AC290 EXTREME COMPUTING: FINAL REPORT

Bleeding Edge

Gioia Dominedo, Christian Junge, Abhishek Malali and Isadora Nun

October 16, 2015

Introduction

The Influence Maximization Problem examines the behavior of nodes of a graph influencing one another. For this assignment, we are looking at a graph of nodes and the edges between them, and examining how the nodes within this graph activate one another. The nodes represent individual Yelp reviewers within the United States, the edges represent the connections between these reviewers, and activating a node represents that reviewer being influenced by one of its connections. The motivation of this problem is to find the optimal set of starting nodes, that is, the set with the highest average number of nodes activated when the simulation is run repeatedly. In this way, we can select an initial set of reviewers to target with advertising and incentives that will have the maximum influence that spreads throughout the network of Yelp reviewers.

The influence process is simulated with the Independent Cascade, in which the starting nodes stochastically activate their connections throughout the graph. The success of a particular set of starting nodes in activating other nodes within the graph is measured by the Influence function, which returns the average total number of nodes that are activated in many runs of the Independent Cascade.

This problem was broken into three scales: The full graph of 350,620 US reviewers, the subset of 24,244 reviewers in North Carolina, and a 240 reviewer subset of the North Carolina reviewers. We started our optimization implementations at a small scale to develop our code with reasonable computing time, and then attempted to scale our approaches to the larger problem. We were able to implement three different methods for finding the optimal set, and we applied each of these three methods to the two smaller scales of the problem. Along the way, we examined the parallelization and computing considerations that went into scaling this problem. Ultimately, we did observe that our two more complicated implementations returned sets with comparable Influence to the sets returned by the simplest implementation, the Naive Method, but we attempted to apply an algorithm to the Full US set for the purpose of gaining experience dealing with a distributed computing problem at a large scale.

Methods

A) **Uncertainty on the Influence Function (F)**

The influence function returns an expected value that has a certain amount of uncertainty due to the fact that it is estimated from a finite number of randomized simulations from within the large sample space of all possible outcomes. While the standard error of the independent cascade function is endemic to a given set of starting nodes, regardless of the number of simulations, the standard error of the Influence function shrinks as the number of independent cascade simulations (N) increases. Therefore, a larger number of simulations will yield a more precise estimate of the average influence for a given set. This tightening of the standard error is proportional to $\frac{1}{N^{-1/2}}$, therefore, the uncertainty in the value of the Influence Function can only be reduced by increasing the number of simulations. All optimization algorithms that use the Influence Function are impacted by its inherent uncertainty, so an unavoidable tradeoff exists between the uncertainty in Influence values used to find the optimal set and computation time.

B) Optimization Methods

i. Naive Method

The Naive set of size k contains the k nodes with the highest number of edges. The Naive Method is the fastest and simplest optimization method considered because it does not require running any simulations of the Independent Cascade. This makes it suitable as a baseline for other methods.

ii. Greedy Algorithm

The Greedy Algorithm is a method for finding a starting set of size k with a high number of activations without searching every possible set of starting nodes. In its first iteration, it finds the single node that has the highest value of the influence function when run over over N simulations. In each subsequent iteration, it finds the next node which, when added to the starting set, returns the highest expected value of activated nodes when run over N simulations. In this way, additional starting nodes are incrementally added to the set which was previously determined to be the best set until the set contains k starting nodes.

The limitation of the Greedy Algorithm is that by not considering every possible starting set, it may not find the true optimum. Furthermore, it is impacted by uncertainty within the Influence Function.

iii. Simulated Annealing

Simulated Annealing is a process to stochastically find the optimal starting set of nodes. This optimization process was developed for discrete search spaces where an exhaustive search would take a prohibitively long time. In the case of the Maximum Influence Problem, testing all possible combinations of nodes would take a very long time, so simulated annealing is a sensible approach.

