Report for Project 2: Influence Maximization Problems

Zhan Zhuang 11811721
Computer Science and Engineering
Southern University of Science and Technology
11811721@mail.sustech.edu.cn

1. Preliminaries

1.1. Problem description

This report focuses on the algorithm and solution of an abstract problem named Influence Maximization Problem (**IMP**) as well as the way to tune parameters for a better program performance with given conditions. The program has been developed as the first project for the course of CS303 Artificial Intelligence, held in the Fall 2020 at Southern University of Science and Technology.

For **IMP**, to be specific, a social network is modeled as a directed graph $G = (V, E)$ with nodes in V modeling the individual in the network and each edge $(u, v) \in E$ is associated with a weight $w(u, v) \in [0, 1]$ which indicates the probability that u influences v. Let $S \subseteq V$ to be the subset of nodes selected to initiate the influence diffusion, which is called the seed set. The stochastic diffusion models specify the random process of influence cascade from S, of which the output is a random set of nodes influenced by S. The expected number of influenced nodes $\sigma(S)$ is the influence spread of S. The goal of **IMP** is to maximizes the $\sigma(S)$ with a number of seed $|S| = k$.

This problem has been proved to be NP-hard [1] and the influence spread estimation (**ISE**) is P-hard [1] under the definitions, so the target of **IMP** is to find a approximately optimal solution with given conditions.

Kempe et al [1]. first formalize the **IMP** and define two basic propagation models, the independent cascade (**IC**) model and the linear threshold (**LT**) model to simulate the influence spread. In addition, the author prove that a natural greedy strategy obtains a provable solution in 63% optimal range using an analytical framework based on submodular functions. In this project, I used an algorithm implemented with a **D-SSA** framework which was proposed by H. T. Nguyen et al. [2]. The method is also based on the reverse impact sampling (**RIS**) [3] framework.

1.2. Software

This project is all written in Python 3.9 with PyCharm and Jupyter lib. The python library used including argparse, sys, random, time, multiprocessing, numpy, queue and math.

1.3. Problem applications

As a key problem in viral marketing [4], **IMP** plays a role in many practical problems such as epidemic control and assessing cascading. Viral marketing means that a company selects a few influential individuals in a social network and provides them with incentives (e.g., free samples) to adopt a new product, hoping that the product will be recursively recommended by each individual to his/her friends to create a large cascade of further adoptions [4], [5], [6].

With this program, we can try two diffusion modes to run the influence maximization problems using D-SSA framework with given conditions by python. Also, it can be compared with other algorithms such as simulation-based Monte Carlo method and pruning techniques, proxy-based influence ranking proxy, and sketch-based forward influence FI-Sketch [7].

2. Methodology

2.1. Notation

The primary notations used in this report are listed in **Table 1**.

TABLE 1. NOTATIONS

Symbol	Definition
G	a social network G
G^T	G^T constructed by reversing the direction of each edge in G
n	the number of nodes in G
m	the number of edges in G
k	the size of the seed set for influence maximization
w_{uv}	the propagation probability of an edge from u to v
$\mathbb{I}(S)$	Influence Spread of seed set $S \subseteq V$
$\hat{S}_k$	The returned size-k seed set of D-SSA
S_k^*	An optimal size-k seed set
R_j	A random RR set
$\mathcal{R}$	A collection of random RR sets
$Cov_{\mathcal{R}}(S)$	#RR sets $R_j \in \mathcal{R}$ covered by S
$\mathbb{I}_{\mathcal{R}}(S)$	$\frac{\mathrm{Cov}_{\mathcal{R}}(S)}{\lvert\mathcal{R}\rvert}$
θ	the size of $\mathcal{R}$
$\Upsilon(\epsilon, \delta)$	$\Upsilon(\epsilon, \delta) = \left(2 + \frac{2}{3}\epsilon\right) \ln \frac{1}{\delta} \frac{1}{\epsilon^2}$

2.2. Data structure

The primary data structure used in this report are listed in **Table 2**.

TABLE 2. DATA STRUCTURE

Function	Definition
active_set	A set to store the active node set currently
network	Two two-dimensional arrays stored G and G^T
find(v)	A list of the RR set covered node v
seed_list	A list of seeds selected

2.3. Model design

For **ISE**, the most conventional method is monte Carlo simulation to determine whether a node affects the next node through random number, and then take an average of all simulation results to get the final result. For the two different cascading methods **IC** and **LT**, the propagation way will be described later.

