NTNU
Kunnskap for en bedre verden

Faculty of Information Technology and Electrical Engineering

Multi-Agent Optimization and Learning: Resilient and Adaptive Solutions - International Graduate School on Control 2023

Distributed optimization algorithms in NetLogo

Author:
Luka Grgičević

March, 2023

Table of Contents

1 Introduction

There are two methodologies for solving distributed optimization problems, that accompany each other, described in this report. Both of them achieve emergent properties that appear only by implementing local communication between nodes in the network. In the **consensus-based algorithms**, nodes converge to a uniform value either by taking the weighted average value in a synchronous manner or by a local broadcast technique that reaches the consensus even in the presence of lossy and noisy communication. There are 5 similar algorithms explained in the sub-chapters.

Gradient-based algorithms aren't free to define their value as any real number but each node may be bounded by a local cost function. Some key consensus ideas are incorporated for gradient-based methods to work. The goal is to find a minimum of the sum of those local cost functions. There are 3 algorithms explained in the sub-chapters. Some algorithms take into consideration a directed, one-way communication topology.

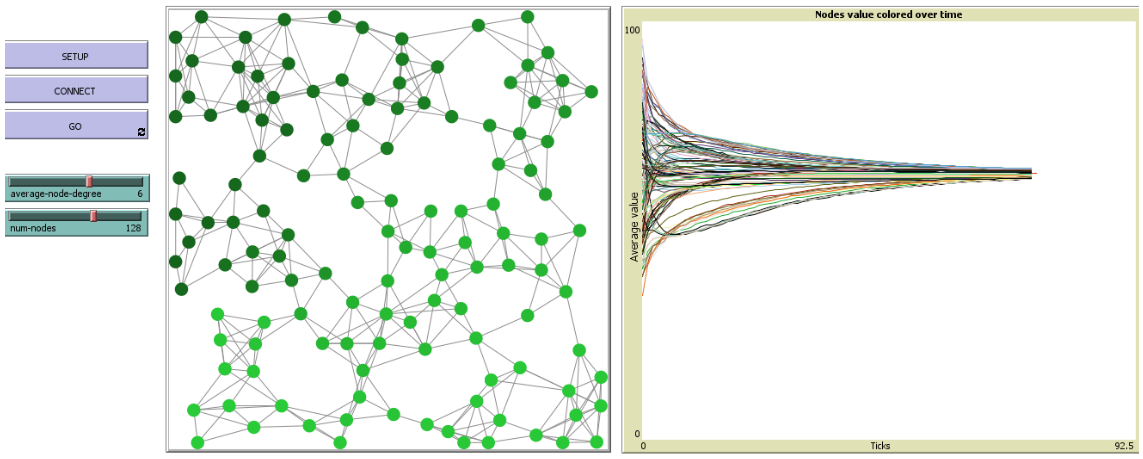
Each algorithm is briefly explained if there is no reference provided. They are implemented in NetLogo programing language. It is an intuitive platform for modelling multi-agent systems. Nodes are represented by *tutles* agent type and *links* agent type are the communication channels.

Each algorithm described has a code in the complementary file. NetLogo program has to be downloaded to run the code (https://ccl.northwestern.edu/netlogo/download.shtml). Simulations could simply be sped up by the slider above the interface.

