Influence Maximization Problem

Project 3 Report of

CS303 Artificial intelligence

Department of Computer Science and Engineering

BY

Kebin Sun

11410151

Southern University of Science and Technology

December 2018

Table of Content

1 Preliminaries

A social network is the graph of relationships and interactions within a group of individuals that plays a fundamental role as a medium for the spread of information, ideas, and influence among its members. The *Influence Maximization Problem* (IMP) asks that, for a given graph, to what k-node seed set the influence spread is maximized. This problem has applications in viral marketing, where a company may wish to spread the rumor of a new product via the most influential individuals in popular social networks.[1]

In this project, we are specifically interested in the IMP of directed graph under two propagation model – *Independent Cascaded* (IC, often called *Weighted Cascaded* – WC in recent published works) and *Linear Threshold* (LT). However, the optimization problem of selecting the most influential nodes is NP-hard. Even the calculation of mean influence spread is #P-hard. Hence, exact methods are often not practical.[2] As a result, the heuristic approaches such as Degree method and Degree Discount method [3], as well as greedy approaches such as CELF [2] are often applied instead.

1.1 Software & Hardware

This project is written in *Python* ( ) with editor *Visual Studio Code* ( ). The main testing platform is *Windows 10 Professional Edition* (version 1809) with Intel® Core™ i7-8700K @ 3.70~4.70GHz of 6 cores and 12 threads.

1.2 Algorithms

This project involves the published research of CELF [2], pure greedy [1] and exact influence spread calculation extended on [4]. The Memetic Algorithm (MA) is additionally used in some cases as well. Some ideas analogizing the Degree Discount method [3] is applied for large graphs instead.

2 Methodology

In this part, I will introduce the representations of the IMP the applied approaches, the structure of my program as well as and the detailed algorithms.

2.1 Mathematics and Representations

2.1.1 General Descriptions

The directed IMP we concern can be described as follows: a graph $G = (V, E)$, with a set of vertices denoted by V, a set of (directed) edges denoted by E is given. Each node v can only be one of two states – activated (i.e. $act(v) = 1$) and inactivated (i.e. $act(v) = 0$). And each edge (u, v) has a weight $w(u, v) = 1/d_{in}(v)$ where $d_{in}(v)$ is the in-degree of node v. The influence spread of a seed set $S \subseteq V$, denoted as $\sigma(S)$ is evaluated depending on the choice of diffusion models – IC and LT.

The IC propagation model calculate influence spread as follow:

1) When a node u gets activated, initially in seed set S or by another node, it has a single chance to randomly activate each inactive neighbor v with the probability proportional to the edge weight $w(u, v)$;
2) Afterwards, the activated nodes remain its active state but they have no contribution in later activations;
3) The iteration stops when there is no node activated in the last iteration.

The LT propagation model calculate influence spread as follow:

1) At the beginning, each node v is assigned with a random threshold $\theta_v \in (0,1)$ and the seed set nodes are set to activated.
2) At each iteration, node v is activated if $\sum_{\text{activated parents } u} w(u, v) \geq \theta_v$;
3) The iteration stops when there is no node activated in the last iteration.

Given a positive integer k, the IMP aims to find the seed set $S \subseteq V$ that maximize the expected influence spread

$$\overline{\sigma}(S) \equiv \mathbb{E}[\sigma(S)] = \lim_{n\to\infty} \sum_{i=1}^{n} \sigma(S)\big/n \tag{2.1}$$

subjected to $|S| = k$.

Overall, the mathematical representation of IMP is:

$$\begin{aligned} &\arg\min_{S} \overline{\sigma}(S) \equiv \lim_{n\to\infty} \frac{\sum_{i=1}^{n} \sigma(S)}{n} \\ &\text{s.t. : } |S| = k. \end{aligned} \tag{2.2}$$

2.1.2 Data Structure of the Program

The main data structure in the program is the graph of `Graph` class which is a node representation graph with `nodes` – a instance of `dict` class. A value of `nodes` is an instance of `Node` class consists the node name `name`, the super node weight `weight`, the parents `parents` and the successors `successors`. Also, in case that the input graph has loop(s) of single node(s) which the above date structure cannot handle, an additional `selfLoopNodes` list is implemented to store the name of these nodes for further usages. Furthermore, a list of previous parents' activation probabilities `lastAffected` is used for evaluating the expected influence spread $\overline{\sigma}(S)$ for DAG.

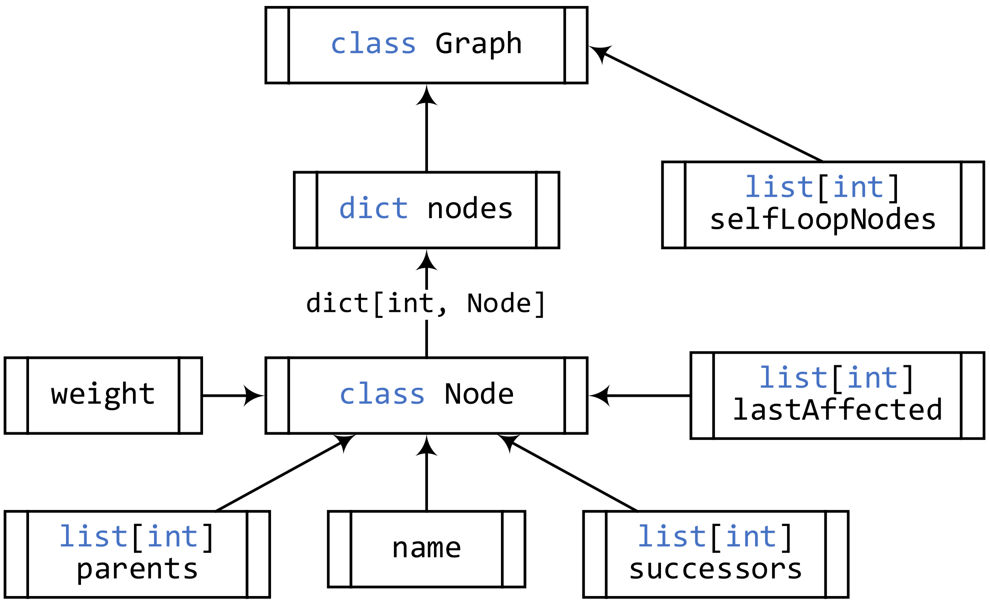


Figure 1 The data structure of the program

2.1.3 Mathematical Results for DAG

As stated above, the evaluation of the expected influence spread $\overline{\sigma}(S)$ for an arbitrary directed graph is a #P-hard problem. However, if the provided graph was a directed acyclic graph (i.e. DAG), the estimation of expected influence spread $\overline{\sigma}(S)$ would be quite simple since $\overline{\sigma}(S)$ of both IC and LT model of DAG have exact and fast solution. The exact solution of LT model of DAG was proposed by [4]. The proof used in [4] is simple but based on an important result in [1], therefore the comprehension may be difficult. Hence, I will first find out the exact $\overline{\sigma}(S)$ of IC model in DAG and then using analogues to obtain the same result of LT model as [4].

2.1.3.1 Expected influence spread of IC model in DAG

Noticing that each node has only one chance of activation its successors in the IC model, the result of the activation of the nodes being sequencing influence immediately comes to us. This implies that the when multiple nodes, say a, b, etc. want to activate a common successor, say u, the influence does not come at the same time but with an arbitrary order.

Theorem: The order of influence by parents' $\{u_1, u_2, \dots, u_n\}$ does not affect the final activation probability of node t $ap(t)$.

Proof: Let the activation probabilities of parents be $\{ap(u_1), \dots, ap(u_n)\}$, the activation probabilities of parents when visiting t last time be $\{ap^0(u_1), \dots, ap^0(u_n)\}$ and the original activation probability of node t be $ap^0(t)$. It is obvious that the weight $w(u_i, t) \equiv 1/d_{in}(t)$.

