IMP Project

Yilin Zheng 11510506
Department of Computer Science and Engineering
Southern University of Science and Technology
Email: 11510506@mail.sustc.edu.cn

1. Preliminaries

1.1. Software and Library

This project is written in Python using editor VSCode with python interpreter 2.7.14. The libraries being used includes argparse, random, collections, time, etc.

1.2. Algorithm

Algorithms used in this project includes greedy strategy, heuristic search and some computing cost saving tricks like computing if necessary.

2. Methodology

2.1. Introduction

This project is to solve IM(influence maximization) problem with two models estimating the results. The IM problem, which si abbreviated as IMP, is first introduced as an algorithmic problem by Domingos and Richardson [1] to studied for marketing products, though they said that was "Black Art". After that, Kempe, Kleinberg, and Tardos abstracted [2] this problem as a formal problem. A social network is modelled as a graph with vertices representing individuals and edges representing connections or relationship between two individuals. Influence are propagated in the network according to a given model. Given a social network graph, for a specific influence cascade model, and a small number k as seeds, the influence maximization problem is to find k vertices in the graph (referred to as seeds) such that under the influence cascade model, the expected number of vertices influenced by the k seeds (referred as the active nodes in the graph) is the largest possible.

2.2. Problem

For this problem, we defined the objective as following: Define a function$f(\cdot)$, which is *submodular* if f satisfies a natural "diminishing returns" property: the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding the same element to a superset of S.

Formally, for selected node v:

$$f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$$

$\forall v$ and $\forall$ pairs of sets $S \in T$. Our goal is to maximize $f(S)$ with certain amount of seeds S.

2.3. Model and Algorithm

There are two basic diffusion models [2] which are used for evaluation of seeds selected according to our IMP algorithms:

- *Independent Cascade(IC)*
- *Linear Threshold(LT)*

Four algorithms are implemented in this source codes and only **CELF** algorithm is used to output the results under the model chosen when running:

- *General Greedy* [1]
- *CELF* [3]
- *DegreeDiscount* [1]
- *MaxDegree*(Self-defined)

2.4. Representation

Representation used in the code includes nested dict by using *defaultdict* from *collections* module, primitive data type *list* and *dict*. The details are as following:

- **network_file**: file object opened from given absolute path read from command line
- **network**: network data stored in nested dict
- **seeds**: seed set stored in list
- **seed_size**: size of seeds predefined in command line
- **nodes**: node set stored in list
- **edge_num**: edge number
- **model**: evaluation model, either IC or LT
- **Dv**: outdegree of node v, used in *DegreeDiscount* algorithm

Some global variables as parameters can be adjusted by hand:

- **p**: parameters used in *DegreeDiscount* algorithm
- **R**: iterations of estimating seeds generated by certain algorithm

2.5. Architecture

The architectures both in **ISE.py** and **IMP.py** are almost the same excluding the finding process of seeds which evolves with IMP algorithms. They both contains file reading function **read_file** with slightly difference, model functions for *IC* and *LT* models, default evaluation function **default_evaluation** with iteration $R = 10000$ and command line function **command_line** for command line arguments passing and assignment. Here only specified the difference in **IMP.py**(functions only exist in IMP.py):

- *MaxDegree*: Select k-element nodes with maximal degree.
- *DegreeDiscount*: Heuristic method combined with greedy strategy.
- *GeneralGreedy*: Greedy based selection strategy method choosing the node with maximal reward when entering the active node set.
- *CELF*: Optimized GreneralGreedy method without repeating the evaluation of reward for large amount of nodes using *submodular* property.
- *LazyForward*: Process of *CELF*.
- *delta_R*: Process of *CELF*.
- *CELF2*: *CELF* re-implementation under own understanding.

Next section will explain the details of some algorithms with clear pseudocode.

2.6. Detail of Algorithm

- ***MaxDegree***: Choose k-element nodes with maximal degree as seeds.

Algorithm 1 MaxDegree

```
Input: network data $network$, node set $nodes$, size of seeds $seed\_size$
Output: a seed set $S$
1: $S \leftarrow \emptyset$
2: for each vertex $v \in nodes$ do
3:    calculate outdegree $d_v$
4: end for
5: sort $d$ in descending order
6: $S \leftarrow$ choose $k$ nodes with maximal outdegree
7: return $S$
```

- ***DegreeDiscount***: Idea comes from following: let v be a neighbour of vertex u. If u has been selected as a seed, then when considering selecting v as a new seed based on its degree, we should not count the edge vu towards its degree. Thus we should discount v's degree by one due to the presence of u in the seed set, and we do the same discount on v's degree for every neighbour of v that is already in the seed set. [4] This algorithm conducting a discount on degree so as to optimize the running time with performance matching the greedy method.