2 Consensus-based algorithms

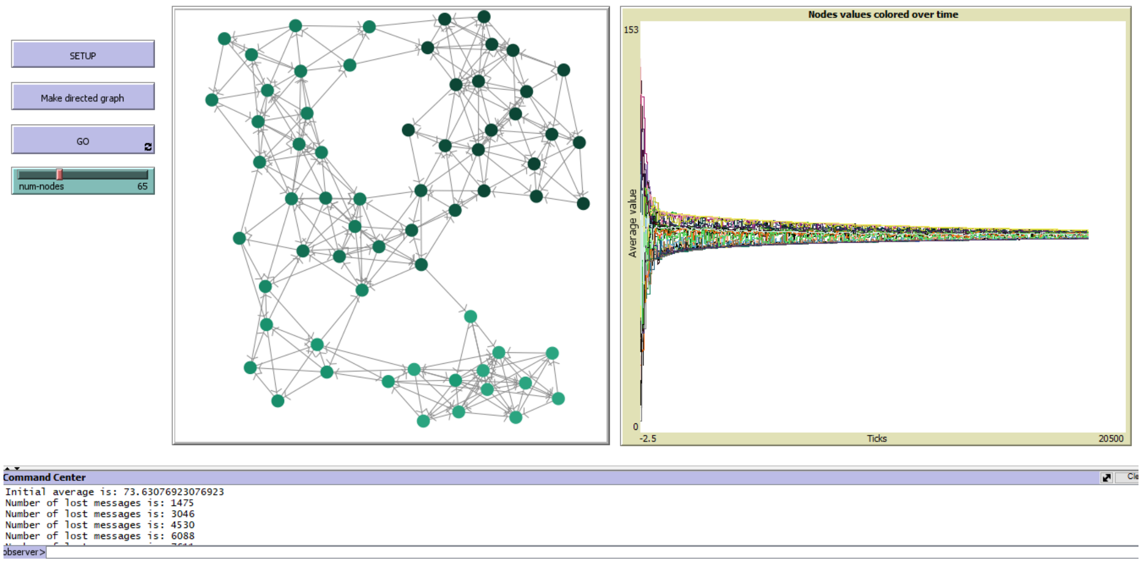
2.1 Ratio consensus

We start with initializing value y_i of each of the nodes in the range from 0 to 139 (colour palette in NetLogo) from the uniform distribution and initial weights sum s_i to one. Each node i in every loop pass (*tick*) then asks all of his neighbours j to send him their weighted values y_j. Weights d_i are related to the node degree. So, if the node has 4 neighbours the d_i will be 1/5. Node i then defines his new colour as his old colour multiplied by his weight plus the weighted sum of his neighbours, all divided by the sum of all the weights (s_i and s_j). So, the nodes i synchronously receive from their neighbours j their current value y_j and the weights s_j. The color is then updated by y_i/s_i. If the initial conditions for y_i and s_i are changed then the algorithm converges slower and not to the average of the initial values. The "Ratio consensus.nlogo" file is in the complimentary folder.

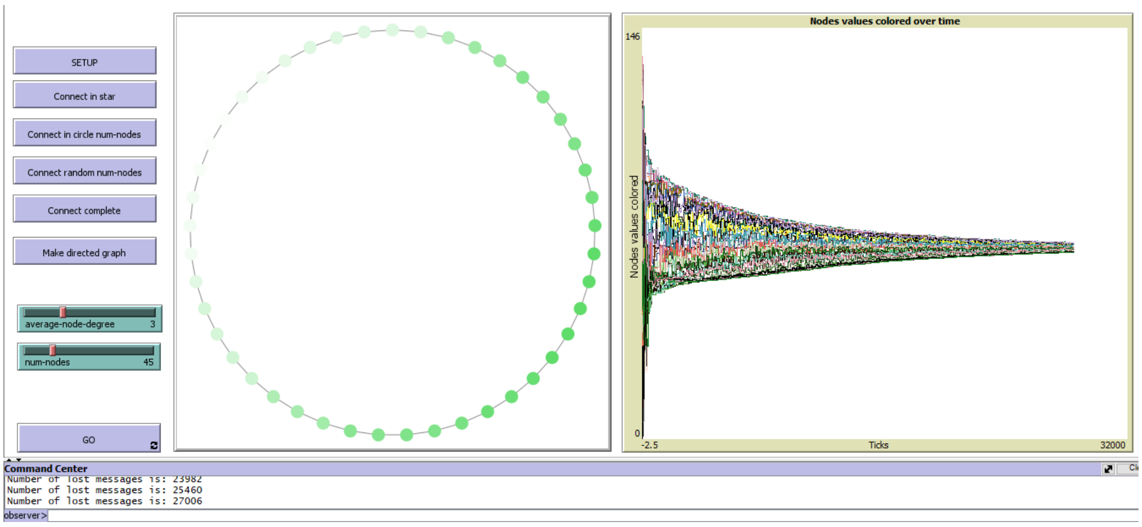


2.2 Robust consensus in directed graphs with packet loss

In this algorithm, each node i stores an integral of values sent and received from the neighbours j. So, those will be 4 lists of the length number of neighbours, two for y_i and two for s_i. Every *tick* one of the nodes is selected. He then updates his new values y_i and s_i by $y_i * d_i$ and $s_i * d_i$. The new color is now y_i/s_i. Newly computed values are added to the previous one in the lists that are storing sent data to the neighbours in the directed graph. Neighbours then subtract the previously saved amount with the received amount in their lists for that node i, for both of the values, y_j and s_j. Eventually they change the color to y_j/s_j. It is necessary to save the previously sent values to compensate for the packet loss. The packet loss is modelled by not sending the information over the channel if the node is randomly picked from the uniform distribution of neighbours' node numbers. The network is constructed so every node has at least one input and output channel. To run the "Robust consensus in directed graphs with packet loss.nlogo" click "SETUP", "Make directed graph" and then "GO". In the "Robust consensus symmetric graph topologies with packet loss.nlogo" one can choose different graph topologies.



