MS&E 231 Project Report
Influence Maximization in the presence of Negative Opinions

Oscar Aguilar, Louis Gautier, Eva Zhang

December 13, 2023

The code for our project is available on this repository

Abstract

Influence Maximization is a Stochastic Optimization problem that has numerous applications in fields such as viral marketing, epidemiology, and social network analysis. Given a stochastic model of influence propagation, it consists of finding a set of seed nodes of size at most k so that the expectation of their influence spread under the proposed model is maximized. In this report, we describe how we implemented an algorithm that solves Influence Maximization when negative opinions may emerge, proposed by Chen [1]. We first depict the stochastic model proposed by the authors which incorporates the concept of negativity bias from social psychology. Then, we implement and benchmark the custom algorithm that they proposed for this problem (called MIA-N) against a greedy simulation-based algorithm. We compare the theoretical complexity and observed run times of these two algorithms, and give insights and interpretation on selected figures that we replicated from their paper.

1 Introduction

1.1 Influence Propagation Models and Influence Maximization Algorithms

In the field of social network analysis, influence propagation models and influence maximization algorithms have drawn significant attention for their ability to identify influential individuals whose actions can propagate through a social network, leading to the maximal spread of influence. A seminal work in this field is the paper by Kempe, Kleinberg, and Tardos [2], which introduced a fundamental model for maximizing influence spread through a social network. This paper introduced both a stochastic model of influence in a social network called the Independent Cascade model and algorithmic tools to solve the combinatorial optimization problem consisting of maximizing the expected influence spread of a set of seed nodes of size k. We provide a high-level description of both in this section.

Stochastic model: The Independent Cascade Model The Independent Cascade model simulates the propagation of influence in a social network. Nodes of the directed network $G = (V, E)$ are either active or inactive, and a subset of them, denoted as the seed set S, is initially active. Each edge (u, v) is associated with a transmission probability $p_{u,v}$ that quantifies how easily node u will successfully influence node v (in other words, make node v adopt an action or a belief). In a social network, these edges would represent (directed) trust or friendship relations between individuals that are represented by the nodes. Under the Independent Cascade model, the activation of each node follows independent processes. When node u becomes active, it tries once to activate each of its neighbors $v \in \mathcal{N}(u)$. The success of the neighbor activation process is modeled by independent Bernoulli random variables with probability $p_{u,v}$. It results in a growing influence spread, which is defined as the set of nodes that are activated at the end of the influence process, when all activated nodes have tried once to activate their neighbors. A possible rollout is shown in Figure 1.

Algorithm Kempe, Kleinberg, and Tardos then formalize the influence maximization problem: a combinatorial optimization problem that, given a network and an influence model, consists of identifying a set of initial seed nodes that, when activated, will result in expectation in the largest possible influence spread.

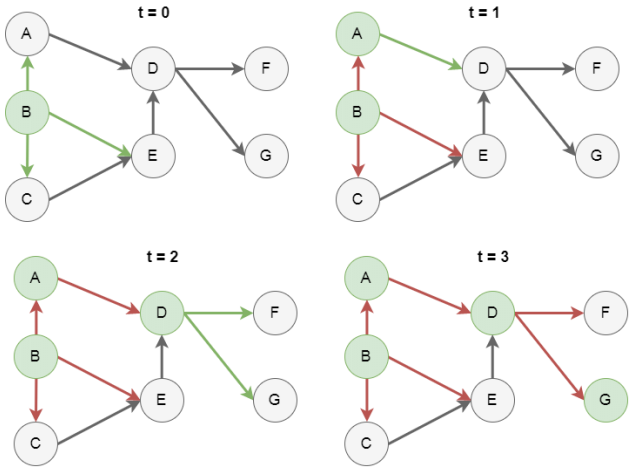


Figure 1: Possible roll-out of the Independent Cascade Model. Green arrows represent successful activations while red arrows represent failed ones.

