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(Reserve Rsearch) 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($nextNode, activeSet$)
    $activeNew \leftarrow activeSet.copy()$
    while $activeNew$ do
        $activeTemp \leftarrow new \quad set()$
        for $i \quad in \quad activeNew$ do
            for $j \quad in \quad nextNode[i]$ do
                if $random < j[1]$ then
                    if $j[0] \quad not \quad in \quad activeSet$ 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.

3. Empirical Verification

References

algorithm 2 LT

```
function LT($nodes, nextNode, activeSet$)
    $threshold \leftarrow []$
    for $i \leftarrow 0 \quad tonodes$ do
        $threshold.append(random())$
    end for
    $activeNew \leftarrow activeSet.copy()$
    while activeNew do
        $activeTemp \leftarrow new \quad set()$
        for $i \quad in \quad activeNew$ do
            for $j \quad in \quad nextNode$ do
                if $j[0] \quad not \quad in \quad activeSet$ 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
        $activeNew \leftarrow activeTemp.copy()$
    end whilereturn $activeSet$
end function
```

algorithm 3 CELF

```
function CELF($nodes, size$)
    $activeSet = set()$
    $que = PriorityQueue()$
    for $i \quad in \quad 0 \quad to \quad nodes$ do
        $que.add(ISE(i, activeSet))$
    end for
end function
```