Report One: Extreme Computing
Project-based High Performance Distributed and Parallel Systems
APCOMP 290R

Group CPU

October 16, 2015

Partners: Xiaowen Chang
Feifei Peng
Zelong Qiu
Qing Zhao
Instructor: Professor Pavlos

1 Introduction of the Problem

A social network is a platform for building social relations among people who share similar interests, activities, backgrounds or real-life connections. One of the most important functions of a social network is to carry the spread of information among people in the same social network. Also, the spread of information in social networks has become the focus of many studies.

Among all issues, finding a small subset of influential individuals in a social network such that they can influence the largest number of people in the network has become the most fundamental one. Formally, the problem is called as influence maximization, which is, for a parameter k, to find a k-node set with the maximum influence. For instance, a newly opened restaurant may plan to invite a small number of influential individuals of the Yelp users to eat for free, expecting that these invited guests will recommend the restaurant on Yelp, other users who have seen the reviews will try the restaurant and continue to influence more Yelp users in the network. Ultimately, the newly opened restaurant will benefit from inviting the right subset of influential individuals at the beginning.

We will be exploring Yelp Review Network in the U.S. in this report. To make the process easier, before scaling up the Influence Maximization Problem on larger network which contains over 350 thousand nodes and 4 million edges, we will work with North Carolina reviewer network which contains 240 thousand nodes. Or we will work with a even smaller network, a sample of North Carolina reviewer network which contains 240 nodes and 920 edges.

2 Method

2.1 Uncertainty on the influence function F

2.1.1 Find the appropriate N

The parameter N for the Influence Function is the number of times running Independent Cascading. In theory, as N gets larger, measuring the influence of nodes becomes more accurate. However, there is a trade-off here: running large N is timing consuming and inefficient, but small N will lead the resulting influence fluctuate widely.

2.1.2 Define the search depth

Instead of expending all the branches to the very end starting from the initial node set, we can set the maximum level we search, the search depth. By doing this, the algorithm is more efficient when dealing with those nodes that has extremely deep vertical connections. The depth should be set to an appropriate value. If the depth is too small, we may miss many nodes that are potential good candidates; on the contrary, a large depth will cause the

algorithm to be time consuming. For this problem, setting a depth does not have a large effect, so we decide not to set search depth. But the concept of search depth can be applied to other problems.

2.2 Naive, Greedy and SA optimization methods

2.2.1 Influence Function

After applying the independent cascading function, we can get the influence that the starting nodes have on the whole networking. However, the result may not be accurate enough, so we repeat the same process N times and get the mean of these N numbers. By doing this, we can get a more accurate result with respect to a certain set of nodes' influence.

2.2.2 Greedy Algorithm

A greedy algorithm is a mathematical process that looks for optimal solution. Based on the current state, it simply decides which next step will provide the most obvious benefit. This algorithm doesn't consider the larger problem as a whole. Once a decision has been made, it is never reconsidered.

1)For node i in the node set
  a.Compute the combined influence of current node and node i
  b.If new influence > max influence, set new influence as max influence, and node i as max node
2)Return max node

2.2.3 Simulated Annealing

The problem we want to solve is what combination of 3 nodes in the graph has the maximum influence. Besides using greedy algorithm, we could also use simulated annealing method to get the 3 nodes. The function we are trying to maximize is the influence function. The following is brief description of our algorithm.

1)Initialize X_0
2)For i through imax:
  a.Find a new x*
  b.Put x* and x to the influence function and get the value
  c.Calculate the value of exp((*newvalue-oldvalue*)/T) as D
  d.Generate u from uniform distribution (0, 1)
  e.If u < D, accept x^* as the new x_i
  f.Update T and L with new x_i

The challenge here is how to set the parameters of Simulated Annealing:

a. **Initialisation**
   It is common to start the simulated annealing algorithm with a random node and let the simulated annealing process improve on that. However, it would be a better choice to start with a node that has been heuristically selected. Here the node heuristically selected is the Naive estimator.

b. **Neighborhood Selection**
   Now in our simulated annealing method, we replace one node by a randomly chosen node in the filtered graph. Another way of doing this is setting level relationship among nodes, and replacing nodes with intended level neighbors in the graph.

c. **Initial Temperature(T)**
   The probability of accepting a worse move is a function of both the temperature of system and the change in the influence function. As the temperature increases, the probability of accepting a worse move is increased. As a result, the starting temperature is very important thus must be high to enable enough flexibility. If we do not do that, the ending result will be very close to even the same as the starting solution. However, on the other side, if the temperature starts at a too high level, the search will be able to move to any neighbor, leading the search to be too random. We need to find a good starting temperature to control the acceptance rate.

d. **Cooling Schedule**
Cool is used to update T(temperature). The way we adopt to decrease the temperature is critical to the success of convergence. There are two popular ways to cool temperature:

1) Using a constant number(reanneal) of iterations at each temperature:

* Linear: temperature decreases as $T_{i+1} = \alpha T_i$, where $0.8 \leq \alpha \leq 0.9$
* Exponential: temperature decreases as α^i, where $0.8 \leq \alpha \leq 0.9$
* Logarithmic: temperature decreases as $\frac{1}{log(i)}$

The drawback for this choice is it is very time consuming because it requires large number of iterations.