Solving this problem is NP-hard. However, the authors show that the influence spread under the Independent Cascade model is a submodular function [2], that's to say that adding seed nodes brings diminishing returns. This brings nice properties as it implies that a greedy algorithm is guaranteed to find a $(1 - \frac{1}{\epsilon})$ approximation of the maximum influence spread. We will develop on how to build this greedy algorithm in section 3.1.

1.2 Influence Maximization when negative opinions may emerge

The work by Kempe et al. laid the groundwork for subsequent research in influence maximization, inspiring the development of various algorithms and extensions. The paper by Chen et al. [1] extends Kempe et al.'s work with negative opinions, which accounts for more realistic behaviors where opinions can turn negative, and negative opinions tend to spread more, which is known as the negativity bias in social psychology. It is this paper that we analyze in depth and replicate in this project. The authors propose both a new stochastic model, the Independent Cascade Model with Negative Opinions (IC-N), that we describe in section 2, and an algorithm, called MIA-N, for *Maximum Influence in Arborescences with Negative Opinions*, that we describe in section 3.3. Our project consisted of implementing the MIA-N algorithm in Python, benchmarking it against a greedy simulation-based algorithm, and replicating some key figures (namely Figures 2 and 3) of the original Chen's paper.

Applications The influence maximization framework is very useful for solving problems such as viral marketing, political campaign design, and disease spread modeling. It is natural to see that negative opinions exist in these models. Not only do negative opinions exist, they tend to propagate faster than positive opinions. For example, we need to read a collection of good reviews to go try out a restaurant, but would likely avoid one because one friend said they had a terrible experience there. This asymmetry called negativity bias thus further strengthens the importance of considering negative opinions.

2 Model

Stochastic model: The Independent Cascade Model with Negative Opinions (IC-N) Just like the classical Independent Cascade model described earlier, the IC-N model is based on a directed graph $G = (V, E)$, where each node is in one of three states: neutral (not activated), positively activated, or negatively activated. Each edge is associated with a propagation probability $p : E \to [0, 1]$. A very important parameter is the quality factor q, which is fixed for the whole graph and quantifies the quality of the product for which the influence is spread, or more specifically the probability that a node turns positive when influenced by a positive neighbor.

Under the model, k seeds are initially activated. Inside this seed set, each node turns positive with probability q, and negative with probability $1 - q$. Then, the propagation of influence unfolds in two ways:

- Every newly activated node tries to influence its out-neighbors. Like in the Independent Cascade Model, it succeeds in doing so with the probability given by the edge weight.
- If some out-neighbors are successfully activated, they become positive with probability q if they were activated by a positive in-neighbor, or negative with probability 1 if they were activated by a negative in-neighbor. This trick allows to model the negativity bias.

A possible roll-out of the IC-N model is presented in Figure 2. Under the IC-N model, the objective of influence maximization algorithms will be to maximize the expected positive influence spread, that's to say the expectation of the number of nodes that are positive at the end of the stochastic process.

IC-N Model Implementation We implemented the IC-N stochastic model in Python. After the initialization of the seed set S, which corresponds to the k seeds mentioned in the previous section, the influence loop continues until no new nodes can be activated. In each iteration of the main loop, we build a set that contains the neutral successors of the newly active nodes. For each node v in the successors set, the algorithm randomly orders v's predecessors, and goes through each of the predecessors until the activation of v or until v runs out of predecessors that can activate it.

3 Algorithms

Given a network and the parameters of the IC-N influence model, finding the initial seed set of size k that maximizes the expected positive influence spread is NP-hard. Therefore, algorithms have to rely on heuristics to find the best possible approximate solution. Fortunately, the submodular nature of the objective function guarantees that a greedy algorithm will approximate the optimal solution within a $1 - \frac{1}{\epsilon}$ ratio, provided we have a way to estimate the increment in positive influence spread given by adding a node to the seed set. We describe the greedy algorithm structure in section 3.1 and benchmark two ways to estimate this increment in positive influence spread:

- An exhaustive Monte-Carlo simulation-based estimation procedure,
- The MIA-N algorithm proposed in the paper: an efficient estimation procedure to estimate this influence spread increment.

We implemented and benchmarked these two estimation procedures in our project.