The process works as follows: First, an initial set is randomly selected and the value of the influence function for N simulations is calculated. Then this starting set is perturbed by switching some of the starting nodes, and the expected value of this new set is calculated. The next iteration of the algorithm will use either the newly perturbed set or the set from the previous iteration, and it will select which of these two sets to move forward with randomly with a probability that is related to the difference in "Energy" (in this case, how many more nodes one set activates versus the other set, denoted ΔE)

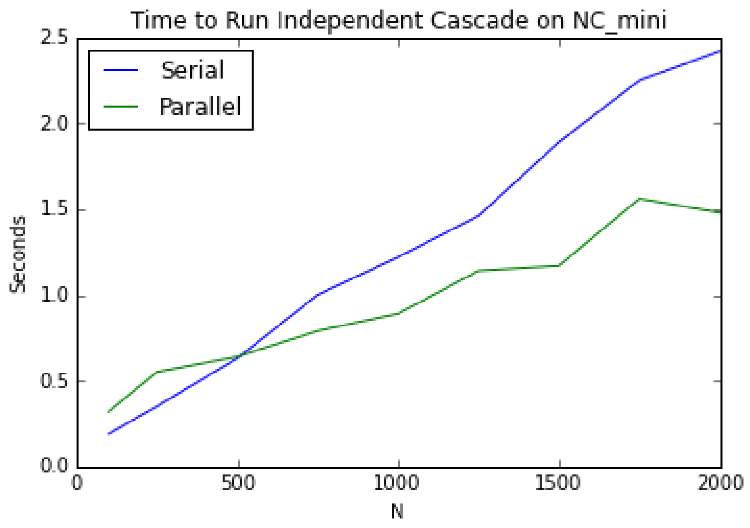


Figure 1: Timing of Serial and Parallel Influence Functions

and the "temperature" (T) of that given point in the algorithm. The temperature is a parameter that changes throughout the algorithm to adjust the probability of accepting the next starting set. This probability of accepting a new set is given by the equation $P(X = j) = \exp(\frac{-\Delta E}{T})$. If the temperature is high, the algorithm is more likely to accept a new set even though it returned a lower value of the influence function. Varying the temperature throughout the iterations of the algorithm allows it to consider sets of nodes that do not look promising at first glance, but may, with slight modification, yield better sets than those that were previously being considered.

The advantages of the simulated annealing algorithm are that it can search a very large sample space without considering an extremely large set of possibilities, and that it does so in a strategic way by iterating upon promising tracks, but not constraining itself to one small subset of the sample space.

C) **Parallelization**

The Influence function lends itself very well to parallelization because individual runs of the Independent Cascade are relatively fast and do not depend on one another. Also, capping the number of cascade steps helps each run of the Influence function to each take roughly the same amount of time. Therefore, within both the Greedy Algorithm and the Simulated Annealing Algorithm, parallelizing the Independent Cascades within each run of the Influence function should lead to significant time improvements. In practice, communication overhead trumps this benefit for smaller jobs, and the benefit of parallelization increases as N grows.

a) **Greedy Algorithm** - Because the Greedy Algorithm considers the behavior of many sets of nodes independently, it also lends itself well to parallelization. The parallelization strategy for the Greedy algorithm is to split the graph nodes over the partitions and independently run them over different cores since one computation does not affect the other.

Sample Spark Code:

```
def greedySearch(graph, k=3, N=1000, t=999999):
    best_s = []
    max_inf = 0
    nodeRDD = sc.parallelize(list(set(list(sum(graph.edges(), ())))), 4)

    for i in range(k):
        infRDD = nodeRDD.map(lambda n: (n, 0.) if n in best_s else \
                          (n, influenceFunctionNotParDetStart(graph, best_s + [n], N, t))
        next_s, next_i = infRDD.reduce(lambda a,b: a if a[1] > b[1] else b)
        best_s += [next_s]
        max_inf = next_i

    return best_s, max_inf
```

b) **Simulated Annealing** - This algorithm executes serially, and the set of nodes on which the Simulated annealing operates in each iteration depend on the set of nodes from the previous iteration. Hence the parallelization cannot be carried out on the nodes as in the Greedy Algorithm. Because of this, we can only benefit from parallelizing the Influence Function.