2) Only do one iteration at each temperature, but let the temperature decrease very slowly:
The formula used is t = t/(1+βt), where β is a small value.

e. **reheat**
The basic idea of reheating is that, once simulated annealing reaches a very low temperature, it is difficult to escape from local maximum, because the probability of accepting a move is very low. Reheating is the idea of increasing the temperature again to help escape from the local maximum.

f. **iterr**
Iterr is the total number of node sets that we run the influence function. As iterr increases, the curve is more likely to converge.

2.2.4 Naive Estimator

Naive estimator is given by running influence function on all the nodes in a graph to get the top three influential nodes. Naive estimator provides a good standard for comparing our result with, that is, our result should be at least as good as the influence of the naive estimators. The ideal naive estimator would be the set of top three influential nodes.

2.3 Parallelization

Rewrite influence function for parallel computing
Instead of dividing a graph into different partitions and conducting BFS on partitioned nodes, our first intuition is to parallel the computation in influence function, that is, we send the graph to each worker and each worker performs cascade functions multiple times (N/ partition number) to compute nodes' influence, and we reduce the results by computing average to get the final influence of the initial node set.

2.4 Graph

In the previous Naive and Greedy algorithms, we randomly choose nodes from the whole graph and in the previous simulated annealing algorithm, we randomly replaces one of the three nodes with an element in the whole graph to create a new solution set. There are two ways improving this selection mechanism to get a faster converge speed.

a. **Max Influence Node Heuristic**
For each node in a graph, we run the influence function and sort the nodes by their influence. Then we filter the graph by the rank of the influence. For Greedy algorithms, the idea is to choose nodes from the filtered graph instead of the original whole graph. For simulated annealing, when we want to replace a node in the three node set with a new node, instead of randomly choosing a node from the whole graph, we now only choose nodes from the filtered graph. Problem with this improvement is that running influence function on each node in a graph is that sometimes it is very time consuming when applying to very big graphs such as U.S. graph. As a result, we will introduce another heuristic method to deal with the graph.

b. **Max Potential Influence Node Heuristic**
We now propose a new way of filtering the node sets to choose from. If a node has many neighbors (or successors), we think of it as having the "potential" of influencing more nodes. So we filter the whole graph by checking the number of neighbors connecting to each node, and only keep those nodes with more than n neighbors. For Naive method, we apply the influence function to the three nodes in the filtered graph instead of the whole graph. For Greedy and Simulated Annealing algorithms, we choose nodes from the filtered graph instead of the original whole graph.

One important point about the above heuristic methods is that they not only improve the performance of Simulated Annealing algorithm, but also they help improve the efficiency of Naive and Greedy algorithms. The reason for this is very straightforward: using the above heuristic methods will reduce the number of nodes we choose from and make it more possible to getting a node with big influence. Note that this method may filter out some potential good candidates and is not very meaningful for relatively small graphs.

3 Experiment Results

3.1 NC mini

3.1.1 Naive Method

NC mini graph has only 240 nodes, thus we did not apply the heuristic method mentioned in the section 2.4. We use influence function to get the influence of each node and pick the top three influential nodes.
Notice that when using influence function to calculate the influence, we need to set an appropriate N such that the influence we get is accurate enough but, at the same time, N cannot be too large in case the running time is too long. To get an appropriate N, we run influence function with different N values each for 100 times and plot the standard deviation of these 100 result for each N.
From the plot attached, choosing N equal to or larger than 250 will make standard deviation less than 1. But we finally choose N to be 100 such that the standard deviation is relatively small and the influence function is efficient.

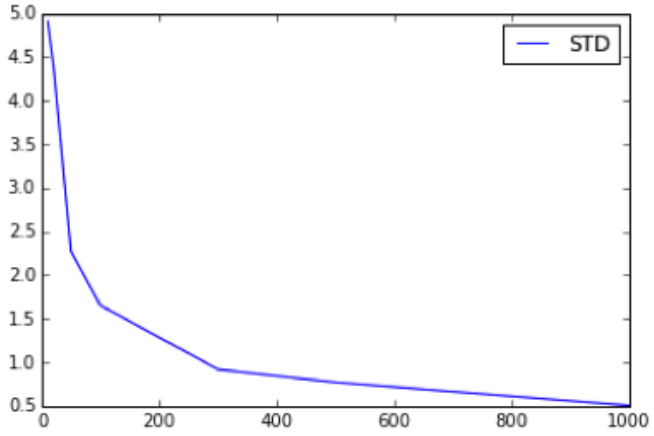

Figure 1: NC mini: standard deviation of 100 influence function runs for different N's

After doing as we mentioned before, we got the top three influential nodes as:
$[ts7EG6Zv2zdMDg29nyqGfA, NzWLMPvbEval0OVg_YDn4g, M-TwsqjrGVH9-qyw2KcvdQ]$
The influence of this node set is 42.
This influence also set a good standard for comparison for the upcoming experiments. That is, the three node set's influence we find later must be at least as large as 42.