3.1 Greedy algorithm

Algorithm 1 Greedy(k, f)

1: Initialize $S = \emptyset$
2: **for** $i = 1$ to k **do**
3: Select $u = \arg\max_{w \in V \setminus S}(f(S \cup \{w\}) - f(S))$
4: $S = S \cup \{u\}$
5: **end for**
6: **Output** S

The greedy algorithm presented in Algorithm 1 is a heuristic approach to solving the Influence Maximization problem under the Independent Cascade with Negative Opinions (IC-N) model. It starts with an empty seed set and iteratively selects the node that yields the highest estimated positive influence spread gain, represented by the monotone and submodular set function f that represents our estimate of the expected positive influence spread.

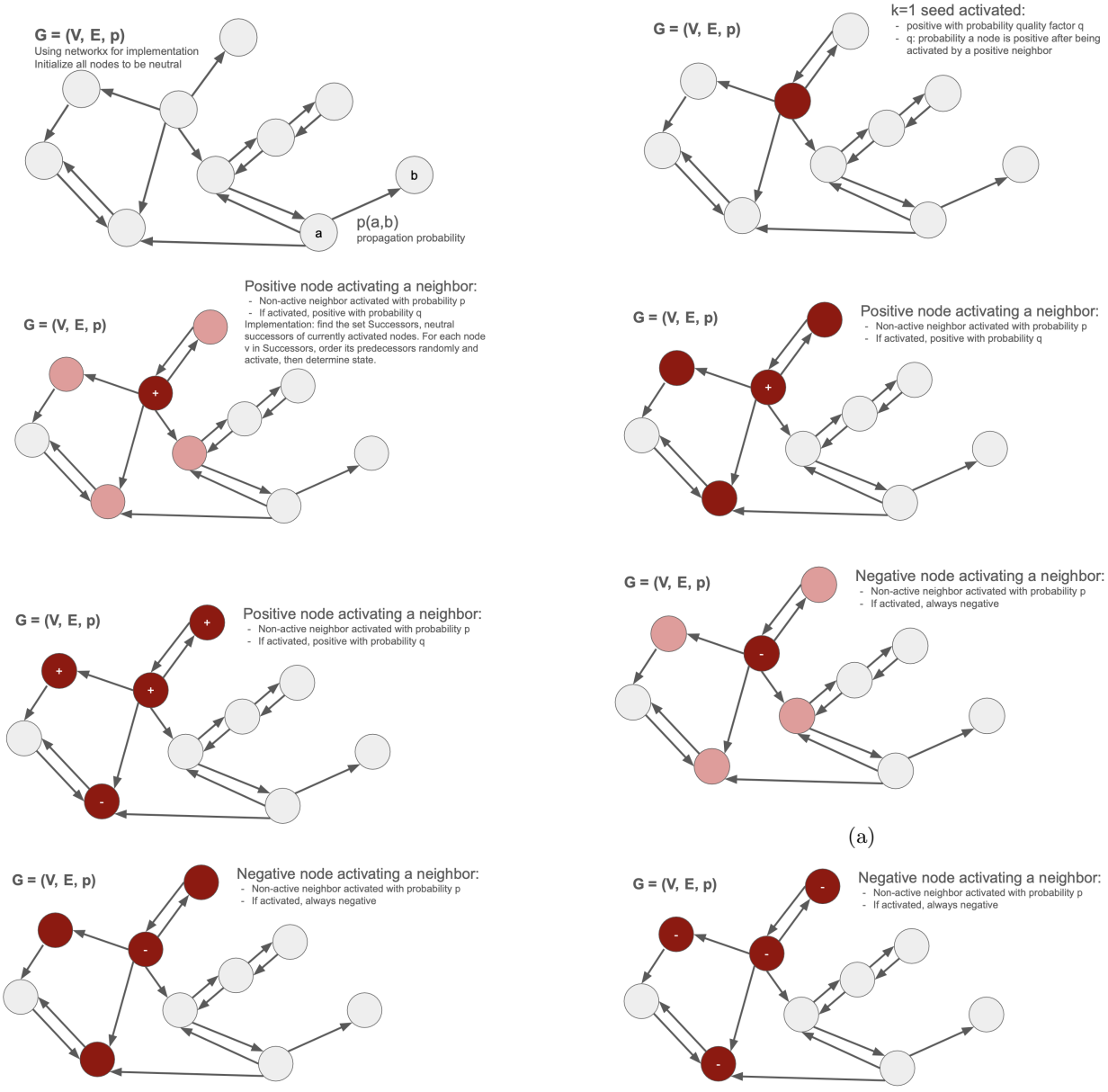


Figure 2: Possible roll-out of the IC-N Model

3.2 Monte Carlo Simulation-based value estimation

The simplest way to design this estimation function f is to use Monte Carlo Simulation: for each input set of f, we run M roll-outs of the stochastic IC-N model and the output of f is computed as the mean of the positive influence spreads obtained in the M roll-outs. This method is very computationally costly, as for all $i \in [|1, k|]$, for all $w \in V \setminus S$, we have to run M times the IC-N model, yielding a complexity of $\mathcal{O}(knMm)$, where n is the number of nodes in the graph and m is the number of edges.

3.3 MIA-N algorithm

The goal of the MIA-N algorithm described in Algorithm 2 is to provide a heuristic to estimate this incremental influence gain without having to evaluate the incremental influence gain of each node of the whole graph at each iteration like in the greedy algorithm. It uses arborescences to approximate local influence regions of every node in the graph and uses efficient methods to compute the positive influence spread in these arborescences. To understand how the MIA-N algorithm works, we thus need two ingredients:

- A way to efficiently compute positive influence spread in arborescences, presented in 3.3.1,
- A way to approximate local influence regions of every node in the graph by arborescences, presented in 3.3.2.

Algorithm 2 MIA-N(G, q, k, θ)

/* Initialization */
Set $S = \emptyset$
Set actual$(v) = 0$ for each node $v \in V$
for each node $v \in V$ **do**
    Compute MIIA(v, q, θ), MIOA(v, q, θ)
