Project 2 Report

Group members:

Yihui Wang UFID# 8316-4355
Wei Zhao UFID# 9144-4835

Topologies Implementation:

1. Full Network: Every actor is a neighbor of all other actors. That is, every actor can talk directly to any other actor.
2. Line: Actors are arranged in a line. Each actor has only 2 neighbors (one left and one right, unless you are the first or last actor).
3. Random 2D Grid: Actors are randomly position at x, y coordinates on a [0-1.0] x [0-1.0] square. Two actors are connected if they are within .1 distance to other actors.
4. 3D torus Grid: Actors form a 3D grid. The actors can only talk to the grid neighbors. And, the actors on outer surface are connected to other actors on opposite side, such that degree of each actor is 6.
5. Honeycomb: Actors are arranged in form of hexagons. Two actors are connected if they are connected to each other. Each actor has maximum degree 3.
6. Honeycomb with a random neighbor: Actors are arranged in form of hexagons (Similar to Honeycomb). The only difference is that every node has one extra connection to a random node in the entire network.

Gossip Protocol Implementation:

1. The main process starts gossiping by send a message to one of the nodes.
2. When a node received the message, it will randomly select a node from his neighbor list which is generated according to the topology it used and send the message to it.
3. Each node counts how many times it has received the message. When a node has received messages 10 times, it will stop sending the message to

its neighbors and send a message to the main process. The main process counts how many nodes are stopped.

4. When the main process find that all the nodes have stopped sending messages, it will print the total time it used to finish the gossiping.

PushSum Protocol Implementation:

1. The main process starts gossiping by send a message {sum: 0, weight: 0} to one of the nodes.
2. Each node initializes their sum to i (the number of the node), and weight to 1. When a node received the message, it will add the sum and weight received to 1/2 of its own sum and weight and calculate the ratio of its new sum and weight.
3. Each node calculates the different of its old ratio and new ratio.
4. If the ratio of the node changes more than 10^{-10}, it will randomly select a node from its neighbor list which is generated according to the topology it used and send 1/2 of its current sum and weight to it.
5. If the ratio does not change more than 10^{-10} for 3 consecutive rounds it stops sending message to its neighbor and send a message to the main process. The main process counts how many nodes are stopped.
6. When the main process find that all the nodes have stopped sending messages, it will print the total time it used to finish the gossiping.

Analyzation of Performance:

Generally, for both type of algorithm, the “line”, “honeycomb” and “random honeycomb” topology has better efficiency than the “Full” and “Random 2D” topology. As those 3 topologies has less change of running time when number of node increase (Figure1 and 3). By inspection of the line diagram, except of torus 3D, the relationship of the number of node (n) and the running time act as similar as log(n) (Figure1 and 3). The torus 3D was not able to run successfully when the number of nodes is larger than 2000, which indicate that there is inefficiency build in the implement of the topology. It takes too long for the system to figure out the neighbor. As it showed in the graph, for the torus 3D, the relationship of

number of node (n) and the running time act as $\Omega(n^2)$. Except for Full and Random 2D topology, the running time of both algorithms is rather close. When implement Full and Random 2D topology, Gossip algorithm running time is much larger than the running time of push sum which shows better efficiency of Push Sum algorithm with Full and Random 2D topology. (Figure 5)

