IMP Report for Project 2

Course: Artificial Intelligence

Name: 卢斓

ID: 11810935

1. Preliminary

1.1 Problem Description

The content of this project is to solve the problem of Influence Maximization Problem, namely IMP. For a given social network which is made up of many nodes and edges, IMP asks us to find a seed set, a subset of nodes, of size k where k is given as a parameter which can maximize the “influence spread”, namely the number of expected influenced nodes.

Through the designing and modifying process, we students are expected to exercise and master the Independent Cascade(IC) model and Linear Threshold model of Stochastic diffusion models for Influence Spread Estimation(ISE) as well as IMM Algorithm for Influence Maximization Problem(IMP).

In the project, PyCharm is used as my python editor and some python libraries like numpy, random and datatime play an important role in designing and testing my own codes. In addition, a specialized website,, namely

http://10.20.107.171:8080/, is provided by the teacher for further testing and examining.

1.2 Problem Applications

Actually, IMP has a wide range of applications in the real world. Take the social media as an example. When an advertiser wants to find one or several celebrities for promoting his services or products, it is crucial for him to calculate and get which persons will possibly have the max number of audiences. Same things happen in a wide range of scenarios in our lives including virus marketing, recommendation system, information diffusion, time detection, expert discovery, link prediction and so on.

2. Methodology

2.1 Notation

2.1.1 Notations for ISE

class Edge(object): Edge class with attributes “u”, “v”, “w”, and “next_e” representing the start node, end node, weight of the edge and the next edge with the same start node.

IC(head, edge, active_set, active): IC model for calculating the influence spread. The input “head” and “edge” give the head and edge information needed to construct the graph while “active_set” is the seed set and “active” records the

information whether a node is visited or not. What is returned is “ans” representing the calculated value for influence spread.

LT(n, head, edge, active_set, active, weight_total): LT model for calculating the influence spread. The input “n”, “head” and “edge” give the number of nodes in a a graph and the head and edge information needed to construct the graph while “active_set” is the seed set and “active” records the information whether a node is visited or not. Besides, “weight_total” tells the weight information of each edge.

main: Graph is constructed in this part. Besides, the execution of IC or LT method is performed here while the repeated number is decided by the given time limit. The result “summ” is printed to the console.

2.1.2 Notations for IMP

class Graph(object): Graph class with an attribute “edge_net” which is a dictionary whose keys are start nodes of the graph and values are the dictionaries storing the mapping information of all this node’s end nodes corresponding to the edges’ weight values. Method “add_edge(self, u1, v1, w1)” add an edge to the attribute “edge_net” where “u1”, “v1”, “w1” are the start node, end node and weight of this edge. Method “get_neighbors(self, node)” returns all neighbors which can reached by node “node” through only one edge.

get_RR(model, node): Method for generating RR sets. The input “model” and “node” tells that a RR set for node “node” should be generated through LT or IC model and then be returned.

IC_RR(node): Method for generating RR of node “nodes” sets through IC model and returning the RR sets generated.

LT_RR(node): Method for generating RR of node “nodes” sets through IC model and returning the RR sets generated.

sampling(time1): Method for repeatedly generating RR sets of randomly selected nodes according to the given time limit “time1” and returning “R” as the set of all generated RR sets.

nodeSelection(R, k): Method for selecting “k” nodes which have the maximum influence spread according to the given RR sets “R”. What is returned is the seed set of size “k” which has the maximum influence spread.

IMM(time1, k): Method for performing IMM algorithm to find and return the seed set “S” of size “k” which has the maximum influence spread.

main: Method for constructing graph, calling IMM algorithm and printing the results.

2.2 Data Structure

2.2.1 Data Structure for ISE

Global data structure:

head: Dictionary to map a start node with its final edge index being added into the graph.

edge: List of all edges in the graph.

active_set: List of the given seed set.

active: List of node information recording whether a node is visited or not.

weight_total: List of current accumulated influence values from neighbors for all nodes.