    Compute pap$(u, \{v\}, \text{MIIA}(u, q, \theta), q)$, $\forall u \in \text{MIOA}(v, q, \theta)$
    IncInf$(v, u) = \text{pap}(u)$, $\forall u \in \text{MIOA}(v, q, \theta)$
    IncInf$(v) = \sum_{u \in \text{MIOA}(v,q,\theta)} \text{IncInf}(v, u)$
end for
/* Main loop */
for $i = 1$ to k **do**
    Pick $u = \arg\max_{v \in V \setminus S}\{\text{IncInf}(v)\}$
    $S = S \cup \{u\}$ /* u is selected as a new seed */
    /* Update incremental influence spreads */
    for each $v \in \text{MIOA}(u, q, \theta)$ **do**
        actual$(v)+ = \text{IncInf}(u, v)$ /* Incremental influence from u to v is realized */
        for each $w \in \text{MIIA}(v, q, \theta)$ **do**
            Compute pap$(v, S \cup \{w\}, \text{MIIA}(v, q, \theta), q)$
            $\Delta = \text{pap}(v) - \text{actual}(v)$
            IncInf$(w)+ = \Delta - \text{IncInf}(w, v)$
            IncInf$(w, v) = \Delta$
        end for
    end for
end for
return S

3.3.1 Computing influence in arborescences

[1] introduces an efficient dynamic programming approach for computing influence spread in tree structures, focusing on arborescences, which are directed trees where edges point either into (in-arborescence) or away from (out-arborescence) a root node. While the general task of determining influence spread using the

Independent Cascade (IC) model is acknowledged as #P-hard, the algorithm uses arborescence structures to estimate the positive influence spread $\sigma_A(S, q)$.

In the case of out-arborescences, the computation of positive influence spread for a seed node is straightforward, requiring a simple traversal of the arborescence. However, for in-arborescences, the computation of positive activation probabilities ($pap(u)$) becomes more intricate, particularly when considering negative opinions. The paper introduces dynamic programming to efficiently compute activation probabilities ($ap(u, t)$) and subsequently derive positive activation probabilities ($pap(u, t)$). The presented theorems (Formulae (4.2) and (4.3)) encapsulate the efficient computation of influence spread in in-arborescences, boasting a time complexity of $O(hn)$, where h and n denote the height of the arborescence, and the number of nodes in the arborescence, respectively.