First, consider the influence order of $\{1,2,\dots,n\}$, the new activation probability $ap(t)$ is derived by (where Pr is the probability)

$$ap^{i+1}(t) = \Pr(act(t) = 1) + \Pr(act(t) = 0) \cdot \Pr(act(u_i) = 1 | act(t) = 0) w(u_i, t); \tag{2.3}$$

$$\begin{aligned} ap^1(t) &= ap^0(t) + [1 - ap^0(t)][ap(u_1) - ap^0(u_1)]/d_{in}(t)\,, \\ ap^2(t) &= ap^1(t) + [1 - ap^1(t)][ap(u_2) - ap^0(u_2)]/d_{in}(t)\,, \\ &\vdots \\ ap(t) = ap^n(t) &= ap^{n-1}(t) + [1 - ap^{n-1}(t)][ap(u_n) - ap^0(u_n)]/d_{in}(t)\,. \end{aligned} \tag{2.4}$$

To show that any order leads to the same result, only the swap law between any consecutive ap is needed which can be proved easily:

$$\begin{aligned} ap^{i+1}_{k,j} &= ap^i + (1 - ap^i) p_j / d_{in} \\ &= ap^{i-1} + (1 - ap^{i-1}) p_k / d_{in} + \{1 - [ap^{i-1} + (1 - ap^{i-1}) p_k / d_{in}]\} p_j / d_{in} \\ &= ap^{i-1} + \frac{1}{d_{in}} [(1 - ap^{i-1}) p_k + (1 - ap^{i-1}) p_j] + \frac{1}{d_{in}^2} [(1 - ap^{i-1}) p_j p_k]. \end{aligned} \tag{2.5}$$

This is clearly irrelevant of the order of k & j, hence $ap^{i+1}_{k,j} = ap^{i+1}_{j,k}$.

Then, using the mathematical induction, the theorem can be proved.

Q.E.D.

During the proof of the theorem, we can see that for a DAG is lacking of loop, the activation of parent $act(u_i)$ is independent of its successors' activation $act(t)$, therefore $\Pr(act(u_i) = 1 | act(t) = 0) = \Pr(act(u_i) = 1)$. For a general graph, if there was(were) path from t to u_i, this simplification would fail and becomes quite hard (#P-hard) to evaluate. In this program, the iteration approach (2.4) is implemented.

2.1.3.2 Expected influence spread of LT model in DAG

Unlike the IC model where each node can only try to activate its successors once, the LT model implies that, in a way, each node will try 'infinite' times to activated its successors. More precisely, as long as the threshold θ of a node t is smaller than its activated parents' number times their weights, t is activated.

$$\left[act(t) \overset{?}{=} 1\right] = \theta(t) \overset{?}{\leq} \sum_{u \in N_{in}(t)} act(u)w(u,t) = \sum_{u \in N_{in}(t)} act(u)/d_{in}(t) \tag{2.6}$$

where $N_{in}(t)$ represents the in-neighbor of t (a.k.a. parents of t).

Again, we should rewrite this equation into the probability form. At first, the distribution of $\theta(t)$ is clearly a unit uniform distribution, i.e. $\theta(t) \sim \mathcal{U}(0,1)$. By multiplying $d_{in}(t)$ on both side and taking the fact that $\sum_{u \in N_{in}(t)} act(u)$ can only be an integer varies from 0 to $d_{in}(t)$ into account, the equivalent form of (2.6) can be obtained:

$$\begin{aligned} \Pr(act(t)=1) &= \Pr\left[\tilde{\theta}(t) \leq \sum_{u \in N_{in}(t)} act(u)\right], \qquad \tilde{\theta}(t) \sim \mathcal{U}\big(0, d_{in}(t)\big) \\ &= \Pr\left[\mathcal{U}\big(0, d_{in}(t)\big) \leq \sum_{u \in N_{in}(t)} \Pr(act(u)=1 | act(t)=0)\right] \\ &= \sum_{u \in N_{in}(t)} \Pr(act(u)=1|act(t)=0)/d_{in}(t). \end{aligned} \tag{2.7}$$

The same as before, $\Pr(act(u)=1|act(t)=0) = \Pr(act(u)=1) \equiv ap(u)$ for a DAG. Consequently,

$$ap(t) = \sum_{u \in N_{in}(t)} ap(u)/d_{in}(t). \tag{2.8}$$

This result is even simpler than the one of IC model.

2.1.3.3 Applications of the results for DAG

Although these exact expected influence spread evaluations are obtained when $n_{\text{loop}} = 0$, they can be used not only in the DAG but also when the n_{loop} of the graph is small or the length of loops are relatively long. Even if the graph has loops whose between-loop connections are weak, I have got an exact solution as well. However, this solution requires much more computational cost and is not implemented in this program. I use the simple DAG results as fast yet accurate heuristics for general graphs instead.

