MP1 Performance Study

Gary Yip - Group 5

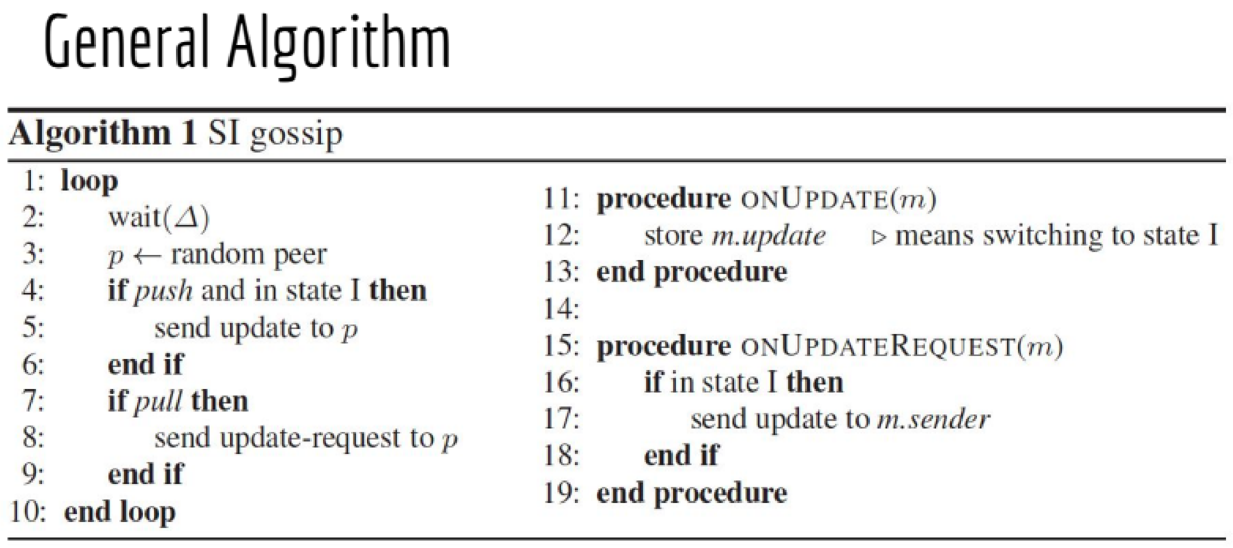
Introduction

The goal of this project was to simulate three different types of gossip, and to find out how the number of nodes affects the convergence time (number of rounds required to fully infect all nodes). The three different types of gossip are: push, pull, and push/pull switch. The program allows for any number of nodes to be tested, as given by user input, as well as any number of iterations in order to get a meaningful and accurate result of the number of rounds required. Due to the random nature of selecting nodes for infection, the number of rounds can vary greatly in between different attempts. For example, by running the pull gossip with 100 nodes, the number of rounds required can be much higher or lower depending on whether the random selection has allowed them to pull from the initial infected node in an early round. My approach have running multiple iterations allows me to lessen the impact of such outliers that may occur if it were only run one time.

In creating this program, with what I had learned about gossip through the lectures, I hypothesized that push/pull switch would be the most efficient of the three methods.

The three types of gossip were the main focus of this project. Push gossip relies on having infected nodes randomly select another node, and send their infected status over to them. Pull gossip works by having uninfected nodes randomly select other nodes and adopt their state if the other node is infected. Push/pull switch gossip combines the two of these together, but starts off with using push until half of the nodes are infected, at which point it switches to using pull gossip.

I based my implementation of gossip on the gossip algorithm by Mark Jelasity as introduced to us in class, as shown below:



General Algorithm

Algorithm 1 SI gossip

```
loop
    wait($\Delta$)
    $p \leftarrow$ random peer
    if push and in state I then
        send update to $p$
    end if
    if pull then
        send update-request to $p$
    end if
end loop

procedure ONUPDATE($m$)
    store $m.update$        ▷ means switching to state I
end procedure

procedure ONUPDATEREQUEST($m$)
    if in state I then
        send update to $m.sender$
    end if
end procedure
```