The idea behind the dynamic program is to recursively compute the probability that each node is activated at time t, without regard for the sign of the activation. At time $t = 0$, only the seed set S is activated with probability 1. Then for each subsequent step t, the probability that a node $u \notin S$ is activated at t is the difference between the probability it is unactivated by $t-1$ and the probability it is unactivated by step t. This is logic is represented by the following formula:

$$ap(u,t) = \begin{cases} 1 & \text{if } t = 0 \wedge u \in S \\ 0 & \text{if } t = 0 \wedge u \notin S \\ 0 & \text{if } t > 0 \wedge u \in S \\ \prod\limits_{w \in N^{in}(u)} [1 - \sum\limits_{i=0}^{t-2} ap(w,i)p(w,u)) & \\ \quad - \prod\limits_{w \in N^{in}(u)} [1 - \sum\limits_{i=0}^{t-1} ap(w,i)p(w,u))] & \text{if } t > 0 \wedge u \notin S \end{cases}$$

With this, the computation of $\sigma_A(S, q)$ is just a matter of multiplying the activation probability with q^{t+1} to get the positive activation probability at t and summing this probability for every node u over every time t. This is defined by the equation below.

$$\sigma_A(S,q) = \sum_{u \in V} pap(u) = \sum_{u \in V} \sum_{t \geq 0} pap(u,t) = \sum_{u \in V} \sum_{t \geq 0} ap(u,t) \cdot q^{t+1}$$

The dynamic programming approach defined by these formulas is implemented in our code with the recursion rolled out to an iterative form for enhanced efficiency.

3.3.2 Approximate influence regions as arborescences

The key contribution of the MIA-N algorithm is to define the local influence regions of each node in the graph. To this extent, the paper introduces two arborescence structures: the Maximum Influence In- and Out-Arborescences. These trees are defined as:

$$MIIA(v,q,\theta) = \bigcup_{u \in V, ppp(MIP(u,v)) \geq \theta} MIP(u,v)$$

$$MIOA(v,q,\theta) = \bigcup_{u \in V, ppp(MIP(v,u)) \geq \theta} MIP(v,u)$$

where $MIP(u, v)$ is the path between u and v with highest positive propagation probability: $ppp(P) = \prod_{i=1}^{m-1} p(v_i, v_{i+1})q^m$, where P is a path $(v_0, ..., v_l)$. θ is a fixed hyperparameter that controls the size of these structures.

The critical assumption that the MIA-N algorithm makes is that the influence to a node v is only propagated through $MIIA(v, q, \theta)$. Then, the positive influence spread of the whole seed set S can be estimated as $\mu(S, q) = \sum_{u \in V} pap(v, S, MIIA(v, q, \theta), q)$ where pap evaluates the positive activation probability of v.

This assumption brings a huge efficiency improvement. Indeed, each time we add a node u to the seed set, we only need to update the incremental influence spread of nodes $w \in MIIA(v, q, \theta)$, for $v \in MIOA(u, q, \theta)$. As a result, at each step of the main loop of the greedy algorithm, we only have to compute $n_o n_i$ incremental influence spread updates, with $n_o \ll n$ and $n_i \ll n$ bounding the sizes of the MIOAs and MIIAs.

