CS 224w Project Report
Influence Maximization

Boris Perkhounkov and Dai Shen

December 11, 2016

1 Introduction

There has been much recent progress on the influence maximization problem. The problem is to find a small set of nodes that will result in the greatest spread of influence to other nodes in a given network. Most generally, for a directed graph (V, E), objective function $f : 2^V \to \mathbb{R}$ and cost function $g : 2^V \to \mathbb{R}$, the problem is to compute

$$\begin{aligned} \max_{S \subseteq V} \quad & f(S) \\ \text{subject to} \quad & g(S) \leq C \end{aligned} \tag{1}$$

The functions f, g depend on the model or the specific instance of the problem. Klempe et al.[2] have shown that in general, influence maximization is an NP-hard problem, and known approximations are still very time consuming on large graphs. Influence maximization has many diverse real world applications. For example, Leskovec et al.[3] find the placement of a small number of sensors in a water distribution network to detect contaminations as quickly as possible. In the same paper, they find small sets of blogs which capture as many information cascades as possible. Thus, further optimizations to known approaches for the influence maximization problem would be useful to a wide range of fields and real world problems.

A common model for influence maximization is the Independent Cascade Model (ICM). In this model, associated with each edge is the probability of the source node influencing the destination node, provided that the source is itself active. In this model, f is taken to be the expected number of influenced nodes as a result of the seed set being activated at the outset. Often times, g is taken to be the number of nodes in the set. A notable example when this is not the case is the field of viral marketing, in which one may spend variable amounts of resources at each node, hoping to start the largest possible cascade of influence in the graph using their given total resources.

Our work in particular will focus on the ICM and will take the simple g from above. We propose and implement two algorithms for finding high influence subsets of a graph. Our first algorithm is an optimization of an algorithm which is guaranteed to yield at least a factor of **(1-1/e)** times the optimal solution, while our second algorithm is entirely novel. We test them against the well known CELF algorithm on the *KDD CUP 2003 Dataset* in order to find the most influential high energy physics (theory) papers published on Arxiv in 2001, 2002 or 2003 and compare the algorithms' speeds and results.

2 Mathematical Background

A set function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is called submodular if for any $A \subseteq B \subseteq \mathcal{U}$ and any $s \in \mathcal{U}$,

$$f(A \cup \{s\}) - f(A) \geq f(B \cup \{s\}) - f(B).$$

The function f given by the ICM model has this property, which we prove in theorem 1 below. Given cost function $g(S) = |S|$ and submodular function f, the optimization problem (1) can be solved up to a factor of $(1 - 1/e)$ using the greedy algorithm.[5] See the pseudocode for the algorithm below:

In order to be able to apply the algorithm to our model with the guarantee of near optimality,

Algorithm 1 KK-Greedy(G,k)

```
1: $S \leftarrow \varnothing$
2: for $1 \le i \le k$ do
3:     $S \leftarrow S \cup \arg\max_{v \in V} f(S \cup \{v\}) - f(S)$
4: end for
5: Return $S$
```

we must demonstrate the submodularity of our function.

Theorem 1. *Let $G = (V, E)$ be a directed graph and let $p : E \to (0, 1]$ assign probabilities to its edges. The function $f : 2^V \to \mathbb{R}$ where $f(S)$ is the expected number of infected nodes under the Independent Cascade Model is submodular.*

Before proving this result, we note that the main principle behind the proof – a probability technique called coupling – is the basis of our two proposed influence maximization algorithms. In the proof, the random variables $Y(S)$ which depend on random subgraph G' are said to be coupled. Our algorithms sample such random G' to estimate marginal influences and find high influence sets of nodes.