3.1.2 Greedy Algorithm

For the same reason in the Naive method, we did not filter the graph using the heuristic methods. We first pick up the node with the max influence in the 240 nodes, then calculate the combined influence of this node with all the

remaining 239 nodes. From which we pick up the two node set with largest influence. Then we repeat this process starting with the most influential two node set we found before to find the most influential three node set.
Here we run the Greedy algorithm 100 times with different parameter N values for the influence function and calculate the ratio of minimum and maximum. The ideal ratio should be larger than 0.67. From the plots attached, we found N=100, 200, 500 all works well since the ratios are all larger than 0.67. Even when N=100, the ratio is already 0.8239, so running influence function with N=100 could give us pretty accurate results.

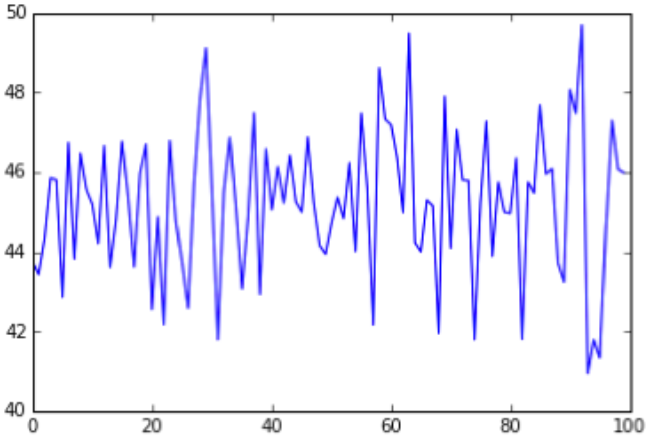

Figure 2: N = 100 ratio = 0.8239

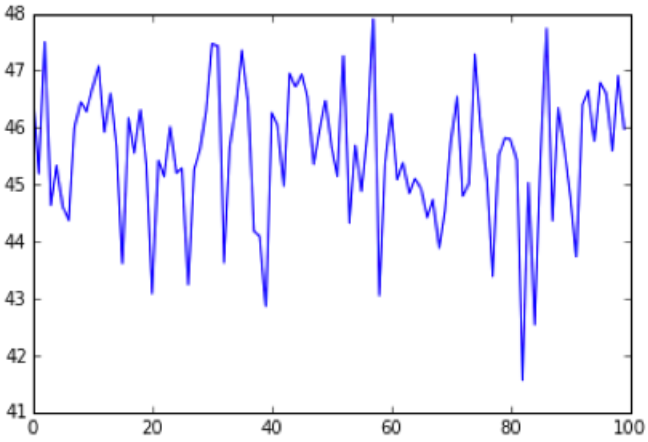

Figure 3: N=200, ratio=0.8541

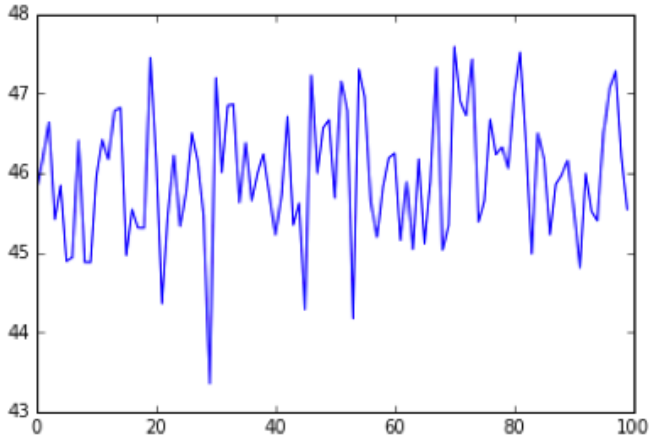

Figure 4: N=500 ratio = 0.9095

From the three plots above, we can also see that the influence of the solution sets given by the Greedy algorithm are mostly above 42, which beats the Naive solution.

3.1.3 Simulated Annealing

- **Initialisation**
  It is common to start the simulated annealing algorithm with a random node and let the simulated annealing process improve on that. However, it would be a better choice to start with a node that has been heuristically selected. Here the node heuristically selected is the Naive estimator [$ts7EG6Zv2zdMDg29nyqGfA$, $NzWLMPvbEval0OVg_YDn4g, M-TwsqjrGVH9-qyw2KcvdQ$].