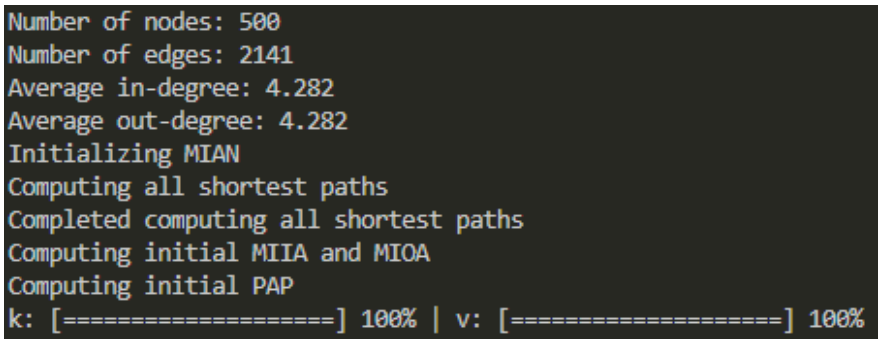


Figure 3: Example log from the implementation

3.4 Complexity and robustness comparison

Complexity comparison On the one hand, the time complexity of the Monte Carlo algorithm is $\mathcal{O}(knMm)$: for each step of adding a node to the seed set (k such steps), we need to run M times the simulation for each of the nodes of the graph (excluding the ones already in S). Running one simulation costs $\mathcal{O}(m)$ as the number of neighbor activation attempts.

On the other hand, the time complexity of the MIA-N algorithm is $\mathcal{O}(kn_o n_i^2 h)$, as for each step of adding a node to the seed set (k such steps), we need to compute $n_o n_i$ incremental influence spread updates, and each of these updates takes $\mathcal{O}(n_i h)$ based on the Dynamic Program of 3.3.1, where h is a bound on the height (depth) of the MIIA trees.

In practice, we have $h \ll n_o, n_i \approx M \ll n$, which means that the time to run the MIA-N algorithm is well below the time it takes to run the greedy simulation-based algorithm. We verified it empirically as described in 4.5.

Robustness comparison On the one hand, the Monte-Carlo simulation is quite sensitive to the number of simulations M that we need to average to obtain the positive influence spread estimates. Lower values of M can give unstable expectation estimates that will lead us to take suboptimal decisions of which nodes to add to the seed set.

On the other hand, the MIA-N algorithm is very sensitive to the choice of the parameter θ, which controls the size of the MIIAs and MIOAs. Empirically, we observed that a small delta of 0.01 on θ could make the execution of the algorithm go from a few minutes to several hours. Besides, the size of the MIIAs and MIOAs is highly dependent on the graph structure and on the edge weights, making the tuning of the θ parameter in the MIA-N algorithm very complex.

4 Replication results

4.1 Our MIA-N implementation

We fully implemented the MIA-N algorithm in Python with compatibility with the NetworkX graph library. To the best of our knowledge, we are the only publicly available repository implementing this algorithm. This implementation is designed to be generic, accommodating any input graph, and offers user-friendly interaction with convenient results evaluation. Figure 4.1 shows the output generated when running it on a 500-node graph.

Some key choices that we made concerned the computations of the MIIAs and MIOAs of each node. We chose to use Dijkstra's algorithm on a modified graph having $-log(p_{uv}q)$ as edge weights, as finding the Maximum Influence Path is equivalent to finding the shortest path on the same graph with these modified edge weights (independent probabilities are multiplicative whereas distances are additive, hence the log transformation).

4.2 Datasets

We evaluated our implementations and replicated the results of the paper on two real-world graphs: the Epinions and the NetHPT datasets.

Epinions dataset The Epinions dataset is a graph that represents trust relationships in a consumer review website. Each node is a member of the site, and the edges are the directed trust relationships that each member gets to decide. There are 75,879 nodes and 508,837 edges in this graph, making it a fairly dense graph.

NetHPT dataset The NetHPT graph is the High Energy Physics - Theory collaboration network from 1993 to 2003. The nodes are authors of papers in the High Energy Physics - Theory category on arXiv, and the edges are undirected co-author relationships. There is an edge between 2 nodes if the 2 authors have co-authored a paper. There are 9,877 nodes and 25,998 edges in this graph.

Random walk subgraphs The size of both datasets were too large to have reasonable run times for the scope of this project. Thus, we implemented a random walk to obtain subgraphs of Epinions and NetHPT. The Epinions subgraph contains 200 nodes, and the NetHPT subgraph contains 1000 nodes. The random walk subgraph algorithm is given in Algorithm 3. It takes as input the large graph G, the target number of nodes in the subgraph *target_num_nodes*, a maximum number of iterations *max_iters*, and a boolean variable directed indicating whether the graph is directed.

