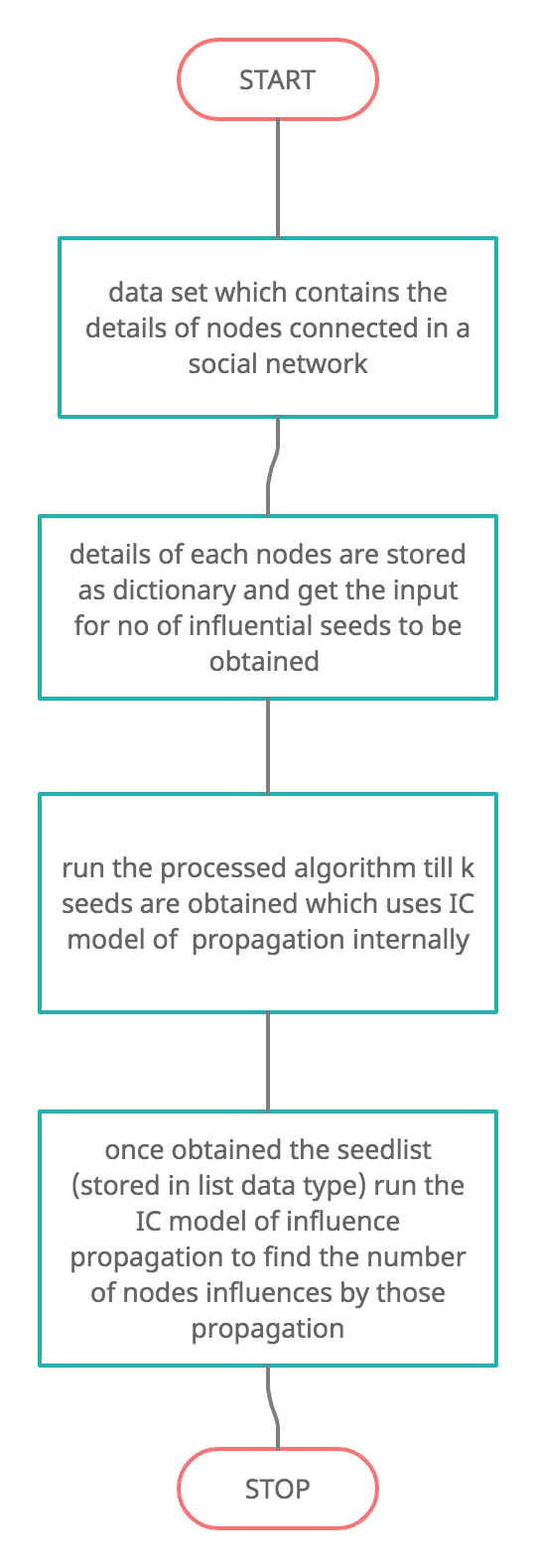
MAXIMIZING THE INFLUENCE SPREAD IN SOCIAL MEDIA USING IMPROVISED CELF ALGORITHM AND CENTRALITY MEASURES

Abstract

Influence maximization in social networks mainly depends on selecting the set of influential users as seed set. Previous algorithms for this were greedy approach, hill climbing algorithm and CELF. For this project will slightly modified the CELF algorithm and we have used the centrality measures property of the nodes, which increases its efficiency And the propagation models are independent cascade and linear threshold. These algorithms are efficient for propagating the influence according to our algorithm.

**Index Terms-Influence Maximization (IM), Influence Propagation, Greedy Algorithms, Sub-modularity, Social Networks.**

DESIGN DIAGRAM



**PROPOSED ALGORITHM**

The following Algorithm describes the CELF algorithm.

We use σ(S) to denote the spread of seed set S. We maintain a heap Q with nodes corresponding to users in the network G. The node of Q corresponding to user u stores a tuple of the form hu.mg1,u.prev best,u.mg2,u.flagi. Here u.mg1 = ∆u(S), the marginal gain of u w.r.t. the current seed set S; u.prev best is the node that has the maximum marginal gain among all the users examined in the current iteration, before user u; u.mg2 = ∆u(S {prev best}), and u.flag is the iteration number when u.mg1 was last updated. The idea is that if the node u.prev best is picked as a seed in the current iteration, we don’t need to recompute the marginal gain of u w.r.t (S {prev best}) in the next iteration. It is important to note that in addition to computing ∆u(S), it is not necessary to compute ∆u(S {prev best}) from scratch. More precisely, the algorithm can be implemented in an efficient manner such that both ∆u(S) and ∆u(S {prev best}) are evaluated simultaneously in a single iteration of Monte Carlo simulation (which typically contains 10,000 runs). In that sense, the extra overhead is relatively insignificant compared to the huge runtime gains we can achieve if it works out.

Algorithm: Proposed Algorithm

Input: G,k

Output: seed set S

1: S ← ; Q ← ; last seed = null; cur best = null.

2: for each u V do

3: u.mg1 = σ({u}); u.prev best = cur best; u.mg2 = σ({u,cur best}); u.flag = 0.

4: Add u to Q. Update cur best based on mg1.

5: while |S| < k do

6: u = top (root) element in Q.

7: if u.flag == |S| then

8: S ← S {u};Q ← Q − {u};last seed = u.

9: continue;

10: else if u.prev best == last seed then

11: u.mg1 = u.mg2.

12: else

13: u.mg1 = ∆u(S); u.prev best = cur best; u.mg2 = ∆u(S {cur best}).

14: u.flag = |S|; Update cur best.

15: Reinsert u into Q and heapify.

In addition to the data structure Q, the algorithm uses the variables S to denote the current seed set, last seed to track the id of last seed user picked by the algorithm, and cur best to track the user having the maximum marginal gain w.r.t. S over all users examined in the current iteration. The algorithm starts by building the heap Q initially (lines 2-4). Then, it continues to select seeds until the budget k is exhausted. As in CELF, we look at the root element u of Q and if u.flag is equal to the size of the seed set, we pick u as the seed as this indicates that u.mg1 is actually ∆u(S) (lines 6-9). The optimization of CELF comes from lines 10- 11 where we update u.mg1 without recomputing the marginal gain. Clearly, this can be done since u.mg2 has already been computed efficiently w.r.t. the last seed node picked. If none of the above cases applies, we recompute the marginal gain of u (line 12-13). The propagation model contributes 40% and then the degree centrality measure contribute 20%, closeness centrality measure contribute 40% to the gain of each node.