IC(head, edge, active_set, active):

The input and output have already been discussed in part 2.1.

head1: Copy of “head”.

edge1: Copy of “edge”.

active_set1: Copy of “active_set” initially. It is used to store the nodes to be used as seeds in the next turn which will change as the code is performing.

active1: Copy of “active”. It is used to store the visiting information of nodes which will change as the code is performing.

new_active: List of nodes which are activated in this turn.

LT(n, head, edge, active_set, active, weight_total):

The input and output have already been discussed in part 2.1.

head1: Copy of “head”.

edge1: Copy of “edge”.

active_set1: Copy of “active_set” initially. It is used to store the nodes to be used as seeds in the next turn which will change as the code is being performed.

active1: Copy of “active”. It is used to store the visiting information of nodes which will change as the code is being performed.

weight_total1: Copy of “weight_total”. It is used to store the current accumulated influence values from neighbors for all nodes which will change as the code is being performed.

thresh: List of numbers between 0 and 1 generated randomly to represent the activating limit of nodes. When the influence given from its neighbors is larger than this node’s activating limit, this node can be activated.

new_active: List of nodes which are activated in this turn.

main:

file_name: File which stores the information of the graph.

seed: Input information of the seed set.

model: Input information of the model which is “IC” or “LT”.

time_limit: Input information of the time limit.

2.2.2 Data Structure for IMP

Global data structure:

n: Number of nodes in the graph.

m: Number of edges in the graph.

graph: Graph object constructed.

seed_num: Expected size of the seed set.

time_limit: Input information of the time limit.

model: Input information of the model which is “IC” or “LT”.

active: List of node information recording whether a node is visited or not.

IC_RR(node):

The input and output have already been discussed in part 2.1.

active_set1: It is used to store the nodes to be considered in the next turn which will change as the code is being performed.

seeds: The generated RR set.

active1: Copy of “active”. It is used to store the visiting information of nodes which will change as the code is performing.

new_active: List of nodes which are chosen in this turn.

LT_RR(node):

The input and output have already been discussed in part 2.1.

active_set1: It is used to store the nodes to be considered in the next turn which will change as the code is being performed.

seeds: The generated RR set.

active1: Copy of "active". It is used to store the visiting information of nodes which will change as the code is performing.

new_active: List of nodes which are neighbors of the node in active_set1 in this turn.

nodeSelection(R, k):

The input and output have already been discussed in part 2.1.

nodes: Dictionary for mapping a node with a list which stores the indexes of RR sets which have not been discussed yet in R of this node.

fre: List of number of RR sets which have not been discussed yet where this node is in.

2.3 Model Design

2.3.1 ISE Model

Formulation: Given the file_name of File which stores the information of the graph, information of the seed set, choice of model which is "IC" or "LT" and time limit, the problem is to calculate the influence spread of the seed set in this graph within time limit.

Solution and Main Steps: To solve this problem, we repeatedly call the corresponding method used to calculate the influence spread if there is still time remained and the final value being printed is the average value of all these calculated influence spread values. Detailed information of how IC and LT models work is given at part 2.1 and 2.4.

2.3.2 IMP Model

Formulation: Given the file_name of File which stores the information of the graph, expected size of the seed set, choice of model which is “IC” or “LT” and time limit, the problem is to find out and print the nodes in the seed set which has the maximum influence spread.

Solution and Main Steps: To solve this problem, we firstly should generate a reverse graph of the given graph and then call the “IMM” method where firstly we call method “sampling” to generate RR sets according to the time remained use the chosen model type. Then, method “nodeSelection” is called to pick out the k nodes which have the maximum influence spread which is evaluated through greedy algorithm. Finally, the expected seed set of size k is generated and nodes are printed out in the required format. Detailed information of how these methods work is given at part 2.1 and 2.4.