- **N**: As we discussed before, N=100 in the influence function is good enough for NC mini. So we will continue to use N=100 for the simulated annealing algorithm.

- **Initial Temperature T**: First we need to choose an appropriate T. If T is too small, it will be hard to move to the neighborhood. On the other hand, a larger T will lead the search to be too random. Here we choose a relatively big T = 4. This can be seen from the plot that at the beginning, the range of the graph is big, which reflects the fact that the acceptance rate is relatively large.

- **Iteration**: We use iteration to stop the process. Iteration is the total number of tries in the simulated annealing algorithm. We first choose iter to be a common value of 10000. If not big enough, we increase the iteration. Actually we find 10000 is enough for this problem.

- **Cool**: Cool is normally between 0.8 and 0.9. We did two experiments with cool = 0.8 and cool =0.9. The graphs for cool = 0.8 and 0.9 are attached here. We found that when cool is 0.8, it is more likely to converge.

- **Cooling Schedule**: Since our iteration number is 10000 which is big enough, we used a constant number(reanneal) of iterations at each temperature. And we used the most popular temperature decrement way, that is, linear decrement. As for reanneal, we choose the typical reannealing interval = 100.

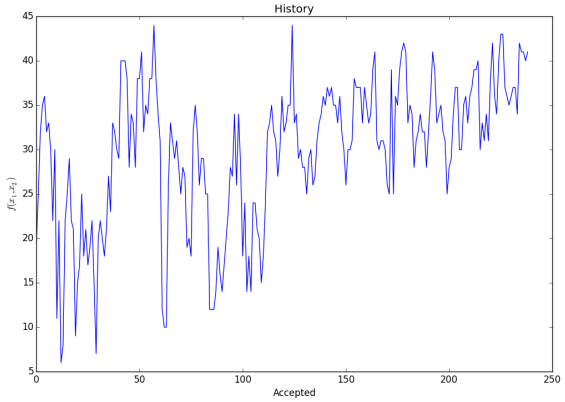

Figure 5: NC mini: T = 4, cool = 0.8, reanneal = 100, iterr = 10000

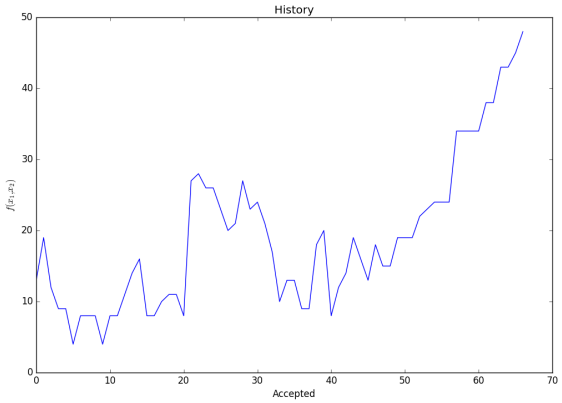

Figure 6: NC mini: T = 4, cool = 0.9, reanneal = 100, iterr = 10000

After our exploration, we decide to choose T = 4, cool = 0.8, reanneal = 100, iterr = 10000.

3.2 NC full

3.2.1 Naive

NC full graph has 24K nodes. Our first intuition is calculating the influence of all the nodes in the graph and choose the top three nodes. Problem with doing this is that running influence function on each node in the whole graph is very time consuming. Even if we run influence function with N = 10, it takes 5 seconds on each node, thus leading to a total running time of 5s*24,000 = around 1.4 days. So we used the Max Potential Influence Node Heuristic method mentioned in 2.4.b.

We first filter the whole graph by checking the number of neighbors connecting to each node, and only keep those nodes with more than 20 neighbors. This gave us a set of 921 nodes. Then we apply the influence function to the three nodes in the filtered graph instead of the whole graph.

Notice that when using influence function to calculate the influence, we need to set an appropriate N such that the influence we get is accurate enough but, at the same time, N cannot be too large in case the running time is too long. To get an appropriate N, we run influence function with different N values each for 30 times and plot the standard deviation of these 30 results for each N.

From the plot attached, choosing N equal to or larger than 450 will make standard deviation less than 5. But we finally choose N to be 100, in which case the standard is about 10, such that the standard deviation is relatively small and the influence function is efficient.

