**COP5615 – Fall 2019**

**Project 2 – Gossip and Push sum simulator**

**Project members:**

* Ankur Sethi (UFID: 9351-2951)
* Yogesh Laxman (UFID: 9451-2517)

**Brief Description:**

The aim of the project is to implement Gossip and Push-Sum algorithm in Elixir and determine the convergence time for different network topologies. The implementation of the respective algorithms and topologies are described below.

**Gossip algorithm:**

The main driver of the algorithm initiates a rumor message, which is send to a node selected at random.

This node starts transmitting the message to its neighbors, picking one at a time in random order. This process is repeated by its neighbors.

A node stops transmitting and listening for messages once it has heard the rumor 10 times.

**Observations for Gossip algorithms:**

* *Transmitting periodically works better*

We Tried two methodologies: 1. A node transmits a message to a neighbor node when it receives a new message and waits for another message before transmitting. 2. A node starts transmitting in periodic intervals to its neighbors and continues this until it has itself heard the rumor 10 times.

In the first case, the chain-like behavior sometimes prevents for the entire network to reach the convergence criteria of 10 times. This happens more often for topologies like *Line and Random 2D Grid(with fewer nodes),* than for densely populated topologies like *Full or 3D Torus.*

* *Convergence achieved was 100% for all topologies*

When using periodic transmission policy, we observed a convergence rate of 100% for all topologies. Exception for minimum nodes was the *Random 2D Grid* topology. It needs at least 100-150 nodes to converge. For that number of 50-150, the algorithm works sometimes but breaks often. The expected behavior is depicted for around 200 or greater nodes.

* *Convergence time:*
* *No limit on number of nodes:*

There was no limit observed on the number of nodes, and it is dependent on the system configuration. On our systems, we observed the number of nodes to max out at around 10,000 processes

**Push Sum Algorithm:**

* The Push Sum algorithm diffuses the value at each node to other nodes to converge with an average value at each node.
* This is achieved by setting a value and its weight at each node, and sending this pair of values sending messages in the form of pairs(s,w).
* When this pair of values is sent to a node, it adds them to its own values. It keeps half of these values to itself and the other half to a random neighbor.
* We assume convergence of a node when its s/w ration doesn’t change more than 10^-10 for three consecutive times.

**Observations for Push Sum algorithms:**

* *Takes longer for convergence than the gossip algorithm:*

We observed the convergence time for push sum to be greater on average than the gossip algorithm for similar topologies. Two reasons for this are: 1. the convergence criteria of *change less than 10e-10,* and 2. The nodes are not periodically sending their values like in the gossip algorithm

* *Convergence ratio:*

The convergence ratio is much less than gossip. The primary reason being the serial wise sending of messages. Because of this, a node which has received a message may not have an active neighbor to send a message to, and the message chaining will be stuck. This is observed quite often and thus, we halt the progress at this time.

* *Observed Convergence:*

As mentioned in the above point, many nodes may not reach the convergence criteria of change less than 10e-10 for three times. But we still see the push sum ratio for all nodes as converging. This was observed by logging the values and the s/w ratio of each node but has been removed from the final code as per the project requirements.

* *Convergence time*

The convergence time increases sharply in case of line, for rest

**Topologies implemented:**

* **Full network:** We have defined the nth node is connected to every other node in the network.
* **Line:** Every nth node has been connected to n-1 and n+1 nodes, except the starting and ending nodes which have been connected to second and second last nodes respectively.
* **Random 2D Grid:** We have produced random values between 0 and 1 for n nodes. If the distance between two nodes is less than 0.1, the nodes are connected to each other.
* **3D torus Grid:** We are using the nearest perfect cube to the number of nodes entered. This makes it to implement a topology in 3D. We set the values accordingly, starting from 1 to n and setting neighbors between them based on the derived relationships.
* **Honeycomb:** In order to implement a honeycomb structure, we have rounded the number of nodes to a perfect square nearest to the given number of nodes. This assumption makes it easier to build a honeycomb like structure. In our implementation, the pattern repeats in a set of four numbers for all the nodes except the nodes at the edge of the structure. The edge nodes also have a repeating pattern which have been implemented. An example is shown below for 100 nodes. As we can see, each node has two neighbors, except at the edges. We created this diagram to build a visual structure for the topology helper.

