CS224W: Regarding spread of influence and its maximization in Social Networks

Sameep Solanki, Jim Liu, Xiongbing Ou

Keywords: influence maximization, spread of behavior, performance algorithms, heuristics

1. Introduction

Understanding the spread of influence and how to maximize it in social networks is an important and relevant topic in social network analysis. There are numerous practical business applications such as viral marketing and understanding where to seed stories to maximize impact. Existing techniques exist for influence maximization such as the independent cascade model and use of the greedy hill climbing algorithm. These techniques, however, are run on graphs that are static. Additionally, many of these algorithms are very expensive to run. In today's rapidly changing online environment, graphs are seldom static and also the cardinality of the seed set could change. Even a short delay in analysis of this evolving graphs can lead to unfavorable results.

Our projects will investigate influence maximization techniques on dynamic graphs as well as dynamic cardinality of seed nodes We will look at potential optimizations to algorithms such as greedy hill climbing and evaluate their performance on different graphs. We will present our findings on potential time improvements with a modified version of Greedy Algorithm particularly when the seed set size increases. We looked at both a random graph and a particular dataset to validate the performance of our algorithm.

2. Prior Work

Influence maximization is the problem of finding a small subset of nodes (seed nodes) in a social network that could maximize the spread of influence. Domingos and Richardson [1, 2] are the first to study influence maximization as an algorithmic problem. Their methods are probabilistic, however. Kempe, Kleinberg, and Tardos [3] are the first to formulate the problem as the discrete optimization problem. A social network is modeled as a graph with vertices representing individuals and edges representing connections or relationship between two individuals. Influence are propagated in the network according to a stochastic cascade model. Given a social network graph, a specific influence cascade model, such as the independent cascade model, the weight cascade model, and the linear threshold model [3], and a small number k, the influence maximization problem is to find k vertices in the graph (referred to as seeds) such that under the influence cascade model, the expected number of vertices influenced by the k seeds (referred to as the influence spread in the paper) is the largest possible.
Kempe et al. prove that the optimization problem is NP-hard, and present a greedy approximation algorithm applicable to all three models, which guarantees that the influence spread is within (1 – 1/e) of the optimal influence spread. They also show through experiments that their greedy algorithm significantly outperforms the classic degree and centrality-based heuristics in influence spread.

There has also been some work done with time-dependent graphs by Wang et. al [6]. They defined a pairwise factor graph with two components - a set of observed variables corresponding to users and a set of hidden variables representing which node a user chooses to follow. For the dynamic case, the PFG has additional time window information and the series of dynamic graphs forms a Markov chain. Using a probabilistic learning algorithm, Wang et al were able to find the most influential researchers in a co-author network.

3. Data Collection

The data set used for this problem was Memetracker data collected between August 2008 and April 2009. Each entry in the data set consists at a minimum of a document URL and timestamp. It is important to note that the entries are arranged chronologically. Optionally, each entry will have a series of phrases extracted from the document text and any hyperlinks in the document pointing to other websites. For our initial test set, we started with 10,000 URLs in the initial graph. Additional groups of 10,00 URLs were added to increase the graph size at each step.

To construct the graph itself, the raw text files were processed using python. Each document prefixed with "P" was parsed and the URL was converted to an integer which serves as a node in the graph. Each phrase prefixed with "Q" would be the other edge which forms as edge with the previous node. The phrases are stored in a dictionary to determine if any future URLs contain phrases from previous URLs. For cases where phrases repeat, we create an edge where the second node points to an existing node rather than a new node. This allows us to see which phrases generate the most edges across all URLs as time progresses.

Note that we are not utilizing the hyperlinks in the document itself. One can envision a case where we use the same process described above to a construct a graph for hyperlinks. Currently we are choosing to focus on phrases but hyperlinks may be an interesting area of observation in the future.