For **IMP**, after reviewing the literature, I found that many algorithms have been proposed based on the two properties (monotonic and submodular as below) include greedy-based GREEDY [1], CELF [8], CELF++ [9] as **Algorithm 1**, and RIS frame-based IMM [10], TIM [6], BKRIS [11], SSA [2] as **Algorithm 2** and can return a result with an approximation ratio of $1 - 1/e$. Finally, I selected the D-SSA [2], which is a stop-and-stare algorithm that automatically selects near-optimal ϵ_1, ϵ_2, ϵ_3 settings and has a better performance.

Monotonic property. For $S, T \subseteq V$ and $S \subseteq T$, we have $\sigma(T) \geq \sigma(S)$

Submodular property. For $S, T \subseteq V, S \subseteq T$ and $u \in V \wedge u \notin S \cup T$, we have $\sigma(S \cup \{u\}) - \sigma(S) \geq \sigma(T \cup \{u\}) - \sigma(T)$

Algorithm 1 Greedy framework(k, f)

```
Input: f: a submodular and monotone function; k : number of seeds;
Output: S: the seed set with size k;
1: initialize $S \leftarrow \emptyset$
2: while $|S| < k$ do
3:     select $u \leftarrow \arg\max_{w \in V \setminus S}(f(S \cup \{w\}) - f(S))$
4:     $S \leftarrow S \cup \{u\}$
5: end while
6: return $S$
```

As shown in the Figure 1, it's the procedure of the algorithm adopted in this paper, where RR is defined as follows.

Definition 1. REVERSE REACHABLE (RR) SET.
Given $G = (V, E, w)$, a random RR set R_j is generated from G by following three conditions:
1) selecting a random node $v \in V$
2) generating a sample graph g from G
3) returning R_j as the set of nodes can reach v in g.

Algorithm 2 RIS Framework(G, k)

```
Input: G: a social network; k : number of seeds;
Output: S: the seed set with size k;
1: $\mathcal{R} \leftarrow \emptyset; S \leftarrow \emptyset$
2: Estimate a sufficient sample size $\theta$
3: Generate RR set and insert into $\mathcal{R}$ until $|\mathcal{R}| \leftarrow \theta$
4: while $|S| < k$ do
5:     Identify node $v$ that covers the largest number of RR sets in $\mathcal{R}$
6:     $S \leftarrow S \cup \{v\}$
7:     Remove all the RR sets from $\mathcal{R}$ covered by $v$
8: end while
9: return $S$
```

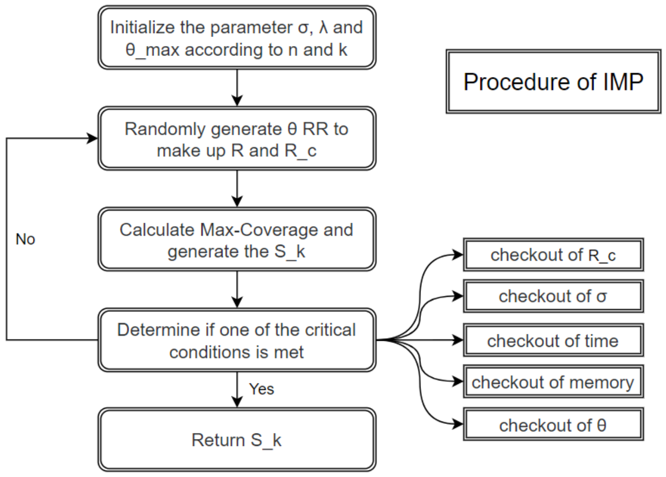

Initialize the parameter σ, λ and θ_max according to n and k
Procedure of IMP
Randomly generate θ RR to make up R and R_c
No
Calculate Max-Coverage and generate the S_k
checkout of R_c
checkout of σ
Determine if one of the critical conditions is met
checkout of time
checkout of memory
Yes
checkout of θ
Return S_k

Figure 1. The Model Design of D-SSA

For **ISE**, different diffusion models have different results for Monte Carlo simulation, and similarly for **IMP**, different diffusion models have different algorithms to obtain RR sets. Two common diffusion models are described below.

2.3.1. Diffusion model: IC. In this model, when a node u gets activated, initially or by another node, it has a single chance to activate each inactive neighbor v with the probability proportional to the edge weight $w(u, v)$. Afterwards, the activated nodes remain its active state but they have no contribution in later activations. When doing a back-propagation calculation, you can treat the process as a reverse ISE and randomly generate an RR set.