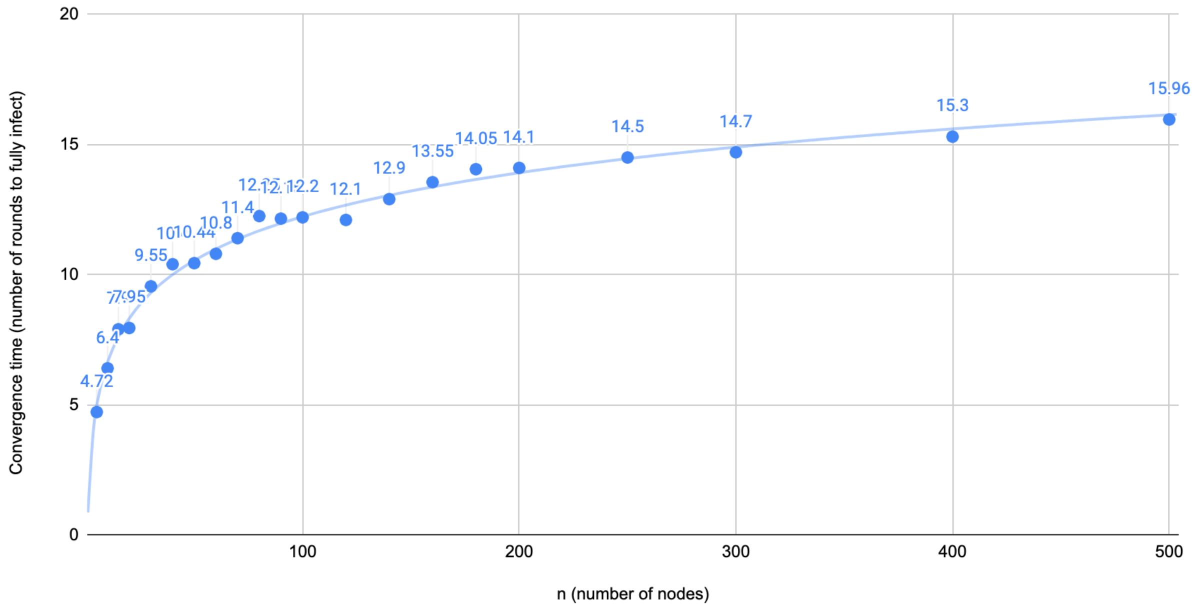
Mark Jelasity also tells us of the time complexity of the gossip algorithm. In our case, the convergence time is represented by the number of rounds required to complete total infection, rather than actual time needed. In particular, Jelasity notes that the time complexity of push gossip is O(N log N), and pull gossip has a time complexity of O(log log N).

In my program, there are details about every round of gossip that happens throughout each iteration. If the user wanted, they would be able to track the entire process of exactly how the gossip is working and what it is doing at every step. Importantly, at the beginning of every round, the program prints out which node that has an action (for example, an infected node when using push gossip) has selected another random node. This happens regardless of if the targeted node needs to have its state changed. Additionally, at the end of the round I print out the status of every node, where true notes that a node is infected and false denotes that a node has yet to become infected. At the end of an iteration, or turn, it will print out the number of rounds required to infect n number of nodes, and will move onto the next turn. Each turn is also printed out on the system, to make it very clear to the user what is happening. Finally, at the end of every turn, it will calculate and print out the average number of rounds required for infecting n number of nodes.