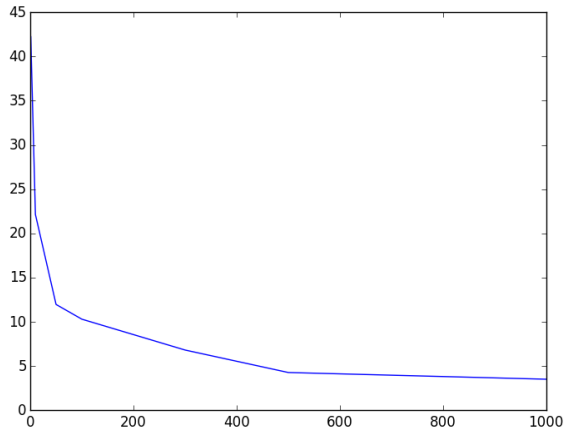

Figure 7: NC full: standard deviation of 30 influence function runs for different N's

After doing as we mentioned before, we got the top three influential nodes as:
$[PM2jXrlVzik1jDwwahLJJQ, eL - pnXODdyMwABMmLOrwKA, Kybm9SJyOfEgDAXFUPRdCQ]$
The influence of this node set is 8958.
This influence also set a good standard for comparison for the upcoming experiments. That is, the three node set's influence we find later must be at least as large as 8958.

3.2.2 Greedy Algorithm

For the same reason in the Naive method, we filtered the graph using the heuristic methods. We first pick up the node with the max influence in the the subset of 921 nodes, then calculate the combined influence of this node with all the remaining 920 nodes. From which we pick up the two node set with largest influence. Then we repeat this process starting with the most influential two node set we found before to find the most influential three node set. Even if we have already filtered the graph, Greedy is still very time consuming.
The solution set that Greedy Algorithm gave us is
$[_X0Xt2z5Q7GESxxRzvy4_g, 9P_5UGjc3t3Bo_yJ5A7iAQ, njfknxaBfd2K3uw_i0a6Uw]$
The influence of this solution set is 8961, which is slightly larger than the Naive solution.

3.2.3 Adapted Simulated Annealing

Because NC full graph has 24k nodes, it will be very time consuming if running 10000 or more iterations. As a result, we may consider reducing the number of iterations. However, it would be hard to converge if the iteration number is not big enough. So we figured out another way mentioned in section 2.4.b to make it more easier to converge. The method is to do only one iteration at each temperature, but let the temperature decrease very slowly. The formula used here is $t = t/(1 + \beta t)$, where β is a small value. Besides, we will add a reheat mechanism to the simulated annealing algorithm. Reheating is basically the idea of increasing the temperature again when it becomes too low to help escape from the local maximum. We will called the adapted method Adapted Simulated Annealing.

When apply simulated annealing to the NC full graph, we filtered the graph using the heuristic methods as we have done with Greedy Algorithm. If a node has many neighbors (or successors), we think of it as having the "potential" of influencing more nodes. So we filter the whole graph by checking the number of neighbors connecting to each node, and only keep those nodes with more than n neighbors. When replacing one node of the origin three node set with a new node, we choose the new node from the filtered graph in stead of the original whole graph.

- **Initialisation**
  It is common to start the simulated annealing algorithm with a random node and let the simulated annealing process improve on that. However, it would be a better choice to start with a node that has been heuristically selected. Here the node heuristically selected is the Naive estimator $[PM2jXrlVzik1jDwwahLJJQ, eL-pnXODdyMwABMmLOrwKA, Kybm9SJyOfEgDAXFUPRdCQ]$.
- **N**: As we discussed before, N=100 in the influence function is good enough for NC full. So we will continue to use N=100 for the simulated annealing algorithm.
- **Initial Temperature T**: First we need to choose an appropriate T. If T is too small, it will be hard to move to the neighborhood. On the other hand, a larger T will lead the search to be too random. Here we choose a relatively big T = 100. This can be seen from the plot that at the beginning, the range of the graph is big, which reflects the fact that the acceptance rate is relatively large.
- **Iteration**: We use iteration to stop the process. Iteration is the total number of tries in the simulated annealing algorithm. As discussed before, we cannot choose iter to be the common value of 10000, which is very time consuming. So we first try 1000 iterations, but it ended up not converging enough. We finally use 2000 iterations.
- **Cooling Schedule**: As discussed before, instead of using a constant number(reanneal) of iterations at each temperature, we used one iteration at each temperature, letting the temperature decrease very slowly. The formula used is $t = t/(1 + \beta t)$, where β is 0.005.
- **Reheat**: Reheating the temperature again when it becomes too low to help escape from the local maximum. Here we reheat after continuously unaccepting 200 nodes.

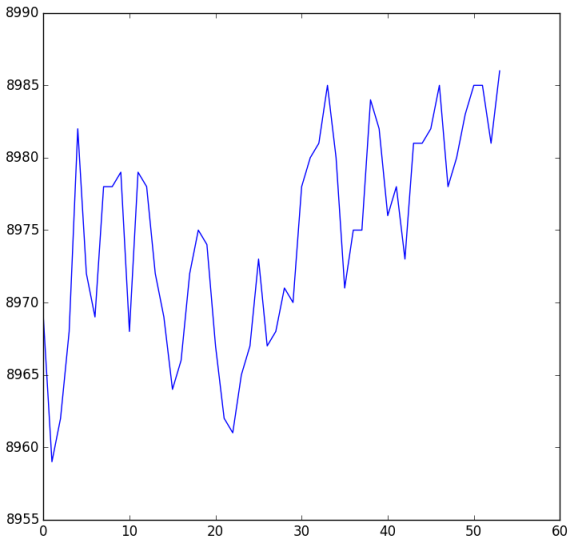

Figure 8: NC full: T = 100, β = 0.005, iterr = 2000

After our exploration, we decide to choose T = 100, β = 0.005, iterr = 2000. After annealing, the influence for the solution set is 8986, which beats the Naive solution. The solution set is $[u'gRKiX_9JUqi4Jy8KjtsehA'$, $u'2KLg9BGxebym4neLPd9dNg', u'R6lRupz_Jo51n8WJGJYdOQ']$