2.2 Architecture

2.2.1 Flow Chart of Whole Program

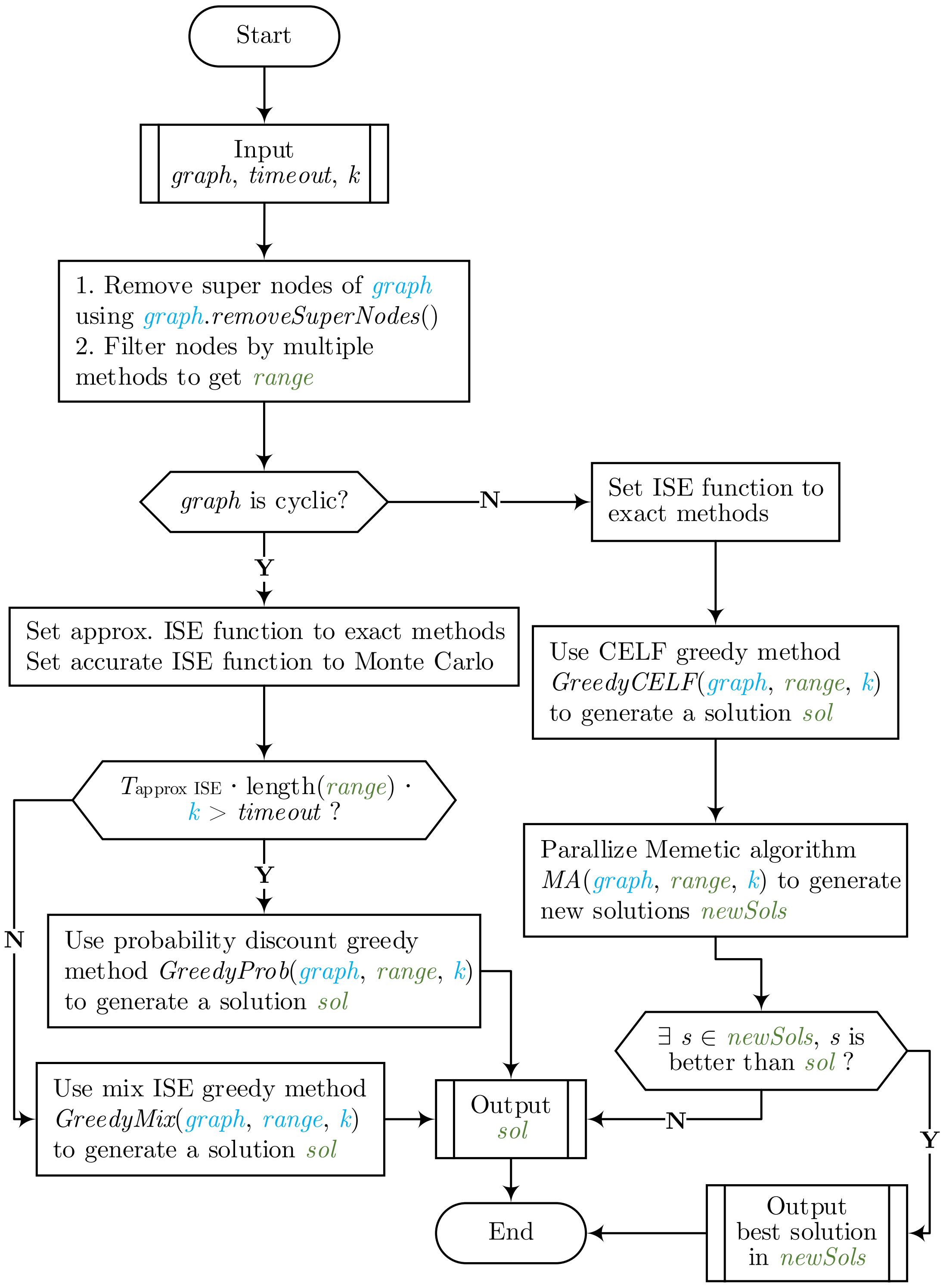

Start
Input
graph, timeout, k
1. Remove super nodes of graph using graph.removeSuperNodes()
2. Filter nodes by multiple methods to get range
graph is cyclic?
N
Set ISE function to exact methods
Y
Set approx. ISE function to exact methods
Set accurate ISE function to Monte Carlo
Use CELF greedy method GreedyCELF(graph, range, k) to generate a solution sol
T approx ISE · length(range) · k > timeout ?
Y
N
Parallize Memetic algorithm MA(graph, range, k) to generate new solutions newSols
Use probability discount greedy method GreedyProb(graph, range, k) to generate a solution sol
∃ s ∈ newSols, s is better than sol ?
Use mix ISE greedy method GreedyMix(graph, range, k) to generate a solution sol
Output
sol
N
Y
End
Output
best solution
in newSols

Figure 2 The flow chart of the whole program

2.2.2 Removing Super Nodes

What is a 'super node'? Noticing that in both diffusion models, when a node, say v, has only one parent (a.k.a. in-neighbor), say p, its weight from the parent $w(p, v) = 1/d_{in}(v) = 1$, which implies that whenever p is activated, v will be activated immediately in the following iteration. Also, v cannot be activated if p is not. This same-activation phenomenon informs us that they can be regarded as one super node without affecting the IMP result at all.

The procedure of removing super nodes is quite simple:

1) Iterating all nodes in graph and let current node be v;
2) If $d_{in}(v) = 1$, remove v from graph and assign the successors of v to the parent p, and the super node weight $w_{sn}(p) \leftarrow w_{sn}(p) + w_{sn}(v)$;
3) The iteration stops when the number of nodes in the graph converges.

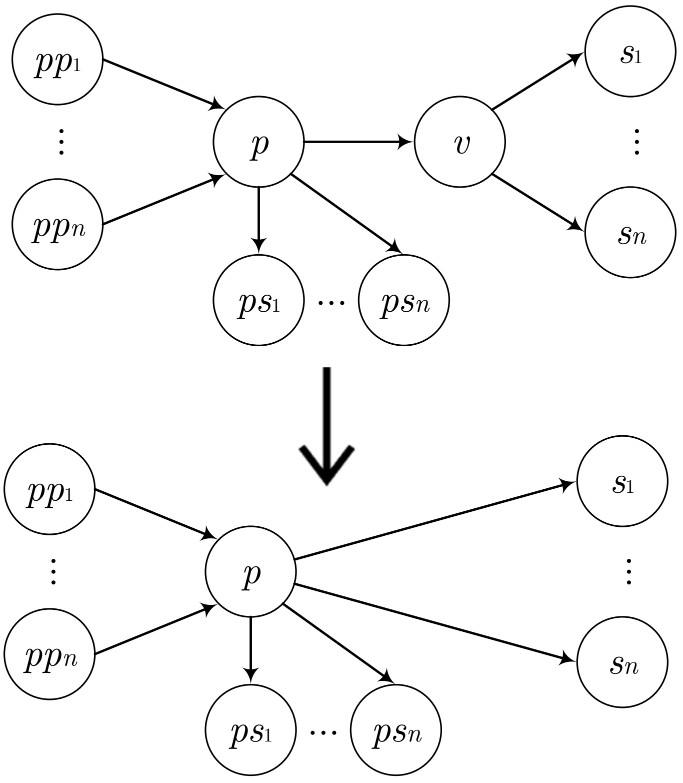


Figure 3 The sketch of removing super node

2.2.3 Multiple Node Filters

In order to shrink the size of candidate nodes where the algorithms find seed set in without reducing the quality of the potential solution, three distinct filters are used – the super node removing mentioned above, the connect component filter and the probability degree filter.

At first, when the super nodes are removed from graph, the in-degree of any node must be greater than or equal to 2 (or just 0) and the number of nodes in the graph will typically be reduced a lot – the node count of *NetHEPT* graph decreases from 15233 to 10648, as an instance.

Then, the connect components of graph is calculated using the *graph.GetComponents*() method whose pseudo code will be shown later. The components whose total weight of nodes is smaller than largest one's than $4k$ times will be deleted from graph. Since components are not connected, this deletion would be quite simple. For example, the node count of *NetHEPT* graph decreases furtherly from 10648 to 5850.

Last but not least, the sum of weights of out-edges of each node

$$d_{\text{prob}}(v) = \sum_{u \in N_{out}(v)} w(v,u) \tag{2.9}$$

is calculated and the candidate nodes are sorted according its d_{prob}. Nodes with $d_{\text{prob}} < \lambda$ will be removed from the candidate list. For $\lambda = 0.05$, the candidate node count of *NetHEPT* graph finally decreases from 5850 to 2868.

2.3 Details of Algorithms

2.3.1 Implementations

In this program, both exact and Monte Carlo ISE for both IC and LT models are implemented, they are `ISE_IC_Exact`, `ISE_IC_MC`, `ISE_LT_Exact` and `ISE_LT_MC`. Additionally, for all IMP methods (`GreedyCELF`, `MA`, `GreedyProb` and `GreedyMix`), parallelization are implemented to achieve higher speed which will not be shown in the following pseudo code.

2.3.2 Pseudo Code

- **ISE_IC_Exact**: IC model ISE using equation (2.4).

```
Function ISE_IC_Exact ($graph$, $seeds$, $threshold = 10^{-5}$)
1   For $s \in seeds$ do
2       $ap(graph[s]) \leftarrow 1$
3   End for
4   While previous TotalActivation($graph$) − the value now $\geq$ $threshold$ do
5       For each node $v$ in $graph$ do
6           Use equation (2.4) to update $ap(v)$ where $u_i \in N_{in}(v)$
            [for $t$ multiple edges of $(u_i, v)$, $u_i$ will occur $t$ times]
7       End for
8   End while
9   Return TotalActivation($graph$) as ISE
```

- **ISE_IC_MC**: IC model ISE using Monte Carlo approach.

```
Function ISE_IC_MC ($graph$, $seeds$)
10  For $s \in seeds$ do
11      $ap(graph[s]) \leftarrow 1$
12  End for
13  $activated \leftarrow seeds$
14  While $activated \neq \emptyset$ do
15      $newActivated \leftarrow \emptyset$
16      For $v \in activated$ do
17          For $s \in N_{out}(v)$ where $ap(s) = 0$ do
18              If Random(0,1) $< 1/d_{in}(s)$ Then
19                  $ap(s) \leftarrow 1$; $newActivated \leftarrow newActivated \cup \{s\}$
20              End if
21          End for
22      End for
23      $activated \leftarrow newActivated$
24  End while
25  Return TotalActivation($graph$) as ISE
```

- **ISE_LT_Exact**: LT model ISE using equation (2.8).

```
Function ISE_LT_Exact ($graph$, $seeds$, $threshold = 10^{-5}$)
26  For $s \in seeds$ do
27      $ap(graph[s]) \leftarrow 1$
28  End for
29  While previous TotalActivation($graph$) $-$ the value now $\geq$ $threshold$ do
30      For each node $v$ in $graph$ do
31          Use equation (2.8) to update $ap(v)$
32      End for
33  End while
34  Return TotalActivation($graph$) as ISE
```

