        s[prev] = 0
16:        for i in range(1000):
17:            s[prev]+=IC_model(new_seeds)-IC_model(
18:                                               seeds)
19:        s[prev]/=1000
20:        current_seed = choose the max value from s{}
21:        if current_seed = prev:
22:            seeds.add(current_seed)
23:            s.pop(current_seed)
24: return seeds
```

3. Empirical Verification

In Empirical Verification, I used two data sets to test my ISE and IMP algorithms separately. In IMP, I will use the basic greedy algorithm and the CELF++ algorithm to compare the results and time, and analyze which algorithm will get more effective results within a limited time.

3.1 Data set

In this project, the dataset I use is network.txt and NetHEPT.txt. These two datasets are small datasets with 62 vertices and 159 edges, and the other is a large dataset with 15233 Vertices and 58891 edges.

3.2 Test environment

The test environment for this experiment is pycharm community version 2018.2, python version is 3.6, then, the hardware part is, the host is Intel(R) Core I7-6700HQ, the benchmark speed is @2.60GHz, the kernel is 4, the logical processors are eight, L1 caches 256KB, L2 caches 1.0MB, and L3 caches 6.0MB. No multi-process was used in this experiment.

3.3 Performance measure

In the performance test, I first show the test results of the code in the online judge, and then show the comparative analysis results of the two algorithms in IMP**.**

ISE result table

Data set	Run Time	Result
Network-seeds-IC	**0.703**	**5.0173**
Network-seeds-LT	**0.644**	**5.038**
Network-seeds2-IC	**1.266**	**30.504**
Network-seed2-LT	**1.728**	**37.0186**
NetHEPT-50seeds-IC	**20.094**	**1003.808**
NetHEPT-50seeds-LT	**30.371**	**1268.766**
NetHEPT-5seeds-IC	**28.762**	**277.09**
NetHEPT-5seeds-LT	**54.836**	**377.609**

IMP result table

Data set	Run time	Result
NetHEPT-5-IC	4.522	323.4887
NetHEPT-5-LT	7.916	392.975
NetHEPT-50-IC-bonus	28.045	1046.3255
NetHEPT-50-LT-bonus	1433.8	1433.87

Basic greedy vs CELF++ table

Algorithm	IC-5	IC-10	Time1	Time2
Basic-greedy	30.63	41.67	2.92	8.445
CELF++	30.75	42.19	0.706	1.389

From the analysis in this table, when the data set is network.txt and the model is the IC model, the CELF++ time is fast and the basic greedy algorithm is found when looking for 5 nodes and 10 nodes. Therefore, it can be concluded that the CELF++ algorithm is faster than the basic greedy algorithm when the scale parameter and the calculated average parameter are the same, and the solutions obtained by the two are very similar.

Next step, I will show the performance result of some large network graphs such as **Amazon2** and **DBLP2** in the ISE project and comparing the basic greedy algorithm and CELF++'s performance. In the first table , I will show you the result of Amazon2 and DBLP2 with given seeds = [10599, 10609, 10599, 10609, 10596, 10613] and seeds = [16759, 16760, 17247, 16088, 15489, 4515, 7206, 6939]

ISE result table of Amazon2 and DBLP2

Data set	Run time	Result
Amazon2	0.09	13.802
DBLP2	0.071	25.076

IMP result table of Amazon2 and DBLP2

Algorithm	Data set	Run time	Result
CELF++	Amazon2	4.89	86.44
CELF++	DBLP2	9.88	116.16

From the above table, we can get the running time and results of the datasets in Amazon2 and DBLP2, the goal is to find 5 seeds in the specified time.

3.4 Hype-parameters

In ISE, my Hype-paramete**r** is the average calculation parameter, and I set it to **10000** in order to get a stable solution. In IMP, I set the average calculation parameter **K=500** in the basic greedy algorithm and CELF++. However, when the value of seed_size is greater than 30 or the fixed point and number of edges of the graph are too large, I set my average calculation parameter **K = 50.** The purpose is to be able to draw results within a specified time and speed up the algorithm.

4.Conclusion

4.1 Result

In this project, after my set average calculation parameters, my ISE algorithm tends to be stable under different data sets and different given conditions. In IMP, I used the basic greedy algorithm and CELF++ respectively. Although the results obtained by the two are almost the same, the algorithm speed of CELF++ is higher than that of the basic greedy algorithm. So at the end of the experiment, I chose optimized CELF++ for my final code submission.

4.2 Analysis

This experiment is about maximizing influence. The algorithm based on this problem is widely used in the research of social networks. In this ISE experiment, I implemented two algorithms based on the IC and LT models respectively. There is no advantage or disadvantage in the algorithm as long as it is implemented correctly. In IMP, the **advantage** of my algorithm is that in a moderately sized network diagram, it is possible to get near optimal seeds within a specified time. The **disadvantage** is that in a large-scale network diagram, since I set the value of the average calculation number K There is still a big gap between my CELF++ algorithm and the optimal seeds. However, through this experiment, I also understand that it is not easy to read the article published by others to understand his pseudo code, and then optimize the parameters in the code to get the near optimal solution.

5.Acknowledgments

I am grateful to the teaching assistant Zeng because he helps me a lot. He gave me an enlightenment on the algorithm and answered some questions that are crucial to me.

6.References

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