4. Mathematical Background

When we study the evolution of the seed nodes when the origin network grows, computing the influence maximization is the base of the project. The social network is modeled as an directed graph G = (V,E), with vertices in V modeling the individuals in the network and edges in E modeling the relationship between individuals. We use n to denote the number of vertices and m to denote the number of edges. Let S be the subset of vertices selected to initiate the influence propagation, which we call the seed set. Let RanCas(S) denote the random process of influence cascade from the seed set S, of which the output is a random set of vertices influenced by S. The algorithm takes the graph G and a number k as input and generate a seed set S of size k, with the intention that the expected number of vertices influenced by the seed set S, which we call influence spread, is as large as possible.

Though the search result of the traditional greedy algorithm can be guaranteed, its efficiency has a serious drawback. The limitation of the greedy algorithm is twofold: (i) The algorithm requires repeated

computation of the spread function for various seed sets. The problem of computing the spread under both the independent cascade model(IC) and the linear threshold model(LT) models is NP-hard. As a result, Monte-Carlo simulations are run 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.

Several recent studies aimed at addressing this efficiency issue. Considerable work has been done on resolving the first issue, by using efficient heuristics for estimating the spread. In [4], W. Chen et al. present a simplified method to estimate the spread. In this scheme, each edge is selected for propagation or not based on some propagation probability, and remove all edges not for propagation from the original graph G to generate a new graph G'. With this treatment, the spread function of S is simply the set of vertices reachable from S in G'. Let RG' (S) denote the set of vertices reachable from S in graph G'. It is convenient to obtain RG' (S) and RG' ({v}) for all vertices $v \in V$ with a linear scan of the graph G' by BFS. Therefore, by randomly generating G' for enough times, the spread function for various seed sets can be exactly estimated.

The most notable work to improve the quadratic nature of the greedy algorithm is [5], where Leskovec et al. present an optimization in selecting new seeds, which is referred to as the "Cost-Effective Lazy Forward" (CELF) scheme. The CELF optimization uses the submodularity property of the influence maximization objective to greatly reduce the number of evaluations on the influence spread of vertices. Their experimental results demonstrate that CELF optimization could achieve as much as 700 times speedup in selecting seed vertices, which is a very impressive result.

4. Algorithms

Our experiments show that the improved algorithm is still not quick enough to complete in a graph with a few tens of thousands of vertices in a reasonable span of time (say 20 mins) , so it is still not efficient for large-scale networks. So we have to further optimize the greedy algorithm. Our implementation inherits the ideas of [4] and [5], but with further simplification by reduction of repetitive processing.

4.1 Algorithm improvements when our seed nodes are increased.

When we add a new node to the seed set, the best candidate should be the node that can influence more nodes and there is no overlap with nodes influenced by new added node with any existing influenced nodes. In other words, if the new added node doesn't related to the existing seed sets, then the new added node can generate greater margin gain. So if we can estimate the influenced vertices set by each node directly, then we can evaluate the margin gain directly after add each new node. The following describes the detail of the optimized algorithm:

1:Based on the origin graph G and the propagation probability to generate the random graph G' for R times as in [4];

2: For each randomly generated graph G', obtain the influenced nodes set and the margin gain for each node v ∈V by BFS. Here let $I_r(v)$ denote the nodes influenced by node v for the r-th random graph G', and $|I_r(v)|$ denote the cardinality of $I_r(v)$, which is the marginal gain of node v for the the r-th random graph G';
3: Compute the average margin gain for each node: $|I(v)|' = (\sum|I_r(v)|)/R$, where r = 1, …, R;
4: Estimate the influenced vertices set by each node by two steps:

a) First generate the whole influenced vertices set I(v) by combining all $I_r(v)$ together, where r = 1, …, R; Let $I(v) = I_1(V) \cup I_2(V) \cup ... \cup I_R(V)$. Note that in I(v), we not only record the influenced vertices, but also record how many times that vertex occurs. So in fact I(v) can be represented as [(u1, times1), (u2, times2),...] where u1 and u2 are in the influence set of v and times1 denotes how many times u1 has been present and so on .

b) Rank the vertices in I(v) based on the occurring times, and then only keep the |Ir(v)|' vertices with the largest occurring times. The left vertices is the estimation of the influenced vertices set by node v. Here let I(v)' denote the estimation of the influenced vertices set by v, then its size is |I(v)|', which is obtained in step 2.

