CS303 Project3: Solving Influence Maximization Problem Using the IMM Algorithm

Shijie Chen
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, Guangdong, China
Email: 11612028@mail.sustc.edu.cn

***Abstract*—The Influence Maximization Problem (IMP) has many real-world applications and is a NP-hard problem. In this project, I first developed a influence propagation model based on Linear Threshold (LT) and Independent Cascade (IC). Then, the IMM algorithm is used to solve the IMP problem. Computational experiments have shown that IMM can solve IMP problems very efficiently.**

I. Preliminaries

A. Notation

The following notations are used in this report:

TABLE I
REPRESENTATION

Name	Variable
The social network	$G = (V, E)$
Set of edges	E
Set of nodes	V
Influence weight of node i to node j	w_{ij}
Reversed reachable set	RR
A node in the social network	$node$
Incoming edges for a node	$incoming[]$
Outgoing edges for a node	$outgoing[]$

B. The Influence Maximization Problem

The influence maximization problem examines the process of the spread of influence is a social network G. The influence spread process is as follows:

- Initially, activate some nodes (seeds)
- Each node that is activated in last iteration (active nodes) stop try to activate its inactive neighbors according to a certain diffusion model.
- Repeat until no node is activated in an iteration.

Performance is given by the total number of nodes that is activated during the process.

C. Diffusion Model

Diffusion Model defines how influence spread in a social network. More specifically, it determines how an inactive node is activate by its active neighbors.

1) Linear Threshold: In the LT model, each node has a threshold that is assigned randomly. A node j is activated if the sum of influence weight of its active neighbors $\sum_i w_{ij}$ exceeds its threshold.

2) Independent Cascade: In the IC model, the process of each activated node trying to active its inactive neighbors are independent. An active node i actives its inactive neighbor j by probability w_{ij}

II. Methodology

A. Data Structure

Data structures used in this project are $list$ and set.

The social network is stored as adjacency lists in both directions to support large scale problems. Set is used to perform set operations like intersection.

B. Influence Spread Estimate (ISE)

As is described above, ISE is implemented by applying diffusion models on a social network G with a given initial seed set S.

Note that we just need to look at neighbors of newly activated nodes to find inactive nodes that may be activated in the next iteration. In this way, we can reduce redundant computation.

Since influence spread is a stochastic process, we take the average of 10,000 runs as the final result.

Algorithm 1 ISE IC

```
function IC(outgoing,seedSet)
    activatedNodes ← seedSet
    activity ← seedSet
    while activity ≠ ∅ do
        newSet ← ∅
        for i ∈ activity do
            for j ∈ outgoing[node] do
                if j is not activated then
                    prob ← random(0, 1)
                    if prob ≤ w_ij then
                        activatedNodes.add(j)
                        newSet.add(j)
                    end if
                end if
            end for
        end for
        activity ← newSet
    end while
    return len(activatedNodes)
end function
```

Algorithm 2 ISE LT

```
function LT(incoming, outgoing,seedSet)
    activatedNodes ← seedSet
    activity ← seedSet
    randomly assign threshold to each node
    while activity ≠ ∅ do
        newSet ← ∅
        for node ∈ activity do
            for j ∈ outgoing[node] do
                if neighbor is not activated then
                    influence ← 0
                    for i ∈ incoming[j]) do
                        if i is activated then
                            influence ← influence + w_ij
                        end if
                    end for
                    if influence ≤ j.threshold then
                        activatedNodes.add(j)
                        newSet.add(j)
                    end if
                end if
            end for
        end for
        activity ← newSet
    end while
    return len(activatedNodes)
end function
```

C. Influence Maximization Problem (IMP)

There are many algorithms that can solve IMP. In this project, I implemented an algorithm that is based on IMM [1] and introduced parallelism to accelerate the algorithm. The IMM algorithm provides a $(1 - 1/e - \varepsilon)$-approximate solution to IMP and performs well in large scale problems. The algorithm is composed of two phases:

- **Sampling** This step generates random reversed reachable sets and put them together as a set $\mathcal{R}$ until a stopping criteria is met.
  To generate a RR set, we start from a randomly chosen node v, apply influence spread in a reversed direction and return the activated nodes. Which is to say, active nodes will try to activate inactive neighbors in $incoming[node]$ according to IC or LT model. The process is almost the identical to ISE, so details are omitted in this report.

Algorithm 3 genRR

```
function GENRR(v)
    RR ← reversedISE(v)
    return RR
end function
```

Algorithm 4 Sampling

```
function LT(G, k, ε, l)
    R ← ∅
    LB = 1
    ε' ← √2 · ε
    for i ← 1 to log_2 n − 1 do
        x ← n/2^i
        θ_i ← λ'
        while len(R) ≤ θ_i do
            randomly select a node v from G
            RR ← GENRR(v)
            R.append(RR)
        end while
        S_i ← NODESELECTION(R)
        if n · F_R ≥ (S_i)/(1 + ε') then
            LB ← n · F_R(S_i)/(1 + ε')
            break
        end if
    end for
    θ ← λ*/LB
    while len(R) ≤ θ do
        randomly select a node v from G
        RR ← GENRR(v)
        R.append(RR)
    end while
    return R
end function
```

- **Node Selection** This step selects k solutions from $\mathcal{R}$ to form a size-k solution set S_k^* that covers the most RR sets. S_k^* is returned as the final result. $F_{\mathcal{R}}(S)$ returns the number of RR sets in $\mathcal{R}$ that overlaps S.

$$F_{\mathcal{R}}(S) = |\{RR | RR \cap S \neq \emptyset, RR \in \mathcal{R}\}|$$

This step can be optimized to reduce computation of $F_{\mathcal{R}}(S)$. We can keep track of the number of sets a node

v can activate if it is added to S_k^* with a list $hit[]$. Each time we take the v with the largest $hit[v]$, insert v to S_k^* and update $hit[]$.