2.4 Detail of Algorithm

2.4.1 ISE Detail

IC(head, edge, active_set, active): IC model for calculating the influence spread. The input “head” and “edge” give the head and edge information needed to construct the graph while “active_set” is the seed set and “active” records the information whether a node is visited or not. What is returned is “ans” representing the calculated value for influence spread.

```
IC(head, edge, active_set, active):
    initial ans to be the length of active_set
    while active_set is not empty:
        initialize new_active to be an empty list
        for each u in active_set:
            for each outgoing edge of u:
                let v be the end node and w be the weight value
                if v is not visited:
                    randomly generate ran between 0 and 1
                    if ran <= w and activate successfully:
                        add v to new_active and set it visited
        ans += len(active_set)
        active_set = new_active
    return ans
```

LT(n, head, edge, active_set, active, weight_total): LT model for calculating the influence spread. The input “n”, “head” and “edge” give the number of nodes in a a graph and the head and edge information needed to construct the graph while “active_set” is the seed set and “active” records the information whether a node is visited or not. Besides, “weight_total” tells the weight information of each edge.

```
LT(n, head, edge, active_set, active, weight_total):
    thresh = []
    for i from 0 to n:
        add a randomly generate number between 0 and 1 to thresh[i]
    initial ans to be the length of active_set
    while active_set is not empty
        initialize new_active to be an empty list
        for each u in active_set:
            for each outgoing edge of u:
                let v be the end node and w be the weight value
                if v is not visited:
                    weight_total[v] += w
                    if weight_total1[v] >= thresh[v]:
                        add v to new_active and set it visited
        active_set1 = new_active
        ans += len(new_active)
    return ans
```

main: Graph is constructed in this part. Besides, the execution of IC or LT method is performed here while the repeated number is decided by the given time limit. The result “summ” is printed to the console.

main():

start = current time

get parameters from input and construct graph

let sum = 0

let start1 = current time

calculate the influence spread for 10 times and add the results to sum

let end = current time

let gap = (end - start1) / 10

let N = ((time_limit - 1)*0.8 - (end - start) - gap * 10) / gap

repeatedly calculate the influence spread for N times and add the results to sum

sum = sum/(N+10)

print sum

2.4.2 IMP Detail

get_RR(model, node): Method for generating RR sets. The input “model” and “node” tells that a RR set for node “node” should be generated through LT or IC model and then be returned.

```
get_RR(model, node):
    if model is 'IC':
        return IC_RR(node)
    else:
        return LT_RR(node)
```

IC_RR(node): Method for generating RR of node “nodes” sets through IC model and returning the RR sets generated.

```
IC_RR(node):
    initially active_set and seeds to be a list containing only node
    set node visited
    while active_set is not empty:
        initialize new_active to be an empty list
        for each u in active_set:
            outgoing_edges = get_outgoing_edges(u)
            for each edge in outgoing_edges:
                let v, w be the end node and weight of the edge
                if v is not visited:
                    randomly generate ran between 0 and 1
                    if ran <= w and activate successfully:
                        add v to new_active and seeds and set it visited
```

```
            active_set = new_active
        return seeds
```

LT_RR(node): Method for generating RR of node “nodes” sets through IC model and returning the RR sets generated.

```
LT_RR(node):
        initially active_set and seeds to be a list containing only node
        set node visited
        while active_set is not empty:
            initialize new_active to be an empty list
            for each u in active_set:
                neighbors = get_neighbors(u)
                for each v in neighbors:
                        add v to new_active
            if new_active is empty:
                    break
            active_set = [randomly select a node in new_active]
            if active_set[0] is not visited:
                    add active_set[0] to seeds and set it visited
            else:
                    set active_set empty
        return seeds
```

sampling(time1): Method for repeatedly generating RR sets of randomly selected nodes according to the given time limit “time1” and returning “R” as the set of all generated RR sets.

```
sampling(time1):
    set R to be a empty list
    cur = current time
    while (cur - time1) < (3*time_limit/5) and len(R) < 4000000:
        let v be a randomly selected node
        RR = get_RR(model, v)
        add RR to R
        cur = current time
    return R
```

nodeSelection(R, k): Method for selecting “k” nodes which have the maximum influence spread according to the given RR sets “R”. What is returned is the seed set of size “k” which has the maximum influence spread.

```
nodeSelection(R, k):
    let nodes be an empty dictionary and fre be an empty list
    for i from 0 to n:
        add 0 to fre[i]
    let limit = len(R)
```

```
for i from 0 to limit-1:
    let rr = R[i]
    for each node in rr:
        fre[node] += 1
        if node not in nodes:
            initial nodes[node] to be an empty set
        add i to nodes[node]
initial s to be an empty list
for i from i to k
    let val = max(fre)
    let ind = index of val in fre
    add ind to s
    let nodes1 = copy of nodes[ind]
    for each u in nodes1:
        for each node in R[u]:
            remove u from nodes[node]
            fre[node] -= 1
return s
```

IMM(time1, k): Method for performing IMM algorithm to find and return the seed set “S” of size “k” which has the maximum influence spread.

