Gossip Algorithm Report

Jacob Ville - 4540-7373
Shaifil Maknojia - 7805-9466

Implementation

We implemented the gossip algorithm by creating a network of actors (nodes) that pass messages between each other until each has received 10 messages. At this point the actor stops propagating messages.

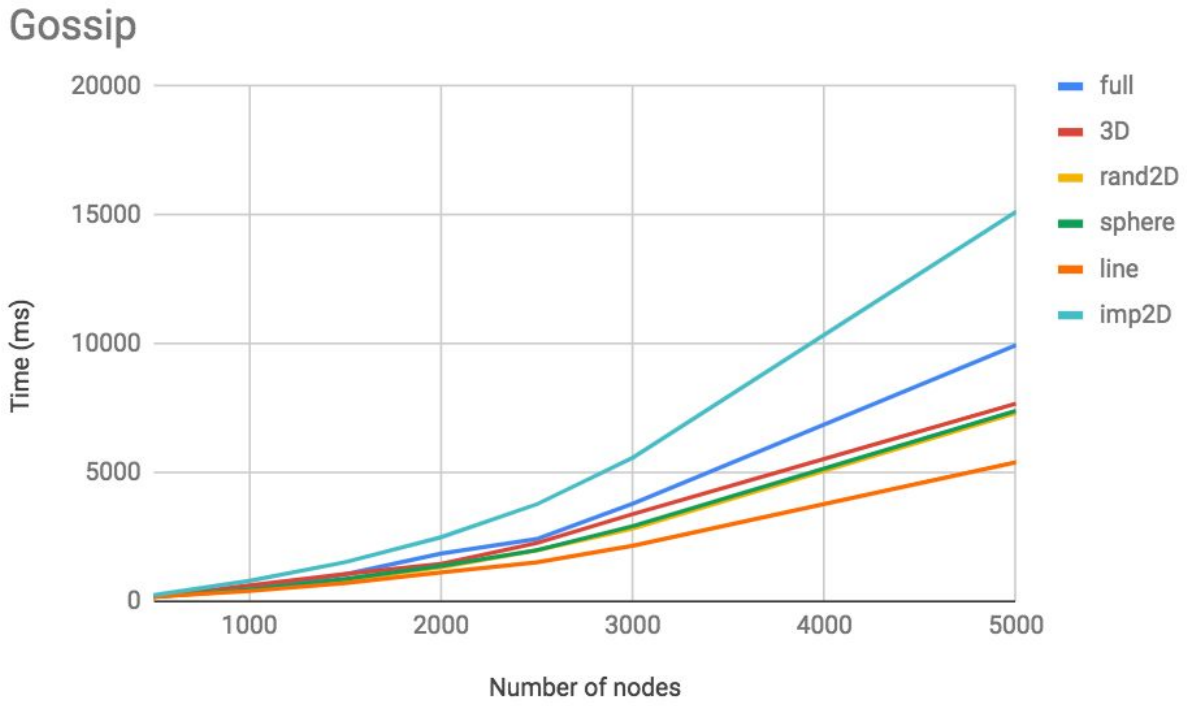
Push sum had a similar implementation, however the message exchanged included two values: s and w. When sending a message, these values are halved, with half remaining in the state of the actor, and the other half sent as a message. Upon receiving a message, the actor will add the values in the message to the values in its own state. Once the ratio of s/w does not change more than 10^{-10} in three consecutive rounds, the actor terminates.

Timing

To determine how long it takes for all actors to terminate, we used the System.monotonic_time() function. Unlike System.os_time(), monotonic time does not decrease or leap. Once all processes have been spawned, we take the time with this method, and pass it into the module that kicks off the algorithm. Once the last actor has terminated, we take the time again, and compare this to the first to determine how much time has elapsed.

Gossip Results

A notable finding from our results of the gossip algorithm on the following page is that the line topology converged faster than the others. This is due to how we implemented the line topology. A scenario arises where both of an actor's neighbors have terminated. In this case we found the next closest actor in the line and passed the message to that actor, resulting in the fast convergence time observed.



Push Sum Results

Unlike the results from gossip algorithm, the line topology performed the slowest. This is because actors would terminate very quickly in gossip (after receiving only 10 messages), and as mentioned previously the next closest neighbor is sent the message. In the push sum implementation however, many more messages had to be received before the ratio of s and w would converge, resulting in the slow times observed below.

Along with this observation, it was interesting to see that imperfect line (imp2D) converged almost as quickly as a full network despite the slow convergence time of the line. This is a great example of how adding a degree of randomization can affect the convergence time, and although we discussed this concept during lecture, we weren't expecting the effects to be this drastic.