3.3 US

3.3.1 Strategy Selection

We discussed three methods above: Naive, Greedy and Simulated Annealing methods. Naive method can only give us an appropriate estimate or starting point, so we will not use Naive method to get the final result. But since it

is not that time consuming, we run it first to get standard and the result we get later must be better than what Naive method gives.
Greedy algorithm based proposals are able to achieve good accuracy. However, they are very slow on large scale social networks. Since Simulated Annealing algorithm can escape the local maximum and can be applied flexibly by adjusting the parameters, we finally choose to use Simulated Annealing algorithm, specifically Adapted Simulated Annealing here, to solve the U.S. graph.

3.3.2 Balancing Noise vs Runtime Tradeoff in Selecting N

The parameter N for the Influence Function is the number of times running Independent Cascading. In theory, as N gets larger, measuring the influence of nodes becomes more accurate. However, there is a trade-off here: running large N is timing consuming and inefficient, but small N will lead the resulting influence fluctuate widely.
As a result, we run influence function with different N each for 30 times and record the standard deviations for different N's. From the plot attached, choosing N equal to or larger than 200 will make standard deviation less than 10. But this is extremely time consuming. Since in the U.S. graph, the average influence of each node is very large, about 180K, a relatively big standard deviation like 60 is reasonable. So we finally choose N to be 10 such that the influence function is still efficient.

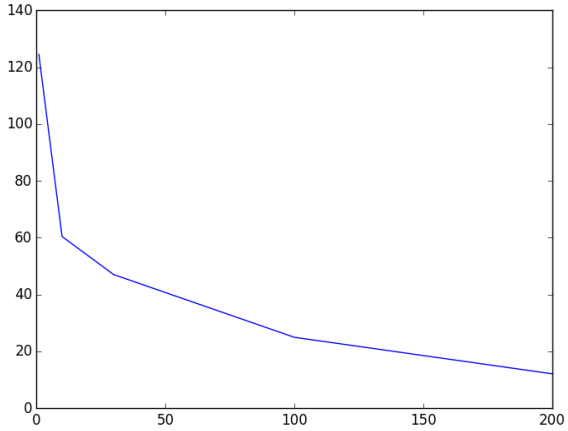

Figure 9: US: standard deviation of 30 influence function runs for different N's

3.3.3 Define the search depth

Instead of expending all the branches to the very end starting from the initial node set, we can set the maximum level we search, the search depth. By doing this, the algorithm is more efficient when dealing with those nodes that has extremely deep vertical connections. The depth should be set to an appropriate value. If the depth is too small, we may miss many nodes that are potential good candidates; on the contrary, a large depth will cause the algorithm to be time consuming. Like we discussed in section 2, for this problem, setting a depth does not have a large effect, so we decide not to set search depth. But the concept of search depth can be applied to other problems.

3.3.4 Graph Filtering

Since U.S. has 350k nodes, it is extremely time consuming to run over the whole graph. Thus, we use the same heuristic method to filter the graph as in the NC full case. Specifically, we used the Max Potential Influence Node Heuristic method mentioned in 2.4.b. We first filter the whole graph by checking the number of neighbors connecting to each node, and only keep those nodes with more than 200 neighbors. This gave us a set of 1642 nodes.

3.3.5 Naive Estimator

Naive estimator provides a good standard for comparing our result with, that is, our result should be at least as good as the influence of the naive estimators. The ideal naive estimator would be the set of top three influential

nodes. But like in the previous section 3.3.4 said, computing influence function on all the nodes in the U.S. graph is very time consuming and unrealistic. So we applied the influence function to the nodes in the filtered graph (1642 in our case) and choose the top three influential ones.
Using the above method, we find the naive estimators as below:
$[u'Yw - Q_4QrwWffjnHWLvo4kw', u'DcgO7qiYKS2VuAJj2dQpcg', u'EGiAtB4sgZhDdYpRkDneig']$
The influence of this node set is 184118.

3.3.6 Adapted Simulated Annealing Parameters

In the simulated annealing algorithm, we plot the accepted node sets versus their influence that calculated from the influence function. We would like to have the cure finally converge, for which we can achieved by adjusting the parameters.