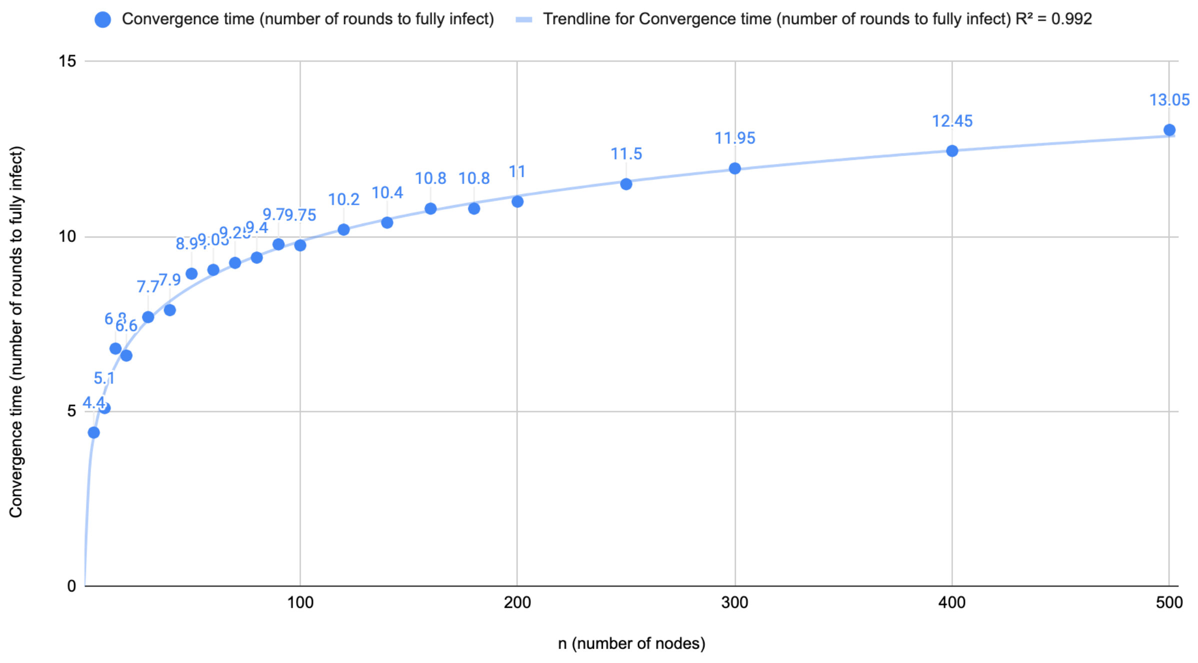
Graphs

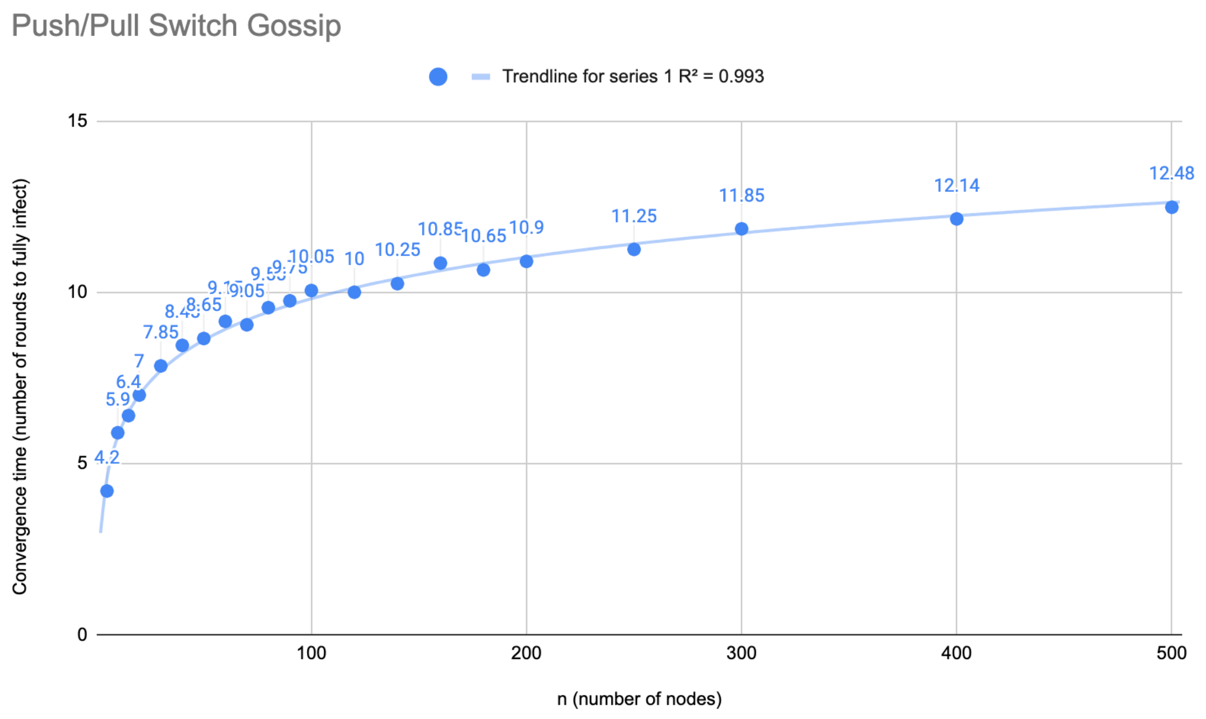
Each data point for all graphs is the average of 50 iterations.

Push Gossip



Pull Gossip





Results

As it is clearly seen from the graphs above, push gossip takes a considerable amount more time than pull does. This was predicted by use of Jelasity's time complexity for both of these two algorithms. Additionally, it was unsurprising to see push/pull switch perform slightly better than pull, making it the best overall, just as I hypothesized. I believe that push works slightly more efficiently than pull when there are fewer infected nodes, and pull works slightly more efficiently than push when there are more than 50% infected nodes. This program was able to test my hypothesis. We also see that the line of best fit for the graph is a logarithmic one, which echoes the time complexity that Jelasity had calculated for the algorithm.

What was interesting to see was that despite running 50 iterations on every data point on the graphs, there was still some variance as the data was not a perfect logarithmic curve as I expected. While there were no complete outliers, we did see smaller, sudden spikes across two neighboring points on the graph. This is due to the random nature of the program, and as such there will always be some sort of variance caused by it no matter what. With that being said, I do believe that I was still able to make the results much more accurate.

Overall, I believe that this project was a success. I was able to accurately simulate the three different types of gossip protocol, and compare the number of nodes with the convergence time for all three of them. However, looking back I definitely see some flaws in my program. To start with, unlike some other groups I did not have a feature to collect data from a range of number of nodes, so I would have to run the program separately multiple times in order to get my data rather than just once. Indeed, because of this limit I decided to take less data points and this could have affected the result. If I had a data point for every single n up to 500 (which is where I stopped) it would be an even more accurate and precise study. Furthermore, I did not implement a plot feature in this program, and although it is very easy to analyze the data and make graphs through other means, I definitely think that it would be a great function to have.