Sample Spark Code:

```
def simulated_annealing_tsp(function, initial_X, graph, N, initial_temp, cool, reanneal, ite

    accepted = 0
    X = initial_X
    T = initial_temp

    history = list()
    prev_E = function(graph, N, X, t)
    history.append(prev_E)

    for i in xrange(iterr):
        X_star = swapnodes(X, graph)
        new_E = function(graph, N, X_star, t)
        delta_E = prev_E - new_E

        # Flip a coin
        U = np.random.uniform()
        if U < np.exp(-delta_E / T):
            accepted += 1
            history.append(new_E)
            # Copy X_star to X
            X = X_star
            prev_E = new_E

        # Check to cool down
        if accepted % reanneal == 0:
            T *= cool
            if T < 0.001: # Reheat
                T = 2.

    return X, history
```

Implementation and Results for NC mini and NC full

1. **NC min**
   We implemented the Naive, Greedy, and Simulated Annealing optimizations on the 240

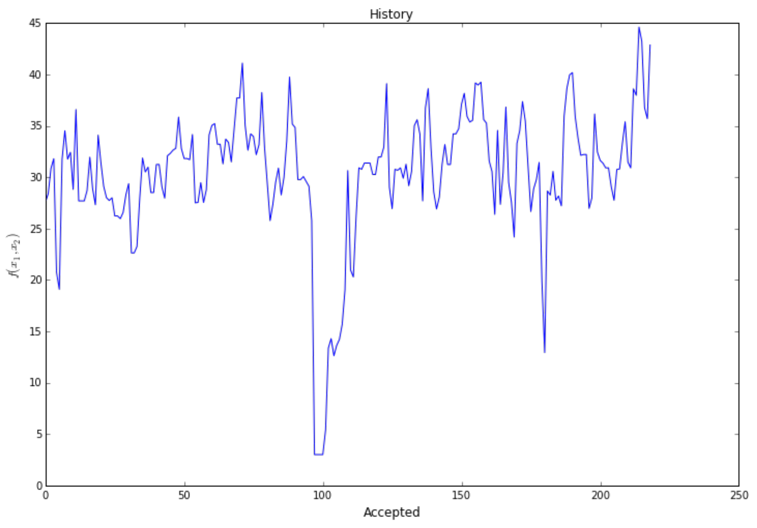


Figure 2: Simulated Annealing for NC mini

node subset of the North Carolina graph to develop our code. The results appear in below in Table 1:

Table 1: Results of Optimizations for NC mini

Algorithm	Influence	Run Time	Optimal Nodes
Naive	46.30 nodes	0.242 sec	`[u'NzWLMPvbEval0OVg_YDn4g', u'ts7EG6Zv2zdMDg29nyqGfA', u'VhI6xyylcAxi0wOy2HOX3w']`
Greedy	46.45	0.710 sec	`[u'NzWLMPvbEval0OVg_YDn4g', u'VhI6xyylcAxi0wOy2HOX3w', u'-_1ctLaz3jhPYc12hKXsEQ']`
Simulated Annealing	42.697	51.245 sec	[u'Q9xJQu-9oCFZozVBjZDETw', u'ts7EG6Zv2zdMDg29nyqGfA', u'VhI6xyylcAxi0wOy2HOX3w']

2. **NC full**
   We implemented the same optimizations for the full 24,000 node graph of North Carolina reviewers, and the results appear in Table 2:

Table 2: Results of Optimizations for NC full

Algorithm	Influence	Run Time	Optimal Nodes
Naive	8965.11 nodes	156.00 sec	`[u'CvMVd31cnTfzMUsHDXm4zQ', u'NzWLMPvbEval0OVg_YDn4g', u'4G68oLRY3aHE5XUt_MUUcA']`
Greedy	8963.51 nodes	2537.23 sec	`[u'm0N4g0eye9y_rXOB4a_MpQ', u'Cxj67usfkzhlReBfDkxezA', u'rO6S3Ai0m10Nuuh9ZCph6g']`
Simulated Annealing	8964.84 nodes	1576.72 sec	[u'kRvBUAnLXzE0Fsv07vBMMw', u'PuymBa7eyYV4RCmWjZUCGQ', u'wtJjo3a8ecMvVFjrGsiuEQ']

Greedy Algorithm
To reduce total compute time, the following restrictions were imposed on the Greedy Implementation for the full NC graph:

a) Depth was capped. Figure ———4——- demonstrates that, for NC full, there is a diminishing return of observed Influence Function value by allowing the Independent Cascades to propogate for an unlimited number of iterations because, after a certain number of steps, almost all cascade steps have terminated in this well-connected graph. A cap was set at 10, which according to the plot has very little risk of underestimating the Influence value, but still shaves several minutes off of the calculation time.

b) The number of simulations N was set to 1000. Figures 5 and——6——- show that uncertainty decays exponentially as N increases, so having very low standard error values would require extremely high values of N. $N = 1000$ was deemed to be sufficiently accurate, with a standard error of just over 200 nodes, while allowing the computation to be completed in a reasonable amount of time.