Until now we have got |I(v)|', the average margin gain for each node and I(v)' , the estimation of the influenced vertices set by node v. Based on these two type of value, we can find the seed sets S by the following steps:
5: s_0 should be the node v that maximize |I(v)|';
6: To find s_1, scan v ∈V, but $v \neq s_0$, to evaluate the margin gain use the following formula:
The margin gain after adding node v = $|I(v)' - (I(v)' \cap I(s_0)')|$.
That is to say, deducting the overlap between I(v)' and the influenced nodes set, $I(s_0)'$ to get the margin gain for node v. Then s_1 should be the node that generates the largest margin gain;
7: After find s_0 and s_1 find, try to find s_2, then scan v ∈V, but $v \neq s_{0,}s_1$, to evaluate the margin gain uses this formulas:
The margin gain after adding node v = $|I(v)' - (I(v)' \cap (I(s_0)' \cup I(s_1)')|$.
That is to say, deducting the overlap between I(v)' and the influenced nodes set, $I(s_0)' \cup I(s_1)'$to get the margin gain for node v. Then s_2 should be the node that generates the largest margin gain;
8: and so on to find $s_{3,}s_4$,..., until enough seed sets are found.

As described in step 5/6/7/8, we use the estimation of the influenced vertices set by node v to evaluate the margin gain after adding node v. This method can reduce the multiple times of random process to estimate the spread function for various seed sets in the traditional method. So it is very obvious that the proposed algorithm can shorten the running time.
The important point is that the proposed algorithm also can use the submodularity property of the influence maximization objective to further reduce the number of evaluations on the influence spread of vertices. In step 5/6/7/8, we can use the same ideas as CELF[5] to sort |I(v)|', and only find the candidate v from the top of the list and doesn't need scan all node in V.

The two metrics we compare the proposed algorithm with the existing algorithm are influence spread and running time. Firstly we evaluate our algorithms by random network and the initial experimental results show that our improved greedy algorithm achieves better running time comparing with the MixGreedyIC algorithm of [4], which is the combination of generating random graph G' to estimate the the spread

function for various seed sets and CELF, with matching influence spread; and more importantly when add the size of seed sets, the running time of the proposed algorithm doesn't change much, so it will be helpful to find larger seed sets or some quick initial estimate.

4.2 Algorithm improvements when new nodes are added to the Graph .

In the real world Scenarios Graphs tend to evolve over time . Lets take an evolving graph G . We take a snapshot of the graph at a particular time t (say G_t). Let it have N_t nodes. We have already computed the seed Set S_t for it . However as the time progresses new nodes are added and now the new total number of nodes in Graph is N_n. We assume that when the new Nodes it does not modify the graph structure of the previous Graph at G_t . Hence the adjacency matrix looks as per Figure 1
.Now the influence set of only those nodes in Gt which are reachable to a new nodes added would be changed . This would require BFS (following inlinks) from each of the new nodes . Lets say that this is Set W where $W \in V_t$.(Since the vertices in that reach new nodes cannot be greater that the total vertices V_t of the G_t Then based on the algorithm 4.1 we just need to recalculate the influence I(v) for $v \in W$. We can proceed exactly similar to algorithm 4.1 to get K seed nodes for Max influence. Hence instead of naively running algorithm on the new graph. We cache the results from the previous Graph and only recompute the influence of certain nodes of Graph .