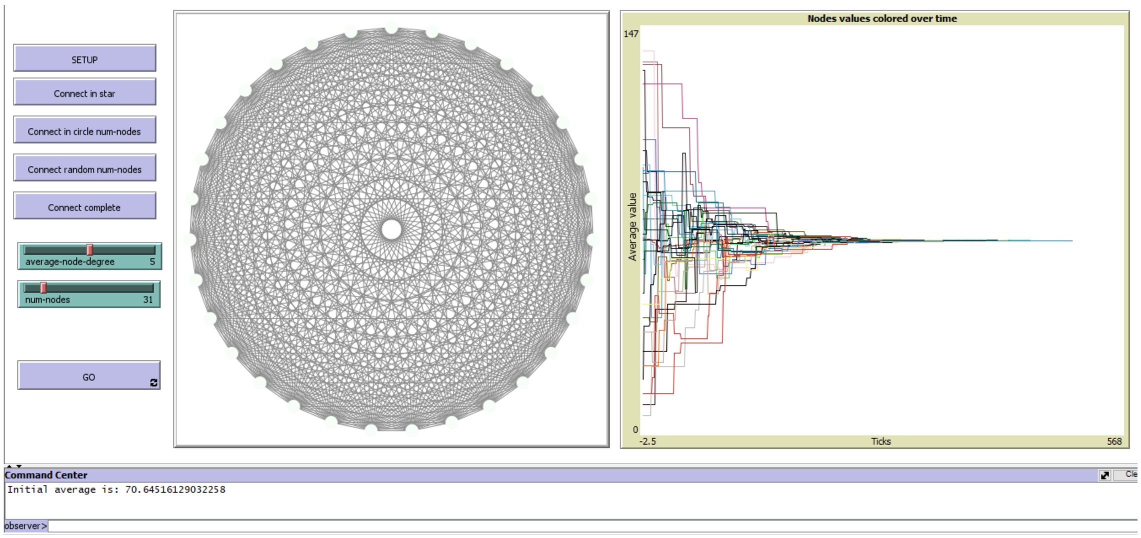
```
Initial average is: 73.63076923076923
Number of lost messages is: 1475
Number of lost messages is: 3046
Number of lost messages is: 4530
Number of lost messages is: 6088
```



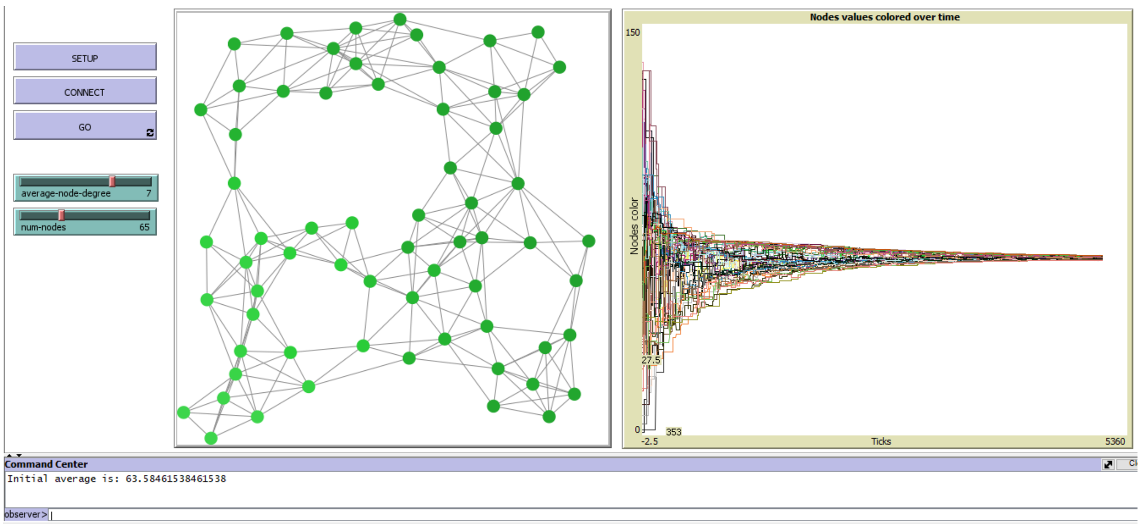
```
Number of lost messages is: 23982
Number of lost messages is: 25460
Number of lost messages is: 27006
```