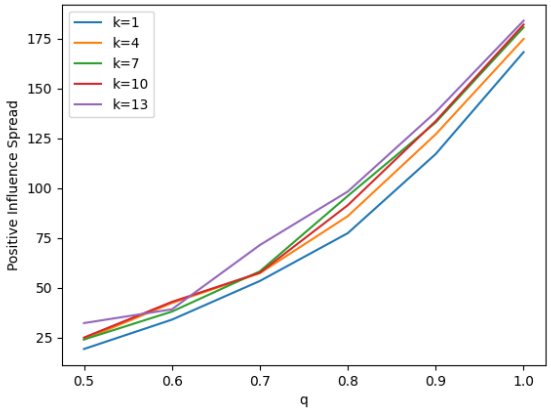
Algorithm 3 Random walk subgraph of size target_num_nodes on graph G, max_iters, boolean directed.

```
start_node a node chosen randomly from G
subgraph_nodes = {start_node}, iters = 0
while length(subgraph_nodes) < target_num_nodes and iters < max_iters do
    if directed then
        neighbors = sorted current node's successors
    else
        neighbors = sorted current node's neighbors
    end if
    if neighbors = ∅ then
        if length(subgraph_nodes) < 10 then
            current_node = randomly choose a node from G
        else
            current_node = randomly choose a node from subgraph_nodes
        end if
    else
        current_node = randomly choose a node from neighbors
        add current_node to subgraph_nodes
    end if
    iters = iters + 1
end while
return subgraph_nodes
```

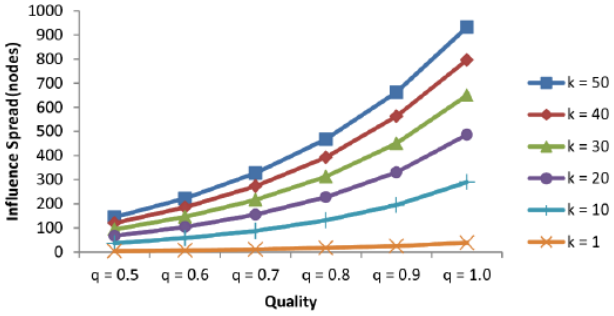
4.3 Positive influence spread against the quality factor

Figure 4 replicates Figure 2 from [1]. It shows the positive influence spread as a function of the quality factor q of the product the influence is spread about. We can make several comments about this figure and our replication.

First, we observe both in our figure and in the figure of the paper a convex trend of the expected positive influence spread with q at a fixed initial number of seed nodes k. This was expected as with the negativity bias modeled in IC-N, only positive nodes can participate in the positive influence spread, therefore the



(a) Our implementation tested on a 200-node subgraph of the Epinions dataset



(b) Paper on the full NetHPT graph

Figure 4: Positive influence spread against q

number of positive nodes grows exponentially with the probability that an activated node turns positive, which is q.

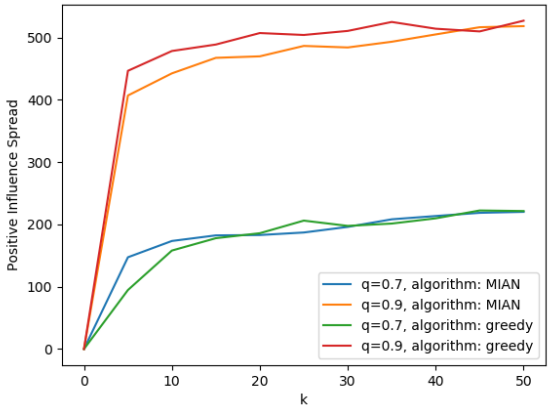
Besides, we can observe that the difference between the curves for several values of k is smaller with our implementation on the Epinions subgraph than in the article. Our interpretation is that it is linked to our graph being much denser. Whereas the nodes in the full NetHPT graph have an average in-degree of approximately 3, the nodes of our Epinions subgraph have an average in-degree of about 15. As a result, adding more nodes to the seed set has very quickly diminishing returns in our case, which is why curves for successive values of k are much closer than in the article. The interpretation is that in a dense social network, such as a small village for example, the quality of a product is the most important factor to obtain a large positive influence spread. It is for example more important than giving free meals to more people in the village, which would be a possible analoguous process to adding nodes to the seed set, and increasing k.

