Influence Maximization Project

Wang Zhiyuan 11610634
CSE
Computer Science and Technology
11610634@mail.sustc.edu.cn

1. Preliminaries

1.1. Software

In this project, I do some work to solve the Influence Maximization problem, which aid to find the initial seed set to let the final influence max in the social network.

To do this work, I do it in two part, one is ISE(Influence Size Estimator), one is IMP(Influence Maximization Problems). And the social network we use is made in two models: IC(Independent Cascade) and LT(Linear Threshold).

In the Project, I write the program by python and no extra package used. The test data is storaged in the txt.

1.2. Algorithm

The algorithm I have came true is CELF and IMM. The CELF is a greedy function with pruning, and the IMM is transfer the result of the active each round to the overlay of the RR(Reverse Reachable) set.

In my test, the CElF can always get the best value in the IC model, but for the property of the LT, CELF can't always get the best result of the LT. But the largest dsiadvantage of the CELF is it's speed, when I apply CELF to a social network with 15k nodes and 30k edges. The CELF will work more than 2 hours.

The IMM can work quickly and accuratly for both IC and LT. For the graph I mentioned last paragraph, IMM can work out in 20 seconds in both IC and LT for a 50 seeds initial set with $\epsilon = 0.1$, number of processing is 8. But IMM is not perfact, this algorithm will consume so much memory when handle a graph has so many nodes with a low ϵ. For example, when I calculate a network with 425k nodes and $\epsilon = 0.1$, it need 13.2GiB memory.

2. Methodology

2.1. Representation

2.1.1. ISE. In the ISE, I do it in 3 parts:

- $BuildMap$: Read data from the file and generate the Adjacency **list**. Then reason that I choose the two dimension list but not the matrix or the dictionary is to get a balance of the memory ad the speed.
- $CreateprocessingPool$: Create a processing pool to do the multiprocessing to calculate quickly.
- $DoICorLT$: Calculate the result of the network and seed given for many time.

2.1.2. CELF. In the CELF, I do it in 3 parts:

- $BuildMap$: Read data from the file and generate the Adjacency **list**. Then reason that I choose the two dimension list but not the matrix or the dictionary is to get a balance of the memory ad the speed.
- $DoISE$: Get the result of the seeds we choice in the given network and storage it in a **heap**
- $Choicenodes$: Choice that use the node in the top of the heap or do ISE again

2.1.3. IMM. In the IMM, I do it in 3 parts:

- $BuildMap$: Read data from the file and generate the Adjacency **list**. Then reason that I choose the two dimension list but not the matrix or the dictionary is to get a balance of the memory ad the speed.
- $Sampling$: Calculate the influence and create the **set** of RR set.
- $NodeSelction$: Get the initial nodes set by compare the number of the RR set covered by the node.

2.2. Architecture

2.2.1. ISE.

- Read data and storage in memory.
- Get the active seeds initial.
- Active the node by the nodes actived last round.
- Do the last step until there are no new node actived in one round

2.2.2. CELF.

- Read data and storage in memory.
- Calculate the influcence of the each nodes
- Choice that use the node on the top of the heap or Calculate the influence again
- Do the last step until the size of the set we get equal to the size we need

2.2.3. IMM.

- Read data and storage in memory.
- Do the sample and generate the list of the RR set
- Do node selection for the list generated last step, get the final set

2.3. Detail of Algorithm

2.3.1. ISE. In the ISE, I do 10000 times estimate in 8 processing, each do 1250 times calculations. In each calculation, the program have two parts: LT and IC, choice which parts by the parameter input. The detail of two part can look at algorithm1 and algorithm2.

algorithm 1 IC

```
function IC(nextNode, activeSet)
    activeNew ← activeSet.copy()
    while activeNew do
        activeTemp ← new set()
        for i in activeNew do
            for j in nextNode[i] do
                if random < j[1] then
                    if j[0] not in activeSet then
                        activeTemp.add[j[0]]
                        activeSet.add[j[0]]
                    end if
                end if
            end for
        end for
        activeNew = activeTemp.copy()
    end whilereturn activeSet
end function
```

2.3.2. CELF. For the CELF, I do once ISE for each node one time and storage them in a heap depend on the influcence increase of the node. Then, choice that using the node on the top of the heap or do the ISE for the nodes leftover. The detail of the CELF will be shown in the algorithm3.

2.3.3. IMM. For the IMM, it translate the ability of the activing nodes to the number of the RR set can be covered.

Defination 1 (Reverse Reachable Set). Let v be a node in network G, and g be a graph obtain by removing an each edge e in G with $1-p(e)$ probability. The reverse reachale(RR) set for v in g is the set of nodes in g that can reach v.(That is, for each node u in RR Set, there is a directed path from u to v)

Defination 2 (Random RR set). Let $\mathcal{G}$ be the distribution of g induced by the randomness in edge removal from G. A random RR set is an RR set generated on an instance of g randomly sampled from $\mathcal{G}$, for a node selected uniformly at random from g.