Algorithm 2 DegreeDiscount

```
Input: network data $network$, node set $nodes$, size of seeds $seed\_size$
Output: a seed set $S$
1: $S \leftarrow \emptyset$
2: for each vertex $v \in nodes$ do
3:    calculate its degree $d_v$
4:    $dd_v \leftarrow d_v$
5:    $t_v \leftarrow 0$
6: end for
7: for $i = 1$ to $seed\_size$ do
8:    $u \leftarrow$ argmax$\{dd_v | v \in V \backslash S\}$
9:    $S \leftarrow S \cup \{v\}$
10:   for each neighbour $v$ of $u$ and $v \in V \backslash S$ do
11:      $t_v \leftarrow$ active neighbours of $v \cap$ active neighbours of $u$ //modified to used in directed graph
12:      $dd_v \leftarrow d_v - 2t_v = (d_v - t_v)t_v p$
13:   end for
14: end for
15: return $S$
```

- ***GeneralGreedy***: Access every nodes each iteration after selecting a node as seed, then select the node with maximal reward as new seed. This algorithm is a basic algorithm we can be aware of and it returns the optimal empirical solutions so far.

Algorithm 3 GeneralGreedy

```
Input: network data $network$, node set $nodes$, size of seeds $seed\_size$
Output: a seed set $S$
1: $S \leftarrow \emptyset$
2: $R \leftarrow 20000$ //iterations for acceessing nodes under certain model
3: for $i = 1$ to $seed\_size$ do
4:    for each vertex $v \in V \backslash S$ do
5:       $s_v \leftarrow 0$
6:       for $i = 1$ to $R$ do
7:          $s_v += |RanCas(S \cup \{v\})|$ //RanCas is the access model
8:       end for
9:       $s_v \leftarrow s_v / R$
10:   end for
11:   $S \leftarrow S \cup \{\text{argmax}_{v \in V \backslash S}\{s_v\}\}$
12: end for
13: return $S$
```

Algorithms for *CELF*:

- ***delta_R***: Calculate the reward after selecting v as seed.

Algorithm 4 delta_R

```
Input: network data $network$, node $v$
Output: reward $delta_v$
1: $delta_v \leftarrow RanCas(S \cup \{v\}) - RanCas(S)$
2: return $delta_v$
```

- ***LazyForward***: Optimized greedy algorithm using the property of *submodular* with only re-evaluate the top node with largest reward. This algorithm can be used under **uniform cost(UC)** case or **cost benefit ratio(CB)** case.

Algorithm 5 LazyForward

```
Input: network data $network$, node set $nodes$, size of seeds $seed\_size$, compute type $type$
Output: a seed set $S$
 1: $S \leftarrow \emptyset$
 2: for each vertex $v \in nodes$ do
 3:   $\delta_v \leftarrow +\infty$
 4:   $cur_v \leftarrow false$
 5: end for
 6: while true do
 7:   if $type = UC$ then
 8:     $v^* \leftarrow \text{argmax}_{v \in V, c(S \cup \{v\}) \le seed\_size} \delta_v$
 9:   else if $type = CB$ then
10:     $v^* \leftarrow \text{argmax}_{v \in V, c(S \cup \{v\}) \le seed\_size} \frac{\delta_v}{c(S)}$
11:   end if
12:   if $cur_v$ then
13:     $S \leftarrow S \cup \{v^*\}$
14:     break
15:   else
16:     $\delta_v \leftarrow delta\_R(S, v)$
17:     $cur_v \leftarrow true$
18:   end if
19: end while
20: return $S$
```

- ***CELF***: Consider the maximal result for *LazyForward* under **UC** and **CB** and add the node yielding the result.

Algorithm 6 CELF

```
Input: network data $network$, node set $nodes$, size of seeds $seed\_size$
Output: a seed set
 1: $S_{UC} \leftarrow LazyForward(network, nodes, seed\_size, UC)$
 2: $S_{CB} \leftarrow LazyForward(network, nodes, seed\_size, BC)$
 3: return agrmax$\{R(S_{UC}), R(S_{CB})\}$ //return the set with maximal result
```

- ***CELF2***: Choose the top node with greatest reward each evaluation.