   Further support for setting N to 1000 appears in Figure 7, which shows the ratio of the minimum Influence Value to the maximum Influence Value. We can see that the gains from increasing N level off and there is a diminishing return for increasing the number of simulations.

   Figure 8 displays a similar point for the Greedy Algorithm: as the number of runs of the independent cascade within the greedy algorithm increases, the distance between the maximum and minimum activations decreases. The minimum bound also increases as expected with the increase in the number of runs for the greedy algorithm. This means that uncertainty due to stochasticity in the influence function can be reduced by increasing the value of N. There is a trade-off here, as increasing N increases the amount of computation, and displays diminishing returns.

Simulated Annealing

Within the Simulated Annealing Algorithm, high levels of uncertainty in the value of the Influence Function for each set of starting nodes considered can cause the algorithm to fail to come to local maxima. This is displayed conceptually in Figure ——8—— for the Traveling Salesman problem, where varying levels of uncertainty have been introduced into the distances between cities. The ideal case is to be very certain of the distances so that the algorithm can converge properly on minimum distances. Analogously in the Influence Maximization Problem, if the value of N is made too low to speed up computing time, Influence function values will be too noisy for the Simulated Annealing algorithm

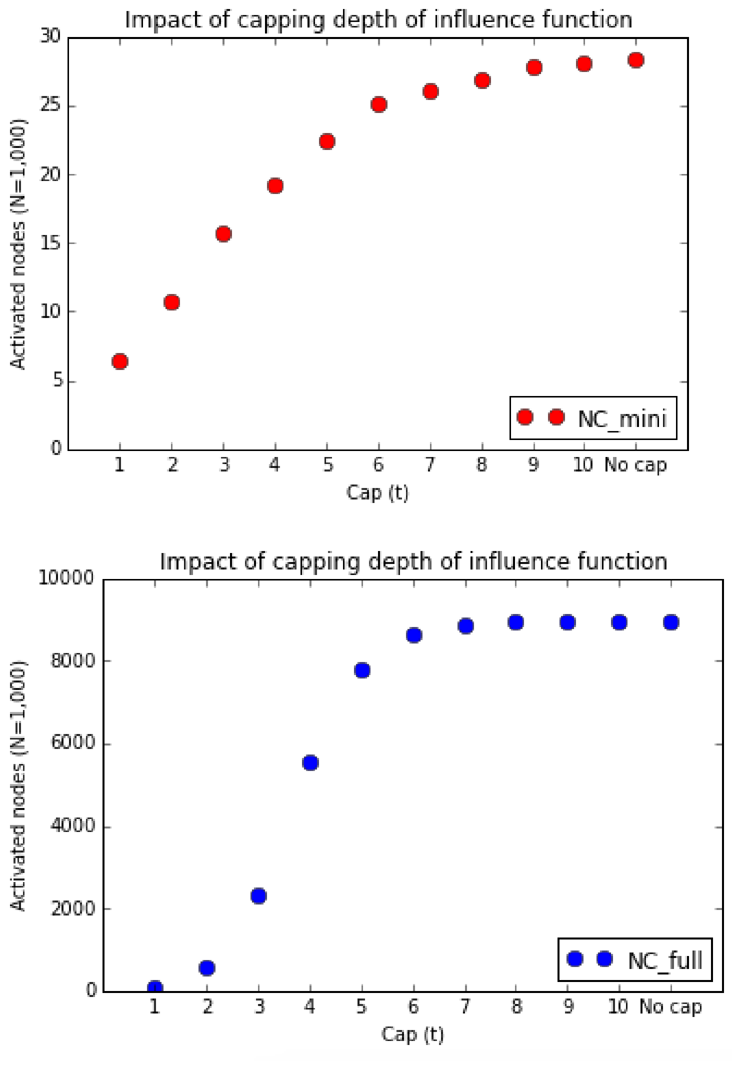


Figure 3: Impact of Capping Depth on Influence Function Value

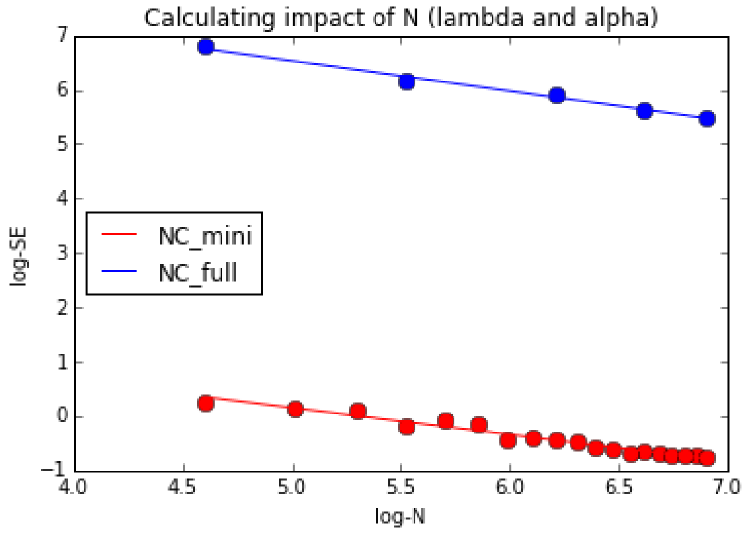
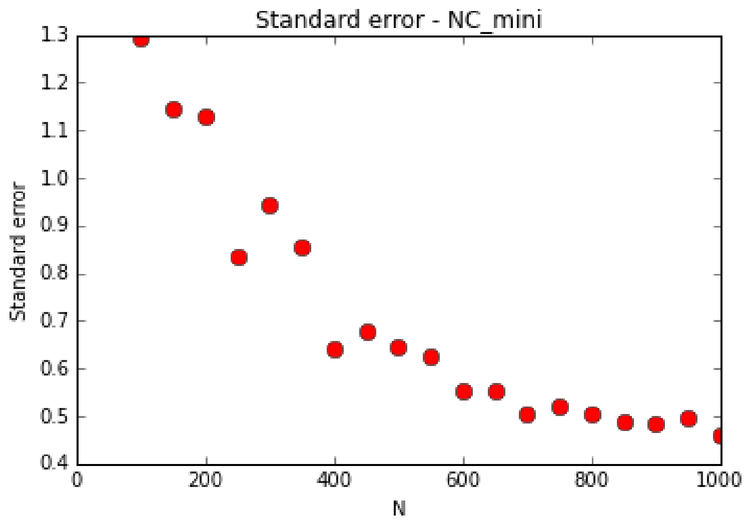