2.3 Average consensus symmetric graph topologies

Every *tick* one link between the nodes is selected. The nodes then agree to change their values midway synchronously. In the ”Average consensus symmetric graph topologies.nlogo”, there are options to choose network topologies (*star*, *circle* or *complete*). There exist global variables to change the node degree and the number of nodes. In the ”Gosip symmetric algorithm.nlogo” there is an undirected random graph with the same principles.



```
Initial average is: 70.64516129032258
```

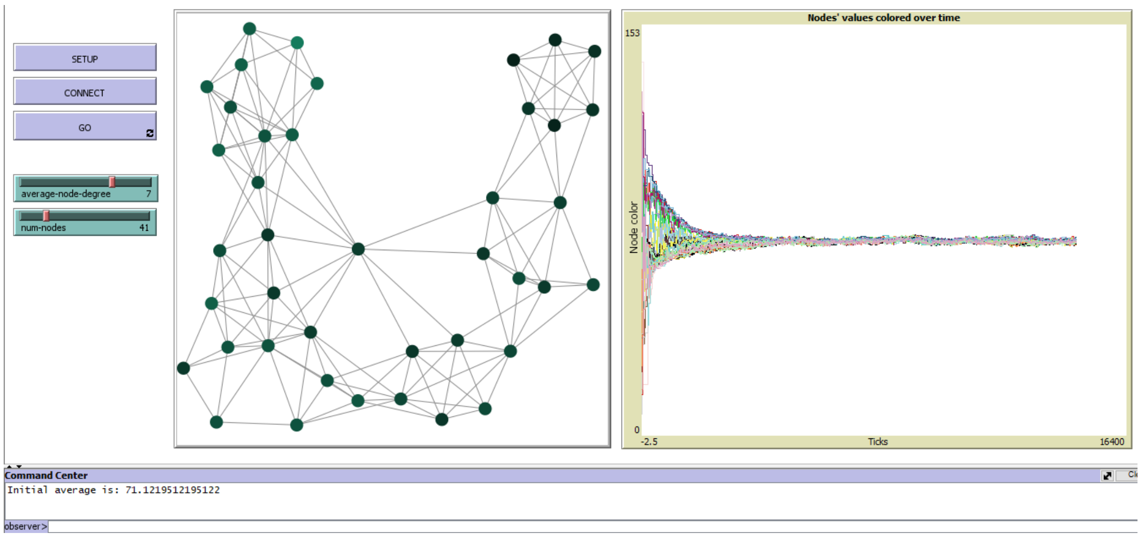


```
Initial average is: 63.58461538461538
```

2.4 Gossip symmetric algorithm plus white Gaussian noise

This algorithm has the same working principle as the previous one, only we added white Gaussian noise in the communication channel. In the image, a slight drift of the consensus can be depicted, which is a known consequence. To address the question of what happens if the graph is not connected, it is clear to see that the nodes' values are just going to decrease and converge to zero.

Of course, the natural way to prevent it would be not to do anything before we receive the packet from the node j.