2.3.2. Diffusion model: LT. In this model, at the beginning, each node v selects a random threshold θ_v uniformly at random in range [0,1]. If round $t \geq 1$, an inactive node v becomes activated if $\sum_{\text{activated neighbors } u} w(u, v) \geq \theta_v$. When doing a back-propagation calculation, it can be reduced to randomly selecting one of the opposite edges for activation.

2.4. Detail of Algorithm

2.4.1. D-SSA Algorithm. The following is a basic D-SSA algorithm. In this project, I also added the control of time and space. If I use a third of the time limit at the end of a loop, I abort and extract the current optimal solution.

Algorithm 3 D-SSA Algorithm

```
Input: G: network; $0 \leq \sigma, \delta \leq 1$; k : number of seeds;
Output: $\hat{S}_k$: An optimal seed set with size k;
 1: $N_{\max} = 8\frac{1-1/e}{2+2\epsilon/3}\Upsilon(\epsilon, \frac{\delta}{6}/\binom{n}{k})\frac{n}{k}$
 2: $t_{\max} \leftarrow \lceil \log_2(2N_{\max}/\Upsilon(\epsilon, \frac{\delta}{3})) \rceil; t \leftarrow 0$
 3: $\Lambda \leftarrow \Upsilon(\epsilon, \frac{\delta}{3t_{\max}}); \Lambda_1 \leftarrow 1 + (1+\epsilon)\Upsilon(\epsilon, \frac{\delta}{3t_{\max}})$
 4: repeat
 5:     $t \leftarrow t + 1$
 6:     $\mathcal{R}_t \leftarrow \{R_1, \ldots, R_{\Lambda 2^{t-1}}\}$
 7:     $\mathcal{R}_t^c \leftarrow \{R_{\Lambda 2^{t-1}+1}, \ldots, R_{\Lambda 2^t}\}$
 8:     $< \hat{S}_k, \hat{\mathbb{I}}_t(\hat{S}_k) > \leftarrow$ Max-Coverage $(\mathcal{R}_t, k)$
 9:     if $\mathrm{Cov}_{\mathcal{R}_t^c}(\hat{S}_k) \geq \Lambda_1$ then
10:         $\mathbb{I}_t^c(\hat{S}_k) \leftarrow \mathrm{Cov}_{\mathcal{R}_t^c}(\hat{S}_k) \cdot n/|\mathcal{R}_t^c|$
11:         $\epsilon_1 \leftarrow \hat{\mathbb{I}}_t(\hat{S}_k)/\mathbb{I}_t^c(\hat{S}_k) - 1$
12:         $\epsilon_2 \leftarrow \epsilon\sqrt{\frac{n(1+\epsilon)}{2^{t-1}\mathbb{I}_t^c(\hat{S}_k)}}$
13:         $\epsilon_3 \leftarrow \epsilon\sqrt{\frac{n(1+\epsilon)(1-1/e-\epsilon)}{(1+\epsilon/3)2_t^{t-1}{}^c(\hat{S}_k)}}$
14:         $\epsilon_t \leftarrow (\epsilon_1 + \epsilon_2 + \epsilon_1\epsilon_2)(1 - 1/e - \epsilon) + (1 - \frac{1}{e})\epsilon_3$
15:         if $\epsilon_t \leq \epsilon$ then
16:             return $\hat{S}_k$
17:         end if
18:     end if
19: until $|\mathcal{R}_t| \geq N_{\max}$;
20: return $\hat{S}_k$
```

2.4.2. Max-Coverage procedure. It is a standard greedy algorithm used to find $\hat{S}_k$. The algorithm repeatedly selects node v with maximum gain, the number of RR sets that are covered by v but not the previously selected nodes.

Algorithm 4 Max-Coverage procedure

```
Input: $\mathcal{R}$ : RR$sets$; $k$ : number of seeds; $n$ nodes;
Output: $\hat{S}_k$ and its estimated influence $\hat{\mathbb{I}}_{\mathcal{R}}(\hat{S}_k)$;
 1: $\hat{S}_k \leftarrow \emptyset$
 2: while $|\hat{S}_k| < k$ do
 3:     $\hat{v} \leftarrow \arg\max_{\{v \in V\}}(\mathrm{Cov}_{\mathcal{R}}(\hat{S}_k \cup \{v\}) - \mathrm{Cov}_{\mathcal{R}}(\hat{S}_k))$
 4:     $\hat{S}_k \leftarrow \hat{S}_k \cup \{v\}$
 5: end while
 6: return $< \hat{S}_k, \mathrm{Cov}_{\mathcal{R}}(\hat{S}_k) \cdot n/|\mathcal{R}| >$
```

3. Empirical Verification

3.1. Dataset

I just use one real-world networks NetHEPT which frequently used in the **IMP** to test my model. NetHEPT is a given network and the numbers of nodes and directed edges in the graph are 15K and 62K respectively.