- **TotalActivation**: Calculate the influence spread of a graph.

```
Function TotalActivation ($graph$)
35  Let $graph = G(V, E)$
36  $result \leftarrow 0$
37  For $v \in V$ do
38      $result \leftarrow ap(v) \cdot w_{sn}(v)$
39  End for
40  Return $result$
```

- **ISE_LT_MC**: LT model ISE using Monte Carlo approach.

```
Function ISE_IC_MC ($graph$, $seeds$)
For $s \in seeds$ do
    $ap(graph[s]) \leftarrow 1$
End for
$thresholds \leftarrow$ generate Random(0, 1) with count the same as nodes in $graph$
$activated \leftarrow seeds$
While $activated \neq \emptyset$ do
    $newActivated \leftarrow \emptyset$
    For each node $v$ in $graph$ do
        If $\sum_{p \in N_{in}(v)} ap(p)/d_{in}(v) \geq thresholds[v]$ Then
            $ap(v) \leftarrow 1$; $newActivated \leftarrow newActivated \cup \{v\}$
        End if
    End for
    $activated \leftarrow newActivated$
End while
Return TotalActivation($graph$) as ISE
```

- **MA**: Memetic algorithm approach of IMP.

```
Function MA ($graph$, $seedNumber$, $searchRange$, $FunctionISE$,
            $P_{ls} = 0.2$, $popSize = 60$)
$pop \leftarrow$ random select length $seedNumber$ from $searchRange$ for $popSize$ times
While stopping criterion is not met do
    $pop_t \leftarrow pop$
    For $i = 1$ to $opsize$ do
        $child \leftarrow$ Crossover($pop$)
        If Random(0,1) $< P_{ls}$ Then
            $child \leftarrow$ LocalSearch($child$)
        End if
        If $child \notin pop_t$ Then
            $pop_t \leftarrow pop_t \cup \{child\}$
        End if
        $pop \leftarrow$ Sort($pop_t$)$[\![0{:}\,opsize]\!]$
    End for
End while
Return the best feasible solution in $pop$
```

- **GreedyCELF**: CELF approach of IMP.

```
Function GreedyCELF ($graph$, $seedNumber$, $searchRange$, $FunctionISE$)
72  $output \leftarrow \emptyset$
73  $influenceMap \leftarrow \{r \mapsto FunctionISE(graph, \{r\})$ For $r \in searchRange\}$
74  $output \leftarrow output \cup \left\{\arg\max_k influenceMap[k]\right\}$
75  $previousInfuence \leftarrow$ pop $output[\![-1]\!]$ from $influenceMap$
76  For $i = 1$ to $seedNumber - 1$ do
77      $maxIncrease \leftarrow -\infty$; $best \leftarrow$ None
78      For $p \mapsto influence$ in sorted $influenceMap$ do
79          If $influence \leq maxIncrease$ Then Break
80          If $FunctionISE(graph, p)$ is better than $maxIncrease$ Then
81              $best \leftarrow p$; $maxIncrease \leftarrow influence$
82          End if
83      End for
84      $output \leftarrow output \cup \{best\}$
85      pop $output[\![-1]\!]$ from $influenceMap$
86      $previousInfuence \leftarrow previousInfuence + maxIncrease$
87  End for
88  Return $output$ as IMP result
```

- **GreedyProb**: Probability degree discount approach of IMP.

```
Function GreedyProb ($graph$, $seedNumber$, $searchRange$, $FunctionISE$)
89  $output \leftarrow \emptyset$
90  $degrees \leftarrow \left\{r \mapsto \sum_{u \in N_{out}(r)} w(r,u) \cdot w_{sn}(u)\ \text{For}\ r \in searchRange\right\}$
91  For $i = 1$ to $seedNumber$ do
92      $candidates \leftarrow$ NLargest($\boldsymbol{n}$, $degrees$) where $\boldsymbol{n}$ is determined by timeout
93      $output \leftarrow output \cup \left\{\arg\max_{k \in candidates} FunctionISE(graph, k)\right\}$
94      $successors \leftarrow N_{out}(output[\![-1]\!])$
95      temporally reduce all $successors$' $w_{sn}(u)$ to $[1 - 1/d_{in}(u)] \cdot w_{sn}(u)$
96      recalculate the $degrees$ of nodes $\{v | v \in N_{in}(s), s \in successors\}$ by line 90
97      change back all $successors$' $w_{sn}(u)$
98      For $s \in successors$ do $degrees[s] \leftarrow [1 - 1/d_{in}(u)] \cdot degrees[s]$
99  End for
100 Return $output$ as IMP result
```

- **GreedyMix**: This approach of IMP is basically the same as the **GreedyProb** method. When the equation-based ISEs are not accurate and using **GreedyCELF** with *FunctionISE* being the MC ISE causing large time cost, the basic ISE is set to equation-based ISE and furtherly apply MC ISE after retrieving candidates.

3 Performances & Time Control

In this part, performance test of this program as well as the time control approach will be implemented. As mentioned above, the testing platform is *Windows* 10 with Intel® Core™ i7-8700K @ 3.70~4.70GHz.

3.1 Theoretical Performance & Time Control

The time complexity of the MC ISE approach of one iteration is of course

$$\tilde{T}_{\text{MC ISE}} = \Theta(|E|) \tag{3.1}$$

where input graph $G = (V, E)$.

Therefore, to get a decent ISE, R iterations' average are needed:

$$T_{\text{MC ISE}} = \Theta(R|E|). \tag{3.2}$$

Hence, the greedy CELF approach of IMP have upper bound

$$T_{\text{CELF}} = O(kR|E| \cdot |N|) \tag{3.3}$$

where k is the seed set size.

For equation-based ISE calculation in acyclic graph, the time complexity is

$$T_{\text{equation-based ISE}} = \Theta(|E|). \tag{3.4}$$

However, when the input graph is cyclic with c loops, the time complexity becomes larger:

$$T_{\text{equation-based ISE}} = O(c|E|). \tag{3.5}$$

It is pretty hard to calculate the equation-based ISE method cost, therefore the parallel running of actual case before performing IMP is implemented. By this mean, both $T_{\text{MC ISE}}$ and $T_{\text{equation-based ISE}}$ can be obtained. The $\boldsymbol{n}$ in the **GreedyProb** function can be calculated by

$$\boldsymbol{n} = \left\lfloor \frac{T_{\text{timeout}}}{k \cdot T_{\text{equation-based ISE}}} \right\rfloor \text{ or } \left\lfloor \frac{T_{\text{timeout}}}{k \cdot T_{\text{MC ISE}}} \right\rfloor. \tag{3.6}$$

3.2 Empirical Performance

Since the provided sample graph is a small scale-free acyclic graph which cannot meet the variety requirement of empirical performance benchmark, algorithm-generated graphs are necessary.

A scale-free graph is a graph whose degree distribution follows a power law, at least asymptotically, i.e., randomly select a node v, its degree d follows

$$\Pr(d = k) \propto \frac{1}{k^\gamma}. \tag{3.7}$$

Many realistic networks are scale-free, e.g. Internet and social networks we concern in this project. Thus, generating graphs obey (3.7) is needed.

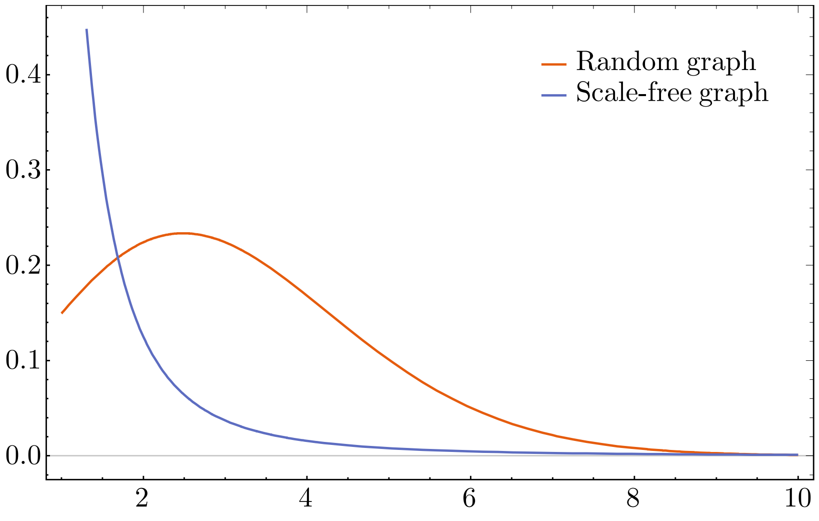


Figure 4 The degree distributions (probability vs. degree) of random graphs and scale-free graphs.

