AC290 EXTREME COMPUTING: FINAL REPORT

Bleeding Edge

Gioia Dominedo, Christian Junge, Abhishek Malali and Isadora Nun

October 14, 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 goal 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.

For Christian To Fill In:
——————————Motivations——————————————
——————————Quick Summary of Work——————————
——————————Outline———————————————

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) 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**

a) **Greedy Algorithm** - 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, with a cap on the number of cascade steps, each run of the Influence function should each take roughly the same amount of time. Therefore, within both the Simulated Annealing algorithm and the Greedy algorithm, parallelizing the Independent Cascades within each run of the Influence function should lead to significant time improvements. The parallelization strategy for the Greedy algorithm will be 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——————

————————————————————-

——Also, I need to verify that these parallelization strategies are the ones that you used, and the wording of these sections can be improved—————-

————————————————————-

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. Instead, we can parallelize the Influence Function by parallelizing the N runs of the Independent Cascade. Since each of the runs operates independently on a set of starting nodes, we can thereby reduce execution time of the Simulated Annealing Algorithm.

————————————————————

——————Sample Spark Code——————

————————————————————

Implementation and Results for NC mini and NC full

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

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

	Influence	Run Time	Optimal Nodes
Naive	???	???	???
Greedy	???	???	???
SA	???	???	???

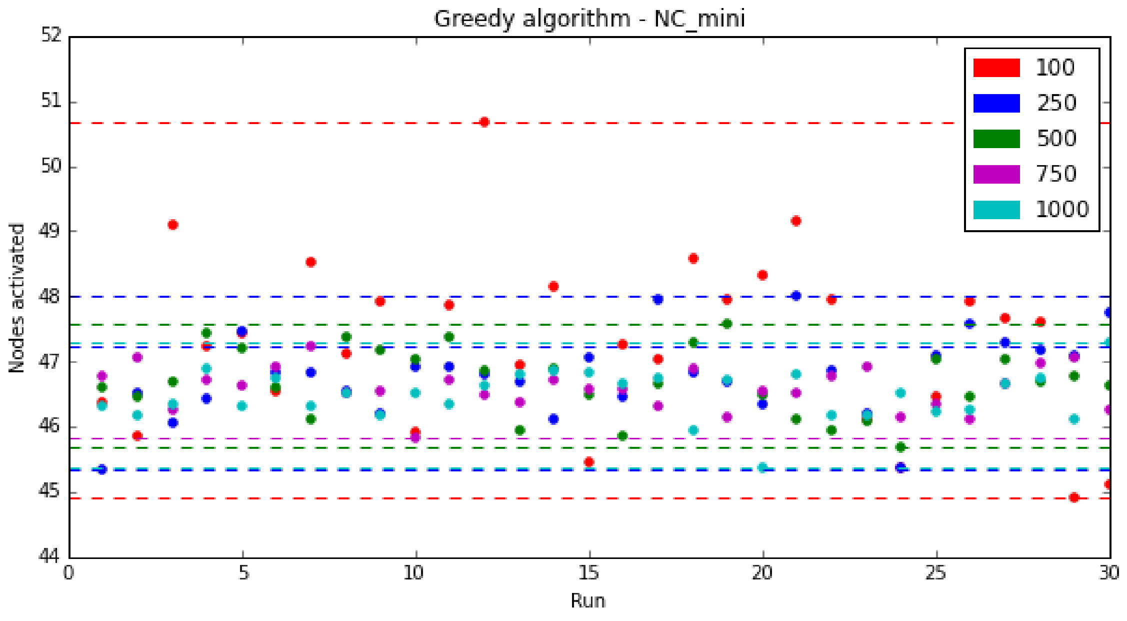


Figure 1: Plot demonstrating how minimum and maximum activations changes with respect to N

As can be seen in Figure 1, 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.