- **Initialisation**: It is common to start the simulated annealing algorithm with a random node and let the simulated annealing process improve on that. However, it would be a better choice to start with a node that has been heuristically selected. Here the node heuristically selected is the Naive estimator $[u'Yw - Q_4QrwWffjnHWLvo4kw', u'DcgO7qiYKS2VuAJj2dQpcg', u'EGiAtB4sgZhDdYpRkDneig']$.
- **N**: As we discussed before, N=10 in the influence function is relatively acceptable for U.S. graph. So we will use N=10 for the simulated annealing algorithm.
- **Initial Temperature T**: First we need to choose an appropriate T. If T is too small, it will be hard to move to the neighborhood. On the other hand, a larger T will lead the search to be too random. Here we choose a relatively big T = 500. This can be seen from the plot that at the beginning, the range of the graph is big, which reflects the fact that the acceptance rate is relatively large.
- **Iteration**: We use iteration to stop the process. Iteration is the total number of tries in the simulated annealing algorithm. As discussed before, we cannot choose iter to be the common value of 10000, which is very time consuming. So we choose 1000 iterations, which tends out to converge.
- **Cooling Schedule**: As discussed before, since we can only use limited iterations, instead of using a constant number(reanneal) of iterations at each temperature, we used one iteration at each temperature, letting the temperature decrease very slowly. The formula used is $t = t/(1 + \beta t)$, where β is 0.001.
- **Reheat**: Reheating the temperature again when it becomes too low to help escape from the local maximum. Here we reheat after continuously unaccepting 200 nodes.

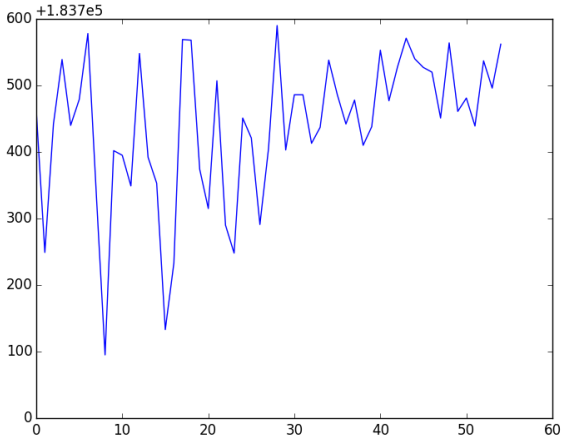

Figure 10: US: T = 500, β = 0.001, iterr = 1000

After our exploration, we decide to choose T = 500, β = 0.001, iterr = 1000.

The resulting node set of simulated annealing algorithm is [$zTWH9b_ItSdLOK9ypeFOIw$, $NvDR3SPVPXrDB_dbKuGoWA$,$ikm0UCahtK34LbLCEw4YTw$].
The influence of the resulting node set is 184201.

4 Conclusion

The resulting node set of simulated annealing algorithm is [$zTWH9b_ItSdLOK9ypeFOIw$, $NvDR3SPVPXrDB_dbKuGoWA$,$ikm0UCahtK34LbLCEw4YTw$].

The influence of the resulting node set is 184201. The influence of the Naive solution is 184118. Our solution by far is only a little better than the Naive solution. The main reason for this is that we do not have enough computing power and good equipment. If we had better hardware, we could:

Increase N: Now we use N=10, which leads to the standard deviation of influence to be around 60. This is relatively big, which may damage the accuracy of the algorithm. If we could control the standard deviation to be 5, for which N is greater than 500, we will have a much more accurate result.

Enlarge Solution Space: Since we filtered the graph, when we select nodes from the graph, we limited our selection to 1642 nodes, which can effectively speed up the convergence. However, this limits our solution set to be within the 1642 node set, which will reduce the possibility of getting a dramatic improvement

Improve Neighborhood Selection Method: Now in our simulated annealing method, we replace one node by a randomly chosen node in the filtered graph. Instead of simply doing this, we can set level relationship among nodes, and replacing nodes with intended level neighbors in the graph. This will bring us more possibility and flexibility.

We can further explore more improvement about many aspects of simulated annealing algorithm, such as the influence function, the acceptance criteria, etc., which needs more research and mathematical calculations and proof. We will not discuss further here.