To generate graphs following this law, I choose the Barabási-Albert random graph which consecutively add nodes, each new node u will connect node v with probability

$$\Pr(u \leftrightarrow v) = \frac{d_v}{\sum_{i\in V} d_i}. \tag{3.8}$$

After generating graph using BA model, edges are randomly assigned directions with different probabilities to simulate distinct cyclic degrees of graphs. Since running greedy CELF approach by MC ISE using *Python* is not practical in large graphs, the generated graphs shown below have at most 200 nodes.

Graph name	*# of nodes*	*# of edges*	*# of reverse edges*	*Acyclic?*
BA1	200	397	171	No
BA2	200	594	303	No
BA3	200	397	0	Yes
BA4	200	397	41	No
BA5	150	297	0	Yes
BA6	100	197	0	Yes
network	62	159	0	Yes

Table 1 The name and the properties of generated graphs and provided sample

Using my program to test the performances relative to the CELF with MC ISE of 2000 iterations' average, the following diagrams are obtained. (These plots are arranged in the order of increasing graph complexity.)

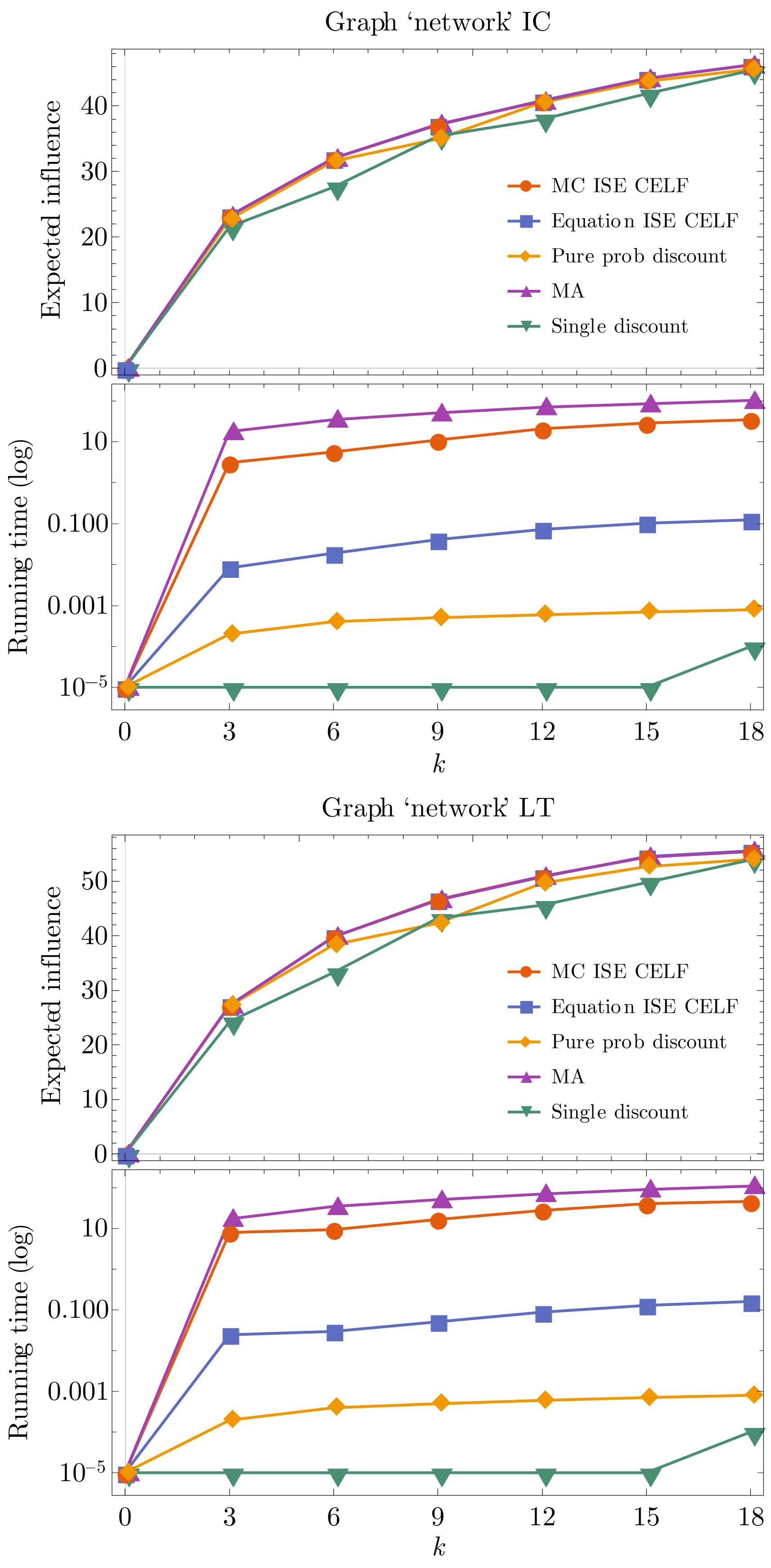


Figure 5 The benchmark result of graph 'network'