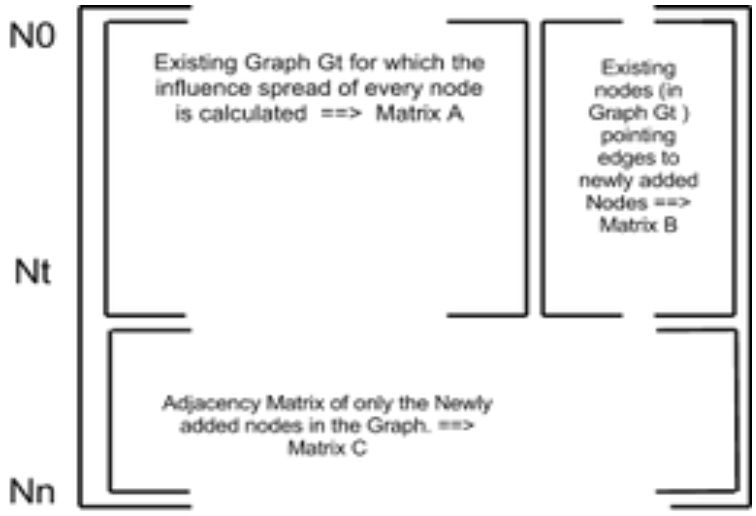


Figure 1: The innner matrix Nt * Nt represents the Snapshot of the Graph at time t. At time N it becomes Gn. The submatrices inside are labled according to what they represent .

Algorithm ::

1 : Compute the Seed Set S_t of Graph $G_t(V_t,E_t)$ as per algorithm 4.1 . Also store the individual influence values of each $v \in Vt$ i.e. I(v)

2:. Take another snapshot of Graph at time n (say G_n) . Calculate the new nodes added to the graph. Perform BFS (following inlinks) to get the set of Nodes from Gt that are reachable to the new nodes. Let this set be W

3: Generate the random graph G'_n for R times as in [4] using the propagation probability

4: . For each randomly generated graph G', obtain the influenced nodes set and the margin gain for each node $v \in W$ in the Vt and for each v outside of Vt .(These are the set of nodes whose influence set could be affected due to addition of new nodes.) Here let $I_r(v)$ denote the nodes influenced by node v for the r-th

random graph G', and $|I_r(v)|$ denote the cardinality of $I_r(v)$, which is the marginal gain of node v for the the r-th random graph G';

5: Compute the average margin gain for each node: $|I(v)|' = (\sum|I_r(v)|)/R$, where r = 1, …, R;

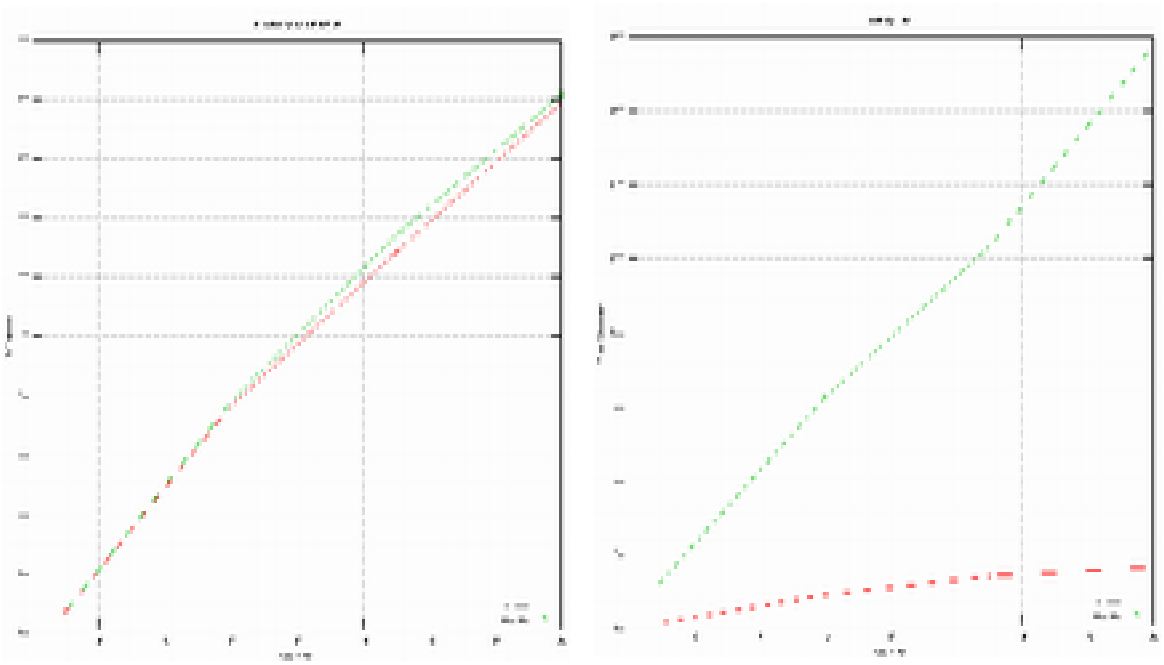
6: For nodes $v \notin W$ and $v \in Vt$ use the I(v) value that was computed during the time set St was generated. The influence set of these nodes would not have been changed