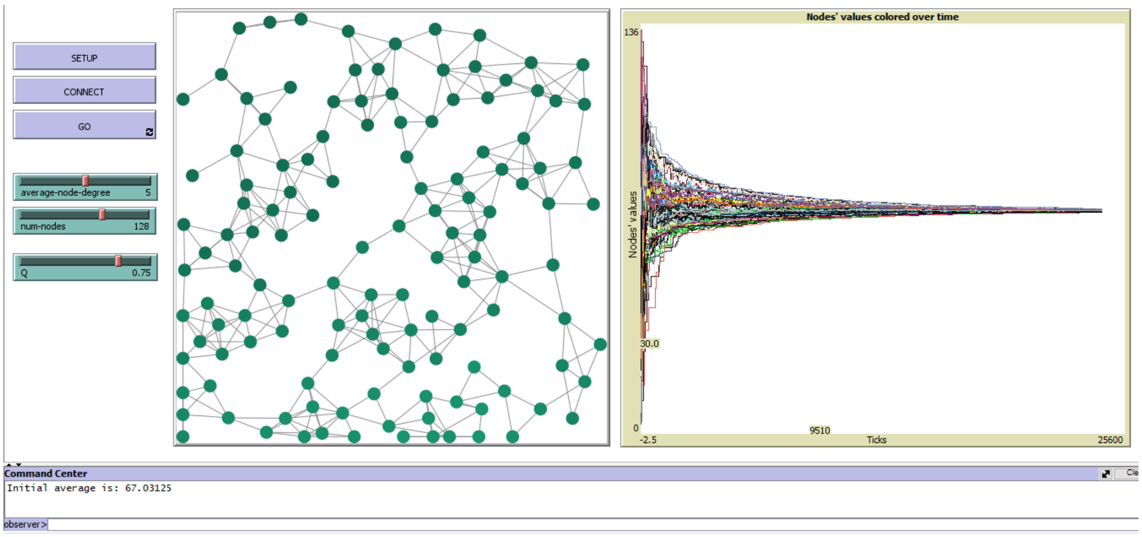


average-node-degree 7
num-nodes 41

```
Initial average is: 71.1219512195122
observer>
```

2.5 Asymmetric broadcast consensus

In this algorithm, one node i is activated and it broadcasts the colour to its neighbours j. Neighbours' new value x_j is then calculated to be somewhere in between the caller node i and node j, depending on the variable Q which determines the percentage node j is going to change its values toward the caller. The algorithm works also with multiple callers activated and broadcasting its colour to its neighbours at the same time.



average-node-degree 5
num-nodes 128
Q 0.75

```
Initial average is: 67.03125
observer>
```

3 Distributed optimization algorithms

3.1 Alternating direction method of multipliers (ADMM), edge-based classical implementation

In this case, we have an undirected graph. Each node has a local cost function $f_i(x_i)$. The goal is to minimize the sum of the local function only using communication with neighbours.

$$min \quad F(x) = \sum_i f_i(x_i)$$

such that

$$x_i = x_j$$

This is the implementation of the algorithm from [2]. Once a *tick* one edge e is activated. Nodes i and j compute their new values x_i^{k+1} and x_j^{k+1} based on the estimate (past values) of their neighbours sorted in the lists p and z (look below). After that, they exchange those values over the edge and update their lists z and p.

More precisely, we have a problem formulated as:

$$min \quad \sum_{x_i,z}^{N_i} f_i(x_i)$$

with constraints,

$$A_{ei}x_i = z_{ei}, \quad e = 1, ..., M, \quad i \in N(e)$$

N_i is a number of nodes. $N(e)$ number of edges connected to node i. M is the number of edges in the graph, N_i is the number of nodes. Matrix $A \in 2MxN$ and contains only 1, 0 and -1. Vectors are $x \in Nx1$ and $z \in 2Mx1$. Here we form an augmented Lagrangian function.

$$L(x, z, p) = F(x) - p^T(A_{ei}x_i - z) + \beta/2\|A_{ei}x_i - z\|_2^2$$

Vector $p \in Nx1$ is a vector of positive multipliers associated with each edge e_{ij}. This is a list stored in the memory of node i. Bridge variables $z_{ij}, z_{ji} \in z$ are assigned to each particular edge. Those variables are the same just different signs. β is positive *quadratic penalty* parameter. We follow the procedure for a primal update in a distributed fashion. $q \in i, j$ is for node i and node j.