Figure 4: Estimating impact of N on uncertainty



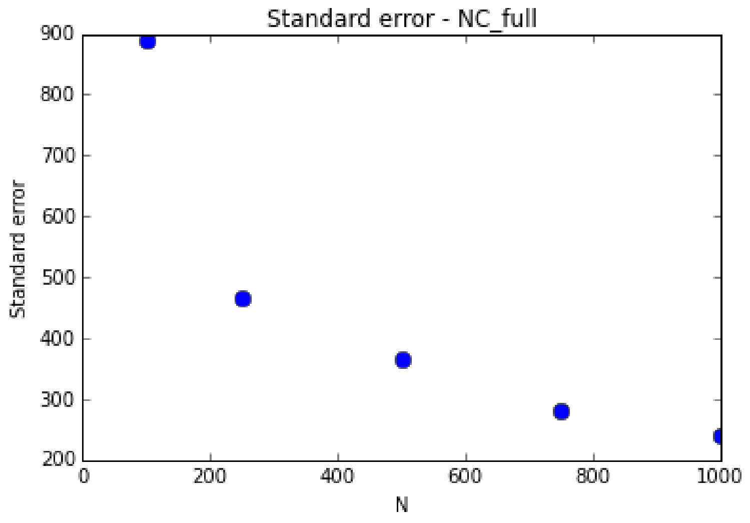


Figure 5: Impact of N on uncertainty for greedy algorithm

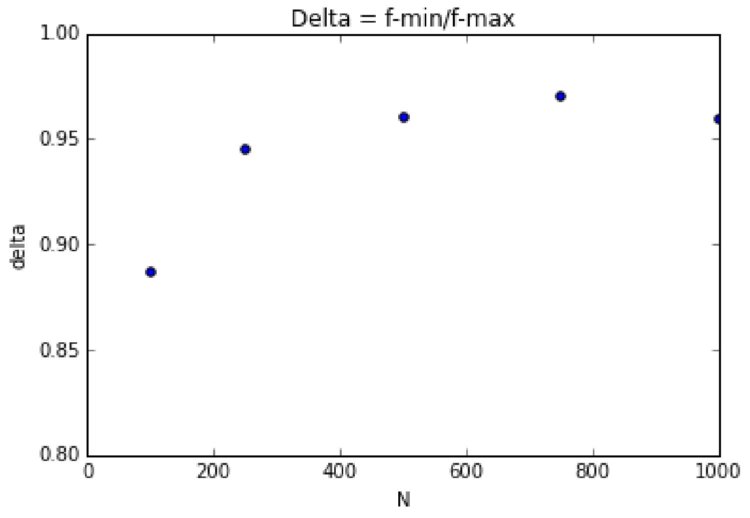


Figure 6: Ratio of min to max Influence Value for Different Values of N

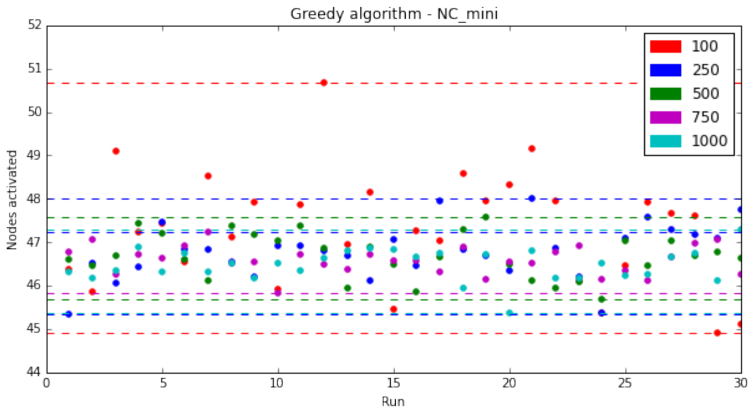


Figure 7: Plot demonstrating how minimum and maximum activations changes with respect to N

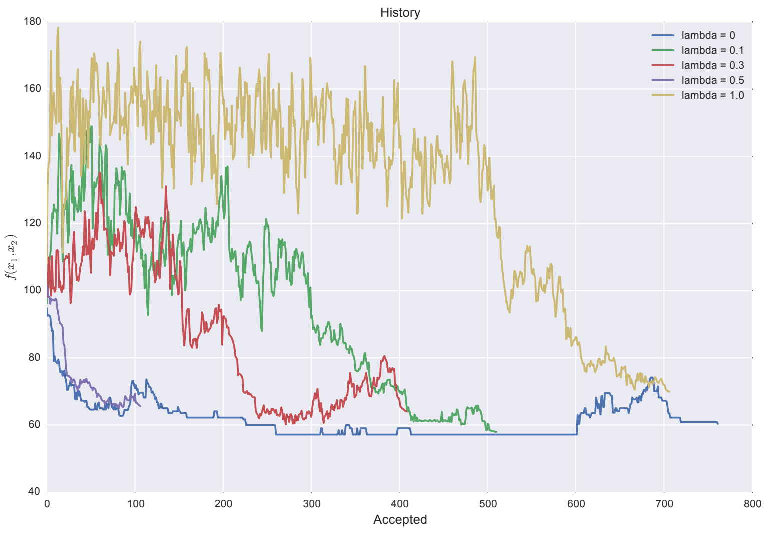


Figure 8: Simulated annealing with noise $= \lambda * random.uniform(0, 1)$ for different values of λ

to converge on maximum influence values.