Algorithm 7 CELF2

```
Input: network data $network$, node set $nodes$, size of seeds $seed\_size$
Output: a seed set $S$
 1: $S \leftarrow \emptyset$
 2: $R \leftarrow 1000$
 3: while $c(S) < seed\_size$ do
 4:   if $S \neq \emptyset$ then
 5:     $v \leftarrow \text{argmax}_{v \in V \setminus S}\{R(v)\}$
 6:     $s_v \leftarrow 0$
 7:     for $i = 1$ to $R$ do
 8:       $s_v \leftarrow s_v + RanCas(S \cup \{v\}) - RanCas(S)$
 9:     end for
10:     $s_v = s_v / R$
11:     $new\_v \leftarrow \text{argmax}_{v \in V \setminus S}\{R(v)\}$
12:     if $new\_v = v$ then
13:       $S \leftarrow S \cup v$
14:       $s.remove(v)$
15:     else
16:       continue
17:     end if
18:   else
19:     for each vertex $v \in V$ do
20:       $s_v \leftarrow RanCas(\{v\})$
21:     end for
22:     $S \leftarrow S \cup$ argmax$_{v \in V \setminus S}\{R(v)\}$
23:     $s.remove(v)$
24:   end if
25: end while
26: return $S$
```

3. Empirical Verification

Empirical verification is compared between these algorithms. Note the result output acquiescently is computed by *CELF2*. The following table shows the results of these algorithms with seed size 4 and 10, parameters $p = 0.001$, $R = 1000$, command line argument keep the random seed 10. Data set being used it the given network data with 62 nodes and 159 edges.

TABLE 1. RESULT OF ALGORITHMS

model	IC		LT	
seed size	4	10	4	10
MaxDegree	21.4458	31.964	24.1294	39.2762
DegreeDiscount	21.4541	31.9626	24.1085	39.2421
GeneralGreedy	**27.2694**	**42.8165**	**31.8258**	**54.8086**
CELF	23.5466	37.9507	27.0816	47.3416
CELF2	**27.2997**	**42.7756**	**32.8008**	**54.7855**

From the table, we can see that *CELF2* gives as good results as *GeneralGreedy* gives, since *CELF2* just use *submodular* property to lower the computations without losing the quality of outcome.

3.1. Design

The IMP is NP-hard and its computation is #P under *IC* model and *LT* model. In this project, we just test for a given small network data so it might need further test on larger network data, but we still can see the difference performance between various algorithms. We design the algorithm using both greedy strategy and heuristic method, and some algorithm has a tendency for combination of both motivations. Of course, the design of algorithms should also considers the difference of models being estimated. And a significant thing is usually the result activated by same seeds under *LT* models are better than that under *IC* model, which cause our consider of the accuracy of these two basic diffusion models, though we cannot arbitrarily say which model is more closed to real network in our physical world. In summary, the model, the strategy, and the size of data are all the factor when designing IMP algorithms.

3.2. Data and data structure

Data used in this project includes a file of small network data with 62 nodes and 159 edges, a seeds files including 4 seeds. Data in the source code includes list, dict. Data structure used by the algorithms mostly is queue.

3.3. Performance

TABLE 2. RUNNING TIME

Algorithm	Complexity
MaxDegree	$O(n)$
GeneralGreedy	$O(knRm)$
CELF	$O(knR)$
DegreeDiscount	$O(klogn + m)$

3.4. Result

The empirical results can be referenced by TABLE 1. Here we just analysis the discovery from the experiments: *LT* model can get better result compared with *IC* model with the same seeds and heuristic way can faster the computing with matching result to greedy strategy. However, *CELF* and *GeneralGreedy* can get the best selections we know so far.

3.5. Analysis

This project is to solving an abstract problem originally generated from a marketing problem. To be studied as an algorithmic problem, this problem is closely related to *submodular* property and we can use this property to optimized the original greedy algorithm without lower the quality of result. The basic models are important reason for algorithm design and optimization direction.

Acknowledgments

I would like to thank Yulian Mao who help me go in deep in understanding the heuristic algorithm and modified the algorithm originally applied to undirected graph to simply applied to direct graph. And I also want to thanks Miss. Hong who help explain the expressions in the paper [4].

References

[1] P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the 7th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5766, 2001.

[2] D. Kempe, J. M. Kleinberg, and . Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 137146, 2003.

[3] 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* - KDD 07, 2007.

[4] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2009.