$$x_q^{k+1} \in argmin_x L(x, z, p)$$

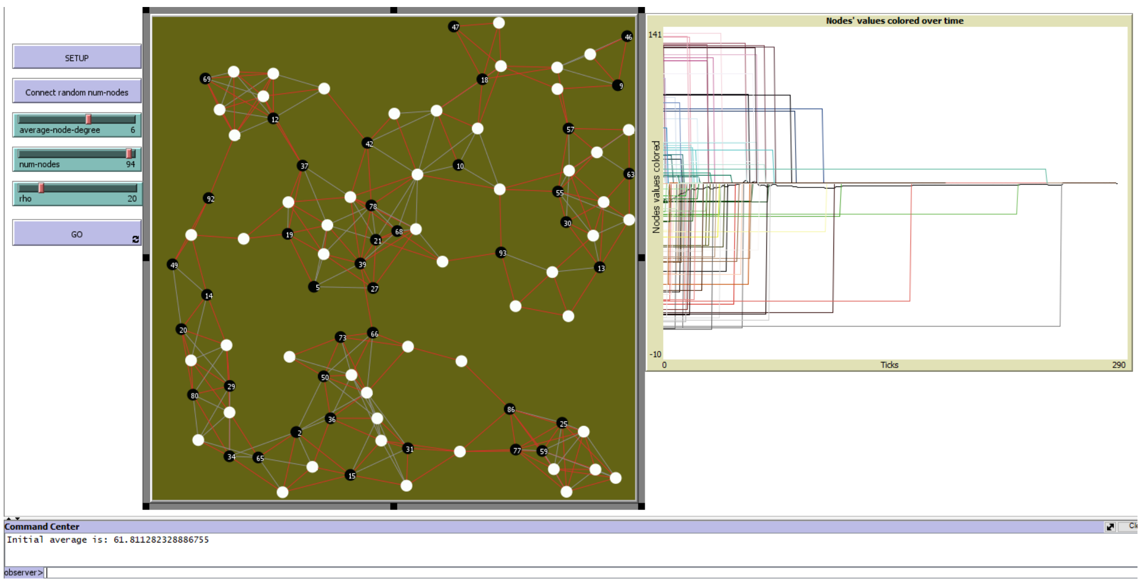
$$z_{ei}^{k+1}, z_{ej}^{k+1} \in argmin_{z_{ei},z_{ej}} L(x, z, p)$$

Then we update the dual variable p:

$$p_{eq}^{k+1} = p_{eq}^k - \beta(A_{eq}x_q^{k+1} - z_{eq}^{k+1})$$

In the code, x_q^{k+1} update is solved numerically by searching for zeros of the function's explicit derivative. The local cost function is of the form:

$$f_i(x) = c_i e^{a_i x} + d_i e^{-b_i x}, \quad a_i, b_i \sim \cup(0, 0.2], \quad c_i, d_i \sim \cup(0, 1]$$



average-node-degree 6
num-nodes 94
rho 20

```
Initial average is: 61.811282328886755
```

observer>

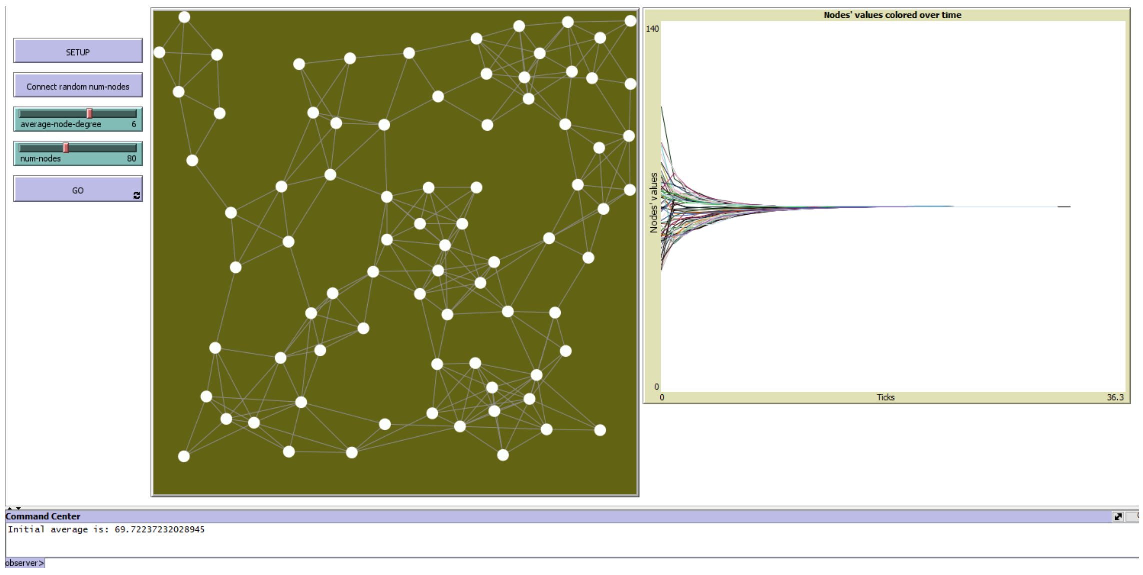
Please refer to chapter III in the [2] for a detailed description. See [1] for the more advanced version of ADMM.