—————-Plot showing how Simulated annealing goes up and down—————
———-Should have places where it flattens out, drops again, and comes back up before —————-flattening out again to show that this was implemented properly—————-
——-Keep in mind that the Simulated Annealing Algorithm should store the absolute max that it comes across (even though it will switch away from this as it heats and cools)— ——and should return the absolute max at the end——————————

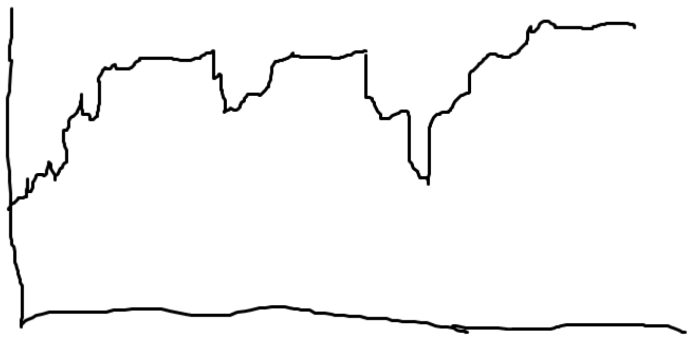


Figure 2: Simulated Annealing

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:

	Influence	Run Time	Optimal Nodes
Naive	???	???	???
Greedy	???	???	???
SA	???	???	???

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

   ——————-Cap depth? Discussion of how the depth cap was selected, ideally an experiment where this was selected to shorten compute time without severely coming up short on the calculated influence——

   ——————Only consider nodes with connections, or with a min number of connections?—-

   —————-Value of N, picked to be large enough to get good values of F, but small —-——-enough to shorten compute time, discussion of how this N was selected systematically–

   ———Refer to plot showing impact of N on uncertainty——————-

   ———————-Plot of Simulated Annealing for NC Full——————

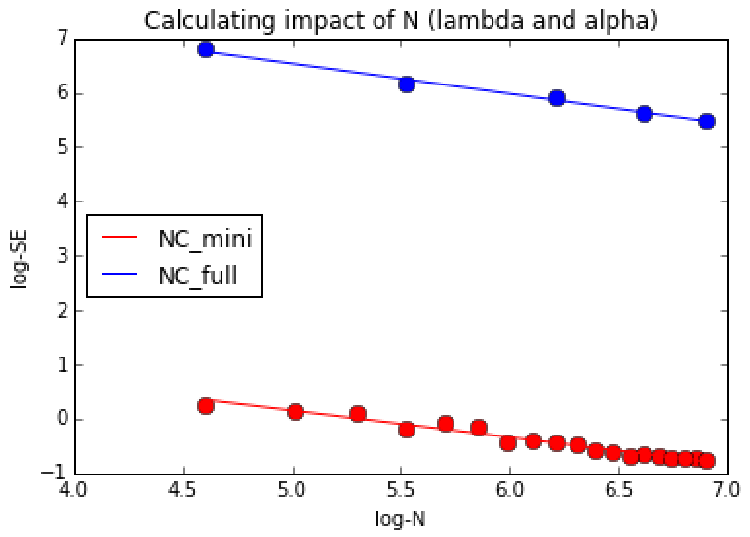
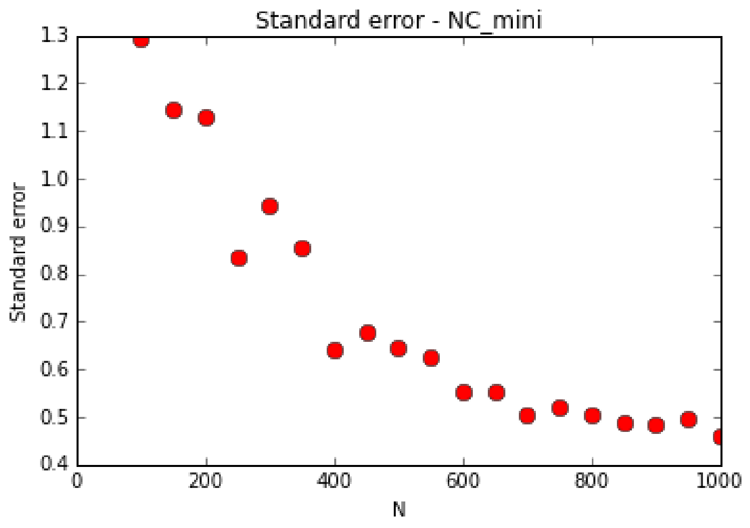