After the defination of the RR set and random RR set, how IMM work? By the paper Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency [] we

algorithm 2 LT

```
function LT(nodes, nextNode, activeSet)
    threshold ← []
    for i ← 0 tonodes do
        threshold.append(random())
    end for
    activeNew ← activeSet.copy()
    while activeNew do
        activeTemp ← new set()
        for i in activeNew do
            for j in nextNode do
                if j[0] not in activeSet then
                    threshold[j[0]]− = j[1]
                    if threshold[j[0]] <= 0 then
                        activeTemp.add(j[0]
                        activeSet.add(j[0]
                    end if
                end if
            end for
        end for
        activeNew ← activeTemp.copy()
    end whilereturn activeSet
end function
```

algorithm 3 CELF

```
function CELF(nodes, size)
    activeSet = set()
    que = PriorityQueue()
    for i in 0 to nodes do
        que.add(i, (ISE(i, activeSet)))
    end for
    activeSet.add(que.get()[0])
    while len(activeSet) < size do
        B ← que.get()
        if ISE(B[1], activeSet) < que.get()[1] then
            activeSet.add(B[0])
        else
            for i in 0 to nodes do
                if i not in activeSet then
                    que.add(i, (ISE(i, activeSet)))
                end if
            end for
        end if
    end while
    return activeSet
end function
```

can know that if we add many enough RR set in the list R, and select the nodes that can cover the most number of the RR set, we must can get a good enough solution. But how many RR set we need to add in it? This is the work that IMM do more then the algorithm called TIM in the paper Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency [] do. First, we set some parameter that we will use in the sampling of the RR set.

$$\lambda^{'} = \frac{(2 + \frac{2}{3}\epsilon^{'}) \cdot (\log(\binom{n}{k}) + l \cdot \log n + \log\log_2^n) \cdot n}{\epsilon^{'2}} \quad (1)$$

$$\lambda^* = 2n \cdot ((1 - \frac{1}{e}) \cdot \alpha + \beta)^2 \dot{\epsilon}^{-2} \quad (2)$$

$$\alpha = \sqrt{l \log n + log2} \quad (3)$$

$$\beta = \sqrt{(1 - \frac{1}{e}) \cdot (\log(\binom{n}{k}) + l \log n + \log 2)} \quad (4)$$

First of all, let's look at the algorithm the NodeSelection, in this part, we will get a given size set from the nodes in the G that let the nodes in the set can cover as more as possible. This part not only use in the last step to get the target set, but also use in the sampling part to determine the size of the RR set list we need. In the NodeSelection part,

algorithm 4 NodeSelection

1: **function** NODESELECTION($\mathcal{R}, k$)
2: $S_k^* \leftarrow set()$
3: **for** $i \leftarrow 1 \quad to \quad k$ **do**
4: $v \leftarrow Maximizes \quad num(RR \quad Set \quad Covered)$
5: $S_k^*.add(v)$
6: **end for**
7: **end function**
8: **return** S_k^*

how to let the speed of line 4 be highest is the key of the optimization of the algorithm. When I implment it, I use a **map** to storage many **list** as the **value**, each **list** storage the nodes that can be reached by the **key** node of this **list**. And at the same time, I count that the number of the nodes that can be reached for each **key** node Then, I will do the work that choice the node to add to the target set. I get the node that have the max number of the nodes can be reached, then update the map and the list that used to statistic last step. I will repeat this step until the target set is full.

Then, I will do the Sampling part work. The detail of the sampling is in the algorithm5

algorithm 5 Sampling

1: **function** SAMPLING(G, k, ϵ, l)
2: $\mathcal{R} \leftarrow []$
3: $LB \leftarrow 1$
4: $\epsilon^{'} \leftarrow \sqrt{2} \cdot \epsilon$
5: **for** $i \leftarrow 1 \quad to \quad \log_2 n - 1$ **do**
6: $x \leftarrow n/2^i$
7: $\Theta_i \leftarrow \lambda^{'}/x$
8: **while** $|\mathcal{R}| \leq \Theta_i$ **do**
9: $v \leftarrow random(G.node)$
10: $\mathcal{R}.append(v)$
11: **end while**
12: $S_i \leftarrow NodeSelection(\mathcal{R}, k)$
13: **if** $n \cdot \mathcal{F}_{\mathcal{R}}(S_i) \geq (1 + \epsilon^{'} \cdot x$ **then**
14: $LB \leftarrow n \cdot \mathcal{F}_{\mathcal{R}}(S_i)/(1 + \epsilon^{'}$
15: **break**
16: **end if**
17: **end for**
18: $\Theta \leftarrow \lambda^*/LB$
19: **while** $|\mathcal{R}| \leq \Theta$ **do**
20: $v \leftarrow random(G.node)$
21: $\mathcal{R}.append(v)$
22: **end while**
23: **return** $\mathcal{R}$
24: **end function**

3. Empirical Verification

References