Algorithm 5 NodeSelection

```
function NodeSelection(R, k)
    S_k* ← ∅
    for i ← 1 to k do
        v ← argmax{F_R(S_k* ∪ v) − F_R(S_k*)}
        S_k* ← S_k* ∪ v
    end for
    return S_k*
end function
```

The IMM algorithm put together the above two phases.

Algorithm 6 IMM

```
function IMM(G, k, ε, l)
    l ← l · (1 + log_n 2)
    R ← Sampling(G, k, ε, l)
    S_k* ← NodeSelection(R, k)
    return S_k*
end function
```

D. Parameters

There are many parameters in the IMM algorithms. The values are chosen so as to guarantee a $(1 - 1/e - \varepsilon)$-approximate result. l and ε are two external parameters. For IMM in this report, I used $l = 1$ and $\varepsilon = 0.1$.

Parameters are computed as follows:

$$\alpha = \sqrt{l \log n + \log 2} \tag{1}$$

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

$$\lambda^* = 2n((1 - 1/e) \cdot \alpha + \beta)^2 \cdot \varepsilon^{-2} \tag{3}$$

$$\lambda = (8 + 2\varepsilon)n \cdot (l \log n + \log \binom{n}{k} + \log 2) \tag{4}$$

IMM is designed to finish in $O((k + l)(n + m) \log n/\varepsilon^2)$ time and returns a $(1 - 1/e - \varepsilon)$-approximate solution with at least $1 - 1/n^l$ probability under triggering model(In this report, IC and LT) [2]. For IMM, we set $l = 1$.

$\varepsilon \in (0, 1)$ is a parameter that can be set by the user to control accuracy. A lower ε value will increase accuracy but will increase computation time dramatically. I used $\varepsilon = 0.1$ to balance accuracy and computation time.

III. Parallel Implementation

In this project, I used parallel programming in ISE and Sampling phase of IMM.

In ISE, the process of computing average score is conducted in parallel.

In IMM, the process of generating RR sets is conducted in parallel.

Unfortunately, experiments show that most of the computation time in IMM is used by $nodeSelection$ method.

IV. Validation

A. Environment

- OS: windows 10
- RAM: 16G
- CPU: Intel Core i7 8700k @ 3.7GHz
- Python: 3.7.1

The parallel part of the program uses 8 processes.

B. ISE

ISE is run 10000 times on the $network$ data set with two seed sets and the result is the average value.

TABLE II
REPRESENTATION

Benchmark	Diffusion Model	Time(s)	Result
network-seeds	IC	1.108	5.037
network-seeds	LT	1.101	5.053
network-seeds2	IC	1.110	30.423
network-seeds2	LT	1.239	37.145

C. IMP

IMP is run on the $network$ and $NetHeap$ data sets.

TABLE III
REPRESENTATION

Benchmark	Diffusion Model	k	Time(s)	Result
network	IC	5	0.387	30.636
network	LT	5	0.229	37.653
NetHeap	IC	5	6.954	323.585
NetHeap	LT	5	6.312	392.521
NetHeap	IC	50	9.766	1296.445
NetHeap	LT	50	8.915	1698.013

V. Discussion

Computational experiments show that my implementation of ISE can estimate influence spread process quite accurately and my implementation of IMP can obtain good result in an acceptable time.

While my IMP implementation perform well under LT model, performance is not so good under IC model compared to others' algorithms.

Moreover, experiments also show that most of the computation time of my implementation of IMP is consumed by node selection, as is shown in

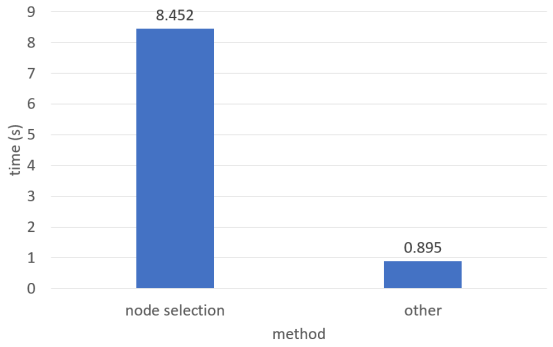

Fig. 1. Running time of IMM under NetHeap, $k = 50$, IC

A better implementation of node selection may be used and hopefully it can make use of parallelism.

VI. Conclusion

In this project, I implemented ISE with IC and LT model and IMP using IMM algorithm. The performance is generally good. Some improvement can be made to improve the efficiency of implementation.

Acknowledgment

The authors would like to thank the TAs for their hint in the Lab and maintaining a online runtime platform.

References

[1] 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*, SIGMOD '15, (New York, NY, USA), pp. 1539–1554, ACM, 2015.

[2] D. Kempe, J. Kleinberg, and E. 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*, KDD '03, (New York, NY, USA), pp. 137–146, ACM, 2003.