Proof. Enumerate the edges of G as $E = \{e_i\}_{i=1}^N$ and let $X = (X_i, 1 \le i \le N)$ be a random vector with each coordinate i.i.d. $U((0, 1))$. Define $G' = (V', E')$ to be the random graph with $V' = V$ and $E' = \{e \in E : X_i \le p(e_i)\}$. Let $Y(S)$ be a family of random variables, each one associated with a subset $S \subseteq V$, where $Y(S)$ is defined to be the number of nodes that can be reached in G' from nodes in S. I claim that $f(S) = E[Y(S)]$. For any path from u to v, the probability of this path existing in G' equals to the probability of a successful chain of influence along the same path in G, because $P(X_i \le p(e_i)) = p(e_i)$. Then the probability that a node is reachable from S in G' is equal to the probability that an influence cascade reaches that node from S in G. This is sufficient for the claim to hold true. All that remains is to see that for any $A \subseteq B \subseteq V$ and any $s \in V$, $Y(A \cup \{s\}) - Y(A) \ge Y(B \cup \{s\}) - Y(B)$. The LHS of this inequality counts nodes in a particular instance of random G' reachable from s but not from A, while the RHS counts nodes in the same instance of G' that are reachable from s but not from B. Since $A \subseteq B$, the inequality must hold. Taking the expectation proves the theorem. □

We see now that the greedy algorithm gives a good approximation of the optimal solution in our case. The only problem is that it is still very slow.

3 Related Work

3.1 Cost-effective Outbreak Detection in Networks

Leskovec et al.[3] propose an optimization to the KK-Greedy algorithm for the influence maximization problem which exploits the submodularity of the problem. Although they use a slightly different objective function from ours, it is similar in that it is the expected value of a random process propagating across edges of the graph, which makes it challenging to compute. This makes naively applying the KK-Greedy algorithm extremely inefficient, because each computation $f(A)$ is expensive. One of the big results of the paper is "Lazy Forward" cost selection which reduces the number of computations of f needed to run the algorithm. This version of the algorithm is called CELF. The speed improvement is up to a factor of 700. Below is the pseudocode of this algorithm and a brief explanation of the main idea. It maintains a priority queue Q⟨u, u.mg, u.iter⟩ sorted by u.mg in decreasing order, where u is a node, u.mg is the most recently computed marginal influence of u, and u.iter is the iteration in which that marginal gain was computed.

If the node chosen in the main loop in line 9 of Algorithm 2 had marginal influence computed in the current iteration of S, then it must have the maximal marginal influence of all the other nodes, even if their gains were computed in previous iterations. This is because, by submodularity, the gains those nodes will have on the current set must be no larger than the gains on the smaller set from the previous iteration. In this way, one can eliminate many costly computations of f and still produce the same output as the KK-Greedy algorithm, which is at least a factor of $(1 - 1/e)$ times the optimal.

Algorithm 2 CELF(G,k)

```
1: $S \leftarrow \varnothing$
2: $Q \leftarrow \varnothing$
3: for $u \in V$ do
4:     u.mg$\leftarrow f(\{u\})$
5:     u.iter$\leftarrow 0$
6:     $Q \leftarrow$ (u, u.mg, u.iter)
7: end for
8: while $|S| < k$ do
9:     $u = Q[0]$
10:    if u.iter$== |S|$ then
11:        $S = S \cup \{u\}$
12:        $Q = Q \setminus \{u\}$
13:    else
14:        u.mg$= f(S \cup \{u\}) - f(S)$
15:        u.iter$= |S|$
16:        Resort $Q$
17:        $S \leftarrow S \cup \arg\max_{v \in V} f(S \cup \{v\}) - f(S)$
18:    end if
19: end while
20: Return $S$
```

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

Goyal et al.[1] proposes an improvement on CELF that further exploits submodularity, dubbing this new version of the algorithm CELF++. The main change is that the priority queue from CELF now stores Q⟨u, u.mg1, u.prev_best, u.mg2, u.iter⟩, where u.mg1 is the same as u.mg in CELF, u.prev_best is the best node found before u in the current iteration, and u.mg2 is the marginal gain of u with respect to $S \cup$ {u.prev_best}. The idea is that u.mg2 can be computed in the same Monte-Carlo simulation as u.mg1 is, so it is not costly to add. The payoff is that, when we run the main while loop from CELF, when u.iter$== |S| - 1$ and u.prev_best is the node added to S in the previous iteration, we have already computed the marginal gain of u in the current iteration so we can access it at u.mg2 instead of having to compute it.

In short, CELF++ reduces the number of times the objective function f needs to be computed compared to CELF by using one Monte Carlo simulation to compute several marginal gains simultaneously and storing this additional information to be used later. This algorithm has been tested by previous work and an improvement of 35-55% over CELF has been observed.

