Experience of the implementation of IMP

Project 2 Report of

CS303 Artificial intelligence

Department of Computer Science and Engineering

BY

Ting Sun

11710108

December 2019

1. **Preliminaries**

1.1 Problem Description

Influence Maximization Problem (IMP) is a problem whose goal is to maximize the influence among the fixed-number seeds you choose and you can select any node in the network [1]. After you choose these seeds as a set, there is a kind of algorithm to calculate the influence of the seeds set.

1.2 Problem Applications

The problem is very common in the real world, especially in the social network. For example, if you are a advertiser, you have to consider how to advertising in order to get the maximum profits and in this case, there is no doubt that you may use the IMP algorithm.

2. **Methodology**

2.1 Notation

In my description, symbol notation is not needed.

2.2 Data Structure

ISE:

activateQueue: a queue to store all activate nodes.

now_list: contains the nodes activated in last turn.

new_list: used in every new round influence and all the newly-activated nodes this new round will be added to it.

IMP:

activateQueue: a queue to store all activate nodes.

candidateHeap: a heap to store all the inactivate nodes with the priority of their estimated potential influence. The top element of the heap is the one whose estimated influence is biggest in the heap.

seeds_set: a list to store all the selected nodes.

net: a adjacent list with its index means the identify number of the node and its corresponding list of the index is the adjacent list of the node.

2.3 Model design

ISE:

I just wrote as what the pseudo-algorithm described in slides of Zhao Yao [1]. In LT or IC type, I use a while loop to try to get the newly-activated nodes. If we cannot find that, then the loop will be broken. The judging algorithms themselves are constrained by LT or IC themselves.

IMP:

The user inputs will be given by command parameters and I use a function to get them. Then, the network will be stored in the “net” adjacent list.

I use the algorithm similar to CELF [2] to solve the problem.

First, we will initial the candidateHeap whose priorities is their estimated current potential influences.

Then, we will chose the top element in the heap every turn. And every time when we choose a node, we will maintain the heap to make the next chose convenient.

2.4 Detail of algorithms

ISE:

As mentioned in slides of Zhao Yao [1], I used while loop similar to BFS. In every, I will use a null list named new_list to contain the newly-activated nodes activated by activated nodes in last round contained in now_list. And before the new round, I will assign new_list to now_list. The check test of the while loop is whether the now_list is null.

IMP:

First, we push all the nodes into the candidateHeap whose priorities are their estimated potential influences at that time, i.e if we select it as the seed what is the additional influence can we get.

Then, we will chose the top element in the heap every turn. After we choose the element, we will maintain the heap. First, we will check whether the top element of the current heap is in the activateQueue. Then, we will recalculate the influence of the top node to check whether the newly-calculated influence is smaller than the second or the third node in the heap. If any answer of these two questions is True, we will discard the top element and re-push it into the heap with its newly-calculated influence. These procedure will be executed in a while loop until all the answer of the two questions are False. Finally, if we can not satisfy these two constraints any more, we will stop the while loop and just select some un-selected nodes and add them into seeds set.

Pseudo code:

ISE:

```
seed, net = get all parameters from cmd
choose a number as cycly_time
cnt=1
run it cycly_time:
    activatedSet = [seed]
    now_list =[seed]
    if type =="LT":
        while now_list is not null:
            new_list = null
            for node in now_list:
                for neibor in net[node]:
                    if neibor in activatedSet:
                        continue
                    total_wei=all activated neiborhood node of neibor
                    if total_wei>threshold of neibor:
                        activatedSet.push(neibor)
                        new_list.push(neibor)
                        cnt+=1
            now_list = new_list
    if type == "IC":
        while now_list is not null:
            new_list = null
            for node in now_list:
                for neibor in net[node]:
                    if neibor in activatedSet:
                        continue
                    if random(0,1,float number)<weight between neibor and node
                        activatedSet.push(neibor)
                        new_list.push(neibor)
                        cnt+=1
            now_list = new_list
print cnt/cycly_time
```