3.2 Distributed gradient descent

In this algorithm, we have an undirected graph where each node is updating its variable based on the consensus X_c^k at *tick* k and discounted gradient $f_i'(x_i)^k$ by $\alpha \in U(0,1]$, which represents the learning rate. The local cost function and the goal are the same as in the previous example.

$$x_i^{k+1} = X_c^k - \alpha * f_i'(x_i)^k$$
$$X_c^k = d_i * x_i^k + \sum_j d_i * x_j^k$$

Where d_i is a ratio $1/(1 + \sum j)$.



average-node-degree 6
num-nodes 80

```
Initial average is: 69.72237232028945
```

observer>

3.3 Asynchronous broadcast consensus, Newton-Raphson method with packet loss (ra-NRC)

This algorithm is NetLogo implementation from the research paper in [3]. Since we should account for packet loss we have to implement mass preservation property. In other words to follow the packets sent and received from each node i. This means having 4 lists, 2 for each variable y_i and s_i for packets sent to and received from node j. We define two functions g_i and h_i.

$$g_i(x_i) = h_i * x_i + f_i'(x_i)$$
$$h_i(x_i) = f_i''(x_i)$$

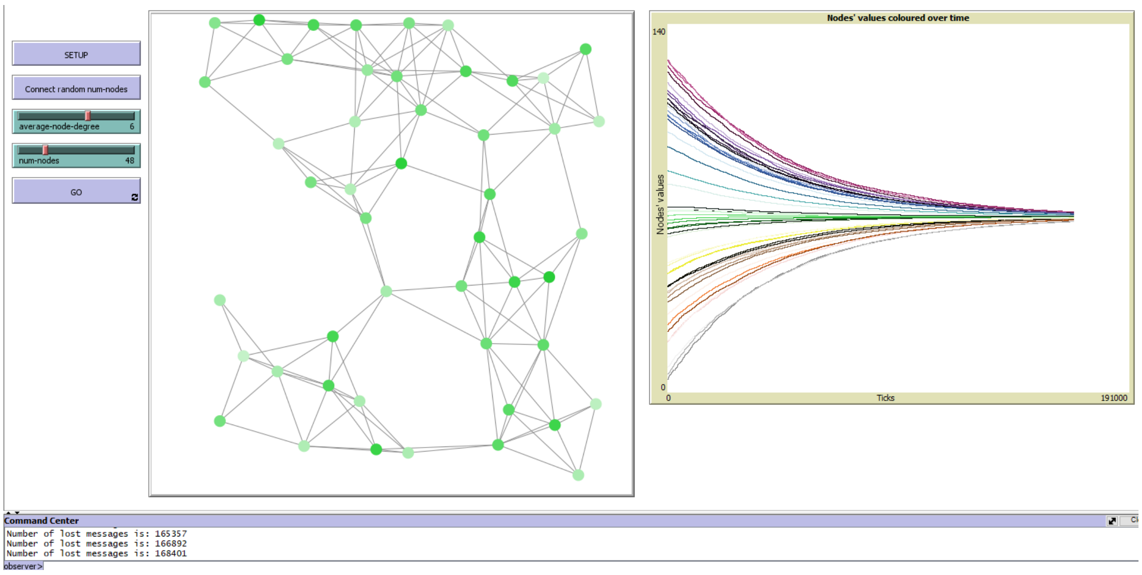
Ones a *tick* one node is updating his variables s and y and computing x_i^{k+1}

$$y_i^{k+1} = d_i * (y_i^k + g_i^{k+1} - g_i^k)$$
$$s_i^{k+1} = d_i * (s_i^k + h_i^{k+1} - h_i^k)$$

$$x_i^{k+1} = (1 - \eta) * x_i^k + \eta * \frac{y_i^{k+1}}{s_i^{k+1}}$$

Node i is then adding variables s and y to the sum of packets sent to the neighbouring nodes and broadcast this information. Nodes j find a difference Δ between the new sum received and the sum it previously had from that node. Then it updates variables s and y but doesn't change the value x. Learning factor $\eta \in U(0, 1]$ depends on network complexity and choosing it to be too big can cause instability and the algorithm doesn't converge. On the other hand, choosing it to be too small is good is slow. There is a point during the simulation and scheduled broadcasting where η can be increased to speed up the consensus. It is depicted in the last image where η was changed from 0,001 to 0,01 and then to 0,1.