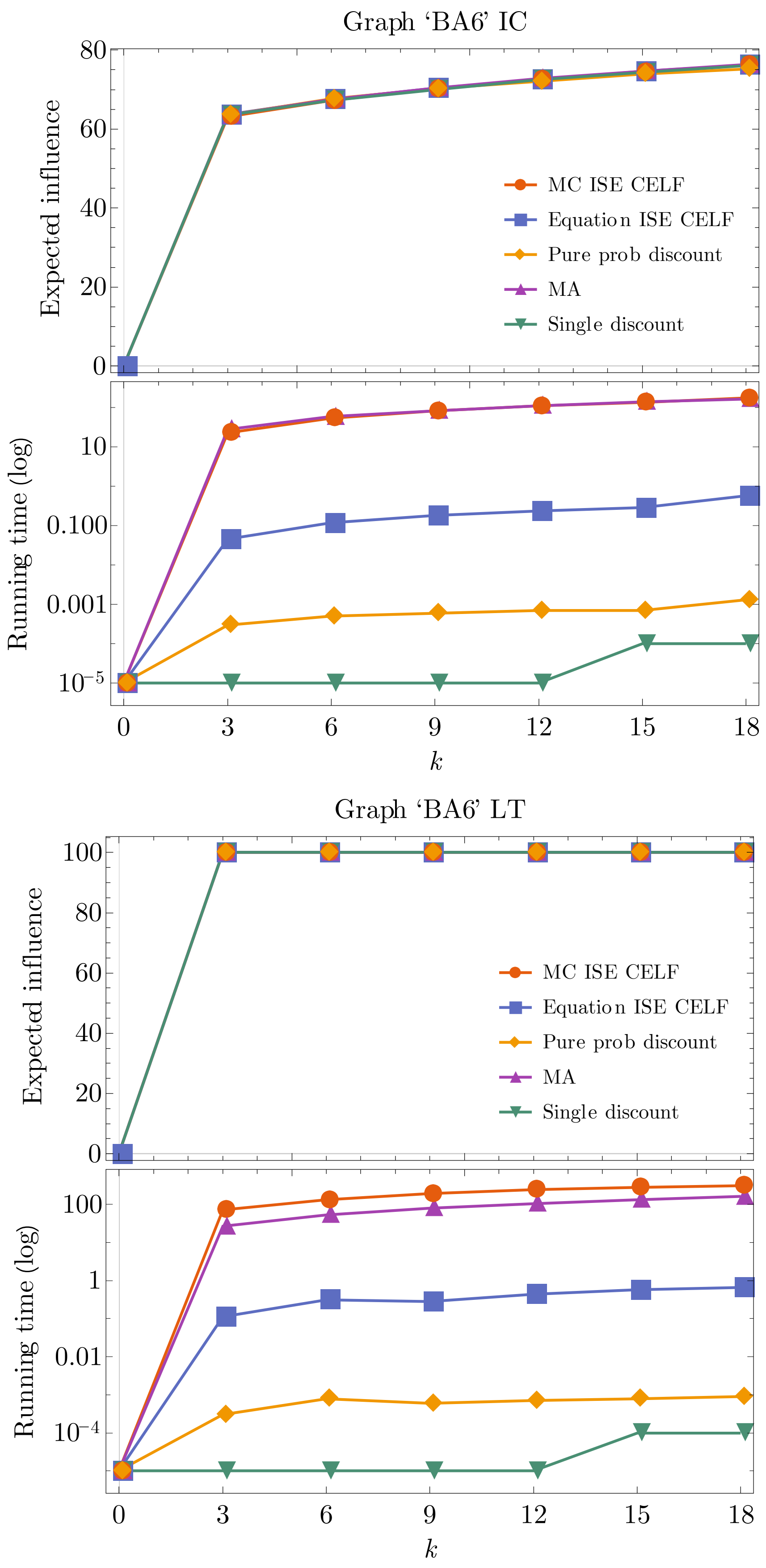


Figure 6 The benchmark result of graph 'BA6'

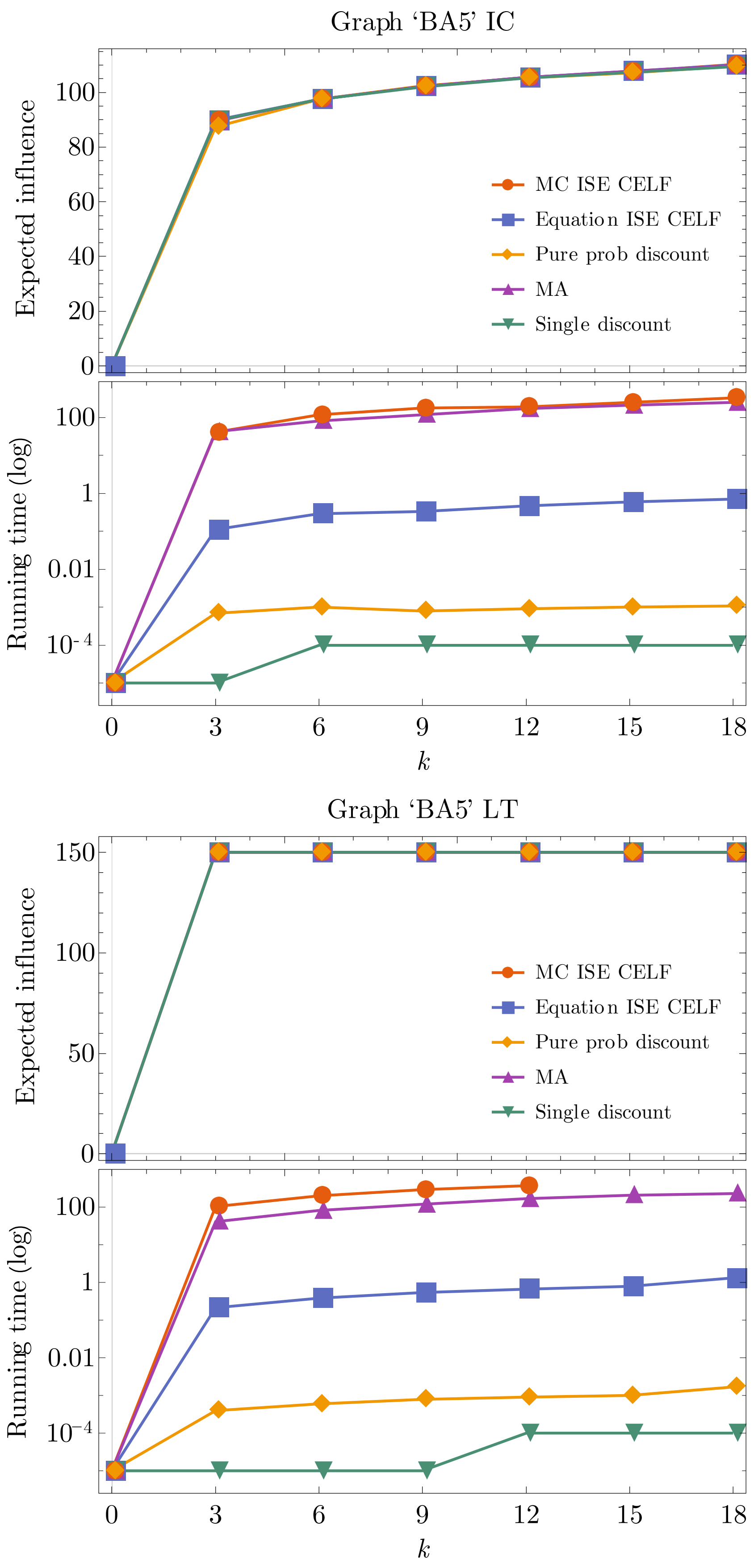


Figure 7 The benchmark result of graph ‘BA5’

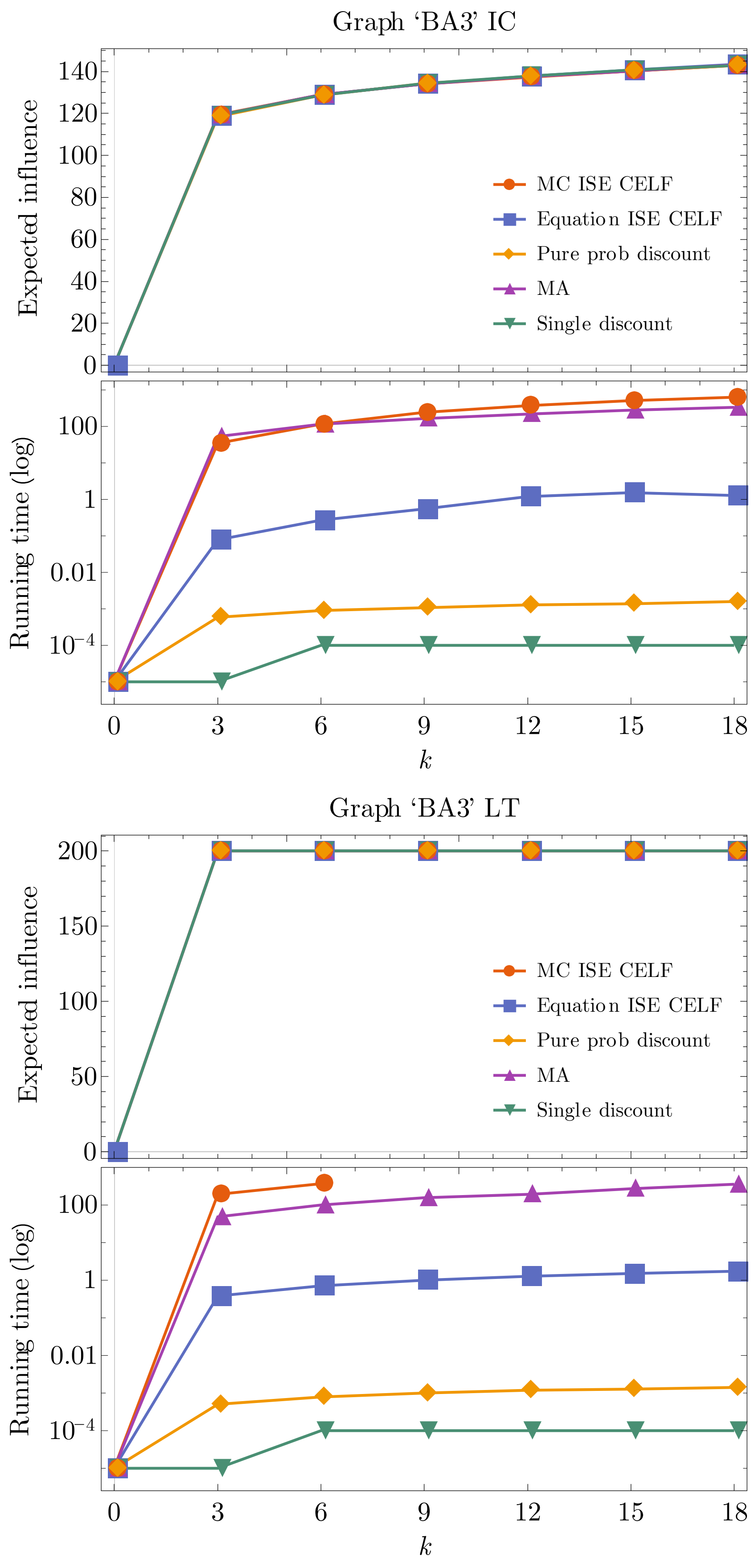


Figure 8 The benchmark result of graph 'BA3'