5 Attachment

5.1 Independent Cascade Function

```
# Cascade Function
def cascade(init_nodes, nodes_set):#, dist_d):
    #nodes_set = nodes_set_broadcast.value
    action = {}
    n = len(init_nodes)
    #np.random.seed(random_d)
    #init_nodes = np.random.choice(NC_digraph.nodes(), 1)[0]
    for i in init_nodes:
        action[i] = 1
    #st = set()
    #st.add(init_nodes)
    init_list = zip([0]*len(init_nodes),init_nodes[::])
    inter = 0
    while len(init_list) != 0 and inter <10 :
        curr_node = init_list.pop(0)
        #print curr_node
        for i in nodes_set[curr_node[1]]:
            if i not in action:
                b = nodes_set.node[i]['review_count']
                a =  nodes_set[curr_node[1]][i]['weight']
                #np.random.seed(12)
                b_dist = np.sqrt(np.random.beta(a = a, b = b))
                #np.random.seed(12)
                u_dist = np.random.uniform(0,1)
                if b_dist > u_dist:
                    action[i] = 1
                    #st.add(i)
                    inter = curr_node[0] + 1
                    init_list.append((inter, i))
                    n = n + 1
    return n
```

Figure 11: Independent Cascade Function

5.2 Influence Function

```
# Influence Function
def influence_function(N, init_nodes, partition_num, nodes_set_broadcast):
    nodes_set = nodes_set_broadcast.value
    activated_num_rdd = sc.parallelize([init_nodes]*N, partition_num)
    activated_num = activated_num_rdd.map(lambda x: cascade(x, nodes_set))
                                          .reduce(lambda x, y: x+y)
    return activated_num/N
```

Figure 12: Influence Function

5.3 Greedy Algorithm

```
# Greedy Algorithm Implementation
def getMaxGreedy(nodes_set, N, curr_nodes):
    result = []
    max_node = None
    max_influence = 0
    for i in nodes_set:
        tmp = influence_function(N, curr_nodes + [i], partition_num, nodes_set_broadcast)
        if tmp > max_influence:
            max_node = i
            max_influence = tmp
    return max_node

#N: number of runs N in mul_cascade
#K: number of initial nodes for influence maximization
def calculateSK(nodes_set, N, K):
    # Perform getMaxGreedy for three times to get the solution for S3
    opt = []
    curr_nodes = []
    for i in range(K):
        tmp = getMaxGreedy(nodes_set, N, curr_nodes)
        curr_nodes = curr_nodes + [tmp]
        nodes_set.remove(tmp)

    max_influence = influence_function(N, curr_nodes + [i], partition_num, nodes_set_broadcast)
    return (curr_nodes, max_influence)
```

Figure 13: Greedy Algorithm

5.4 Simulated Annealing Algorithm

```
#Simulated Annealing
def simulated_annealing_rst(function, N, partition_num, initial_X,
                            initial_temp, cool, reanneal, iterr, nodes_set):

    accepted = 0
    X = initial_X[::]
    T = initial_temp

    history = list()
    # Evaluate E
    prev_E = function(N, X, partition_num)
    history.append(prev_E)

    for i in xrange(iterr):
        # Propose new path.
        X_star = next_step(X, nodes_set)
        # Evaluate E
        new_E = function(N, X_star, partition_num)
        delta_E = new_E - prev_E

        # Flip a coin
        U = np.random.uniform()

        if U < np.exp(delta_E / T):
            accepted += 1
            history.append(new_E)
            print 'value:', new_E
            # Copy X_star to X
            X = X_star[::]
            print 'set:', X
            prev_E = new_E

        # Check to cool down
        if accepted % reanneal == 0:
            T *= cool

    return X, history


def next_step(X, nodes_set):
    tmp = X[0]
    while tmp in X:
        tmp = np.random.choice(nodes_set)
    X[np.random.choice(range(len(X)))] = tmp
    return X
```

Figure 14: Simulated Annealing Algorithm

5.5 Adapted Simulated Annealing Algorithm

```
# Adapted Simulated Annealing Algorithm Implementation
def next_step(X, nodes_set):
    tmp = X[0]
    while tmp in X:
        tmp = np.random.choice(nodes_set)
    X[np.random.choice(range(len(X)))] = tmp
    return X


def simulated_annealing_rst(function, N, partition_num, initial_X, initial_temp,
                            beta, iterr, nc_N_width_nodes, nodes_set_broadcast):

    accepted = 0
    unaccept = 0
    #X = initial_X.copy()
    X = initial_X[::]
    T = initial_temp

    history = list()
    # Evaluate E
    prev_E = function(N, X, partition_num,nodes_set_broadcast)
    history.append(prev_E)

    for i in xrange(iterr):
        # Propose new path.
        X_star = next_step(X, nc_N_width_nodes)
        # Evaluate E
        new_E = function(N, X_star, partition_num,nodes_set_broadcast)
        delta_E = new_E - prev_E

        # Flip a coin
        U = np.random.uniform()
        unaccept = unaccept + 1
        if U < np.exp(delta_E / T):
            print "iteration", i
            unaccept = 0
            accepted += 1
            history.append(new_E)
            print 'value:', new_E
            # Copy X_star to X
            #X = X_star.copy()
            X = X_star[::]
            print 'set:', X
            prev_E = new_E
            # cool down the temperature very slowly
            T = T/(1+(beta*T))
        if unaccept == 200:
            T = T * 1.1

    return X, history
```

Figure 15: Adapted Simulated Annealing Algorithm