Figure 3: Estimating impact of N on uncertainty



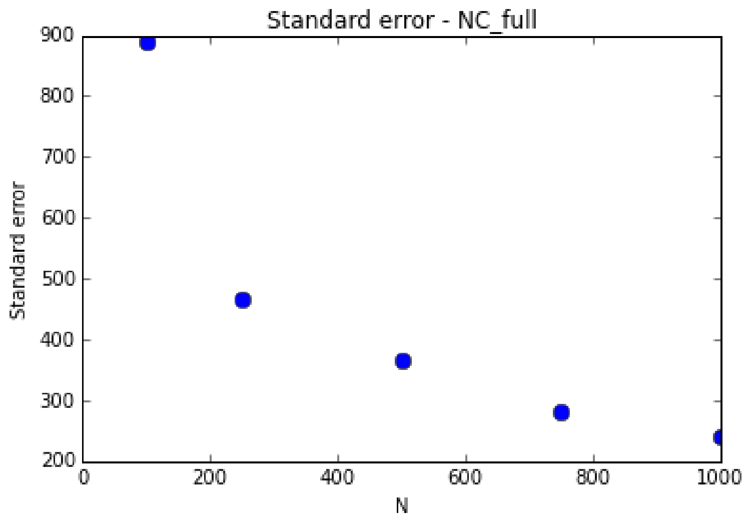


Figure 4: Impact of N on uncertainty for greedy algorithm

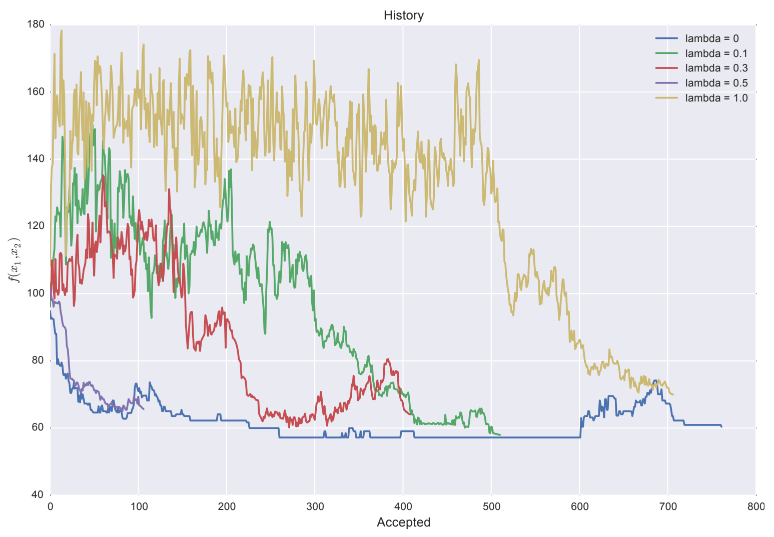


Figure 5: Simulated annealing with noise = $\lambda * random.uniform(0, 1)$ for different values of λ

——————————————————————————————————————

——————————————————————————————————————-

————Discussion of the parallelizations, with numbers and figures to support this——-

——————————————————————————————————————-

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 5 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 to converge on maximum influence values.

——————————————————————————————————————-

——————————————————————————————————————-

–If it is possible to make an analogous plot using different values of N for simulated annealing, even if it is just for NC mini, this would display this point better than the Traveling Salesman analog, and give more credibility to the value of N that we selected to use for Simulated Annealing

——————————————————————————————————————-

——————————————————————————————————————-

Scaling the algorithm to the US

——————————————————————————————————————-

——————————————————————————————————————-

——————————————————————————————————————-

——————————————————————————————————————-

——————————————————————————————————————-

What strategy did you use to scale this problem? Discussions can include:
A) Balancing Noise vs Runtime tradeoff in selecting N (naturally, they should include a plot of standard deviation of influence vs N and runtime vs N)
B) Reducing the depth of the cascade and its consequences
C) Pruning methods - eliminating disconnected or low-out-degree nodes
D) For SA: neighborhood selection - random vs direct neighboring node

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?