CELF++: Optimizing the Greedy Algorithm for Influence Maximization in Social Networks

Amit Goyal
Dept. of Computer Science
University of British Columbia
Vancouver, BC, Canada
goyal@cs.ubc.ca

Wei Lu
Dept. of Computer Science
University of British Columbia
Vancouver, BC, Canada
welu@cs.ubc.ca

Laks V.S. Lakshmanan
Dept. of Computer Science
University of British Columbia
Vancouver, BC, Canada
laks@cs.ubc.ca

ABSTRACT

Kempe et al. [4] (KKT) showed the problem of influence maximization is **NP**-hard and a simple greedy algorithm guarantees the best possible approximation factor in PTIME. However, it has two major sources of inefficiency. First, finding the expected spread of a node set is #**P**-hard. Second, the basic greedy algorithm is quadratic in the number of nodes. The first source is tackled by estimating the spread using Monte Carlo simulation or by using heuristics [4, 6, 2, 5, 1, 3]. Leskovec et al. [6] proposed the CELF algorithm for tackling the second. In this work, we propose CELF++ and empirically show that it is 35-55% faster than CELF.

Categories and Subject Descriptors H.2.8 [Database Management]: Database Applications - *Data Mining*
General Terms: Algorithms, Performance
Keywords: Social networks, Influence Propagation, Viral marketing, Greedy Algorithm, Submodularity, CELF.

1. INTRODUCTION

In influence maximization, we are given a network G with pairwise user influence probabilities (as edge labels) and a number k, and want to find a set S of k users (nodes) such that the expected spread of influence (spread for short) is maximized. In their seminal work [4], Kempe et al. (KKT) studied this problem, focusing on two fundamental propagation models – *Linear Threshold Model* (LT) and *Independent Cascade Model* (IC). They showed, under both models, the problem is **NP**-hard and a simple greedy algorithm successively selecting the node with the maximum marginal influence spread approximates the optimum solution within a factor of $(1 - 1/e)$. This is due to the nice properties of *monotonocity* and *submodularity* that the spread function exhibits under these models. In terms of spread, monotonocity says as more neighbors of some arbitrary node u gets active, the probability of u getting active increases. Submodularity says the marginal gain of a new node shrinks as the set grows. Function f is submodular iff $f(S \cup \{w\}) - f(S) \geq f(T \cup \{w\}) - f(T)$ whenever $S \subseteq T$.

A major limitation of the simple greedy algorithm is twofold: (i) The algorithm requires repeated computes of the spread function for various seed sets. The problem of computing the spread under both IC and LT models is #**P**-hard [1, 3]. As a result, Monte-Carlo simulations are run by KKT for sufficiently many times to obtain an accurate estimate, resulting in very long computation time. (ii) In each iteration, the simple greedy algorithm searches all the nodes in the graph as a potential candidate for next seed node. As a result, this algorithm entails a quadratic number of steps in terms of the number of nodes.

Considerable work has been done on tackling the first issue, by using efficient heuristics for estimating the spread [2, 5, 1, 3] to register huge gains on this front. Relatively little work has been done on improving the quadratic nature of the greedy algorithm. The most notable work is [6], where submodularity is exploited to develop an efficient algorithm called CELF, based on a "lazy-forward" optimization in selecting seeds. The idea is that the marginal gain of a node in the current iteration cannot be better than its marginal gain in the previous iterations. CELF maintains a table $\langle u, \Delta_u(S)\rangle$ sorted on $\Delta_u(S)$ in decreasing order, where S is the current seed set and $\Delta_u(S)$ is the marginal gain of u w.r.t S. $\Delta_u(S)$ is re-evaluated only for the top node at a time and if needed, the table is resorted. If a node remains at the top, it is picked as the next seed. Leskovec et al. [6] empirically shows that CELF dramatically improves the efficiency of the greedy algorithm.

In this work, we introduce CELF++ that further optimizes CELF by exploiting submodularity. Our experiments show that it improves the efficiency of CELF by 35-55%. Since the optimization introduced in CELF++ is orthogonal to the method used for estimating the spread, our idea can be combined with the heuristic approaches that are based on the greedy algorithm to obtain highly scalable algorithms for influence maximization.

2. CELF++

Algorithm 1 describes the CELF++ algorithm. We use $\sigma(S)$ 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 $\langle u.mg1, u.prev\_best, u.mg2, u.flag\rangle$. Here $u.mg1 = \Delta_u(S)$, 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; $u.mg2 = \Delta_u(S \cup \{prev\_best\})$, 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 $(S \cup \{prev\_best\})$ in the *next* iteration.