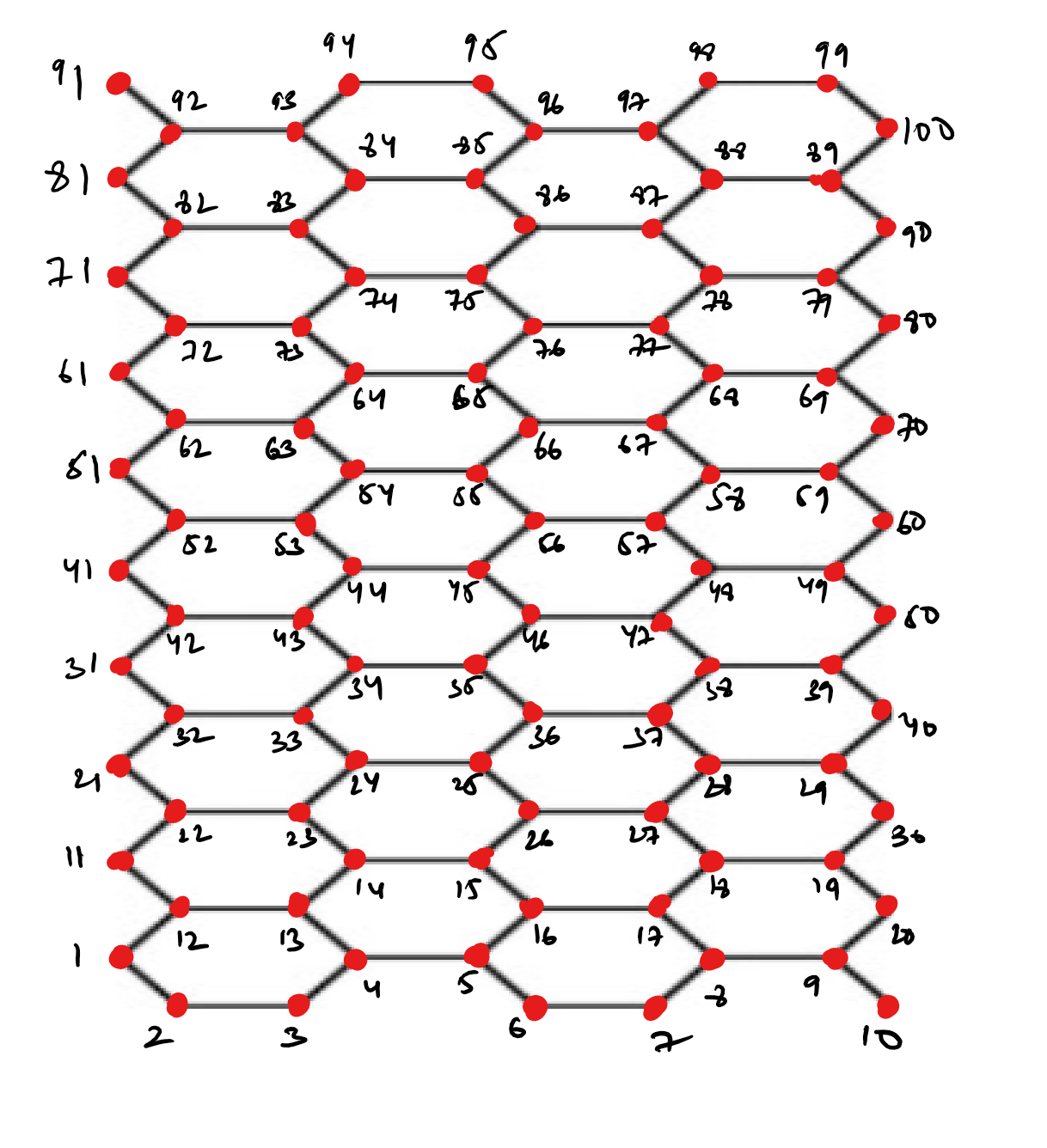


Figure 1: Sample of 10x10 honeycomb structure

* **Honeycomb with a random neighbor**: In the above-mentioned honeycomb structure, each node has been connected to an extra random node apart from its regular neighbors.

**Charts depicting behavior of various topologies:**

* Gossip Algorithm
* Push sum Algorithm

**General Observations:**

* Invariably, line topology performs the worst, as it takes time to converge, sending messages in only one direction.
* Convergence time for the full topology rises sharply, and sometimes can be seen performing even worse than line.
* Torus is the best performer out of all the topologies. This is because the number of connections for each node is 6 and this proves to be a great advantage.
* The patterns and behavior observed are similar for both, gossip and push sum algorithm.
* *Important observation in contrast to the graph: as per the graph for push sum algorithm, it seems that the random honeycomb topology performs better than torus, but that is not the case. Theoretically, we know that torus should perform the best, because not only does it have maximum connections at each node, but also because the entire system is very symetrically connected. The dijoint between this theory and practical observations is explained as follows:*

*We verfied this strange behavior and oberserved that the condition of terminating nodes when they don’t have active neighbors was causing this. Evidently, it is often that a node is stuck at a point with no active neighbor and that leads to our program being terminated. In a real world scenario, we could improve and optimize this behaviour by adding certain conditions to prevent the algorithm to being stuck.*