3.2. Performance measure

For **ISE**, I used the platform to test and measure the performance of my algorithm by comparing leaderboard data (results and elapsed time)

TABLE 3. HYPERPARAMETERS

Parameter	Value	Explanation
core	8	the number of cores used
ϵ	0.001	optimal $(1 - 1/e - \epsilon)$
δ	0.1	probability at least $(1 - \delta)$
mintime	3	Automatically stop when the time is greater than the mintime of the time limit

TABLE 4. RESULT OF NETHEPT

Seed	Time (s)	IC_Result	LT_Result
1	1	21.193	9.330
1	5	92.009	101.597
1	25	92.009	101.670
1	125	92.009	101.670
1	625	92.009	101.670
10	1	92.578	134.032
10	5	126.165	289.682
10	25	312.70	633.746
10	125	509.850	634.443
10	625	509.850	634.443
100	1	0	0
100	5	487.957	709.233
100	25	1880.941	2475.336
100	125	1887.900	2481.369
100	625	1888.915	2481.713
1000	1	0	0
1000	5	0	0
1000	25	2551.230	3244.643
1000	125	6165.648	7644.465
1000	625	7631.074	7801.76
1000	1250	7631.074	7801.23

For **IMP**, I still use the data from the rankings for comparison. At the same time, the influence value can be calculated by substituting the seed generated by the program into the **ISE** program.

The test environment of my laptop is as follows:

CPU: i7-10710U 1.10GHz
Core: 12 (8 threads in the program)
RAM: 16.0GB

3.3. Hyperparameters

The hyperparameters selection of this project is mainly the default parameter or given value, and the mintime is the result obtained based on multiple attempts to commit the task. When the value is reduced, the program will have the risk of timeout, and when the value is increased, the program's effect will decline. Finally, the hyperparameters trained are shown in the Table 3.

3.4. Experimental results

The experimental results are in the Table 4 and the Figure 2. We can see that with the increase of seed number, the time to find a stable solution also tends to increase approximately linearly.

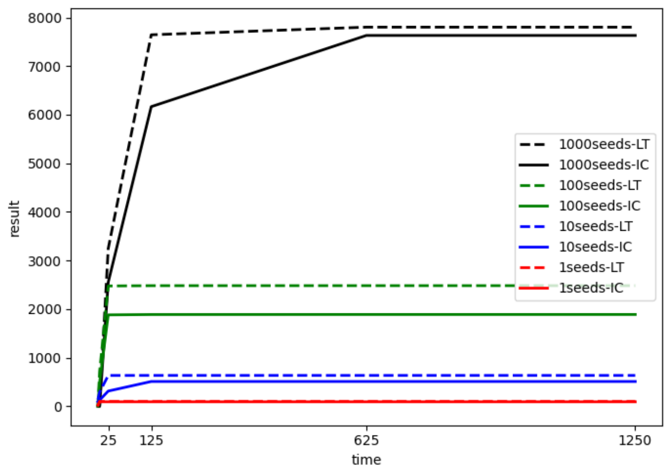
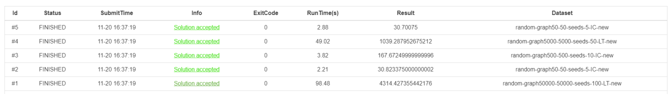


Figure 2. The Result of NetHEPT

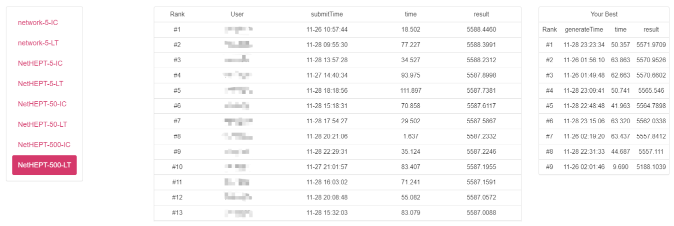
3.5. Conclusion

Based on the **ISE** test results and the results on the **IMP** leaderboard, I did this task well in the first part and passed all the tests as shown in the Figure 3, but in the second part, my program did not have a good result. In the test of 500 seeds, it was far from the top of the list as shown in the Figure 4. Below I summarize some of the advantages and disadvantages of the algorithm I used.