IMM(time1, k):

R = sampling(time1)

S = nodeSelection(R, k)

return S

3. Empirical Verification

3.1 Dataset

For both ISE and IMP, localhost tests are conducted. Three files are provided by the teachers through sakai website for showing the standard input format. They can be used to test the structure correctness of my code and the structure design is regarded to be true if there is output in correct format. Other sample inputs are generated randomly by myself for localhost tests.

Besides, the testing website http://10.20.107.171:8080/ also helps a lot when I want to do experiments on time cost and answer correctness with its specific given sample inputs. Different return contents can be used to discover the errors in my code and then I can make adjustments. Specific description is given in part 3.2.1.

3.2 Performance Measure

3.2.1 Performance Measurement

3.2.1.1 ISE Performance

As discussed above in part 3.1, localhost tests are conducted to verify the structure correctness of my code. What's more, the feedback from the provided IMP website is greatly important. Two main types of error type are "Timed out" and "Bias Too Large" with exit code 1 showing execution time exceeding the time limit and exit code 0 showing the result not within the limits of error respectively.

3.2.1.2 IMP Performance

Similar with measurement of ISE, methods through localhost and IMP platform are provided. Exit code 0 means a success execution of this code without errors or interrupts while exit code 1 implies a time exceeding limit error. Especially for IMP, outstanding results shown on the rank list of other students can be used to measure the accuracy of my own results. Communications among peers plays an important role in adjusting and perfecting my own code.

3.2.2 Test Environment

All tests are conducted through either the testing platform provided by the course or the localhost tests. The localhost is supported by window 10 system with processor information being Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz and RAM size being 8.00GB. Detailed description about testing steps is provided in part 3.1.

3.3 Hyperparameters

3.3.1 ISE Hyperparameters

Time_limit and number of repeatedly calculating times N are two important parameters tightly related with the accuracy of ISE result.

For the above two parameters, time_limit is set to be the input and thus can not be modified. And for the second parameter, we should make it as large as possible within time_limit. So I design the code which can dynamically calculate the suitable repeating times according to time_limit. Specific information is given at part 2.4.1 method “main”.

3.3.2 IMP Hyperparameters

Time_limit and size of RR sets are two important parameters tightly related with the accuracy of ISE result.

For the above two parameters, time_limit is set to be the input and thus can not be modified. And for the second parameter R size, we should make it as large as possible within time_limit as well as memory limit. So I design the code which can dynamically calculate the suitable repeating times of generating RR sets according to time_limit. Besides, an upper bound of size is set to avoid memory exceeding limit. Specific information is given at part 2.4.2 method “sampling”.

3.4 Experimental Results

3.4.1 ISE result

All test cases provided as self-judgement and final judgement are passed successfully. For detailed information, among all test cases in final judge, only two test cases happen to be timed out for one time among all 5 times opportunities for each test case.

#5	FINISHED	11-17 04:47:54	Solution accepted	0	49.42	8489.571942446044	random-graph5000-5000-seeds50-IC
#4	FINISHED	11-17 04:47:54	Solution accepted	0	47.68	9032.194764397906	random-graph5000-5000-seeds50-LT
#3	FINISHED	11-14 15:49:59	Solution accepted	0	90.49	1457.0947795998195	NetHEPT-seeds50-LT
#2	FINISHED	11-14 15:48:06	Solution accepted	0	84.57	1127.219479653102	NetHEPT-seeds50-IC
#1	FINISHED	11-14 15:46:36	Solution accepted	0	46.27	37.02320237869905	network-seeds5-LT