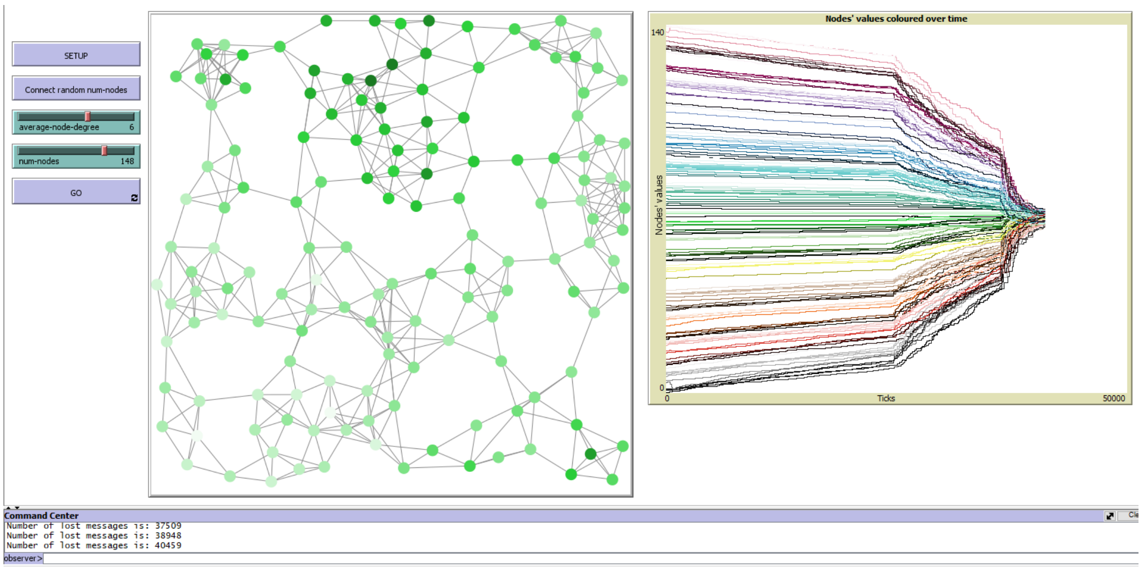
$$y_j^{k+1} = \Delta_y + y_j^k + g_j^{k+1} - g_j^k$$
$$s_j^{k+1} = \Delta_s + s_j^k + h_j^{k+1} - h_j^k$$



average-node-degree 6

num-nodes 48

```
Number of lost messages is: 165357
Number of lost messages is: 166892
Number of lost messages is: 168401
observer>
```



average-node-degree 6
num-nodes 148

Command Center
```
Number of lost messages is: 37509
Number of lost messages is: 38948
Number of lost messages is: 40459
observer>
```

4 Info

This is the report from the course at the International Graduate School on Control named "Multi-Agent Optimization and Learning: Resilient and Adaptive Solutions", organized by the European Embedded Control Institute (http://www.eeci-igsc.eu/) at the University of Paris-Saclay. It is one of the 23 modules offered in 2023. lasting one week each. Lecturers were Nicola Bastianello [1], Ruggero Carli [2] and Luca Schenato [3]. Apart from the listed bibliography, lecturers also provided additional scripts, slides and reading references that might be obtainable upon request. The NetLogo code can be found on the author's Github website [4]

Bibliography

[1] Nicola Bastianello et al. 'Asynchronous Distributed Optimization Over Lossy Networks via Relaxed ADMM: Stability and Linear Convergence'. In: *IEEE Transactions on Automatic Control* 66.6 (2021), pp. 2620–2635. DOI: 10.1109/TAC.2020.3011358.

[2] Ermin Wei and Asuman Ozdaglar. 'On the O(1/k) Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers'. In: *2013 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2013 - Proceedings* (July 2013). DOI: 10.1109/GlobalSIP.2013.6736937.

[3] Filippo Zanella et al. 'Newton-Raphson Consensus for Distributed Convex Optimization'. In: *IEEE Transactions on Automatic Control* 61 (Dec. 2011), pp. 5917–5922. DOI: 10.1109/CDC.2011.6160605.

[1] https://scholar.google.com/citations?user=HGHD4SIAAAAJ&hl=en
[2] https://scholar.google.com/citations?user=3-U0bHQAAAAJ&hl=th
[3] https://scholar.google.com/citations?user=LTL9MjwAAAAJ&hl=en
[4] https://github.com/GrgicevicLukaNTNU