Id	Status	SubmitTime	Info	ExitCode	RunTime(s)	Result	Dataset
#5	FINISHED	11-20 16:37:19	Solution accepted	0	2.88	30.70075	random-graph50-50-seeds-5-IC-new
#4	FINISHED	11-20 16:37:19	Solution accepted	0	49.02	1039.287952675212	random-graph5000-5000-seeds-50-LT-new
#3	FINISHED	11-20 16:37:19	Solution accepted	0	3.82	167.67249999999996	random-graph500-500-seeds-10-IC-new
#2	FINISHED	11-20 16:37:19	Solution accepted	0	2.21	30.823375000000002	random-graph50-50-seeds-5-IC-new
#1	FINISHED	11-20 16:37:19	Solution accepted	0	98.48	4314.427355442176	random-graph50000-50000-seeds-100-LT-new

Figure 3. The Result of ISE



Rank	User	submitTime	time	result
#1	[redacted]	11-26 10:57:44	18.502	5588.4460
#2	[redacted]	11-28 09:55:30	77.227	5588.3991
#3	[redacted]	11-28 13:57:28	34.527	5588.2312
#4	[redacted]	11-27 14:40:34	93.975	5587.8998
#5	[redacted]	11-28 18:18:56	111.897	5587.7381
#6	[redacted]	11-28 15:18:31	70.858	5587.6117
#7	[redacted]	11-28 17:54:27	29.502	5587.5867
#8	[redacted]	11-28 20:21:06	1.637	5587.2332
#9	[redacted]	11-28 22:29:31	35.124	5587.2246
#10	[redacted]	11-27 21:01:57	83.407	5587.1955
#11	[redacted]	11-28 16:03:02	71.241	5587.1591
#12	[redacted]	11-28 20:08:48	55.082	5587.0572
#13	[redacted]	11-28 15:32:03	83.079	5587.0088

Your Best			
Rank	generateTime	time	result
#1	11-28 23:23:34	50.357	5571.9709
#2	11-26 01:56:10	63.863	5570.9526
#3	11-26 01:49:48	62.663	5570.6602
#4	11-28 23:09:41	50.741	5565.546
#5	11-28 22:48:48	41.963	5564.7898
#6	11-28 23:15:06	63.320	5562.0338
#7	11-26 02:19:20	63.437	5557.8412
#8	11-28 22:31:33	44.687	5557.111
#9	11-26 02:01:46	9.690	5188.1039

Figure 4. The Result of IMP

3.5.1. Advantage. The D-SSA algorithm is proved with rigorously bounded solutions and low time complexities.

This method requires less memory and low computational overhead.

3.5.2. Disadvantage. The actual efficiency of the D-SSA algorithm may be worse than that of the proxy-based approach because it needs to ensure the approximate ratio in the worst case

The algorithm does not make full use of monotonicity and submodularity

3.5.3. Future. There is a big shortage in the operation results this time, but Under the premise of the balanced development of various subjects, I have tried my best. In the next project, it is very important to prepare and set the plan in advance. The second is to determine the algorithm to be used and understand the advantages and disadvantages of the algorithm as well as the difficulties.

Acknowledgments

First of all, I would like to express my gratitude to our Computer Science and Technology department and Teacher Ke TANG, Xiaofen LU and Zhao YAO.

I am also deeply indebted to a lot of previous research can be found on the Internet, which really raise me up.

References

[1] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 137–146.

[2] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 695–710.

[3] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2014, pp. 946–957.

[4] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 57–66.

[5] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 61–70.

[6] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 75–86.

[7] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1852–1872, 2018.

[8] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 420–429.

[9] A. Goyal, W. Lu, and L. V. Lakshmanan, "Celf++ optimizing the greedy algorithm for influence maximization in social networks," in *Proceedings of the 20th international conference companion on World wide web*, 2011, pp. 47–48.

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

[11] X. Wang, Y. Zhang, W. Zhang, X. Lin, and C. Chen, "Bring order into the samples: A novel scalable method for influence maximization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 2, pp. 243–256, 2016.