#5	FINISHED	11-17 04:47:54	Solution accepted	0	48.22	8487.870632672333	random-graph5000-5000-seeds50-IC
#4	FINISHED	11-17 04:47:54	Solution accepted	0	38.66	1110.916730495678	random-graph1000-1000-seeds10-IC
#3	FINISHED	11-17 04:47:54	Solution accepted	0	99.09	53527.20795107033	random-graph50000-50000-seeds100-LT
#2	FINISHED	11-17 04:47:54	Solution accepted	0	102.98	53530	random-graph50000-50000-seeds100-LT
#1	FINISHED	11-17 04:47:54	Solution accepted	0	46.66	731.3043856091767	random-graph500-500-seeds10-IC

3.4.2 IMP result

For self-judgement, all provided test cases are passed successfully. When it comes to the accuracy of my code, it is found that the code performs greatly on small size datasets with just the same answer as the top persons' in the rank list. However, for larger datasets with size up to 500, my result turns out to be a bit different from the top ones' results while it is still in the error limit.

#5	FINISHED	11-28 11:06:48	Solution accepted	0	41.81	30.6324	network-5-IC
#4	FINISHED	11-28 11:05:12	Solution accepted	0	41.96	37.5412	network-5-LT
#3	FINISHED	11-28 11:03:15	Solution accepted	0	38.47	323.4887	NetHEPT-5-IC
#2	FINISHED	11-28 11:00:20	Solution accepted	0	38.31	392.975	NetHEPT-5-LT
#1	FINISHED	11-28 10:58:18	Solution accepted	0	75.22	1295.6602	NetHEPT-50-IC

#5	FINISHED	11-28 10:51:26	Solution accepted	0	75.12	1701.9953	NetHEPT-50-LT
#4	FINISHED	11-28 10:48:59	Solution accepted	0	75.82	4320.0699	NetHEPT-500-IC
#3	FINISHED	11-28 10:44:00	Solution accepted	0	75.81	5575.907	NetHEPT-500-LT
#2	FINISHED	11-28 10:40:41	Solution accepted	0	51.00	5566.1586	NetHEPT-500-LT
#1	FINISHED	11-28 10:31:01	Solution accepted	0	28.08	37.5412	network-5-LT

Since final judgment has not be conducted, the result of this process is not discussed here.

3.5 Conclusion

3.5.1 Advantages

For the IMM algorithm itself, it performs more accurately compared with other algorithms for IMP.

Secondly, for both ISE and IMP, codes for time control are written to prevent time exceeding limit error. Specifically, for ISE, since the the result will get more accurate if there are more times of execution for one test input while more times of execution costs more time to finish, I adjust the times of execution with the information of the total given time and time cost for one complete execution which is calculated by 10 times execution time divided by 10. in this way, I can make the most of the given time while making sure that my code will not exceed the time limit. Similar to ISE, for IMP, I adjust the time used for generating the RR sets by the given time instead of the episode parameter used for controlling the RR sets' size by accuracy. Detailed description about the time control paragraph is given in part 2 of this report.

3.5.2 Disadvantages

Generally, IMM is slower than most other greedy and heuristic algorithms.

Secondly, although the time control of my code is realized well, on the contrast, the accuracy of results may lost in a degree because I can not guarantee the repeated times of calculating ISE or generating RR sets for IMP if time is short and not enough.

3.5.3 Summary

It seems that my code provides good guarantee for controlling time while the accuracy may be lost in a degree, However, as long as the time limit is set suitably to match the size of the test dataset, the result of my code can be as accurate as the the code realized with accuracy control. When the tie limit is set to be small, my results may be not that accurate while the the programs realized with accuracy control will directly run into time exceeding error with no results provided at all.

Besides, experiments can be done to improve the accuracy of the node selection part by designing a more optimized selection strategy in the overall state instead of the greedy optimized selection strategy in the part.

4. References

[1] Y. Tang, Y. Shi, and X, Xiao, "Influence maximization in near-linear time: A martinagle approach", Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, (New York, NY, USA), pp. 1539-1554, ACM, 2015.

[2] Poornima Nedunchezhian, Shomona Gracia Jacob, "Social Influence Algorithms and Emotion Classification for Prediction of Human Behavior: A Survey", Recent Trends and Challenges in Computational Models (ICRTCCM) 2017 Second International Conference on, pp. 55-60, 2017.