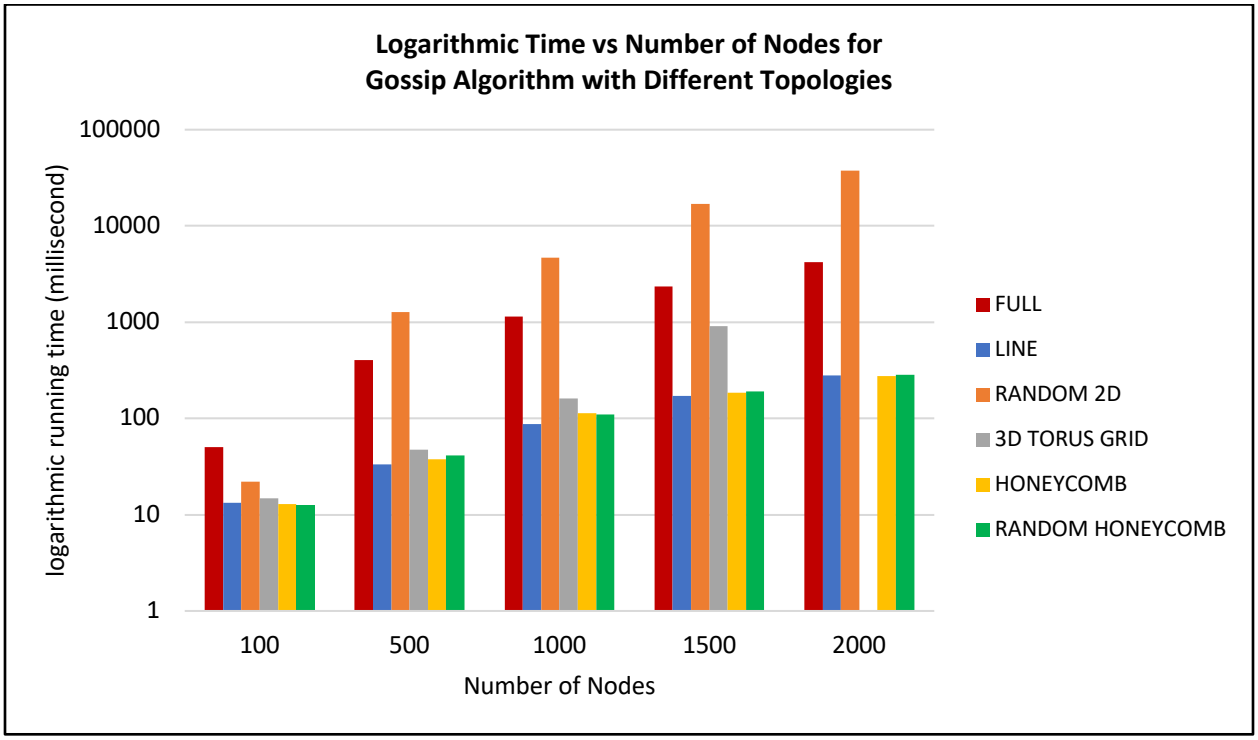

Figure 1

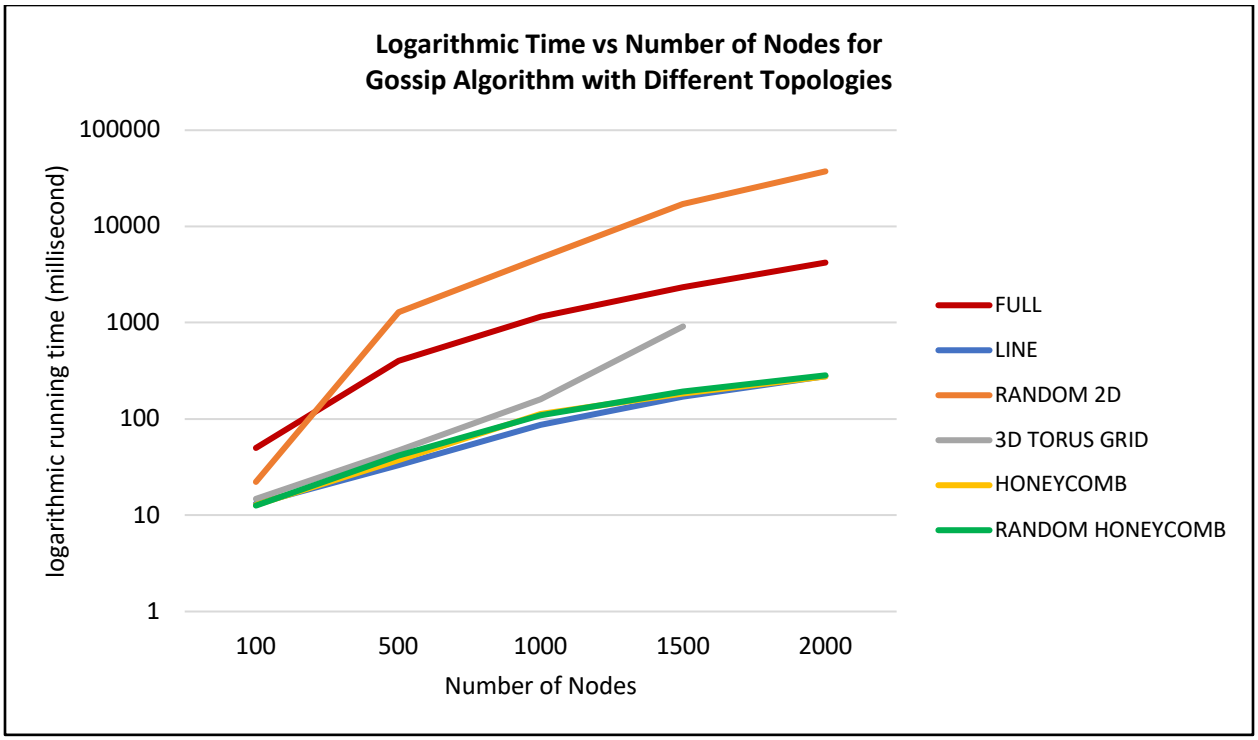

Figure 2

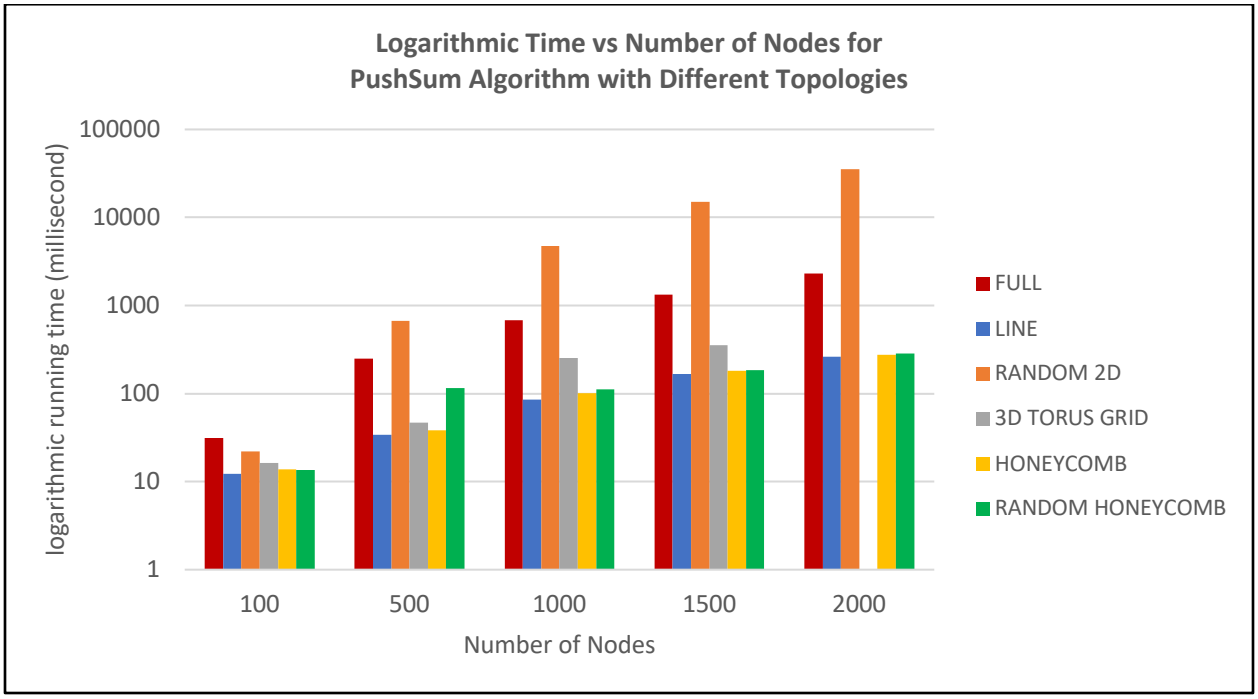

Figure 3

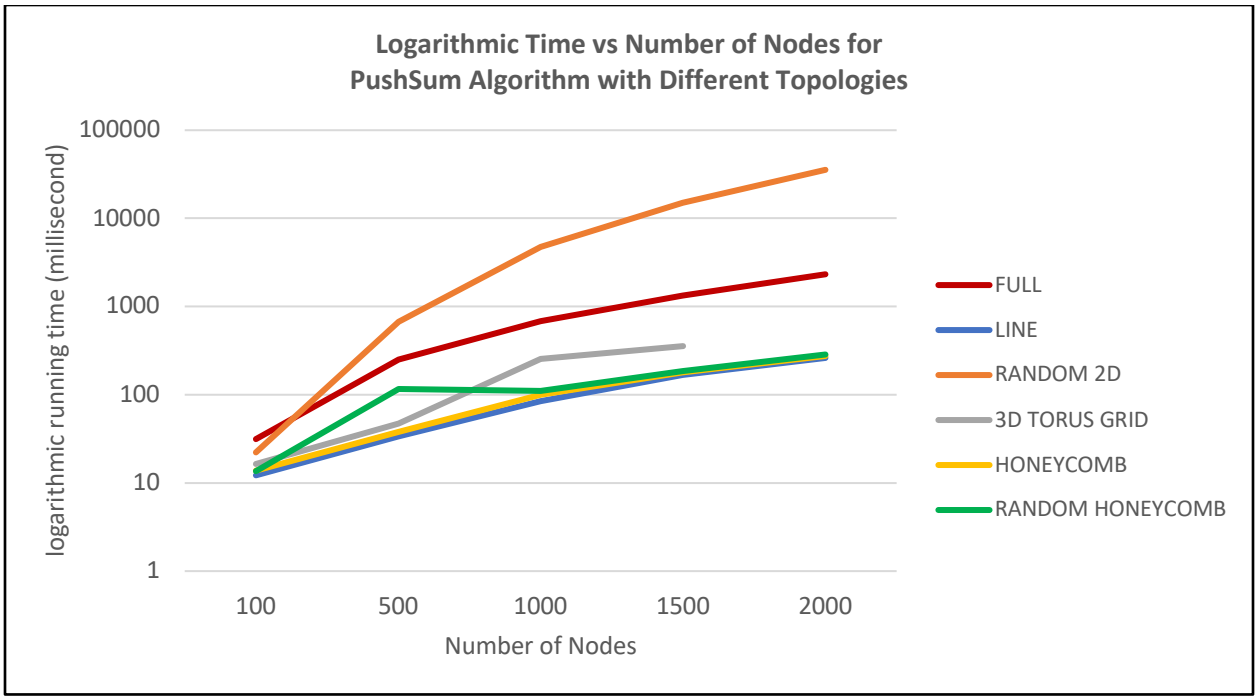

Figure 4

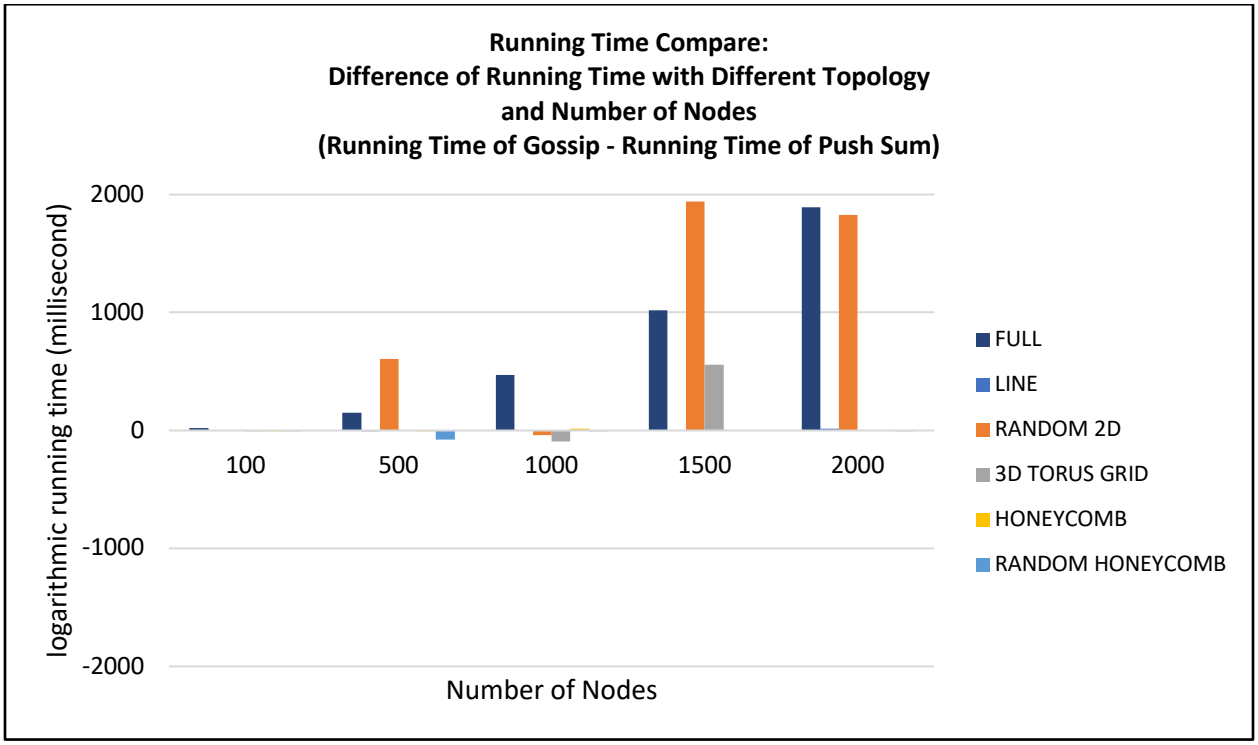

Figure 5

Interesting finding:

In some topologies such as Line and Random 2D, the situation that a node is out of neighbors before finishing its job often happens. It makes the main process

confused about whether the process has finished. In that case, the main process will not print the total running time of the program. To solve this problem, we think that there are two ways that may work. The first method is that we can set a timer in the main process and start to countdown from a certain time. When a node sends a message, it also sends a message to the main process to reset the timer. In that case, if there is no more transmission between nodes for a certain time which was set to the timer, the countdown in the main process will finish and the main process will know that all the transmissions have ended. The second method is that when a node finishes its jobs, it doesn't stop transmission. The finished nodes still send messages to its neighbors. For example, in PushSum the nodes send (sum: 0, weight: 0) to their neighbors after finishing their jobs. In that case, the connections between nodes will never be cut off, and the result will not be affected greatly.