IMP:

```
seed_num, net = get all parameters from cmd
choose a number as cycly_time
seeds_set = []
candidateHeap = a big end heap
for node in net:
    value = estimated( node, run appropriate times)
    put (value, node) into candidateHeap
run seed_num times:
    top = candidateHeap[0][1]
    seeds_set.push(top)
    activated top and neibors influencced by top
    candidateHeap.pop()
    while True:
        top = candidateHeap[0][1]
        if top is activated:
            candidateHeap.pop()
            continue
        new_val = estimated( top, run appropriate times)
        if new_val >= candidateHeap[1][0] and new_val >= candidateHeap[2][0] :
            break
        new_ele = (new_val, new_val < candidateHeap[1][0] )
        candidateHeap.pop()
        candidateHeap.push(new_ele)
print ele in choose
```

3. **<u>Empirical Verification</u>**

3.1 Dataset

ISE:

In ISE, I just submitted a primary edition python file and after I passed all the test cases I forgot to optimize it. Therefore, I used no data to test it except for the data that carp platform supply.

IMP:

In IMP, I wrote a function to generate network files, which could randomly output a network of any nodes number and any edges number we wanted.

I used it to generate a network of various sizes, from 100 nodes to 100000 nodes, to test my IMP.py file.

Surely, this test data could also been used in ISE, but time could not turn back and since the deadline of ISE had passed, all I did for ISE were in ruin.

3.2 Performance measure

ISE:

As I said before, I was too careless and too busy to notify the deadline of ISE, and thus the only measure of it is that I passed all the test of ISE.

IMP:

The time cost of each part of the code is measured in this part. So I used it to ensure my algorithm can run over a network of 100000 within 80 seconds. If my algorithm fail to do that, I will optimize it by pruning, parameters adjusting, and so on.

What I did not do but is also important is that test the correctness of the output.

Test envirment:

This project is written in Python3.7 with editor Visual Studio Code.

The main testing platform is Windows 10 Professional Edition (version 1803) with Intel® Core™ i5-8250U @ 1.60GHz   1.80GHz.

3.3 Hyperparameters

ISE:

The main hyperparameters of ISE.py of mime is the cycle_times, which means how many times should the estimation part to run before it stops. Had I not forgot the deadline of ISE, I would like to use multi-process method and run estimation part as many times as possible and stop this part 3 seconds before it run out of the given time.

IMP:

Multi-process in IMP is more difficult than it in ISE, and when I tried to use multi-process in IMP, something always went wrong. But obviously multi-process and the number of cores you used might throw hugely significance in IMP.

In IMP, I also choose to adjust the cycle_times according to the size of the network and given time. If we have much time, I would set cycle_times higher, otherwise lower it. The key point is that we want to maximize the given time and thus we want to run the code in almost the given time.

3.4 Experimental results

ISE:

Since I had forgotten the deadline of ISE, there is only submit results for ISE:

Info	ExitCode	RunTime(s)	Result	Dataset
Solution accepted	0	37.72	1127.0720666666666	NetHEPT-seeds50-IC
Solution accepted	0	3.74	36.9786	network-seeds5-LT
Solution accepted	0	2.20	30.473266666666667	network-seeds5-IC
Solution accepted	0	5.63	26.9939	network-5-IC
Bias Too Large	0	1.52	5.037	network-seeds5-IC

Info	ExitCode	RunTime(s)	Result	Dataset
Solution accepted	0	6.24	32.7272	network-5-LT
Solution accepted	0	5.92	23.6169	network-5-IC
Solution accepted	0	66.34	1456.5132	NetHEPT-seeds50-LT
Timed out	-1	121.94	0	NetHEPT-seeds50-LT
Timed out	-1	121.28	0	NetHEPT-seeds50-LT

Info	ExitCode	RunTime(s)	Result	Dataset
Solution accepted	0	11.80	943.9742	random-graph5000-5000-seeds-50-IC
Solution accepted	0	10.01	191.1216	random-graph1000-1000-seeds-10-LT
Solution accepted	0	8.36	190.6334	random-graph1000-1000-seeds-10-LT
Timed out	-1	62.00	0	NetHEPT-5-IC
Timed out	-1	62.21	0	NetHEPT-5-IC