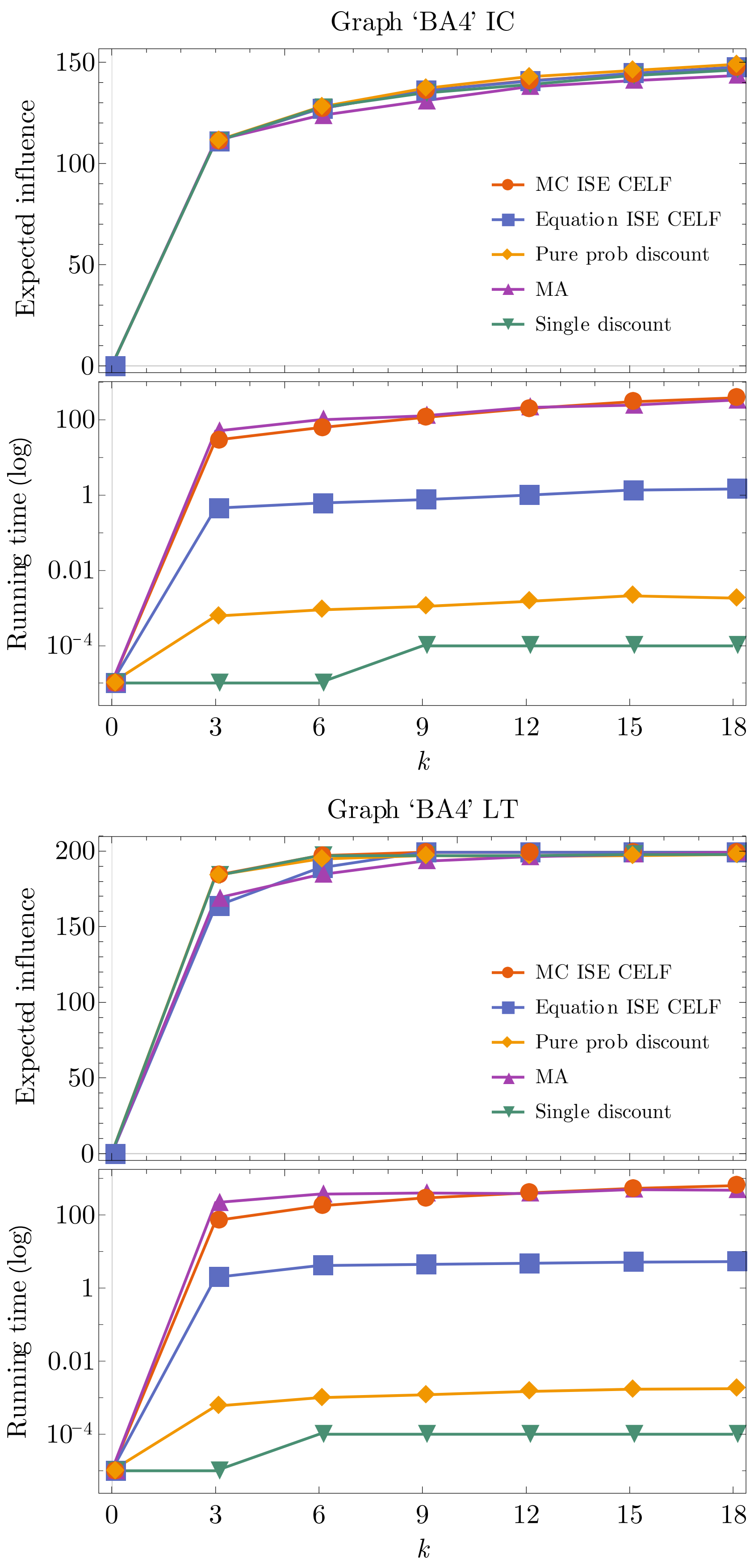


Figure 9 The benchmark result of graph 'BA4'