7: From this step onwards the algorithm follows the Steps 4 to 8 of the previous Algo 4.1

5. Findings

5.1 Influence spread and the running time based on the random graph

The following plot compares the experimental result for the influence spread and the running time. Here, we use SNAP to generate the random directed graph with nodes number 10,000 and 20000 edges .The value of R is 50 and IC mode is used,with propagation probability 0.1.

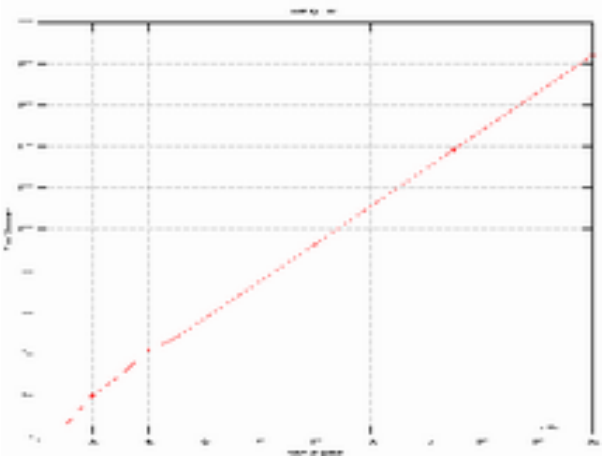


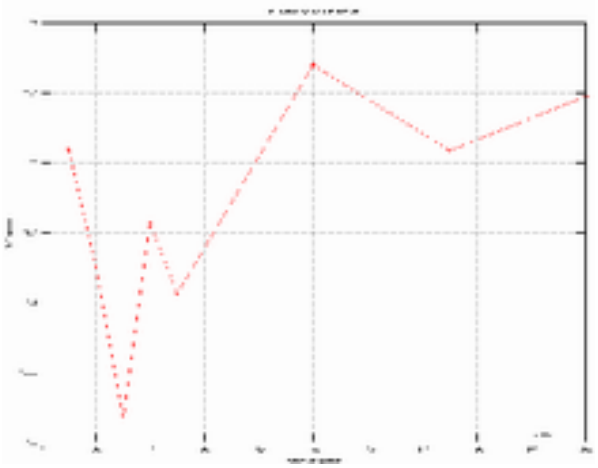
(a)influence spread when increasing seed size (b) running time when increasing seed size

The above two plots show that our proposed greedy algorithm achieves matching influence spread with the MixGreedyIC algorithm (in the plot FastGreedy means the MixGreedyIC, and MyGreedy means our proposed algorithm); when adding the size of the seed sets, the running time of the proposed algorithm is nearly stable, but the running time of the MixGreedyIC algorithm linearly increases with an increase in the size of the seed set.

5.2 Influence of the iteration times for the real data

Also we have compared the variation in Influence set S for different iteration times.We use the real Memetracker data with 10,000 nodes. We varied the value of R from 10 to 200 and independent cascade mode (IC)is used, the propagation probability is 0.1 and the size of the seed set(k) is 20.