ExitCode	RunTime(s)	Result	Dataset
0	36.68	1038.9296	random-graph5000-5000-seeds-50-LT
0	4.12	167.6494	random-graph500-500-seeds-10-IC
0	39.52	1037.607	random-graph5000-5000-seeds-50-LT
0	62.99	4094.8506	random-graph50000-50000-seeds-100-IC
0	2.18	30.7988	random-graph50-50-seeds-5-IC

IMP:

Part of my submission:

Info	ExitCode	RunTime(s)	Result	Dataset
Solution accepted	0	50.68	1252.8301	NetHEPT-50-IC
Solution accepted	0	23.95	1245.2347	NetHEPT-50-IC
Solution accepted	0	80.92	1652.7056	NetHEPT-50-LT
Solution accepted	0	60.62	1647.9688	NetHEPT-50-LT
Solution accepted	0	10.92	1246.4681	NetHEPT-50-IC
Solution accepted	0	11.41	392.8023	NetHEPT-5-LT
Solution accepted	0	2.80	313.9682	NetHEPT-5-IC
Solution accepted	0	2.64	35.4045	network-5-LT
Solution accepted	0	2.33	29.8062	network-5-IC
Solution accepted	0	14.70	1268.2723	NetHEPT-50-IC
Solution accepted	0	65.03	1643.0191	NetHEPT-50-LT
Solution accepted	0	4.75	1250.424	NetHEPT-50-IC
Solution accepted	0	14.38	1638.5253	NetHEPT-50-LT
Solution accepted	0	17.19	1628.6448	NetHEPT-50-LT
Vaule Error! Not int.	0	3.76	0	NetHEPT-50-IC
Solution accepted	0	3.17	1243.6685	NetHEPT-50-IC
Solution accepted	0	16.57	1237.3101	NetHEPT-50-IC
Solution accepted	0	8.34	1248.8709	NetHEPT-50-IC
Solution accepted	0	4.14	1226.2967	NetHEPT-50-IC
Solution accepted	0	28.10	1560.5396	NetHEPT-50-LT
Solution accepted	0	1.59	236.0224	NetHEPT-5-IC
Solution accepted	0	1.81	276.055	NetHEPT-5-IC
Solution accepted	0	6.32	321.2869	NetHEPT-5-LT
Solution accepted	0	1.86	987.0401	NetHEPT-50-IC
Solution accepted	0	6.36	1334.2458	NetHEPT-50-LT

Following record are based on time limit=60

Test record of the final code of IMP:

Nodes #	Edges #	Run times	Avg time (sec)	Ever time exceed?
150	300	10	55.85	No
500	2000	10	43.32	No
2000	5000	10	57.18	No
5000	15000	10	49.52	No
10000	30000	10	51.56	No
30000	60000	10	50.63	No
70000	140000	10	58.69	No
100000	200000	10	53.23	No

Additional, the total score I get in this lab is 105/200.

The influence estimation of outputted seeds set was not conducted due to my poor memory.

3.5 Conclusion

ISE:

Evidently, my ISE.py can pass all the test cases. However, it is not scalable at all and it is not amazing that it failed in the final test of the platform.

IMP:

The experimental results show that the time distribution of the code in different network sizes are almost even and never exceed the time limit.

However, when Nodes number is 500 and edges number is 2000, the time cost is too little, which shows that the algorithm does not make full use of time in this case.

Also, the influence verification of outputted seeds set was not conducted and this can remain to be future work if I have time.

4. References

[1] Zhao, Yao. (2019). IMP.pdf. Retrieved from https://sakai.sustech.edu.cn/access/content/group/f6baff92-a8d1-4805-a353-c737451f5a9a/LAB/Lab1 on 2019 Nov 19th.

[1] Z J. Leskovec et al. Cost-effective outbreak detection in networks. In *KDD 2007.*