4.4 Positive influence spread against the number of seed nodes

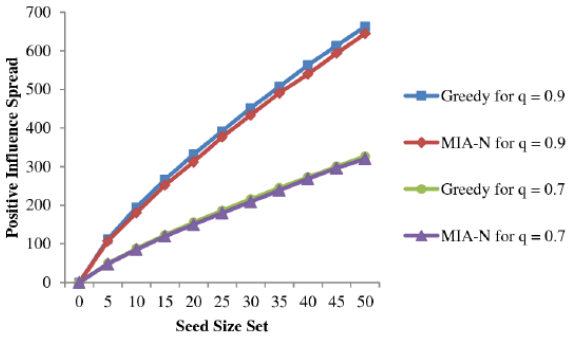
Figure 5 (Figure 3 in [1]) compares the results of running our implementations of the MIA-N and greedy simulation-based algorithms on a 1000-node subgraph of the NetHEPT graph is compared to the results obtained by Wei Chen et. al for the full netHEPT graph. We observe a similar approximation of performance for positive influence spread by MIAN as for greedy. However, the positive influence spread for both algorithms does not increase by the same function of k for us as for the original paper. We attribute this discrepancy to the method by which we sampled the subgraph of NetHEPT. The random walk that we performed is more likely to sample nodes of higher degree. As a result, the subgraph we obtained has a higher average in/out degree than the full graph, and is generally much more connected. This imbalance of high-degree nodes would consist of "low-hanging fruit" for both algorithms to initially exploit, which would explain the steepness of the initial curves. Subsequently, any further nodes added would have a comparatively much smaller contribution, which would account for the plateau in influence spread observed for $k \geq 7$.

4.5 Empirical run times comparison

Finally, we experienced quite similar run times to the ones described in the paper. Running one iteration of the simulation-based greedy algorithm on the 1000-node subgraph of the NetHPT graph took us 1.62 hours, while running one iteration of the MIA-N algorithm took us about 45 seconds on average (giving a ratio of 130 between the two algorithms), which approximately matches the ratio of 112 between the two algorithms that the authors reported in their paper (in Figure 5).



(a) Our implementation on a 500-node subgraph of the NetHPT graph



(b) Paper on the full NetHPT graph Influence

Figure 5: Positive influence spread against k

5 Conclusion and future work

To conclude, the MIA-N algorithm is a very powerful heuristic to efficiently maximize the expected positive influence spread in a network when negative opinions may emerge, which has numerous applications in marketing and epidemiology. Understanding and implementing the algorithm was a great opportunity to understand how to practically solve those problems in a scalable way to apply them to real networks. We managed to replicate the key figures of the paper with the implementation that we built from scratch in Python, and we were able to provide intuitive interpretations of these results.

We can see several promising opportunities for future work in influence maximization when negative opinions may emerge and propagate. First, in real networks, we might have different levels of certainty about the input graph that models the network and more specifically the edge weights. We could incorporate uncertainty about edge weights, by for example fitting linear models to predict the edge weights based on node features. Then, we could evaluate the impact of this uncertainty on the uncertainty about the positive influence spread.

Besides, we could envision implementing the algorithm in totally different and more current application contexts, such as viral marketing. With the rise of TikTok and other social media platforms, viral marketing is essential for most products and services. The IC-N model with the MIA-N algorithm can simulate the spreading of both positive and negative information, allowing companies to potentially identify the best seeds to conduct viral marketing. This is just one useful application we can investigate further, in addition to epidemiology, political campaigns, and more.

References

[1] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincon, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *Proceedings of the 2011 siam international conference on data mining*, pages 379–390. SIAM, 2011.

[2] David Kempe, Jon Kleinberg, and Éva 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*, pages 137–146, 2003.