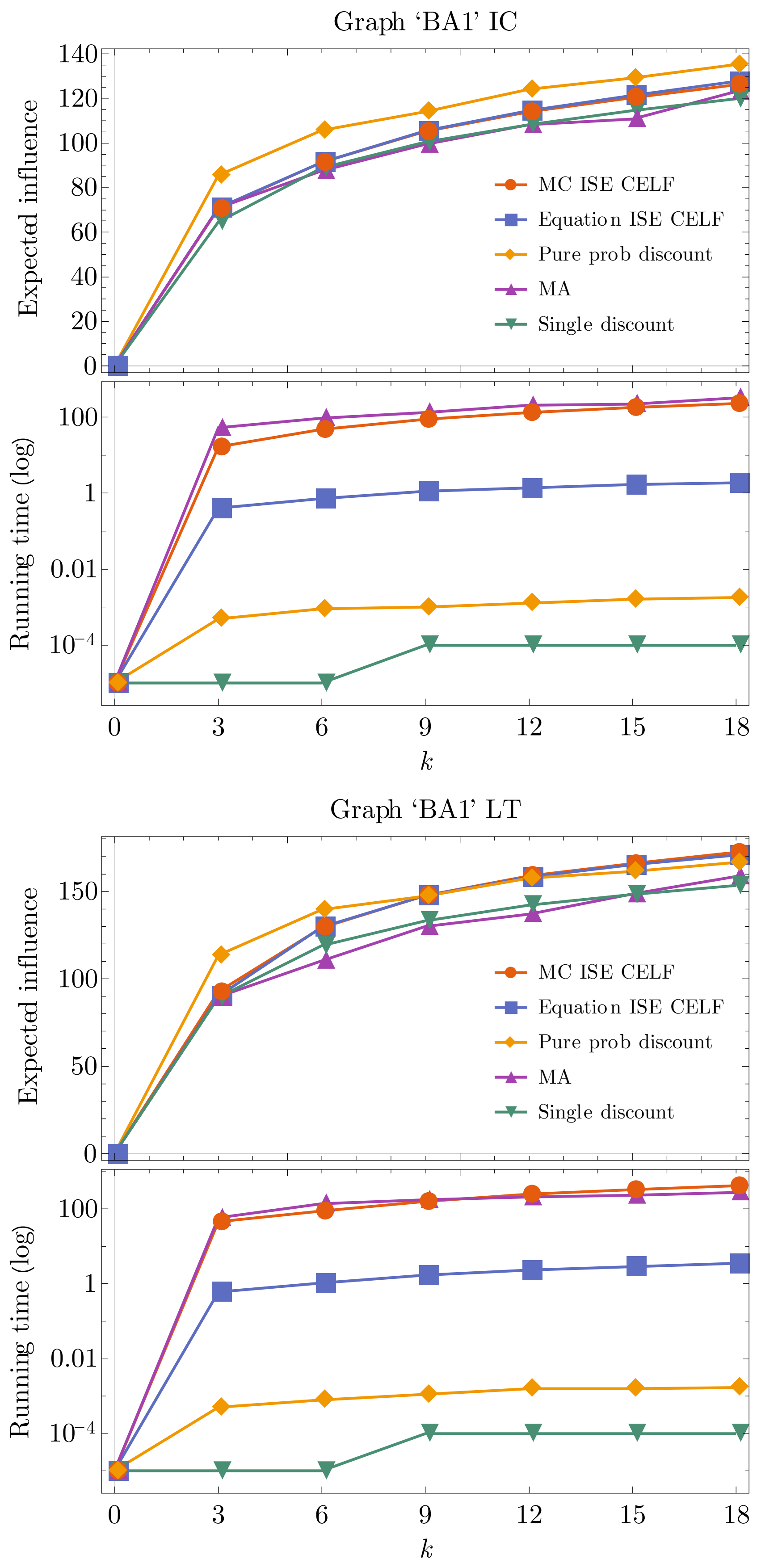
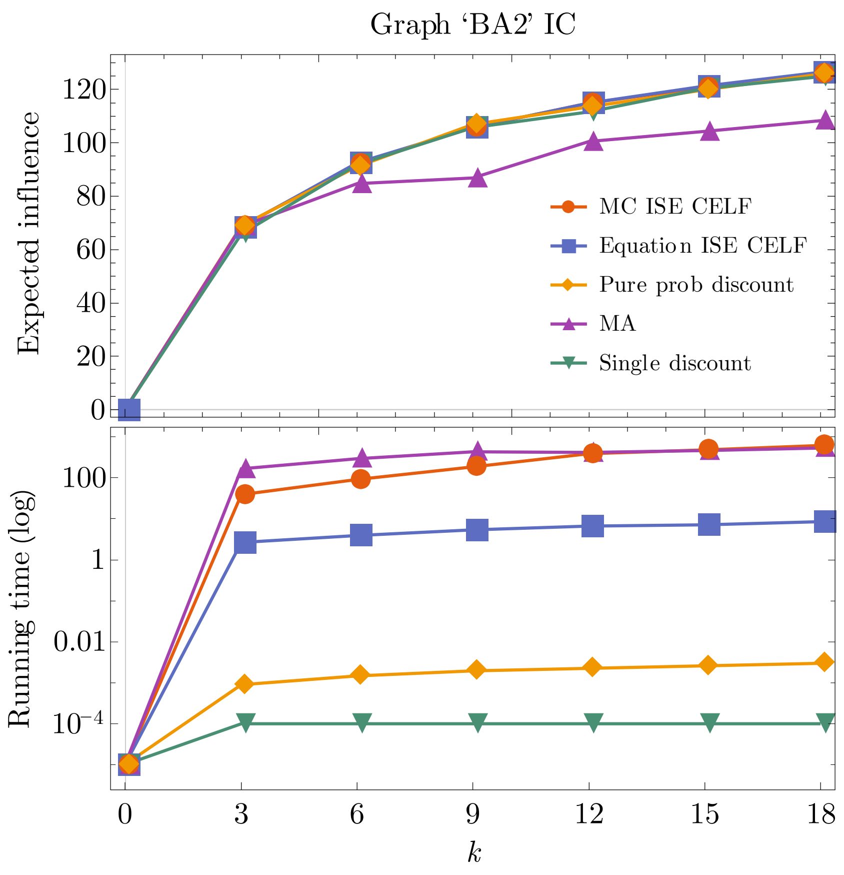


Figure 10 The benchmark result of graph ‘BA1’



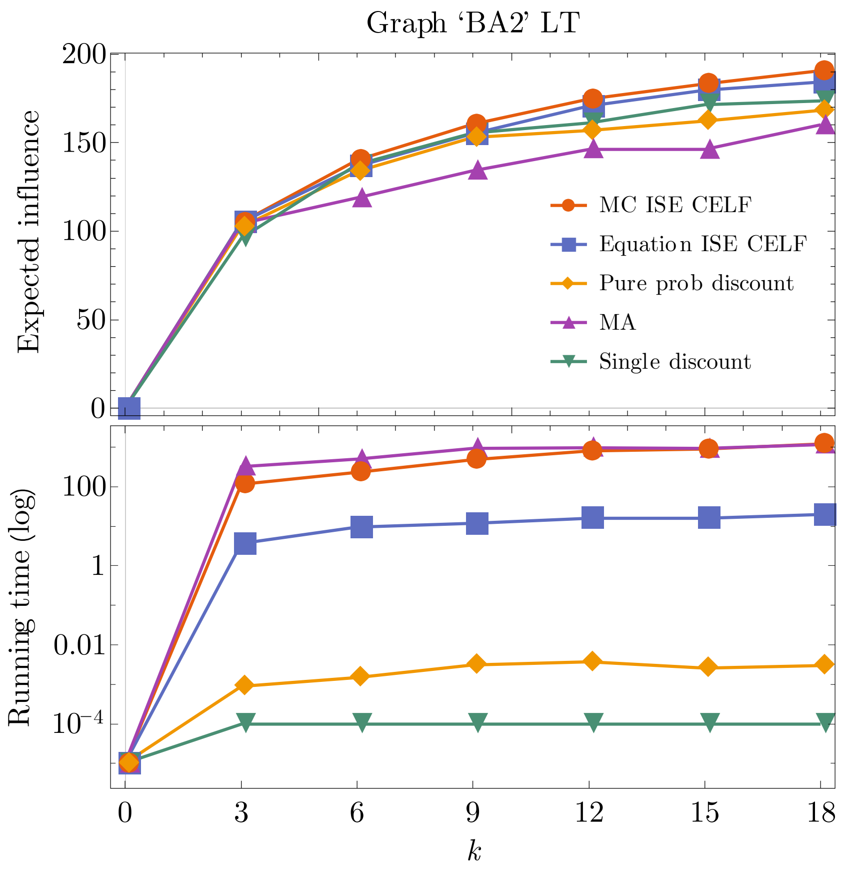


Figure 11 The benchmark result of graph ‘BA2’

3.3 Analysis

We can imply from these benchmarks that when the graph is acyclic and scale-free, the performances between methods become smaller as the graph size grows. However, for cyclic graphs, thing becomes much more complicated.

It is not hard to see that the Memetic Algorithm performs best overall in acyclic graphs: in most cases, MA returns the same result as CELF using MC ISE but with lower time cost; in other cases, MA even gets better result than CELF. Also, in the acyclic graphs, as proved before, the CELF using equation-based ISE equations performs exactly the same as MC ISE, however, the time cost is reduced by nearly three orders of magnitude. As for cyclic graphs, the CELF using equation-based ISE equations still obtain a balance between speed and quality. In this program, as mentioned before, I use the Probability Discount (PD) heuristic with extra nearest-neighbor weight discounting for large graphs. This approach performs almost the same as CELF using MC ISE in most cases and even outperforms all the rest algorithms. But it is clear that the time cost of PD is only one of a hundred thousand of the latter (CELF). Comparing with the Single Discount method in [3] which has significant poorer performances at larger seed set size, this altered Probability Discount heuristic together with parallel computing of equation-based ISE equations provides a fast yet accurate way of solving IMP for large networks.

Acknowledgment

I would like to thank TA *Zhao Yao* for her guidance during writing the program.

References

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

[2] Goyal, A., Lu, W., & Lakshmanan, L. V. (2011, March). Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web* (pp. 47-48). ACM.

[3] Chen, W., Wang, Y., & Yang, S. (2009, June). Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 199-208). ACM.

[4] Chen, W., Yuan, Y., & Zhang, L. (2010, December). Scalable influence maximization in social networks under the linear threshold model. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (pp. 88-97). IEEE.

[5] Onnela, J. -P.; Saramaki, J.; Hyvonen, J.; Szabo, G.; Lazer, D.; Kaski, K.; Kertesz, J.; Barabasi, A. -L. (2007). Structure and tie strengths in mobile communication networks. *Proceedings of the National Academy of Sciences.* 104 (18): 7332–7336.