3.3 Improved Algorithms of CELF and CELF++ for Influence Maximization

Lv et al.[4] proposed a further improvement on the greedy approach to solving the influence maximization problem. Their optimization tried to predict nodes which will provide no marginal gain in future iterations because they will already be influenced by the current seed set. By keeping track of such nodes and removing them from the set of candidates to add to the seed set, Lv obtained a performance improvement. This optimization was applied to both CELF and CELF++ and compared to the original two algorithms on an Epinions dataset. Of the four algorithm variants, lv_CELF is reported to be the fastest.

What is most notable in this paper, aside from the Lv optimization itself, is that CELF consistently outperformed CELF++, despite the results of [1]. We find this result to be consistent with out work.

4 Coupling Approximation

The above papers primarily focus on decreasing the overall number of times that f must be computed through Monte-Carlo simulation. One thing that has not been given as much consideration is how to compute f more efficiently in the first place. We propose the following alternative to Monte-Carlo approximation of f, which we refer to as "coupling approximation," named after the probability technique used to prove theorem 1, which inspired this approach.

Given parameter N and a graph $G = (V, E)$ with influence probabilities $p : E \mapsto (0, 1]$, precompute i.i.d. random graphs $\{G_n\}_{n=1}^{N}$ according to the distribution of G' from the proof of theorem 1. To compute $f(S)$ for a given set S, we determine the number of nodes reachable from S on each G_n using breadth first search and take the average.

Given a set S and precomputed $\{G_n\}_{n=1}^N$, this algorithm is equivalent to a Monte-Carlo approximation with N iterations. The only difference between the two is the time at which random numbers are generated to determine, in the current simulation, whether influence passes along a given edge or not. This does not change the expected value of the output for a given iteration, so running N iterations of Monte-Carlo simulation approximates $f(S)$ exactly as well as the coupling approximation with parameter N.

Using the coupling approximation requires storing N graphs as adjacency lists, so it increases the memory complexity by $O(N(|E| + |V|))$. All optimizations of the KK-Greedy algorithm compute $f(\{u\})$ for every node $u \in G$ to find the first node (node 1) in the seed set, which means that we generate a random number for each edge in G at least once per iteration when we utilize Monte-Carlo simulation. This is exactly the computational complexity of finding $\{G_n\}_{n=1}^N$. However, using Monte-Carlo simulation means that the computations of f required for finding nodes 2 through k of the seed set necessitate further random edge crossing computations. It is in this way that the coupling approximation improves run time at the cost of memory.

5 Sampling with Recombination

The idea of using precomputed graphs $\{G_n\}_{n=1}^N$ to more quickly compute marginal influences inspired a wholly new potential solution for the influence maximization problem. The basic principle is to sample random graphs $\{G_n\}_{n=1}^N$ as before and run a variant of the greedy algorithm to find a high influence subset of k nodes on G_n, with the caveat that the probabilities assigned to edges on G_n are all 1, for all n. We use CELF to find the k nodes on a given G_n. Computing $f(S)$ on G_n no longer requires Monte-Carlo simulation because we are in a deterministic setting, so a breadth first search approach suffices for direct computation. Thus, we can efficiently sample seed sets $K_n = (u_1, \dots u_k)$ corresponding to G_n.

The next part is to recombine $\{K_n\}_{n=1}^N$ into one set of k nodes which is the output of the algorithm. There is some flexibility in how to combine n sets of size k into one. Our implementation naively orders nodes appearing in $\{K_n\}_{n=1}^N$ by number of appearances and selects the first k. We expect the runtime of this algorithm to far surpass the runtimes of CELF and other greedy variants, but there is no longer any mathematical guarantee on the optimality of the output set of k nodes. We check experimentally whether or not sampling and recombination can compare to CELF, which is guaranteed to be a factor of $(1 - 1/e)$ within the optimal, in performance.

6 Data Set

The data set we are using is the *KDD CUP Dataset* which contains information of high energy physics (theory) papers published on arXiv from 1992 to 2003. The primary metadata we use is the citations which are listed in the format of [paper cited from] [paper cited to] which provides a straightforward way to generate the network of all the papers. The full data set has 27770 nodes and 352807 edges. In order to more reasonably test our algorithms, we chose a subset of the original data set corresponding to the subgraph generated by papers written in 2001, 2002 and 2003. This trimmed data set results in a graph with 5647 nodes and 44727 edges. We also performed further experiments on smaller random graphs.