Figure ——— demonstrates the behavior of the Simulated Annealing algorithm with different values of N. If we had more time, we would like to repeat this plot with a fixed set of starting nodes for the algorithm to control for the randomness that dominates this visualization due to different starting sets.

In practice, the Simulated Annealing Algorithm was very time-consuming, and a low value of N (N=100) was used to allow for more cycles to complete. Depth was again capped at 10 to reduce compute time without significantly underestimating Influence Value. Temperature, cooling, and reannealing parameters were selected through trial and error, ultimately yielding a model that matched the results of the Greedy Algorithm and Naive Method.

Scaling the algorithm to the US

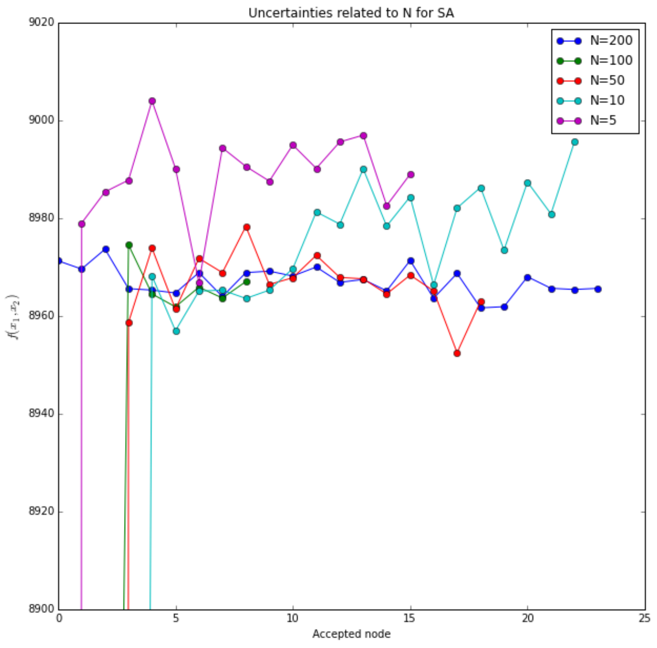


Figure 9: Impact of different N values on Simulated Annealing

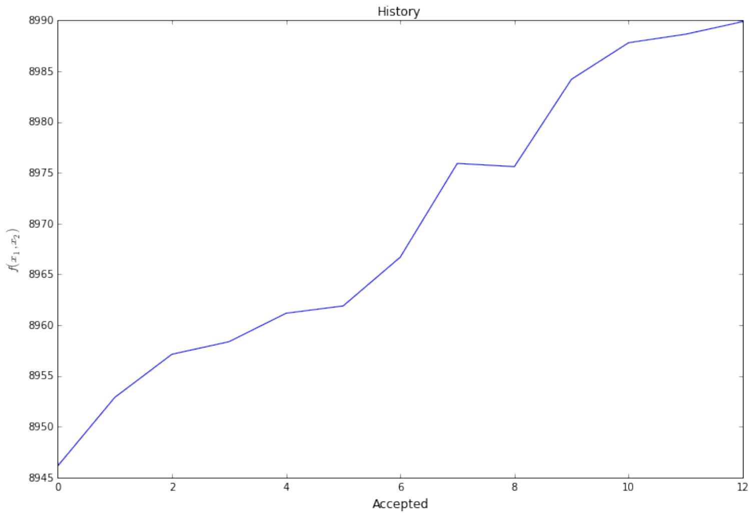


Figure 10: Simulated Annealing for NC full

We attempted to scale the Greedy Algorithm to the full US graph. We selected the Greedy Algorithm because it was parallelizable, and straightforward to impose some reasonable constraints to reduce computing time. The constraints were as follows:

1) Depth was capped at 10. As observed in Figure ———-4—, the Influence Function exhibited diminishing returns more quickly for the larger graph, presumably because it is even more well-connected than the smaller subset, so we decided that it would be reasonable to keep this cap from our North Caroline-scale Greedy implementation

2) N

3) pruning plus graphs

Conclusion

A) Who are the top 3 influential reviewers?
B) Did your solution beat the Naive Method?
C) Future considerations: how could your algorithm be improved to solve this problem?