It is important to note that in addition to computing $\Delta_u(S)$, it is not necessary to compute $\Delta_u(S \cup \{prev\_best\})$

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

from scratch. More precisely, the algorithm can be implemented in an efficient manner such that both $\Delta_u(S)$ and $\Delta_u(S \cup \{prev\_best\})$ 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, as we will show from our experiments.

Algorithm 1 Greedy CELF++

```
Require: $G, k$
Ensure: seed set $S$
 1: $S \leftarrow \emptyset$; $Q \leftarrow \emptyset$; $last\_seed = null$; $cur\_best = null$.
 2: for each $u \in V$ do
 3:     $u.mg1 = \sigma(\{u\})$; $u.prev\_best = cur\_best$; $u.mg2 = \sigma(\{u, cur\_best\})$; $u.flag = 0$.
 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:         $S \leftarrow S \cup \{u\}; Q \leftarrow Q - \{u\}; last\_seed = u$.
 9:         continue;
10:     else if $u.prev\_best == last\_seed$ then
11:         $u.mg1 = u.mg2$.
12:     else
13:         $u.mg1 = \Delta_u(S)$; $u.prev\_best = cur\_best$; $u.mg2 = \Delta_u(S \cup \{cur\_best\})$.
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 $\Delta_u(S)$ (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).

3. EXPERIMENTS

We use two real world data sets consisting of academic collaboration networks: NetHEPT and NetPHY, both extracted from arXiv[1]. NetHEPT is taken from the "High Energy Physics – Theory" section and has 15K nodes and 32K unique edges. NetPHY is taken from the full "Physics" section and has 37K nodes and 174K unique edges. The graphs are undirected, however we make them directed by taking for each edge the arcs in both the directions. We consider the IC model and assign the influence probability to arcs using two different settings, following previous works (e.g., see [4, 2, 1]). In the first setting, for an arc (v, u) we set the influence probability as $p_{v,u} = 1/d_{in}(u)$, where d_{in} is the in-degree of the node u. In the second setting, we assign a uniform probability of 0.1 to all arcs. In all the experiments, we run 10,000 Monte Carlo simulations to estimate the spread.

Dataset	Running time (min)			Avg. # node lookups		
	CELF	CELF++	Gain	CELF	CELF++	Gain
Hept WC	245	159	35%	18.7	13.4	28.3%
Hept IC	5269	2439	53.7%	190.5	101.5	46.7%
Phy WC	1241.6	667.7	46.2%	18.6	15.2	18.3%

Table 1: Comparison between CELF and CELF++. Number of seeds = 100.

The results are shown in Table 1. We use WC (*weighted cascade*) to refer to the case when the probabilities are non-uniform and IC for the uniform probability 0.1 setting. We only show the results corresponding to NetHEPT WC, NetHEPT IC, and NetPHY WC for brevity. The results for NetPHY IC are similar. In these settings, we found that computing $u.mg2$ for all nodes in the first iteration results in large overhead. So, we apply CELF++ starting from the second iteration. Notice that the optimization behind CELF++ can be applied starting from any iteration. As can be seen, CELF++ is significantly faster than CELF. This is due to the fact that the average number of "spread computations" per iteration is significantly lower. Since we apply the optimization starting from the second iteration, we report the average number of nodes examined starting from the third iteration.

Memory Consumption: Although CELF++ maintains a larger data structure to store the look-ahead marginal gains ($u.mg2$) of each node, the increase of the memory consumption is insignificant. For instance, CELF consumes 21.9 MB for NetHEPT and 39.7 MB for NetPHY, while CELF++ uses 22.4 MB and 41.2 MB respectively.

4. CONCLUSIONS

In this work, we presented CELF++, a highly optimized approach based on the CELF algorithm [6] in order to further improve the naive greedy algorithm for influence maximization in social networks [4]. CELF++ exploits the property of submodularity of the spread function for influence propagation models (e.g., Linear Threshold Model and Independent Cascade Model) to avoid unnecessary re-computations of marginal gains incurred by CELF. Our empirical studies on real world social network datasets show that CELF++ works effectively and efficiently, resulting in significant improvements in terms of both running time and the average number of node look-ups.

5. REFERENCES

[1] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD 2010*.

[2] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD 2009*.

[3] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM 2010*.

[4] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD 2003*.

[5] M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In *PKDD 2006*.

[6] J. Leskovec et al. Cost-effective outbreak detection in networks. In *KDD 2007*.

[1] http://www.arXiv.org