7 Preliminary Tests

We started by implementing several algorithms in python to determine which ones are worth exploring further. All tests, including our main results, were performed on a 13" MacBook Air with 2.7GHz Intel Core i5 processor with 8GB of RAM. This limited setup means that not all algorithms can proceed to the final, more extensive testing. The algorithms we wrote in python were the KK-Greedy algorithm, the CELF optimization, the CELF++ optimization, and CELF using coupling approximation instead of Monte-Carlo estimation.

All these algorithms need to compute the expected marginal influence of a node u given a seed set S. We implemented three versions of this computation: one for the case that every edge (u, v) guarantees that u infects v whenever u is itself infected, while the other two versions associate a probability of infection with each edge. In other words, when all probabilities assigned to edges are equal to 1, we can just use breadth first search to deterministically compute the marginal gain. We use the deterministic case to test our algorithms for accuracy on small graphs and to set rough benchmarks on the runtime of the random case. We also use this version of the computation extensively in coupling approximation.

When not all probabilities are 1, we use Monte-Carlo or coupling approximation. They both let the influence expand from S until it terminates, and then do the same for u, counting how many nodes u successfully influences which S would not have. If u is in S's influence set, in that particular run u would net a marginal gain of 0. We do Monte-Carlo for $N = 10,000$ iterations and the expected marginal gain $f(S \cup \{u\}) - f(S)$ is taken to be the average. We use $N = 10,000$ iterations following Goyal in his paper on CELF++.[1] As we noted before, the accuracy of the coupling approximation with parameter N is the same as that of N iterations of Monte-Carlo simulation, so we use parameter $N = 10,000$ for coupling approximation.

We tested these implementations on two Erdos-Renyi graphs, one with $|V| = 50, |E| = 625$ and the other with $|V| = 150, |E| = 1875$. We chose the ratio of $|E|/|V|$ to roughly match the ratio in the KDD Cup Dataset. The parameters of the influence problem are $k = 5$ and uniform probabilities of 0.1 for the source of an edge influencing the destination. This is the same probability that Goyal[1] assigns to edges in his experiments on citation networks. The runtimes are summarized in the table below in minutes. We use CELF_c to denote CELF where we use coupling approximation and CELF to denote CELF with Monte-Carlo estimation.

The fastest algorithm was CELF_c. Using coupling instead of Monte-Carlo estimation resulted in a 83% − 86% decrease in runtime. We were

	$(50, 625)$	$(150, 1875)$
KK-Greedy	6.2	49.6
CELF	5.2	37.0
CELF++	5.3	41.8
CELF_c	0.87	5.1

surprised to see that CELF++ was consistently slower than CELF. This conflicts with Goyal[1] but is consistent with Lv[4]. Based on the runtimes above, we decided to port two of these algorithms to C++ for more extensive testing and to ultimately run them on the KDD Dataset: CELF and CELF_c. We additionally implemented sampling and recombination in C++.

8 Results

Using coupling approximation resulted in a significant speed-up in CELF, as compared to using Monte-Carlo estimation. The precise speed-up depends on the parameter k – the more nodes we seek out, the more times we must estimate marginal influence. Below is the plot of runtime in minutes against number of papers selected in the trimmed version of the KDD Cup dataset:

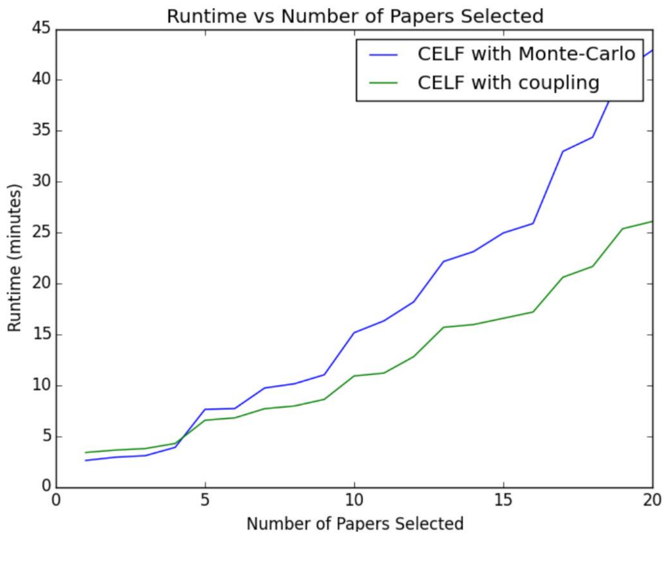

Figure 1: Plot of runtimes using Monte-Carlo and using coupling

Note that selecting just the first node or so is slightly less efficient with coupling approximation than with Monte-Carlo estimation because the precomputation of the graphs ends up being more intensive. For $k = 20$, using coupling resulted in a 39.2% decrease in runtime.

We also ran sampling with recombination on the trimmed dataset for $k = 10$ and $k = 20$. The following tables compare the runtimes in minutes and the influences of the output sets of nodes for these k. We label sampling with recombination as S&R.

	$k = 10$	$k = 20$
CELF	15.2	42.9
CELF_c	10.9	26.1
S&R	10.0	16.9

Figure 2: Runtimes

	$k = 10$	$k = 20$
CELF	597.284	785.864
CELF_c	597.436	786.35
S&R	576.487	774.415

Figure 3: Final Influence

Of note is that CELF produced the same high influence set of nodes no matter which influence estimation we used – the discrepancies in the final influence are due to the random error in estimating it. We see that sampling and recombination does provide a speed boost over CELF. This improvement is a lot more significant for the larger value of k. The cost in terms of the influence of the output set is about $1.5 - 3.5\%$ decrease from what a greedy approach yields. We conclude that sampling and recombination could be worth future investigation, especially since our recombination function could easily be improved.

We ran some further tests to determine that the runtime improvements and relative final influences were not just a product of our particular data set. Specifically, we generated 10 Erdos-Renyi graphs with 150 nodes and 1875 edges (the same as in the preliminary testing section) and ran all three algorithms on these graphs.

CELF using coupling approximation took **20%** less time on average than CELF with Monte-Carlo estimation. The standard deviation was 4.5 percentage points. Both algorithms generated the exact same set of nodes in every iteration – unsurprising because coupling and Monte-Carlo are mathematically equivalent – so the resulting influences were the same.

In comparison, sampling with recombination took on average **71.4%** less time than CELF (with Monte-Carlo estimation), with a standard deviation of 1.1 percentage points. However, sampling with recombination produced a final influence that was **1.4%** less than that given by a greedy approach, on average. The standard deviation was 0.8 percentage points.

9 Conclusion

The results on the random graph agree with the trends we saw on the trimmed KDD Cup 2003 data set. Using coupling instead of Monte-Carlo results in faster runtime, and sampling with recombination far exceeds CELF no matter how we estimate influence. However, with our naive recombination function, the last algorithm results in a slight decrease in influence of the final output.

10 Future Work

Our experiments suggest that sampling with recombination does between 96.5% and 98.5% as well as greedy approaches, while providing an undeniable runtime improvement. Our recombination algorithm naively took the most commonly appearing nodes from the samples. It is possible that a more intelligent approach to selecting the final k nodes from the N samples could yield a higher influence set while maintaining the same runtime advantage. In our future work, we will implement and test different recombination methods to see if the performance can improve past greedy approaches. We do not have any mathematical guarantees for sampling with recombination. Thus, an additional direction we will take is to try to find a recombination function which can result in some form of optimality guarantees. If we are unable to do this, more extensive testing of the algorithm on different kinds of datasets would be useful in determining the reliability of the approach.

Another thing that could bear more extensive testing is how k affects the speed boost that coupling approximation provides over Monte-Carlo estimation. We saw on the trimmed KDD Cup Dataset that coupling gets better as k increases.

11 Work Division

For purposes of grading this project, Dai and Boris have agreed that they have done equal work on the project.

References

[1] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, pages 47–48. ACM, 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. ACM, 2003.

[3] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

[4] Jiaguo Lv, Jingfeng Guo, Zhen Yang, Wei Zhang, and Allen Jocshi. Improved algorithms of celf and celf++ for influence maximization. *Journal of Engineering Science and Technology Review*, 7(3):32–38, 2014.

[5] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.