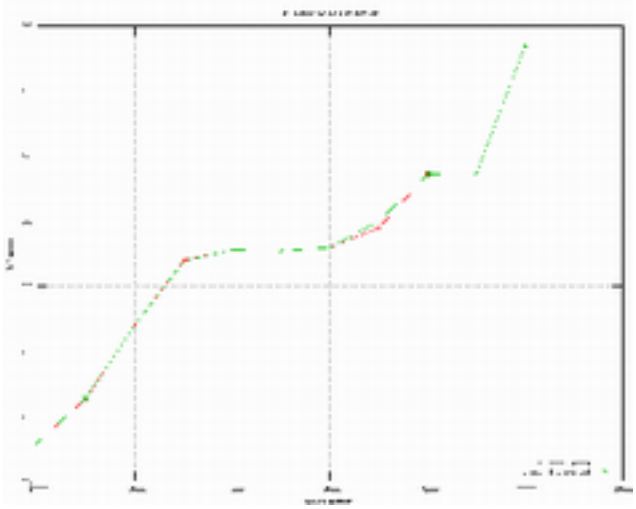


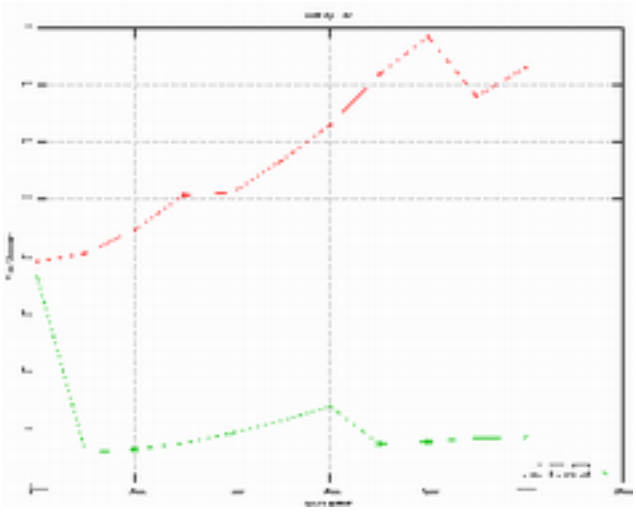
(a) influence spread (S) vs iteration times(R) (b) running time in seconds vs iteration times (R)

Based on the above diagram (a), we can see that the influence spread is between [450, 453] when the iteration times R increases from 10 to 200. This is because our real data belong to the bipartite graph, and it has no complex multiple-level influence propagation, then smaller iteration times R is enough to get the exact estimation result. Since the running time is nearly linearly incremented when increasing the iteration times based on the diagram (b), later we use R=50 for the real data simulation.

5.3 Influence spread and the running time based on the real data

At last we use the real Memetracker data to verify the influence spread when adding new nodes to the existing graph. Here we compare our initial proposed algorithm and the improved algorithm. For Memetracker data, the newly added nodes don't affected the influential nodes sets of the old node, so we can reuse the influential nodes sets of the old nodes and only re-estimate the influential nodes sets of the newly added nodes.(Matrix B in Figure 1 for our use-case is a zero matrix) For this simulation, first we get the initial graph with 10,000 nodes , then add 1000 new nodes to the existing nodes step by step. During the entire simulation, the size of the seed set is kept constant at 20. And the following is the simulation result:





(a) the influence spread when adding new nodes to the existing graph

(b) the running time when adding new nodes to the existing graph

In the above diagram (a) and (b), MyGreedy Total means we use our initial proposed algorithm to compute the influence spread and select the seed set based on the total graph. Every time when we add new nodes to the existing graph, we generate a new graph. MyGreedy Incremental means when add new nodes, only the influential nodes sets of the newly added nodes are re-estimated.

Based on the diagram (a), we can see the initial proposed algorithm and the improved algorithm can achieve the same influence spread. In diagram (b), we can see using the improved algorithm only adding about 30 seconds to get the new result when 1000 nodes to the existing graph. But it will take nearly 300 seconds to compute from the scratch again.

References

[1] P. Domingos and M. Richardson. Mining the network value of customers. In Proceedings of the 7th ACM SIGKDD Conference on Knowledge Discovery and Data Mining,pages 57–66, 2001.

[2] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data
Mining, pages 61–70, 2002.

[3] D. Kempe, J. Kleinberg, E. Tardos. Maximizing the Spread of Influence through a Social Network. SIGKDD, 2003.

[4] W. Chen, Y. Wang, S. Yang. Efficient Influence Maximization in Social Networks. In Proc. KDD, 2009.

[5] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 420–429, 2007

[6] C. Wang. J. Tang. J. Sun. J. Han. Dynamic Social Influence Analysis through Time-dependent Factor Graphs. International Conference on Advances in Social Network Analysis and Mining, 239-246.